future internet

Special Issue Reprint

Edge Intelligence

Edge Computing for 5G and the Internet of Things

Edited by
Yuezhi Zhou and Xu Chen

mdpi.com/journal/futureinternet

MDPI

Edge Intelligence: Edge Computing for 5G and the Internet of Things

Edge Intelligence: Edge Computing for 5G and the Internet of Things

Guest Editors

Yuezhi Zhou
Xu Chen

MDPI

Basel • Beijing • Wuhan • Barcelona • Belgrade • Novi Sad • Cluj • Manchester

Guest Editors

Yuezhi Zhou
Department of Computer Science and Technology
Tsinghua University
Beijing
China

Xu Chen
School of Computer Science and Engineering
Sun Yat-sen University
Guangzhou
China

Editorial Office
MDPI AG
Grosspeteranlage 5
4052 Basel, Switzerland

This is a reprint of the Special Issue, published open access by the journal *Future Internet* (ISSN 1999-5903), freely accessible at: www.mdpi.com/journal/futureinternet/special_issues/173F7J2PUI.

For citation purposes, cite each article independently as indicated on the article page online and as indicated below:

Lastname, A.A.; Lastname, B.B. Article Title. *Journal Name* **Year**, *Volume Number*, Page Range.

ISBN 978-3-7258-3950-6 (Hbk)
ISBN 978-3-7258-3949-0 (PDF)
https://doi.org/10.3390/books978-3-7258-3949-0

Contents

About the Editors

Yuezhi Zhou

Yuezhi Zhou (Senior Member, IEEE) received his PhD in computer science and technology from Tsinghua University, Beijing, China, in 2004. He worked as a visiting scientist with the Computer Science Department at Carnegie Mellon University, in 2005. He is now working as a professor with the Department of Computer Science and Technology, Tsinghua University. He has authored or coauthored more than 100 technical papers in international journals or conferences. His research interests include pervasive computing, distributed systems, and intelligent computing systems. He received the Best Paper Award at the IEEE International Conference on Advanced Information Networking and Applications (AINA) 2007 and IEEE International Conference on High Performance Computing and Communications (HPCC) 2014.

Xu Chen

Dr. Xu Chen is currently a full professor and assistant dean at the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China. He obtained his Ph.D. in Information Engineering from The Chinese University of Hong Kong, Hong Kong, China, in 2012. Before that, he received his B.Eng. degree in Electronic Engineering from South China University of Technology, Guangzhou, China, in 2008. From August 2012 to February 2014, Dr. Chen was a postdoctoral research fellow with Arizona State University, USA. From April 2014 to Aug. 2016, he was a Humboldt scholar fellow with the Institute of Computer Science of University of Goettingen, Germany.

future internet

MDPI

Editorial

Edge Intelligence: Edge Computing for 5G and the Internet of Things

Yuezhi Zhou [1,*] and Xu Chen [2]

[1] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[2] Department of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China; chenxu35@mail.sysu.edu.cn
* Correspondence: zhouyz@mail.tsinghua.edu.cn

1. Introduction

In recent years, the field of artificial intelligence has made a revolutionary breakthrough. The remarkable success of deep learning technologies and large language models (LLMs) has fundamentally reshaped industrial processes and societal dynamics, with their transformative impact poised to persist in the foreseeable future. With the continuous advancement of artificial intelligence (AI) technology, edge intelligence has emerged as a new research hotspot. Just as edge computing was developed to provide faster and lower-latency computing and storage services compared to cloud computing, the development of edge intelligence aims to deliver enhanced responsiveness and diversified intelligent services that surpass conventional cloud-based AI implementations. The emergence of edge intelligence not only brings AI closer to users but also provides new technological support for the development and application of 5G and the Internet of things (IoT).

Specifically, edge intelligence will promote innovative development in the fields of 5G and the IoT in at least three areas. First, it will reconstruct the network infrastructure of 5G and the IoT, making networks more efficient and flexible, thereby laying a solid foundation for intelligent network services and applications. Second, it will provide faster and more diverse intelligent service support for end-user applications. While enhancing existing applications with faster and more personalized intelligent services, it will also give rise to numerous innovative intelligent applications, such as augmented reality and virtual reality. Last but not least, edge intelligence offers new solutions to the technical challenges faced in the era of 5G and the IoT, for example, how to efficiently and rapidly utilize large amounts of local data generated by numerous IoT devices in a manner that protects user privacy.

While demonstrating significant potential, edge intelligence remains at a nascent stage of development, requiring in-depth academic research and continuous industrial exploration. To promote the development of edge intelligence, this Special Issue aims to gather recent advances and novel contributions in the areas of edge intelligence and edge computing for 5G and IoT networks, providing opportunities for academic researchers and industry practitioners to exchange creative ideas, with the goal of inspiring further innovation and fostering progress in academia and industry.

2. An Overview of Published Articles

Yushan Li's article (contribution one) proposes a novel edge-based programming framework tailored for real-time and distributed applications in smart transportation systems. It leverages the Elixir programming language, event-driven architecture (EDA), and the MQTT protocol to enhance edge-based vehicular systems. The proposed solution

Citation: Zhou, Y.; Chen, X. Edge Intelligence: Edge Computing for 5G and the Internet of Things. *Future Internet* **2025**, *17*, 101. https://doi.org/10.3390/fi17030101

Received: 16 February 2025
Accepted: 18 February 2025
Published: 23 February 2025

Citation: Zhou, Y.; Chen, X. Edge Intelligence: Edge Computing for 5G and the Internet of Things. *Future Internet* **2025**, *17*, 101. https://doi.org/10.3390/fi17030101

enhances system performance, achieving low-latency response times, high-concurrency handling, and fault tolerance, as demonstrated through two case studies: a traffic light system and a cab dispatch prototype. This work not only contributes to the development of more scalable and responsive applications in edge computing but also highlights the potential of Elixir in smart transportation applications, paving the way for future research and practical deployment in edge-based smart transportation systems.

The article by Gulshan Saleem et al. (contribution two) introduces Edge-Enhanced TempoFuseNet, a cutting-edge framework designed to address the challenges of intelligent multiclass video anomaly recognition in low-quality surveillance video analytics within 5G and IoT environments. It tackles issues such as poor lighting and low spatial resolution, which are prevalent in real-world surveillance scenarios. The framework employs a two-stream architecture combining spatial and temporal feature extraction for anomaly classification, enhanced by a super-resolution technique using a pre-trained StyleGAN to improve video quality. This approach significantly outperforms traditional methods, achieving high accuracy and low false positive rates in anomaly classification. The study's significance lies in its ability to enhance the effectiveness of video surveillance systems by improving anomaly detection accuracy, thereby contributing to safer and more efficient monitoring solutions in various applications such as security, traffic monitoring, emergency response, and behavior analysis.

The third article is from Rafael Moreno-Vozmediano et al. (contribution three) and presents a comprehensive study on intelligent resource orchestration for 5G edge infrastructures, addressing the challenges of efficiently managing and optimizing resource allocation in distributed edge environments. The proposed Smart 5G Edge-Cloud Management Architecture integrates predictive intelligence mechanisms for workload forecasting and optimization algorithms to achieve the optimal allocation of virtual resources across multiple edge locations. By leveraging various prediction and optimization mechanisms, such as Bayesian regression for CPU usage prediction and integer linear programming for resource optimization, the study demonstrates significant improvements in managing latency-sensitive and data-intensive applications. This work is significant as it provides a robust framework for intelligent edge orchestration, which is crucial for the deployment of next-generation applications in 5G networks.

The subsequent contribution to this Special Issue, written by Laura Lemmi et al. (contribution four), addresses the challenge of ensuring service continuity for edge computing applications in 5G mobile networks, particularly when workloads need to be migrated between edge nodes due to user mobility. The authors propose an innovative solution that integrates Segment Routing over IPv6 (SRv6) into the 5G core data plane alongside the ETSI multi-access edge computing (MEC) architecture. This approach allows for lossless workload migration by implementing a packet buffer as a virtual network function (VNF) within the SRv6 framework, ensuring that services can follow users without packet loss or significant performance degradation. The solution is evaluated through a small-scale testbed and the results demonstrate higher scalability and flexibility compared to traditional IPv6 routing methods, showcasing the potential of SRv6 for enhancing service continuity in dynamic 5G edge environments.

The article by Daniel Christian Lawo et al. (contribution five) addresses the critical issue of securing Internet Protocol security (IPsec) tunnels against future quantum computing threats by integrating post-quantum cryptography (PQC) algorithms. The authors experimentally demonstrate the first IPsec tunnel secured by PQC algorithms such as Falcon, Dilithium, and Kyber, deploying it in two scenarios: a high-performance data center environment with a 100 Gbit/s line rate throughput and a wireless client-to-cloud connection with a 0.486 Gbit/s throughput. The significance of this work lies in its contribu-

tion to the development of quantum-resistant communication protocols and the adoption of PQC in existing network infrastructures, ensuring data security in the post-quantum era.

In the sixth article, Yi Liu and Leonard Barolli (contribution six) tackle the critical issue of determining driver anxiety levels while driving, which is crucial for improving driving safety and reducing the risk of traffic accidents. The authors propose an intelligent system based on fuzzy logic to decide a driver's anxiety level (DAL) by considering various factors, including driving experience, in-car environment conditions, and driver age. They implement and compare two fuzzy-based models (DALM1 and DALM2), with DALM2 incorporating an additional parameter related to accident anxiety state. The simulation results show that the system can effectively evaluate driver anxiety levels, with DALM2 providing more accurate assessments due to its more comprehensive rule base. The study bridges the gap between psychological and technical approaches in assessing driver anxiety, providing a reliable and explainable method for real-time driver state evaluation in intelligent transportation systems.

The research from Youssef Abadade et al. (contribution seven) delves into the problems faced by diagnosing respiratory diseases, particularly asthma, using lung sound recordings. Traditional diagnostic methods, such as the stethoscope, are limited by their reliance on physician expertise, lack of recording functionality, and inability to filter out noise. Cloud-based deep learning solutions, while effective, often face issues related to latency, Internet dependency, and privacy concerns. To overcome these limitations, the authors developed Tiny Machine Learning (TinyML) models that can be deployed on low-power, cost-effective devices like digital stethoscopes to provide the real-time and accurate diagnosis of respiratory conditions. They trained and compared three models—a custom CNN, an Edge Impulse CNN, and a custom LSTM—using a publicly available lung sound dataset. The custom CNN achieved the highest accuracy while maintaining moderate resource usage. This work highlights the potential of TinyML to enhance healthcare by providing accessible, reliable diagnostic tools, especially in remote and underserved areas.

Sakshi Patni and Joohyung Lee's study (contribution eight) presents EdgeGuard, a novel decentralized architecture that addresses the challenges of data privacy, security, and resource management in Internet of Medical Things (IoMT) networks. It integrates blockchain technology, federated learning, and edge computing to enable secure and efficient medical resource orchestration. The key contributions include a federated learning algorithm optimized for medical data with differential privacy, a lightweight blockchain consensus mechanism tailored for IoMT devices to ensure data integrity and security, and an adaptive edge resource allocation method to enhance system scalability and efficiency. Additionally, it introduces an access control system based on smart contracts and a secure multi-party computing protocol for model updates. EdgeGuard significantly improves computational performance, data value, and privacy protection compared to existing solutions. This work provides a comprehensive and effective framework for secure and efficient medical data management in IoMT ecosystems.

The work from Niroshinie Fernando et al. (contribution nine) proposes a novel three-tier architecture integrating edge, fog, and cloud computing to optimize the distributed processing of IoT data. It addresses the challenges of task allocation across heterogeneous resources, dynamic node availability, and maintaining Quality of Service (QoS) in a cooperative edge–fog–cloud environment. The key contributions include the development of a conceptual architecture for dynamic collaboration among these tiers, the implementation of a real-world prototype, and empirical evaluations demonstrating significant improvements in performance, energy consumption, and cost reduction. The results show speedups of up to 7.5 times and energy savings of up to 80%, highlighting the effectiveness of the proposed architecture in supporting the dynamic computational needs of IoT ecosystems.

The last article published in this Special Issue is from Zhiyuan Wang and Yuezhi Zhou (contribution ten), and it provides an in-depth analysis and evaluation of Intel Software Guard Extension (SGX)-based Trusted Execution Environment (TEE) usage in IoT and edge intelligence scenarios. It identifies and addresses key challenges, such as performance overhead, I/O security issues, and vulnerability to side-channel attacks that arise when applying a TEE in resource-constrained edge environments. The study conducts extensive experiments on different SGX implementations, including those based on SGX SDK and LibOS, revealing significant performance bottlenecks and security limitations. It offers critical insights into the limitations of current SGX solutions and proposes performance optimization strategies, such as optimizing enclave entry and exit, bypassing the kernel for I/O operations, and adopting confidential virtual machine (CVM)-based TEE. These contributions provide valuable benchmarks and practical approaches for improving the integration of TEE in IoT and edge intelligence scenarios.

3. Conclusions and Future Directions

This Special Issue on "Edge Intelligence: Edge Computing for 5G and the Internet of Things" showcases the latest advancements and contributions in leveraging edge computing and edge intelligence to empower 5G and the Internet of Things. The articles published in this Special Issue span a diverse range of topics, including innovative programming frameworks, intelligent resource orchestration, secure communication protocols, and healthcare applications. These studies collectively highlight how edge intelligence can revolutionize network infrastructure, enable innovative applications, and address technical challenges facing 5G and the IoT. Together, they underscore the potential and importance of edge intelligence for the future of 5G and the IoT.

Edge intelligence is expected to transform the way we interact with cloud-based AI by bringing AI closer to the end users, thereby enhancing the capabilities of 5G and the IoT. Here are some key future directions for edge intelligence:

Enhanced AI Models for Edge Intelligence: Future research should focus on developing more efficient and lightweight AI models that can run on resource-constrained edge devices without compromising performance. Techniques like model pruning, quantization, and federated or split learning will be crucial in achieving this goal.

Interoperability and Standardization: As edge intelligence continues to evolve, there will be a growing need for standardized protocols and frameworks to ensure interoperability between different edge devices and platforms. This will facilitate seamless integration and communication across diverse 5G and IoT ecosystems.

Security and Privacy: With the increasing adoption of edge intelligence, ensuring the security and safety of intelligent systems and protecting user data privacy and security will remain critical challenges. Future developments should focus on advanced encryption methods, secure communications, robust authentication mechanisms, and trusted or confidential computing technologies to protect sensitive applications and data at the edge.

Dynamic and Scalable Resource Management: Future edge computing systems must be scalable and capable of dynamically managing resources to handle varying workloads and network conditions. Intelligent orchestration frameworks that can predict and optimize resource allocation in real time will be crucial.

Human-Centric Applications: Developing edge intelligence solutions that prioritize human-centric applications, such as healthcare, smart homes, and smart transportation systems, will be an important direction. These applications should leverage edge intelligence to provide more intuitive, responsive, and user-friendly experiences.

By addressing these future directions, edge intelligence can continue to evolve and play a pivotal role in shaping the next generation of 5G and IoT technologies, ultimately leading to a smarter, more efficient, and secure digital ecosystem.

Author Contributions: Conceptualization and methodology, Y.Z.; investigation and validation, X.C.; writing—original draft preparation, Y.Z.; writing—review and editing, X.C.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the National Key Research and Development Program of China under grant no. 2023YFB4503000.

Acknowledgments: We would like to thank all the authors who have contributed their papers to this Special Issue. Their efforts and insights have enriched the content and scope of this collection. Our sincere appreciation also goes to the dedicated reviewers, whose meticulous, responsible, and impartial evaluation not only ensured the selection of high-quality papers but also provided insightful feedback to enhance the quality of the submissions. We are honored to acknowledge the Editorial Board of *Future Internet* for entrusting us with the role of Guest Editors for this Special Issue. Finally, we would like to express our profound thanks to the Editorial Office staff for their rigorous oversight and dedicated management, which have ensured the timely and successful publication of this Special Issue.

Conflicts of Interest: The authors declare no conflicts of interest.

List of Contributions

1. Li, Y.; Fujita, S. A Synergistic Elixir-EDA-MQTT Framework for Advanced Smart Transportation Systems. *Future Internet* **2024**, *16*, 81. https://doi.org/10.3390/fi16030081.
2. Saleem, G.; Bajwa, U.; Raza, R.; Zhang, F. Edge-Enhanced TempoFuseNet: A Two-Stream Framework for Intelligent Multiclass Video Anomaly Recognition in 5G and IoT Environments. *Future Internet* **2024**, *16*, 83. https://doi.org/10.3390/fi16030083.
3. Moreno-Vozmediano, R.; Montero, R.; Huedo, E.; Llorente, I. Intelligent Resource Orchestration for 5G Edge Infrastructures. *Future Internet* **2024**, *16*, 103. https://doi.org/10.3390/fi16030103.
4. Lemmi, L.; Puliafito, C.; Virdis, A.; Mingozzi, E. SRv6-Based Edge Service Continuity in 5G Mobile Networks. *Future Internet* **2024**, *16*, 138. https://doi.org/10.3390/fi16040138.
5. Lawo, D.; Abu Bakar, R.; Cano Aguilera, A.; Cugini, F.; Imaña, J.; Tafur Monroy, I.; Vegas Olmos, J. Wireless and Fiber-Based Post-Quantum-Cryptography-Secured IPsec Tunnel. *Future Internet* **2024**, *16*, 300. https://doi.org/10.3390/fi16080300.
6. Liu, Y.; Barolli, L. An Intelligent System for Determining Driver Anxiety Level: A Comparison Study of Two Fuzzy-Based Models. *Future Internet* **2024**, *16*, 348. https://doi.org/10.3390/fi16100348.
7. Abadade, Y.; Benamar, N.; Bagaa, M.; Chaoui, H. Empowering Healthcare: TinyML for Precise Lung Disease Classification. *Future Internet* **2024**, *16*, 391. https://doi.org/10.3390/fi16110391.
8. Patni, S.; Lee, J. EdgeGuard: Decentralized Medical Resource Orchestration via Blockchain-Secured Federated Learning in IoMT Networks. *Future Internet* **2025**, *17*, 2. https://doi.org/10.3390/fi17010002.
9. Fernando, N.; Shrestha, S.; Loke, S.; Lee, K. On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet* **2025**, *17*, 22. https://doi.org/10.3390/fi17010022.
10. Wang, Z.; Zhou, Y. Analysis and Evaluation of Intel Software Guard Extension-Based Trusted Execution Environment Usage in Edge Intelligence and Internet of Things Scenarios. *Future Internet* **2025**, *17*, 32. https://doi.org/10.3390/fi17010032.

future internet

MDPI

Article

A Synergistic Elixir-EDA-MQTT Framework for Advanced Smart Transportation Systems

Yushan Li [1,2] and Satoshi Fujita [1,2,*]

[1] Graduate School of Advanced Science and Engineering, Hiroshima University, Higashi-Hiroshima 739-0046, Japan; yushanli433@gmail.com
[2] Department of Information Engineering, Hiroshima University, Higashi-Hiroshima 739-0046, Japan
* Correspondence: satoshi.fujita.g@gmail.com

Abstract: This paper proposes a novel event-driven architecture for enhancing edge-based vehicular systems within smart transportation. Leveraging the inherent real-time, scalable, and fault-tolerant nature of the Elixir language, we present an innovative architecture tailored for edge computing. This architecture employs MQTT for efficient event transport and utilizes Elixir's lightweight concurrency model for distributed processing. Robustness and scalability are further ensured through the EMQX broker. We demonstrate the effectiveness of our approach through two smart transportation case studies: a traffic light system for dynamically adjusting signal timing, and a cab dispatch prototype designed for high concurrency and real-time data processing. Evaluations on an Apple M1 chip reveal consistently low latency responses below 5 ms and efficient multicore utilization under load. These findings showcase the system's robust throughput and multicore programming capabilities, confirming its suitability for real-time, distributed edge computing applications in smart transportation. Therefore, our work suggests that integrating Elixir with an event-driven model represents a promising approach for developing scalable, responsive applications in edge computing. This opens avenues for further exploration and adoption of Elixir in addressing the evolving demands of edge-based smart transportation systems.

Keywords: Elixir; edge computing; event-driven architecture; concurrency; smart transportation

Citation: Li, Y.; Fujita, S. A Synergistic Elixir-EDA-MQTT Framework for Advanced Smart Transportation Systems. *Future Internet* **2024**, *16*, 81. https://doi.org/10.3390/fi16030081

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 20 January 2024
Revised: 22 February 2024
Accepted: 25 February 2024
Published: 28 February 2024

1. Introduction

1.1. Background

With the rapid development of Internet of Things (IoT), an exponentially increasing number of smart devices are being connected, generating massive amounts of real-time data that need to be processed instantly. Traditional cloud computing architectures, relying on centralized data centers, are insufficient to meet the low-latency and location awareness requirements of many emerging IoT applications [1]. This has led to growing interest in edge computing, which pushes computation and data storage closer to the location where the data are generated. By processing data at the edge, latency can be reduced significantly while also decreasing bandwidth usage [2].

The global market for the Internet of Things was estimated to be worth around USD 182 billion in 2020 [3], and it is anticipated to triple in size by 2030, reaching over USD 621 billion. At the same time, according to a report by Grand View Research, the global edge computing market size is expected to reach USD 43.4 billion by 2027 [4], driven by the increasing adoption of IoT devices and the need for real-time data analysis and processing at the network edge.

However, existing edge computing solutions predominantly employ imperative programming languages like C/C++, Java, and Python, which incur complexity in developing and maintaining large applications. The tight coupling between components, lack of fault-tolerance mechanisms, and single-threaded execution models in these languages make

them ill suited for the dynamic and distributed nature of edge computing systems. To overcome these limitations, we propose the use of Elixir, a modern functional programming language built on the robust Erlang Virtual Machine, for building highly available and fault-tolerant applications for edge computing.

In our previous work [5], we conducted experiments comparing an Elixir-based message broker with an equivalent Rust implementation under different network conditions. The results validated Elixir's resilience and low latency, confirming its potential for edge computing deployments. Building on these findings, in this paper, we propose a novel programming framework in the edge computing paradigm. The framework combines the benefits of Elixir language, and event-driven architecture, with MQTT protocol. We demonstrate two use cases in the field of smart transportation applications: a traffic light system that optimizes traffic flow in the intersection, and a cab dispatch system that coordinates taxis and passengers based on real-time location data. The evaluation of the cab dispatch system shows the low latency and good performance of the system.

In this paper, we will start our exploration with the introduction. Subsequently, we will provide the essential properties of the language we use in Section 2. In Section 3, we will introduce our architecture design in detail. Two specific case studies are demonstrated in the next section. We next discuss the details of the prototype system in Section 5. Then, Section 6 is provided as the evaluation part. To conclude, we synthesize the key findings and implications of our research in the last section.

This paper is an extended version of a paper [5] presented at CANDAR 2023. The difference to the conference version is summarized as follows: (1) we add recent papers concerned with smart transportation systems and the application of event-driven architecture in smart cities as related work; (2) we add event-driven architecture as an important component in our proposed framework, and utilize it in a smart transportation application; (3) in the evaluation, we add various kinds of experiments to demonstrate the overall performance of our system in smart transportation scenarios; and (4) add a traffic light system for supplement the explanation for our proposed framework.

1.2. Related Research

This subsection overviews related research on this study, focusing on three research areas: event-driven architecture (EDA), smart transportation, and Elixir/Erlang-based systems.

1.2.1. Related Work Concerned with EDA

The adoption of event-driven architecture (EDA) in the development of smart cities has become a widely applied concept in recent research. This trend highlights the versatile application of EDA across various fields, demonstrating its potential to address a broad spectrum of challenges within the smart city paradigm. The diversity observed in these studies reflects EDA's flexibility and its capability to enhance systems in multiple domains, from healthcare, which directly impacts human health and safety, to urban traffic management, aiming at optimizing flow and increasing transportation efficiency. Although these approaches differ in their technical implementations, they share a core objective: leveraging advanced technology to improve human life, whether through safeguarding human health or promoting environmental sustainability.

This work by Amir Rahmani, Babaei, and Souri [6] introduced an event-driven IoT architecture for data analysis of reliable healthcare applications, including context, event, and service layers. Furthermore, the study presents complex event processing (CEP) as an innovative solution, integrating automated intelligence within the event layer to enhance the system's responsiveness and decision-making capabilities. This contrasts with our approach, which integrates Elixir and MQTT for an edge-based programming framework. While CEP offers advanced data processing capabilities ideal for healthcare applications, our methodology leverages Elixir's robust concurrency and MQTT's efficient message handling, tailored to the dynamic nature of smart transportation systems. This divergence highlights the adaptability of event-driven architectures across varying domains. The next

work by Behnam Khazael et al. [7] also utilized complex event processing (CEP) systems. Differently, it introduced Geo-TESLA, an advanced complex event processing language tailored for smart city applications, enhancing the detection and reporting of complex events within urban settings by leveraging spatial data types and operations.

Another work that combines event-driven architecture and smart city is the work by Garcia Alvarez, Manuel, Javier Morales, and Menno-Jan Kraak [8]. They offered an approach for spatiotemporal capabilities in information services for smart cities and developed a reference architecture of event-driven applications. This work demonstrates the feasibility, performance, and scalability of event-driven applications in real-time processing and detecting geographic events, leveraging IoT technologies. Xiao Changjiang [9] offered an event-driven focusing service (EDFS) method that uses cyberphysical infrastructures for emergency response in smart cities.

1.2.2. Related Work Concerned with Smart Transportation

With the development of smart cities, the integration of edge computing technologies plays an important role in transforming urban infrastructure. The adoption of edge computing not only facilitates real-time data processing at the network's edge, enhancing the efficiency and responsiveness of smart transportation applications, but also opens up new avenues for addressing complex challenges inherent in urban environments. This complexity is underscored by both the comprehensive survey by Saeik, Firdose et al. [10] on task offloading in edge and cloud computing and the detailed examination of resource scheduling strategies in edge computing by Luo et al. [11]. These works illustrate the diverse approaches and theoretical foundations developed to optimize task offloading processes and resource scheduling across different edge computing scenarios. The approach also aligns with the broader objectives of improving traffic flow, enhancing vehicular communication, and ensuring safety, thus contributing to the overall efficiency and sustainability of urban living.

The next part is within the smart transportation of edge computing realm. The first related work discussed a decision support method of event-driven architecture for a traffic management system [12]. This paper illustrates how event-driven architecture (EDA) and complex event processing are used for real-time processing and analysis of extensive data streams generated by sensors and vehicles. The core objective is real-time monitoring and control of traffic flow, exemplified in a smart traffic management system prototype in Bilbao, Spain. While the paper effectively demonstrates the use of event-driven architecture for traffic management, our research focuses on leveraging edge computing technologies. This approach significantly reduces communication latency and real-time responses, which is a crucial aspect in smart transportation systems. Wei-Hsun Lee et al. proposed a novel design and implementation of a smart traffic signal control (STSC) system that enhances vehicular communication and traffic management [13]. We were inspired by the design of the smart traffic signal control system; however, we used a different solution that combines Elixir and event-driven architecture to handle the vehicular communication in real time. The work by Ke Ruimin [14] focused on edge computing for real-time near-crash detection in smart transportation. It used IoT devices like Nvidia Jetson TX2 for processing video streams to identify near-crash events.

The research highlighted above offers diverse solutions and implementations for smart transportation systems. Differently, our study introduces a more novel approach by leveraging an edge-based framework using Elixir, combined with event-driven architecture and MQTT, to efficiently handle the real-time processing of huge volumes of data in intelligent transportation systems.

1.2.3. Related Work Concerned with Elixir/Erlang-Based Systems

In addition to research on event-driven architecture and smart transportation, there is also compelling evidence regarding the study of Erlang language compatibility and hardware adaptability in IoT systems. GRiSP is a hardware platform and a bare-metal

Erlang virtual machine designed for real-time embedded systems. Several research studies have attempted to build IoT systems using GRiSP. The papers [15,16] proposed a framework called Achlys, to realize general-purpose edge computing using only nodes on a sensor network without relying on gateways or connections to cloud servers. It offers great suitability for distributed applications in IoT edge networks Hera [17] is a Kalman-filter-based sensor fusion framework whose application programs are written in Erlang running on GRiSP. With this framework, high-level processing for asynchronous and fault-tolerant sensor fusion can be realized directly at the edge of the IoT network. Since GRiSP is bare metal Erlang, it has full compatibility with Elixir which runs on BEAM. We contemplate deploying GRiSP in actual environments in the subsequent phase of our research.

2. Elixir Programming Language

This section identifies three critical properties for implementing robust edge computing frameworks: *fault tolerance*, *real-time processing*, and *support for nondisruptive operation*. Fault tolerance ensures continuous operation despite individual component failures. Real-time processing guarantees timely data processing within specified latency constraints. The nondisruptive operation allows updates and maintenance without service interruptions. This section analyzes how Elixir, leveraging its foundation in Erlang, successfully embodies these crucial properties.

2.1. Erlang: A Foundation for Resilience

Erlang, introduced in 1986 by Ericsson, is a functional programming language designed for concurrent systems with the "run forever" philosophy [18]. This focus on robust, nonstop systems makes Erlang a natural choice for edge computing.

Several key features contribute to Erlang's suitability:

- Concurrent processes: Erlang runs multiple lightweight processes on the Erlang Virtual Machine (BEAM). Individual process failures are handled by automatic termination and restart, ensuring system resilience.
- Efficient resource allocation: Erlang's process scheduling ensures timely responsiveness for real-time tasks. Processes can migrate between execution queues, minimizing wait times and optimizing message exchange.
- Hot code loading: Updates can be applied without service interruptions via hot code loading, enabling nondisruptive operation and continuous maintenance.

These features demonstrate Erlang's strength in building reliable and responsive systems, making it a valuable foundation for edge computing frameworks.

2.2. Elixir: Building on Erlang's Legacy

Elixir, built on top of the BEAM virtual machine, inherits Erlang's core strengths. BEAM compiles Elixir code to bytecode for efficient execution. Lightweight processes and message-passing communication foster concurrency and fault tolerance, as failures in one process do not affect others. This inherent resilience is crucial for edge environments where reliability is paramount. Unlike imperative languages, like Java and C++, that rely on shared memory and heavyweight threads, Elixir's message-passing model avoids complex synchronization issues and performance bottlenecks associated with shared resource contention.

Beyond inheriting Erlang's strengths, Elixir offers additional advantages for building scalable and maintainable edge applications:

- Functional programming paradigm: Elixir encourages a side-effect-free programming style, where functions produce outputs solely based on their inputs, simplifying code comprehension and testing.
- Powerful tools and libraries: Elixir provides a rich ecosystem of libraries and tools designed for building robust and maintainable applications.

These combined benefits make Elixir a compelling choice for developing reliable and performant edge computing frameworks. The next subsection delves deeper into Elixir's programming model and its specific advantages for edge development.

2.3. Elixir's Programming Model

This subsection examines key aspects of Elixir's programming model that contribute to its suitability for developing edge computing frameworks: polymorphism, meta-programming, and code conciseness.

2.3.1. Polymorphism via Protocols

Both Elixir and Erlang achieve polymorphism through pattern matching and function dispatch. However, Elixir introduces the powerful concept of *protocols*, enhancing flexibility and intuitiveness. Protocols define a set of functions that any data type can implement, enabling generic operations for different types and implementations. Elixir dynamically recognizes and calls the corresponding specific implementation, demonstrating inherent polymorphism. Additionally, protocols can have "fallback to Any" mechanisms, providing default implementations for unknown types. This promotes code reusability and simplifies handling heterogeneous data structures.

2.3.2. Powerful Meta-Programming Capabilities

Meta-programming, the ability to manipulate and generate code at runtime [19], empowers Elixir development. Compared to Erlang, Elixir offers a more comprehensive and user-friendly meta-programming toolkit through its macro system. This system provides higher-level abstractions and richer functionalities, including module metadata, annotations, reflection, and code evaluation. Accessing the abstract syntax tree (AST) through macros facilitates powerful code transformations and generation, leading to increased development efficiency and improved code quality.

2.3.3. Concise and Expressive Functional Code

Elixir's functional features contribute significantly to code conciseness. Functional constructs like immutability and explicit function definitions enhance program clarity and control flow visualization. This is particularly beneficial for edge computing applications, where compact and understandable code is crucial for efficient execution and debugging. Furthermore, conciseness reduces development time and complexity, making Elixir a compelling choice for rapid development cycles.

2.4. Summary: Why Elixir for Edge Computing?

This section has identified three key features of Elixir's programming model that make it ideally suited for edge computing applications: robust polymorphism and protocol mechanisms, powerful meta-programming capabilities, and inherent code conciseness through functional idioms. These features, coupled with Elixir's rapidly growing library ecosystem, solidify its position as a top choice for building reliable and efficient edge computing frameworks.

3. Architecture Design

This section outlines the key principles and components of the proposed architecture for smart transportation edge computing. Details of the prototype implementation based on this architecture are presented in the succeeding sections. Our envisioned system continuously collects and stores sensor data from urban areas for efficient processing and response to user requests. It demands scalability, real-time functionality, and fault tolerance, aligning perfectly with the capabilities of the Elixir language, as discussed in the previous section.

The proposed architecture comprises multiple interacting components, designed for specific functionalities. Asynchronous message passing with MQTT for event transport and

Elixir for event processing facilitates concurrent execution and fault isolation. Specifically, Elixir's lightweight concurrency and distributed processing handle asynchronous events in a scalable manner (Section 3.2), while MQTT's publish–subscribe messaging distributes events across service components (Section 3.3).

3.1. Event-Driven Architecture for Edge Computing

Event-driven architecture (EDA) [20] centers around event production, detection, consumption, and reaction, where "event" signifies a significant state change. EDA excels in systems requiring real-time operations, asynchronous communication, and high scalability [20,21]. While traditional EDA often centralizes event handling [22,23], EDA for edge computing, like smart transportation, requires optimization for low-latency and local data processing to minimize network overhead and response time. In other words, edge computing tailors EDA to address its inherent challenges: real-time data processing, resource-constrained environments, and distributed computational nodes.

Prominent EDA systems include Kafka Streams [24], Azure Event Grid [25], and RabbitMQ [26]. Kafka Streams excels in scalability and fault tolerance but might not fit resource-constrained environments due to its complex setup. Azure Event Grid shines within the Azure ecosystem, providing managed autoscaling. RabbitMQ, known for its adaptability, supports varied protocols but can require nuanced configuration.

3.2. Key Components in the Proposed EDA-MQTT Framework

The proposed framework using EDA and MQTT broker for smart transportation comprises five key components, as shown in Figure 1.

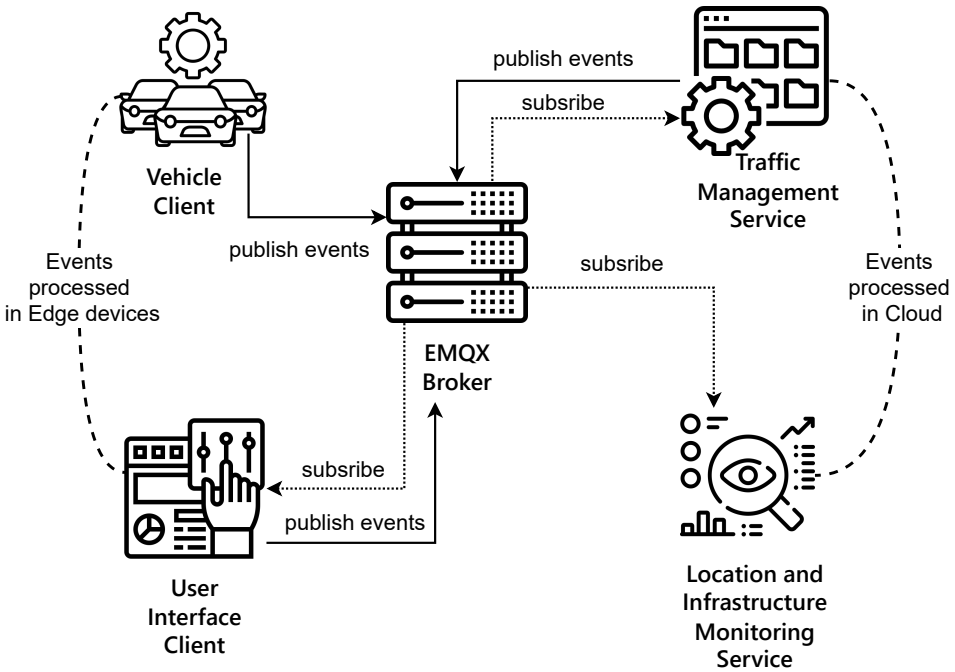


Figure 1. The components in the proposed framework.

- EMQX Broker: The core messaging hub facilitating event-driven communication. It receives events from various clients (user interface client, vehicle client, traffic management service, location and infrastructure monitoring service) and is responsible for accurately forwarding these messages to other clients that have subscribed to them.
- User Interface Client (edge computing component): Serves as the event producer and event consumer. It represents the interface for all end-users, from pedestrians to vehicle operators. It publishes events like traffic congestion reports and listens to updates like route optimizations or transportation schedules.

- Vehicle Client (edge computing component): Mainly serves as the event producer. It caters to all transportation modes, from cars to buses in the edge. This component handles transport-specific events like maintenance alerts, vehicle statuses, and location data.
- Traffic Management Service (hybrid cloud and edge computing component): This service functions as a critical decision-making engine within our framework, operating both at the edge and in the cloud to leverage the strengths of each environment.
  - Edge Deployment: At the edge, the Traffic Management Service focuses on real-time data processing and swift decision making. This proximity to the data sources allows for immediate responses to dynamic traffic conditions, such as adjusting traffic signals to alleviate congestion or responding to unexpected incidents like accidents or road closures. The edge-based component ensures minimal latency and maximizes the responsiveness of the traffic system.
  - Cloud Deployment: In the cloud, the Traffic Management Service undertakes a more comprehensive role. Utilizing the cloud's extensive computational power and vast data storage capabilities, it conducts complex analyses of traffic patterns, predictions of future trends, and development of long-term traffic strategies. The cloud-based service also performs validation and verification of the decisions made at the edge, ensuring overall system accuracy and reliability.
- Location and Infrastructure Monitoring Service (cloud computing component): This service continuously monitors events published by edge servers, facilitating a global analysis of the accumulated data. It rapidly responds to and computes related services, integrating insights from across the network. Additionally, this service is responsible for storing data, ensuring that valuable information is retained for long-term analysis and strategic planning.

Compared to other architectural paradigms, our EDA stands out for its event-focused and asynchronous nature. It contrasts with synchronous patterns like model-view-controller (MVC) and modularity-emphasizing microservices, which can sometimes involve synchronous calls. Our architecture also shares parallels with the event sourcing pattern but emphasizes reactive event handling rather than mere event logging.

Capitalizing on EDA's inherent strengths and MQTT protocol, our architecture strives for scalability, instantaneous responsiveness, and fault resilience, making it suitable for the ever-evolving needs of smart transportation systems.

3.3. MQTT for Event Transport in Edge Computing

Edge computing necessitates efficient communication protocols for real-time data processing, resource-constrained environments, and seamless device/sensor interactions. Choosing the right protocol significantly impacts component interaction, latency, bandwidth utilization, scalability, and security—all crucial factors in edge computing.

Given these demands, MQTT (Message Queuing Telemetry Transport) [27] emerges as a prime candidate. Its lightweight footprint aligns well with the resource limitations of edge devices. Built on a publish–subscribe model, MQTT inherently facilitates event-driven communication, where events are transported through clients subscribing and publishing messages to the broker. Moreover, MQTT offers various quality of service (QoS) levels, ensuring reliable message delivery. This combination of features makes MQTT a well-balanced choice for edge computing requirements.

The following discussion details the rationale behind selecting MQTT for our proposed architecture.

3.3.1. Protocol Comparison for Edge Computing

CoAP (Constrained Application Protocol), AMQP (Advanced Message Queuing Protocol), MQTT, and HTTP (HyperText Transfer Protocol) are popular messaging protocols for IoT and edge computing [28]. While HTTP boasts widespread support and robustness, its resource demands outweigh its benefits in resource-constrained edge environments. Both

CoAP and MQTT excel in low-bandwidth and resource-constrained settings, even running on 8-bit microcontrollers with minimal memory. However, MQTT provides superior throughput and more reliable data delivery options through its QoS levels across low- and high-traffic scenarios. In contrast, AMQP, although adept at complex messaging patterns, falls short in edge computing due to its larger header size and increased complexity [29].

3.3.2. Leveraging MQTT 5.0 for Scalability and Feature Enhancement

The introduction of MQTT 5.0 significantly strengthens its scalability and caters to both large-scale systems and small clients. Features like shared subscriptions and message expiry enhance message management and distribution in large deployments [27]. Session and message expiry intervals offer greater control over session states and message lifetimes, optimizing resource usage, especially in resource-constrained environments. Additionally, MQTT 5.0 introduces new message properties like content type, correlation data, and user properties, enriching the contextual information available for complex data processing and real-time decision making in edge computing systems.

3.3.3. Elixir and MQTT: A Symbiotic Synergy for Edge Computing

Compared to other languages, Elixir's tight integration with MQTT offers robust concurrency processing capabilities. Elixir's lightweight process model and extensive use of asynchronous messaging enable efficient multicore resource utilization, leveraging parallel computing power. Each MQTT connection and session can map to an individual Elixir process, transparently allocated across CPU cores by the scheduler. This one-to-one mapping allows Elixir to handle massive concurrent MQTT connections with superior performance, showcasing its parallel processing capabilities. Studies have demonstrated that Elixir MQTT brokers can support millions of connections with significantly lower latency than other languages.

Furthermore, Elixir's distributed capabilities align seamlessly with MQTT clustering. Through named process registration, Elixir nodes can easily collaborate to construct a geographically distributed, logically unified large-scale MQTT cluster. This cluster automatically load-balances and provides redundancy for fault tolerance, effortlessly managing vast numbers of users and messages.

Finally, Elixir's functional characteristics offer succinct pattern matching capabilities for handling MQTT events, reducing code complexity. Mature MQTT client/server libraries in the Elixir ecosystem expedite development.

In conclusion, Elixir's tight integration with MQTT establishes a powerful programming framework, synergistically combining their strengths to achieve optimal performance and functionality in edge computing applications.

4. Two Case Studies of the Proposed Framework

This section presents two case studies demonstrating the efficacy and versatility of the proposed framework within the domain of smart transportation.

4.1. Traffic Light System for Smart Transportation

4.1.1. Motivation and Approach

Traditional schedule-driven traffic light systems often struggle to adapt to dynamic traffic conditions, leading to unnecessary delays and congestion. To address these limitations, we propose an event-driven architecture (EDA) for adaptive traffic light control within the context of smart transportation. This innovative approach leverages real-time data from diverse sources to dynamically adjust signal timings, potentially minimizing delays and promoting smoother traffic flow compared to fixed-schedule systems [30].

Our proposed system builds upon existing advanced traffic control systems (ATCSs) by incorporating an event-driven paradigm. This enables seamless communication and response between various system components acting as both event producers and consumers. Real-time events such as pedestrian crossings, accidents, and congestion can be

efficiently communicated and acted upon, empowering the system with rapid adaptability to changing traffic conditions. Elixir's inherent scalability and concurrency, as discussed in previous sections, further contribute to the system's responsiveness and effectiveness in handling dynamic traffic patterns.

In addressing the specific scheduling challenges of traffic light signals, our approach incorporates the Oldest Arrival First (OAF) algorithm [31]. The OAF algorithm, known for its efficiency in vehicular traffic scheduling, utilizes real-time position and speed data of individual vehicles. By focusing on isolated traffic intersections, the OAF algorithm aims to minimize delays, enhancing the overall fluidity of traffic movement. By integrating real-time data received from sensors into this algorithm, we can dynamically adjust the scheduling of traffic lights based on real-time analysis. The output of the OAF algorithm, combined with our EDA's responsive policies, allows us to effectively address constantly changing traffic events.

The OAF algorithm's unique capability to process per-vehicle data enables us to dynamically adjust traffic signals in response to immediate traffic conditions. The synergy between the OAF algorithm and our EDA forms the foundation of our proposed smart transportation system.

4.1.2. System Architecture and Implementation

We demonstrate the effectiveness of our approach using a busy crossroads as a testbed, as shown in Figure 2. Sensors strategically placed at the intersection act as event producers, continuously capturing data on vehicle presence, speed, and pedestrian movements. These data points are then transmitted as event messages to roadside units (RSUs) acting as event consumers. The RSUs handle real-time decision making and adjustments based on the received information.

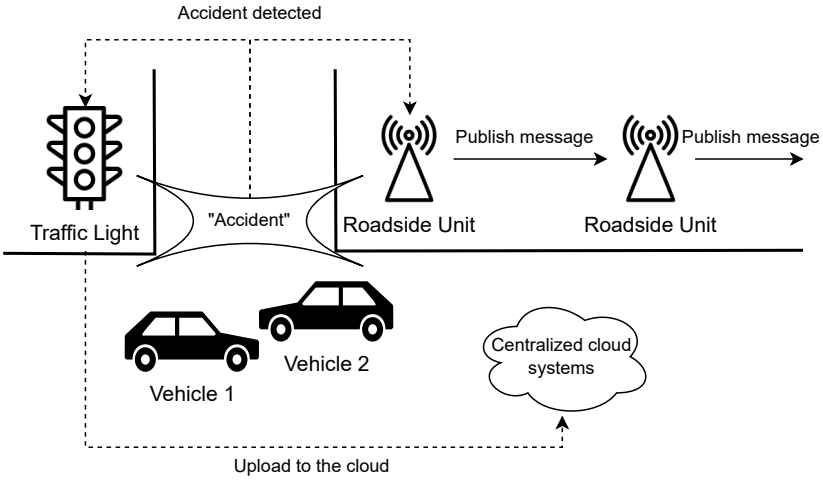


Figure 2. Typical use case of a traffic light system.

The core functionality revolves around event propagation and response:

- When congestion or an accident is detected, the closest RSU publishes a unique event message to the MQTT broker.
- This triggers downstream RSUs on the same street to receive the message and dynamically calculate new route plans for affected vehicles, potentially miles away from the initial event.
- The edge server, running on Elixir, executes immediate control actions based on the updated route plans.
- Centralized cloud systems oversee the broader traffic network and initiate larger adjustments when necessary.

Elixir plays a crucial role in ensuring data integrity and consistency throughout this process. Its functional programming paradigm, featuring immutable data structures,

safeguards data from inadvertent modifications. For instance, an intersection sensor's real-time vehicle density data are represented as an immutable structure shared with two processes: one analyzing city-wide traffic flow and another managing the specific intersection. If the centralized analysis process predicts modifications elsewhere based on this shared data, it does not alter the original structure. This ensures that the intersection control process operates with unaltered data, maintaining consistent and accurate signal timing adjustments based on real-time conditions.

4.1.3. Beyond Traffic Flow Optimization

The benefits of this system extend beyond optimized traffic flow and reduced delays. By minimizing stop-and-go driving, a major contributor to urban vehicle emissions, the system indirectly contributes to a lower carbon footprint for smart transportation systems [32]. This aligns with sustainability goals and leads to improved air quality for urban residents.

4.2. Cab Dispatch System

This subsection presents our second case study: an event-driven cab dispatch system designed for edge computing environments. Such environments demand systems capable of efficiently handling diverse and bursty data streams, effectively utilizing multicore architectures, and dynamically adapting to evolving sensor networks. Our prototype system tackles these challenges and showcases the suitability of Elixir for edge computing applications through its robust concurrency and real-time data processing capabilities.

Figure 3 provides an overview of the system architecture. The detailed implementation aspects of this architecture are discussed in the subsequent section. User-facing passenger and driver applications act as event producers. Passengers initiate ride requests, while drivers simulate location updates, both publishing lightweight messages to a central MQTT broker. A dedicated Elixir-based dispatch service acts as an event consumer, subscribing to these messages and dynamically assigning drivers to passengers based on real-time location and predefined rules.

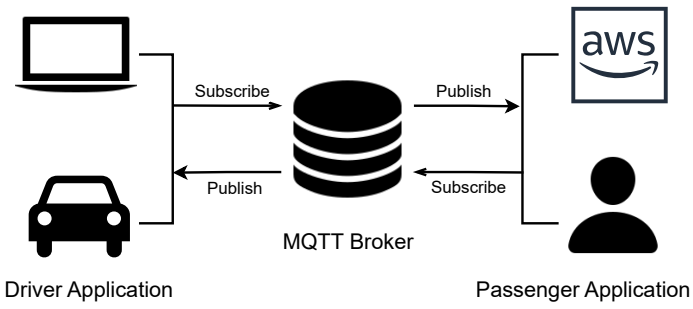


Figure 3. Utilization of the publish/subscribe model in the cab dispatch system.

Elixir's strengths excel in this context, enabling the system to meet the demands of edge computing:

- Lightweight process model: Efficiently manages high concurrency arising from numerous passengers and drivers, eliminating performance bottlenecks.
- Asynchronous messaging: Facilitates real-time responses, ensuring a quick and efficient dispatching experience with minimal latency.

By leveraging these key advantages, our prototype demonstrates the effectiveness of Elixir in latency-sensitive edge computing applications. It paves the way for wider adoption of Elixir in similar scenarios, particularly those requiring robust real-time processing and reliable service delivery.

5. Details of Prototype System

5.1. Event-Driven Design and Components

Figure 4 delves deeper into the event-driven design of the cab dispatch system, highlighting its core components and interactions. In this prototype system, events, event producers, and event channels are implemented as follows.

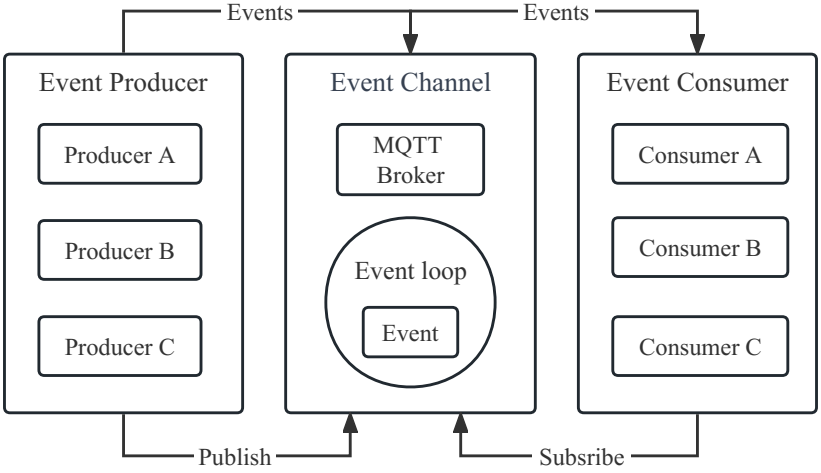


Figure 4. Workflow of the event-driven architecture for the smart transportation system.

5.1.1. Events

The system operates on a foundation of defined events, representing significant occurrences within the system's lifecycle. Core event types include the following:

- CabRequested: Initiated by a passenger application, signifying a ride request with details like passenger ID and destination.
- CabRequestAccepted: Emitted by a driver upon accepting a ride request, confirming their intent to fulfill the request.
- CabArrived: Indicates the driver's arrival at the passenger's pickup location.
- TripStarted: Marks the commencement of the passenger's journey with the chosen driver.
- TripEnded: Signifies the completion of the trip upon reaching the desired destination.

Additional event types can be readily incorporated to cater to future needs and system enhancements. These events are structured data payloads published to the MQTT broker, facilitating efficient and decoupled communication. Upon receiving an event, the dispatch service transitions the corresponding request through its state machine, moving it from pending to assigned, and, subsequently, through other relevant states. For instance, a CabRequest event triggers the transition from pending to assigned, while a TripEnded event marks the final state. This explicit state management allows for clear observation and control of the system's evolution.

5.1.2. Event Producers

Proactive entities within the system act as event producers, encapsulating complex backend details and emitting only meaningful events. Different entities can assume the role of producer depending on the context. In our case, the passenger and driver applications serve as primary producers:

- Passenger application: Publishes CabRequested events upon initiating ride requests.
- Driver application: Publishes CabRequestAccepted events when accepting ride requests and periodically generates LocationUpdate events to broadcast their current location.

This producer-driven approach enables consistent event exposure, simplifying the integration of new data sources and promoting system flexibility.

5.1.3. Event Channels

The MQTT broker serves as the system's central event channel, providing reliable and persistent message transmission between producers and consumers. It acts as a message broker, actively listening for incoming connections, requests, and messages. When a producer publishes an event, the broker filters, validates, and queues it for replayable delivery, ensuring message integrity and resilience. This asynchronous dispatching facilitates decoupled services and enhances the overall effectiveness of the event-driven architecture. The broker's event loop architecture efficiently distributes millions of events per second across geographically distributed edge devices, enabling real-time responsiveness and scalability.

5.2. Cab Dispatch Scenario: A Sequence of Events

The cab dispatch scenario unfolds through a series of orchestrated interactions between key system components: the passenger application, MQTT broker, dispatch service, and driver application. Figure 5 visualizes these interactions using a UML sequence diagram, highlighting the chronological flow of events:

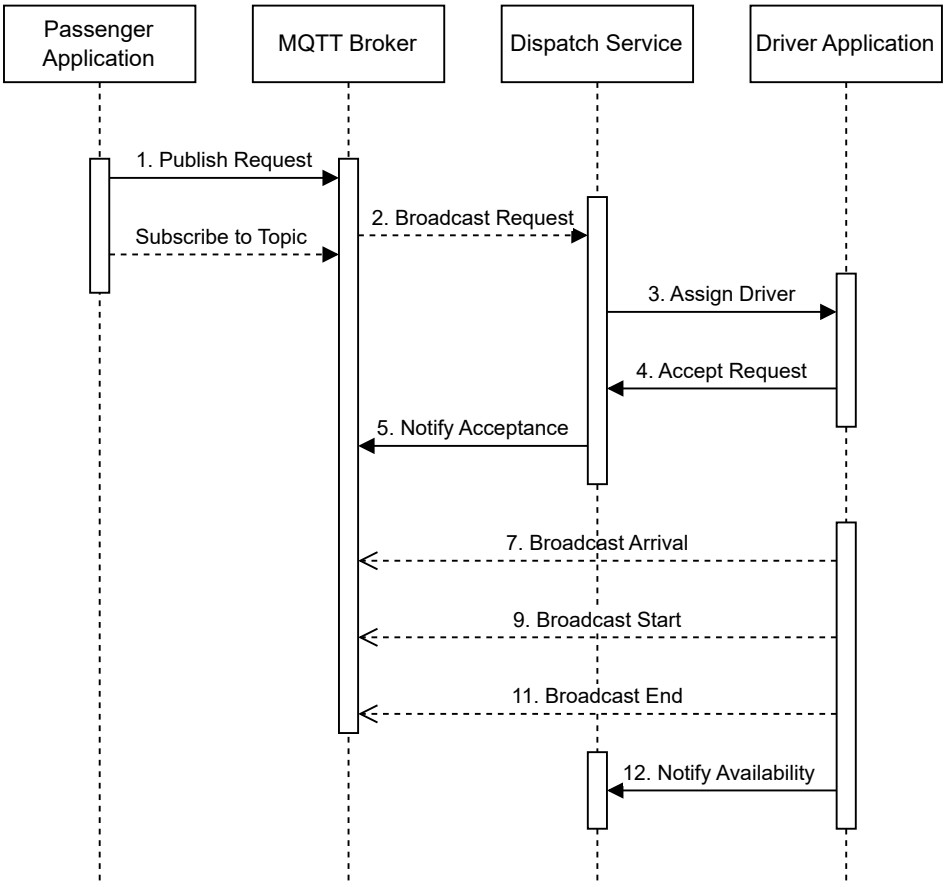


Figure 5. The sequence diagram of the cab dispatch system.

1. Cab Requested: Initiated by the passenger application, this event signifies a ride request and includes details like passenger ID and desired destination. The passenger application publishes this event to the MQTT broker and subscribes to the topic for receiving cab acceptance notifications.
2. Cab Request Broadcasted: Upon receiving the CabRequested event, the MQTT broker acts as a message intermediary, broadcasting it to all subscribed entities, including the dispatch service and driver applications.
3. Driver Assigned: The dispatch service receives the broadcasted request and, applying predefined rules like driver proximity and availability, assigns the request to the most suitable driver.

4. Cab Request Accepted: The assigned driver acknowledges the request by publishing a CabRequestAccepted event to the MQTT broker.
5. Acceptance Notification Broadcasted: Similar to the request, the broker relays the acceptance notification to all subscribers, including the passenger application, confirming the assigned driver.
6. Cab Arrived: The driver application simulates the driver's journey to the passenger's location and, upon arrival, publishes a CabArrived event to notify the system.
7. Arrival Notification Broadcasted: The broker forwards the arrival notification to subscribed entities, informing the passenger that the driver has arrived at the pickup point.
8. Trip Started: Once the passenger boards the cab, the driver application publishes a TripStarted event to mark the commencement of the trip.
9. Trip Start Notification Broadcasted: The broker disseminates the start notification, informing all subscribers, including the passenger application, that the trip has begun.
10. Trip Ended: Upon reaching the destination, the driver application publishes a TripEnded event to signify the completion of the trip.
11. Trip End Notification Broadcasted: The broker broadcasts the final notification to all subscribers, informing them of the trip's completion.
12. Cab Availability Notification: The driver application updates its status to available by notifying the dispatch service, allowing it to assign subsequent ride requests.

This sequential flow exemplifies the event-driven nature of the system, where individual events trigger specific actions and state transitions, orchestrating the entire cab dispatch process.

5.3. Basic Components

5.3.1. Driver Application

The Driver application, designed for edge devices, continuously updates the driver's location to simulate real-time movement. In the experimental evaluation described in the next section, we achieve this by generating random geographical coordinates at regular intervals. The application comprises two primary modules:

- Location Update Module: Generates random location updates for each driver, mimicking continuous position changes through periodic updates. Listing 1 demonstrates a code snippet showcasing this periodic location update functionality.
- MQTT Publishing Module: Publishes location updates as JSON-encoded messages to the MQTT broker under the driver's unique topic. Additionally, it synchronizes all simulated taxis' statuses and publishes pickup requests when drivers become available.

Listing 1. Updating the driver's current location in real time.

```
def update_location(driver_id) do
  :timer.sleep(1000)

  location = generate_random_location()

  :ok = publish_location(driver_id, location)

  update_location(driver_id)  # Keep the loop going
end

defp generate_random_location() do
  %{latitude: rand(0..90), longitude: rand(0..180)}
end

defp publish_location(driver_id, location) do
  event = %{driver_id: driver_id, location: location, timestamp:
      DateTime.utc_now()}
```

```
17
18    event_binary = Jason.encode!(event)
19
20    :ok = :emqtt.publish(conn, "driver/#{driver_id}/location",
          event_binary)
21  end
```

The Elixir application concurrently manages hundreds of taxi objects, including their real-time location and state changes, replicating an actual taxi dispatch scenario. This application runs autonomously, simulating multiple drivers updating and publishing their locations indefinitely, demonstrating Elixir's robustness and suitability for long-running edge computing processes.

5.3.2. Passenger Application

Hosted on the AWS cloud, the passenger application allows passengers to initiate ride requests. It serves as the entry point for submitting requests, specifying details like passenger ID and destination. After submitting a request, the application establishes an MQTT broker connection and subscribes to dedicated topics based on the request ID.

Listing 2 illustrates this subscription using the :emqtt.subscribe/2 function, focusing on topics like events/cab_requested/#passenger_id, Here, #passenger_id is replaced with the actual ID of the passenger, ensuring that the application only receives updates relevant to its specific requests. This ensures the application receives only updates relevant to its specific request. It then enters a listening state, waiting for incoming MQTT messages that provide real-time ride status updates. These messages are processed to inform the passenger about their ride's current status.

Listing 2. Request status monitoring.

```
1  {:ok, conn} = :emqtt.start_link([clientid: "PassengerApp",
       clean_start: false])
2
3  :ok = :emqtt.subscribe(conn,
       "events/cab_requested/#{passenger_id}")
4
5  receive do
6    {:publish, publish} ->
7      IO.inspect(publish)
8  end
```

5.3.3. Dispatch Service

The dispatch service matches passenger requests available taxis based on proximity. It subscribes to MQTT messages containing passenger requests and, upon receiving one, spawns lightweight Elixir processes to concurrently calculate proximity for all idle taxis. This concurrent approach significantly reduces matching computation time. Additionally, we plan to implement load-balancing algorithms in the future.

The dispatch module publishes the assignment result via MQTT to the assigned driver's topic. If no drivers are available, it sends a notification to the passenger about the failed assignment. Leveraging Elixir's efficient concurrency and distributed communication primitives, this module enables fast and reliable order dispatching.

6. Evaluation

Our evaluation methodology is designed to assess the performance of the cab dispatch system within an edge computing paradigm, focusing on several critical metrics: response time latency, concurrency handling under high load conditions, fault tolerance, and multicore processing efficiency. Recognizing the complexity and variability of real-world IoT hardware environments [33], our current testing uses a purely software-based

environment, and future work will include detailed hardware simulations to enhance the practical applicability and robustness of our findings in diverse IoT contexts. Additionally, we also conducted experiments under simulated constrained network conditions to mimic potential real-life challenges. The testbed for our experiments is a MacBook Pro equipped with an Apple M1 chip, featuring 8 cores (4 performance and 4 efficiency cores), and 8 GB of RAM, running macOS Monterey version 12.6. Our test programs were developed in Elixir version 1.14.3, compiled with Erlang/OTP 25. The EMQX MQTT broker was deployed within a Docker container on macOS, and the MQTT clients were operational on Ubuntu VMs, version 22.04.1.

6.1. Latency Testing

We initially focused on response time, defined as the interval between dispatch request initiation by the driver application and its receipt by the passenger application. More specifically, the driver application sends the location to the broker, and the passenger application subscribing to the same topic in the broker will process the location message from the driver application. This test scrutinized system responsiveness, including Phoenix LiveView's hot code loading capabilities. To mimic real-world conditions, the Locust script's "wait_time" parameter controlled the dispatching of real-time location updates at 10 to 30 s intervals. This parameter means that every virtual driver will send the real-time location in every 10 to 30 s.

Further iterations of the test varied the number of virtual drivers and their spawn rate to observe the system's behavior under different loads. As depicted in Figure 6, we executed five sets of tests with varying user counts and spawn rates. The term "spawn rate" here describes the velocity of virtual user generation per second during the test. For instance, in the first group, with a spawn rate of 2 and a target of 100 users, the system incrementally added two users per second until it reached the full count. The results were promising: the median latency stayed below 5 ms, showcasing efficient data processing for the majority of transactions. At the 80th percentile, even with 2000 users, the latency only peaked around 8 ms, indicating that 80% of the transactions were processed within 8 ms. Such performance underscores the framework's aptness for edge computing scenarios, where swift response times are critical.

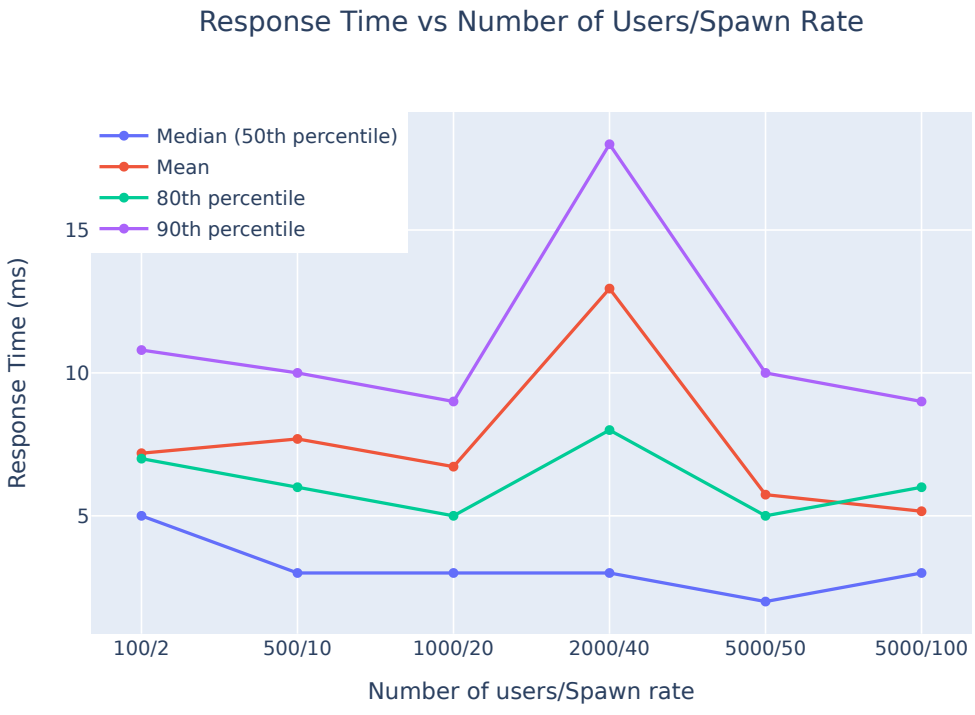


Figure 6. Latency comparison of different groups.

Elixir's functional programming paradigm and the Erlang VM's robust concurrency model contribute to this consistent performance under complex, concurrent processing scenarios. These features equip the system for the real-time demands of edge computing.

Furthermore, to present a more vivid and direct representation of our system's performance, we conducted uninterrupted tests with 2000 and 10,000 users, respectively. Key segments of these tests are illustrated in Figures 7 and 8. The plots predominantly show that the majority of response times are maintained below 8 milliseconds. Notably, while occasional latency spikes were observed, the system leveraged Elixir's fault tolerance capabilities to quickly return to optimal latency ranges under 8 milliseconds. This demonstrates the resilience and stability of our system even under fluctuating conditions.

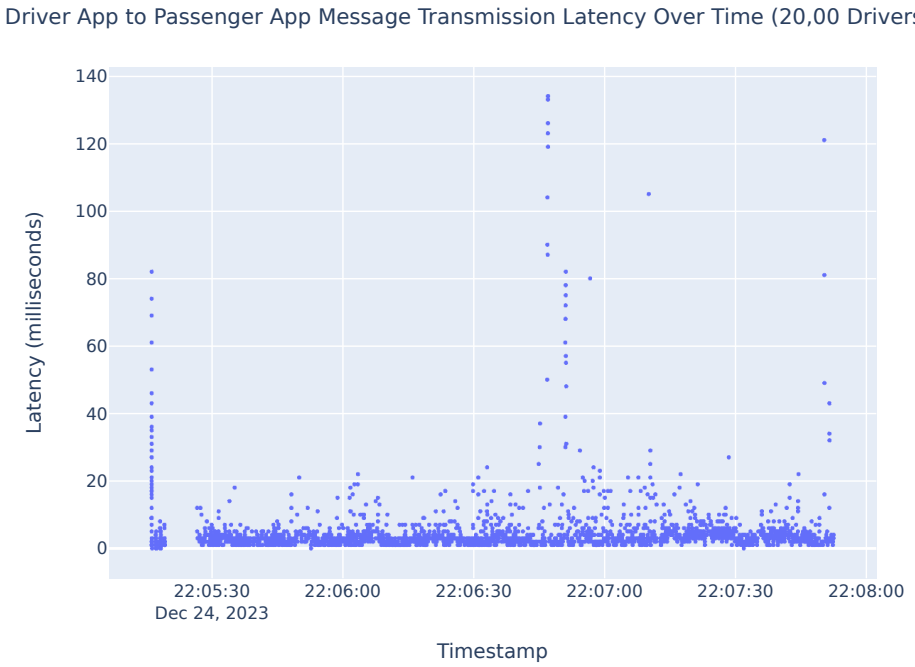


Figure 7. Latency from passenger app to driver app of 2000 users group.

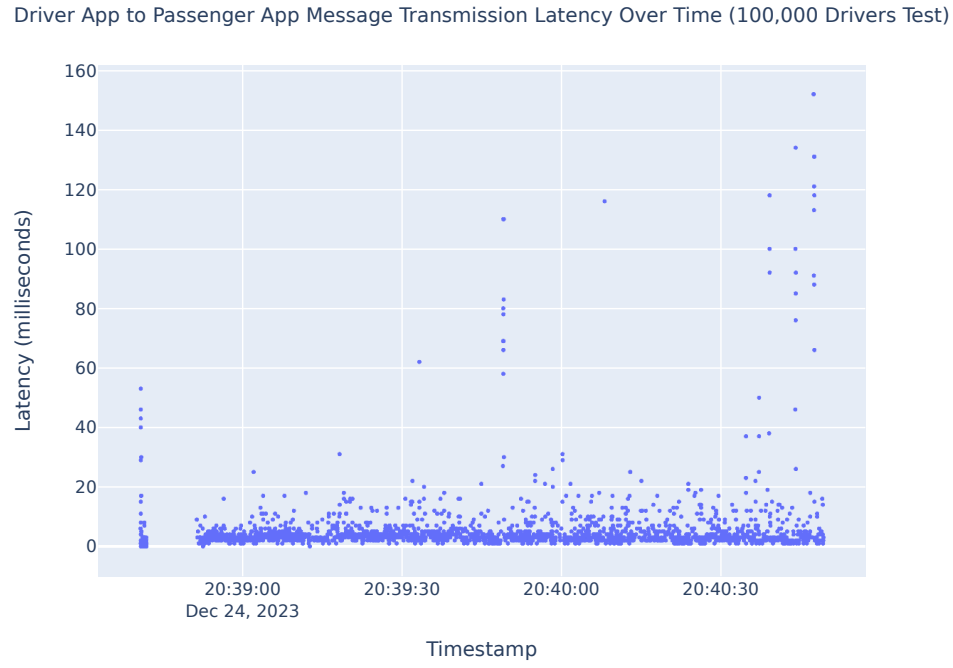


Figure 8. Latency from passenger app to driver app of 10,000 users group.

In scenarios simulating extreme conditions with up to 10,000 users, the system still sustained its performance level. Even with the increased load, the scatterings of peak latency did not exceed 160 ms. This resilience under high user volume is particularly significant, highlighting the system's compatibility with smart transportation applications that demand low latency and high-density user environments.

Latency Comparison with EdgeX Foundry

EdgeX Foundry is a well-known, highly flexible, and scalable open-source edge computing platform that facilitates interoperability between devices and applications at the IoT Edge [34]. We conducted stress tests on the EdgeX Foundry platform under the same testing conditions. Specifically, we configured our test environment on an Ubuntu virtual machine and adjusted the conditions described in Section 6.1. Using EdgeX's built-in device-virtual service, we simulated a cab client and a passenger client, and we conducted the stress tests by using the wrk tool. The tests had the following results: with 8 threads and 50 connections, the average latency was 15.77 ms, with a latency distribution of 75% at 19.04 ms and 90% at 23.52 ms. In comparison, stress tests conducted on our framework using locust demonstrated a lowest average latency of 5 ms, with a 90% latency distribution at 9 ms. This comparative testing showcases the low-latency advantage of our framework. Further detailed comparative research with other frameworks will be conducted in our future work.

6.2. Throughput Testing and Resource Utilization under Load

The next evaluation focused on throughput, analyzing concurrency handling capability under load. The Locust script's "wait_time" was set to 2–8 s to simulate peak traffic conditions, alongside increasing user numbers and spawn rates up to 40,000 users and 5000 spawn rate. Requests per second (RPS) serves as a crucial metric reflecting the system's ability to process incoming requests. Figure 9 illustrates the system's consistent RPS performance across various stress conditions. The results confirm robust performance, efficiently handling high request volumes during peak traffic periods.

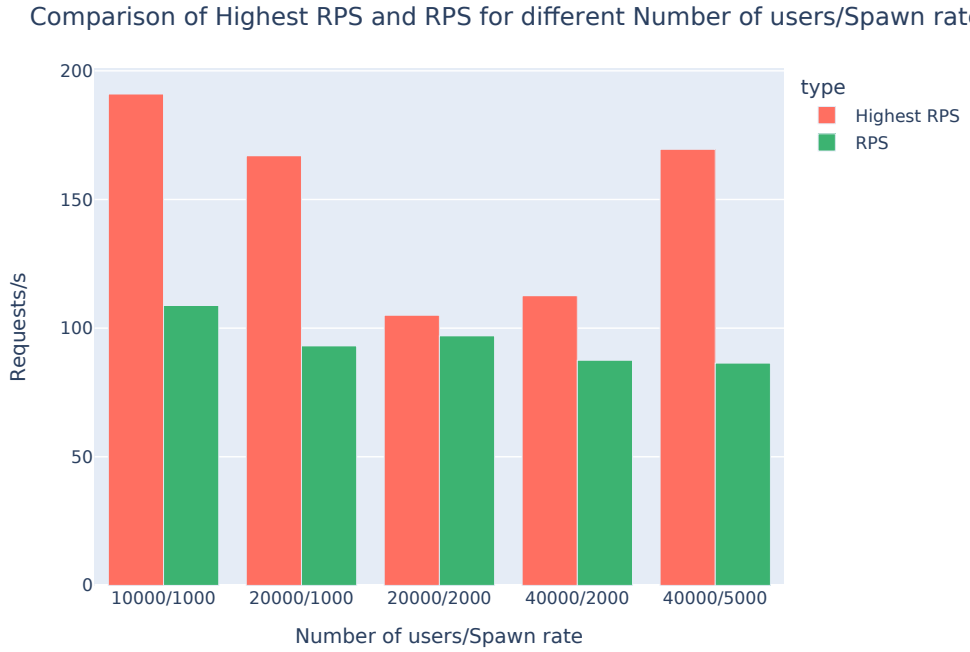


Figure 9. Throughput testing in different sets of groups.

Additionally, resource utilization, defined as the ratio of the actual time spent by resources executing the workload, is another critical metric in the edge computing paradigm [35]. In the throughput testing under load, we also monitored the CPU uti-

lization to assess the overall resource utilization. We focused specifically on the 40,000 user and 5000 spawn rate scenario, and observed a peak CPU utilization of 58%. This figure indicates a balanced resource usage. While lower utilization suggests the availability of additional resources, higher average CPU utilization does not necessarily equate to more efficient orchestration [36]. Indeed, high levels of CPU utilization might indicate load accumulation on other resources due to inefficient load or resource management [37]. The relatively modest resource utilization observed in our tests could be attributed to Elixir's lightweight process model. However, it also highlights opportunities for further optimization of resource usage in future work.

6.3. Constrained Network Conditions Testing

In real-world edge network environments, a variety of factors can contribute to unstable network conditions. Examples in edge networks include varying signal strengths due to geographical locations, network congestion caused by high user density, and intermittent connectivity issues in remote areas. Testing system performance under such unstable network conditions is crucial for improving our framework in edge computing scenarios.

We set three groups of contained network conditions and evaluated them as shown in Figure 10.

Metrics	# Requests	# Fails	Average RPS	Mean	90th percentage
Constraned 1	8184	0	28	114.19	119
Constraned 2	4575	0	20.7	121.32	165
Constraned_3	6358	0	22.6	29.92	20

Figure 10. Different constrained network conditions testing.

- Constrained_1: This scenario involved a network condition with a fixed delay of 100 ms and a packet loss rate of 1%. Under these conditions, the system managed to handle 8184 requests without any failures, maintaining an average RPS of 28.0. The mean latency was recorded at 114.19 ms, with the 90th percentile latency reaching 119.0 ms.
- Constrained_2: This scenario set a base latency of 100 ms with a random fluctuation of up to ±20 ms. The system processed 4575 requests and exhibited an average RPS of 20.7. The mean latency increased slightly to 121.32 ms, and the 90th percentile latency rose to 165.0 ms, reflecting the impact of variable delay conditions.
- Constrained_3: This scenario tested the system's response to a higher packet loss rate of 5%. Despite this challenge, the system processed 6358 requests and achieved an average RPS of 22.6. Without extra latency added in this test, the mean latency in this scenario was significantly lower, at 29.92 ms, and the 90th percentile latency was 20.0 ms.

For comparison, under normal conditions without any imposed network constraints, the average latency of the system was around 8 ms. The observed latency showed an accordingly additional increase over the standard measurements, demonstrating a linear response to the network delay introduced.

6.4. Fault Tolerance Testing

In fault tolerance testing, we designed two primary experimental scenarios to assess the system's performance on fault tolerance: interruptions of the MQTT Broker service and terminations of Elixir processes. The system's response in these scenarios reflects its recovery capabilities and robustness.

- MQTT Broker Service Interruption: In this experiment, we demonstrate the system's resilience when a critical communication component fails. With the MQTT broker positioned at the network edge, maintaining the integrity of complex event transmissions is crucial, especially when encountering unforeseen incidents such as power

outages. Equally important is the ability of MQTT clients to reconnect and quickly return to low-latency operations. As shown in the logs and the accompanying figure, we simulated the latency variations during a broker power outage and subsequent restart. Here, latency refers to the delay experienced by the passenger application in reconnecting to the restarted broker. The log data, with initial latencies exceeding 3000 ms back to the average condition of 10 ms, indicate that the reconnected passenger application rapidly processed messages sent during the outage. When the broker is powered down, the connection between the broker and clients breaks down immediately, the message sent by the driver application will be stored in the broker immediately, and the passenger will automatically keep trying to reconnect and listen to the same topic. When the MQTT broker is powered again, the system will process the omitted events first then quickly return back to normal performance. The system swiftly transitioned from high latency back to low latency, demonstrating rapid recovery capabilities.

- Elixir Process Interruption:
  Before the details of the experimental setup, it is necessary to briefly explain the language logic of Elixir that underpins its fault tolerance.
  In BEAM-based systems, resilience is fundamentally rooted in the complete isolation and independence of each process. This isolation is a key aspect of Elixir's fault tolerance, ensuring that a crash in one process does not affect others. BEAM processes are lightweight, concurrent entities managed by the virtual machine (VM), which schedules their execution. Typically, BEAM employs as many schedulers as there are available CPU cores, with each scheduler operating in its own thread, while the entire VM runs within a single OS process [38]. Figure 11 shows a simplified version of the real experiment environment. Moreover, each process can maintain its state and communicate with other processes to manipulate or access this state. In Elixir, data immutability is a core principle. Processes act as containers for this data, preserving immutable structures over time, sometimes indefinitely.
  In addition to its inherent resilience, Elixir further enhances fault tolerance through the implementation of specialized supervisor processes. These supervisor processes are solely responsible for supervision processes, which are called child processes. Whenever a child process terminates, the supervisor promptly initiates a replacement, utilizing various strategies to manage these processes. This approach effectively reduces the cascading failures and the frequency of restarts that might be triggered by linked process crashes.
  Our experimental setup within the passenger application is depicted in Figure 11, which includes a primary supervisor executing the `rest_for_one` strategy. This strategy ensures that when a child process crashes, the supervisor terminates all subsequently listed processes in the child specification. This is particularly crucial for Process_1, which gathers other system components and needs to maintain consistent state updates to avoid data conflicts. Both the SystemSupervisor and ServerSupervisor employ the `one_for_one` strategy, whereby the crashing and restarting of their supervised processes do not impact the operation of others.
  The experimental results under the supervisor tree are presented in Table 1. Differing from the random process terminations of chaos monkey experiments, we deliberately terminated processes to observe the anticipated restart/no-restart behaviors. As shown in the table, the outcomes matched our expectations: terminating Process_1 led to the restart of all processes; terminating Process_2 did not trigger any restarts; and since Process_3 and Process_4 are linked, the crash of either affected the other. In summary, Elixir's process and supervisor mechanisms flawlessly uphold its fault tolerance capabilities. Within the edge computing paradigm, these features can offer vast prospects for smart transportation applications.

Table 1. Testing of processes to be killed and the existence/nonexistence.

Process \ Killed Process	Process_1	Process_2	Process_3	Process_4
ine Process_1		✓	✓	✓
ine Process_2	✗		✓	✓
ine Process_3	✗	✓		✓
ine Process_4	✗	✓	✗	

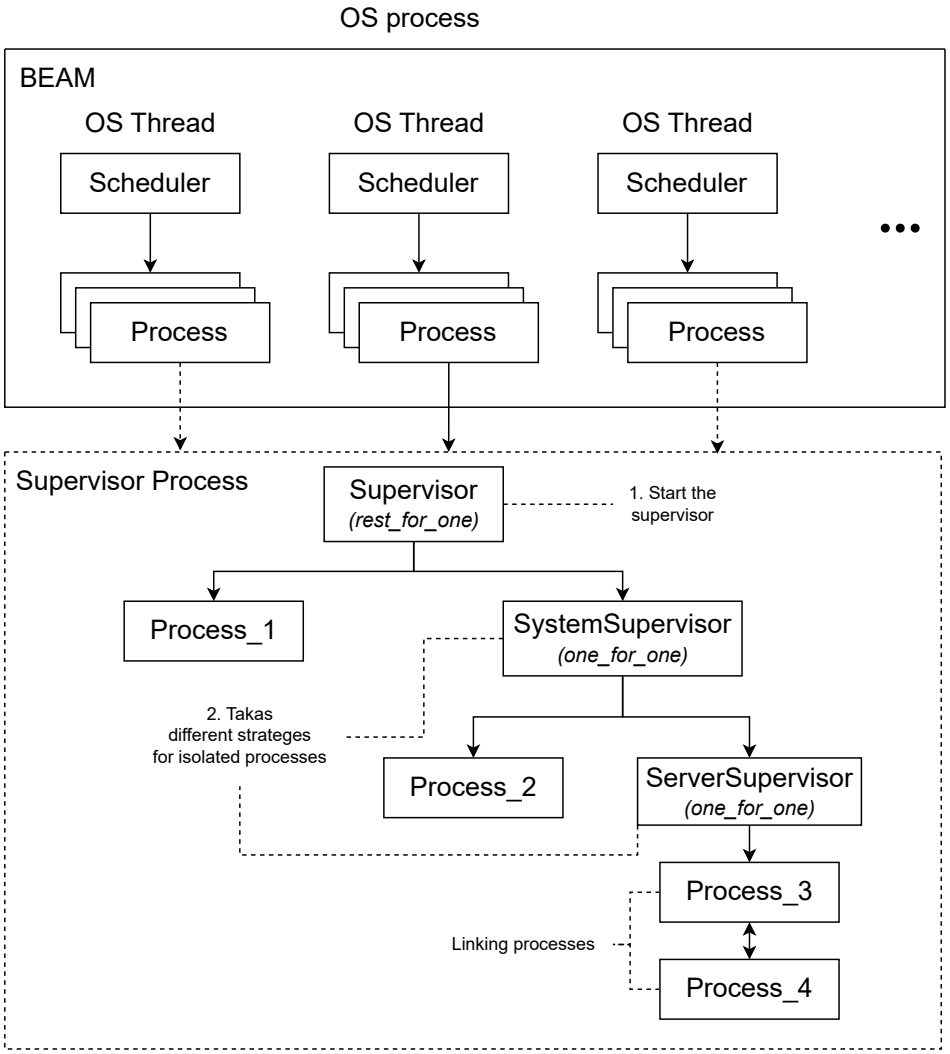


Figure 11. BEAM works as a single OS process and the processes under the supervisor tree.

6.5. Multicore Programming

Listing 3 presents a simulation involving 500 concurrent events, each calculating the Euclidean distance from the origin to a random point. By invoking :erlang.system_info, we determined the number of schedulers used by the Erlang runtime system (typically one scheduler per CPU core). Not only for this individual experiment, but all the above testings were also monitored to use multicore during the process. Our results confirmed

that the Erlang runtime utilized all eight cores of the Apple M1 chip, demonstrating Elixir's effectiveness in leveraging multicore architecture.

Listing 3. Elixir multicore programing example.

```
defmodule ConcurrencyTest do
  def distance_to_origin(x, y), do: :math.sqrt(x * x + y * y)

  def test_concurrency(num_tasks) do
    tasks = 1..num_tasks
    |> Enum.map(fn _ ->
        Task.async(
          fn ->
            x = :rand.uniform(1000)
            y = :rand.uniform(1000)
            distance_to_origin(x, y)
          end) end)
    |> Enum.map(&Task.await/1)

    IO.puts("Schedulers:
        #{:erlang.system_info
        (:schedulers_online)}")
  end
end

ConcurrencyTest.test_concurrency(500)

IO.inspect(:erlang.system_info
    (:schedulers_online))
```

6.6. Summary

The evaluations, encompassing both response time and throughput under high load, demonstrate the system's reliability and scalability. In an edge computing environment, the cab dispatch system exhibits both agile response and sturdy concurrency management, making it well suited for edge-centric solutions.

7. Concluding Remarks

This paper proposed an edge computing architecture tailored for real-time, distributed applications. Our solution leverages Elixir's unique capabilities: its lightweight concurrent processing model for efficient resource utilization, and robust fault tolerance mechanisms inherited from Erlang/OTP for enhanced system resilience. Additionally, we employed the MQTT protocol for asynchronous event transport due to its proven efficiency and reliability in distributed environments.

The evaluation of our framework through a cab dispatch prototype demonstrated its strengths. The prototype achieved low-latency and high-concurrency capabilities, underlining the effectiveness of Elixir's multicore utilization for edge computing scenarios. This successful prototype exemplifies the potency of Elixir in crafting scalable and responsive edge applications.

Our findings offer a valuable contribution to the field by furthering the integration of Elixir and event-driven models within edge computing domains. This paves the way for further scholarly discourse and exploration of Elixir's potential in addressing the evolving demands of real-time distributed systems at the edge.

In this study, we presented two case studies that collectively illustrate the unique advantages of adopting event-driven architecture (EDA) and the Message Queuing Telemetry Transport (MQTT) protocol in smart transportation applications, demonstrating through

testing the irreplaceable benefits of Elixir's integration for system performance enhancement. However, our experiments were conducted in a purely software environment; hence, testing for broader application scenarios remains insufficient. As mentioned in the related research section, we plan to continue framework improvement and testing on hardware systems supporting the BEAM system, such as the Erlang-based GRiSP hardware platform, aiming to propose a more comprehensive and application-enhanced version.

The research into these two case studies has led us to identify the optimal integration of EDA and smart transportation, notably in rapid response to emergency events and in reducing decision-making time during peak periods. At the same time, we have identified other areas requiring further exploration, such as ensuring data protection while maintaining efficient communication. Although Elixir and MQTT 5.0 provide us with high levels of security, further research in this area will also be a part of our future work.

Author Contributions: Conceptualization, Y.L. and S.F.; methodology, Y.L. and S.F.; software, Y.L.; validation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, Y.L. and S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to as the data are part of an ongoing study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pan, J.; McElhannon, J. Future edge cloud and edge computing for internet of things applications. *IEEE Internet Things J.* **2017**, *5*, 439–449. [CrossRef]
2. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A survey on the edge computing for the Internet of Things. *IEEE Access* **2017**, *6*, 6900–6919. [CrossRef]
3. Statista. Internet of Things (IoT) Total Annual Revenue Worldwide from 2020 to 2030. 2021. Available online: https://www.statista.com/statistics/1194709/iot-revenue-worldwide/ (accessed on 15 January 2024).
4. Grand View Research. Edge Computing Market Size, Share & Trends Analysis Report by Component, by Application (Smart Grids, Remote Monitoring), by End Use (Manufacturing, Healthcare), by Region, and Segment Forecasts, 2020–2027. 2021. Available online: https://www.grandviewresearch.com/industry-analysis/edge-computing-market (accessed on 15 January 2024).
5. Li, Y.; Fujita, S. Design of Elixir-Based Edge Server for Responsive IoT Applications. In Proceedings of the 2022 Tenth International Symposium on Computing and Networking Workshops (CANDARW), Himeji, Japan, 21–24 November 2022; pp. 185–191. [CrossRef]
6. Rahmani, A.M.; Babaei, Z.; Souri, A. Event-driven IoT architecture for data analysis of reliable healthcare application using complex event processing. *Clust. Comput.* **2021**, *24*, 1347–1360. [CrossRef]
7. Khazael, B.; Asl, M.V.; Malazi, H.T. Geospatial complex event processing in smart city applications. *Simul. Model. Pract. Theory* **2023**, *122*, 102675. [CrossRef]
8. Alvarez, M.G.; Morales, J.; Kraak, M.-J. Integration and exploitation of sensor data in smart cities through event-driven applications. *Sensors* **2019**, *19*, 1372. [CrossRef] [PubMed]
9. Xiao, C.; Chen, N.; Gong, J.; Wang, W.; Hu, C.; Chen, Z. Event-driven distributed information resource-focusing service for emergency response in smart city with cyber-physical infrastructures. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 251. [CrossRef]
10. Saeik, F.; Avgeris, M.; Spatharakis, D.; Santi, N.; Dechouniotis, D.; Violos, J.; Leivadeas, A.; Athanasopoulos, N.; Mitton, N.; Papavassiliou, S. Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Comput. Netw.* **2021**, *195*, 108177. [CrossRef]
11. Luo, Q.; Hu, S.; Li, C.; Li, G.; Shi, W. Resource scheduling in edge computing: A survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2131–2165. [CrossRef]
12. Dunkel, J.; Fernández, A.; Ortiz, R.; Ossowski, S. Event-driven architecture for decision support in traffic management systems. *Expert Syst. Appl.* **2011**, *38*, 6530–6539. [CrossRef]
13. Lee, W.-H.; Chiu, C.-Y. Design and implementation of a smart traffic signal control system for smart city applications. *Sensors* **2020**, *20*, 508. [CrossRef]
14. Ke, R.; Cui, Z.; Chen, Y.; Zhu, M.; Yang, H.; Wang, Y. Edge computing for real-time near-crash detection for smart transportation applications. *arXiv* **2020**, arXiv:2008.00549.

15. Kopestenski, I.; Van Roy, P. Achlys: Towards a framework for distributed storage and generic computing applications for wireless IoT edge networks with Lasp on GRiSP. In Proceedings of the 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kyoto, Japan, 11–15 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 875–881.
16. Kopestenski, I.; Van Roy, P. Erlang as an Enabling Technology for Resilient General-Purpose Applications on Edge IoT Networks. In Proceedings of the 18th ACM SIGPLAN International Workshop on Erlang, Berlin, Germany, 18 August 2019; pp. 1–12.
17. Kalbusch, S.; Verpoten, V.; Van Roy, P. The Hera Framework for Fault-Tolerant Sensor Fusion with Erlang and GRiSP on an IoT Network. In Proceedings of the 20th ACM SIGPLAN International Workshop on Erlang, Virtual, 26 August 2021; pp. 15–27.
18. Armstrong, J. A History of Erlang. In Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages, San Diego, CA, USA, 9–10 June 2007; pp. 6-1–6-26.
19. Lilis, Y.; Savidis, A. A survey of metaprogramming languages. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–39. [CrossRef]
20. Michelson, B.M. Event-driven architecture overview. *Patricia Seybold Group* **2006**, *2*, 10–1571.
21. Taylor, H.; Yochem, A.; Phillips, L.; Martinez, F. *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*; Addison-Wesley Professional: Boston, MA, USA, 2009.
22. Khriji, S.; Benbelgacem, Y.; Chéour, R.; El Houssaini, D.; Kanoun, O. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *J. Supercomput.* **2022**, 1–28. [CrossRef]
23. Goyal, P.; Mikkilineni, R. Policy-based event-driven services-oriented architecture for cloud services operation & management. In Proceedings of the 2009 IEEE International Conference on Cloud Computing, Bangalore, India, 21–25 September 2009; pp. 135–138.
24. Apache Kafka. Kafka Streams. Available online: https://kafka.apache.org/documentation/streams/ (accessed on 15 January 2024).
25. Microsoft Azure. Azure Event Grid. Available online: https://azure.microsoft.com/en-us/services/event-grid/ (accessed on 15 January 2024).
26. RabbitMQ. RabbitMQ. Available online: https://www.rabbitmq.com/ (accessed on 15 January 2024).
27. *MQTT Version 5.0*, Technical Report; OASIS Standard: Woburn, MA, USA, 2019. Available online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (accessed on 22 June 2020).
28. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017; pp. 1–7.
29. Ouakasse, F.; Rakrak, S. A comparative study of MQTT and COAP application layer protocols via. performances evaluation. *J. Eng. Appl. Sci.* **2018**, *13*, 6053–6061.
30. Zhao, Y.; Tian, Z. An overview of the usage of adaptive signal control system in the United States of America. *Appl. Mech. Mater.* **2012**, *178*, 2591–2598. [CrossRef]
31. Pandit, K.; Ghosal, D.; Zhang, H.M.; Chuah, C.-N. Adaptive traffic signal control with vehicular ad hoc networks. *IEEE Trans. Veh. Technol.* **2013**, *62*, 1459–1471. [CrossRef]
32. Zhao, C.; Wang, K.; Dong, X.; Dong, K. Is smart transportation associated with reduced carbon emissions? The case of China. *Energy Econ.* **2022**, *105*, 105715. [CrossRef]
33. Belson, B.; Holdsworth, J.; Xiang, W.; Philippa, B. A survey of asynchronous programming using coroutines in the Internet of Things and embedded systems. *ACM Trans. Embed. Comput. Syst. (TECS)* **2019**, *18*, 1–21. [CrossRef]
34. EdgeX Foundry. n.d. Available online: https://www.edgexfoundry.org/ (accessed on 15 January 2024).
35. Gill, S.S.; Chana, I.; Singh, M.; Buyya, R. CHOPPER: An intelligent QoS-aware autonomic resource management approach for cloud computing. *Clust. Comput.* **2018**, *21*, 1203–1241. [CrossRef]
36. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet Things* **2020**, *12*, 100273. Available online: https://www.sciencedirect.com/science/article/pii/S2542660520301062 (accessed on 15 January 2024). [CrossRef]
37. Aslanpour, M.S.; Ghobaei-Arani, M.; Toosi, A.N. Auto-scaling web applications in clouds: A cost-aware approach. *J. Netw. Comput. Appl.* **2017**, *95*, 26–41. [CrossRef]
38. Juric, S. *Elixir in Action*; Simon and Schuster: New York, NY, USA, 2019.

future internet

MDPI

Article

Edge-Enhanced TempoFuseNet: A Two-Stream Framework for Intelligent Multiclass Video Anomaly Recognition in 5G and IoT Environments

Gulshan Saleem [1], Usama Ijaz Bajwa [1,*], Rana Hammad Raza [2] and Fan Zhang [3,*]

[1] Department of Computer Science, COMSATS University Islamabad, Lahore Campus, Lahore 54000, Pakistan; gulshnsaleem26@gmail.com
[2] Electronics and Power Engineering Department, Pakistan Navy Engineering College (PNEC), National University of Sciences and Technology (NUST), Karachi 75350, Pakistan; hammad@pnec.nust.edu.pk
[3] Ocean College, Zhejiang University, Hangzhou 316000, China
* Correspondence: usamabajwa@cuilahore.edu.pk (U.I.B.); f.zhang@zju.edu.cn (F.Z.)

Abstract: Surveillance video analytics encounters unprecedented challenges in 5G and IoT environments, including complex intra-class variations, short-term and long-term temporal dynamics, and variable video quality. This study introduces Edge-Enhanced TempoFuseNet, a cutting-edge framework that strategically reduces spatial resolution to allow the processing of low-resolution images. A dual upscaling methodology based on bicubic interpolation and an encoder–bank–decoder configuration is used for anomaly classification. The two-stream architecture combines the power of a pre-trained Convolutional Neural Network (CNN) for spatial feature extraction from RGB imagery in the spatial stream, while the temporal stream focuses on learning short-term temporal characteristics, reducing the computational burden of optical flow. To analyze long-term temporal patterns, the extracted features from both streams are combined and routed through a Gated Recurrent Unit (GRU) layer. The proposed framework (TempoFuseNet) outperforms the encoder–bank–decoder model in terms of performance metrics, achieving a multiclass macro average accuracy of 92.28%, an F1-score of 69.29%, and a false positive rate of 4.41%. This study presents a significant advancement in the field of video anomaly recognition and provides a comprehensive solution to the complex challenges posed by real-world surveillance scenarios in the context of 5G and IoT.

Keywords: edge intelligence; anomaly identification; super resolution; video classification; two-stream architecture; StyleGAN; IoT environment

Citation: Saleem, G.; Bajwa, U.I.; Raza, R.H.; Zhang, F. Edge-Enhanced TempoFuseNet: A Two-Stream Framework for Intelligent Multiclass Video Anomaly Recognition in 5G and IoT Environments. *Future Internet* **2024**, *16*, 83. https://doi.org/10.3390/fi16030083

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 5 January 2024
Revised: 4 February 2024
Accepted: 27 February 2024
Published: 29 February 2024

1. Introduction

In the era of 5G and IoT, video surveillance is a critical component of modern security and monitoring strategies. This surveillance relies on advanced camera technology to observe and analyze diverse environments and contributes to applications such as security, crime prevention, safety, emergency response, traffic monitoring, and behavior analysis [1–3]. Video surveillance contributes significantly to theft prevention, traffic management, and overall safety in the residential, commercial, and industrial sectors.

The incorporation of technology and machine learning [4,5] into video surveillance, particularly in 5G and IoT environments, initiates unprecedented possibilities. Automated video surveillance systems controlled by computer vision algorithms [6–8] detect anomalies, changes in motion, and intrusions in real-time, reducing reliance on human monitoring [9]. However, challenges persist, such as operator errors, false alarms, and limitations in contextual information within video footage [10–12].

In the context of 5G and IoT, this study addresses technical limitations associated with low-quality videos: specifically, poor lighting and low spatial resolution. These difficulties have an impact on the perceptual quality of video streams [13–15] and introduce factors

such as poor lighting, camera noise, low spatial resolution, and low frame rates [9,16–19]. Despite these challenges, various techniques for detecting anomalies in low-quality surveillance videos have been proposed, [20,21]. Two primary approaches are commonly used to address the challenge of detecting anomalies in low-quality videos. The first entails improving video quality with techniques like denoising, dehazing, and super-resolution [22,23]. An alternative strategy is to use deep learning methods directly for anomaly detection in low-quality videos [24,25].

This study outperforms existing approaches by introducing a new super-resolution technique called "TempoFuseNet". For enhanced anomaly detection, this innovative framework employs a two-stream architecture that combines spatial and temporal features. The spatial stream extracts features using a pre-trained Convolutional Neural Network (CNN), whereas the temporal stream captures short-term temporal characteristics efficiently using a novel Stacked Grayscale 3-channel Image (SG3I) approach. The extracted features from both streams are fused via a Gated Recurrent Unit (GRU) layer to leverage long-term temporal dependencies effectively. The contributions of this study include the identification of challenges related to intra-class and inter-class variabilities, the introduction of a super-resolution technique leveraging an encoder–bank–decoder configuration, the incorporation of a StyleGAN for feature enhancement, and the proposal of a two-stream architecture for anomaly classification.

Recognizing the nuanced landscape of automated surveillance systems is essential in the continuum of addressing challenges in video surveillance. These systems play a critical role in overcoming the limitations of manual monitoring. Despite their potential, however, these systems face challenges that necessitate strategic interventions for further refinement. One significant challenge is the generation of false alarms, which can overwhelm security personnel and undermine the effectiveness of surveillance operations. False alarms not only divert attention but also place unnecessary demands on resources. The importance of minimizing false alarms as a fundamental aspect of optimizing automated surveillance systems is acknowledged in this study.

Another problem stems from video's inherent limitation in providing comprehensive context. Surveillance videos frequently capture snippets of events, making it difficult to decipher the intentions of those being watched or comprehend the full scope of a given incident. Improving the contextual understanding of surveillance footage appears to be a critical component in addressing this challenge. Technical constraints obstruct the seamless operation of automated surveillance systems. Poor lighting, low-resolution cameras, and limited storage capacity can all have an impact on the effectiveness of these systems. To improve the robustness and reliability of automated surveillance, a comprehensive approach to addressing these technical limitations is required.

This study focuses on the technical limitations caused by low-quality videos: specifically, poor lighting and low spatial resolution. These difficulties have been identified as critical factors influencing the perceptual quality of video streams and thereby influencing the accuracy of anomaly detection systems according to [16]. The contributions of this study can be summed up as follows:

- This study meticulously identifies and articulates two critical issues inherent in surveillance videos: high intra-class variability and low inter-class variability. These challenges, which are inextricably linked to the temporal properties of video streams, both short- and long-term, are exacerbated by the prevalence of low-quality videos.
- This study makes an outstanding contribution by introducing an innovative super-resolution approach designed to mitigate the impact of low-quality videos caused by downscaling. This approach outperforms traditional bicubic interpolation by using an encoder–bank–decoder configuration to upscale videos. The primary goal is to improve the spatial resolution of videos in order to increase the accuracy of anomaly detection. The addition of a pre-trained StyleGAN as a latent feature bank is a critical step forward that enriches the super-resolution process and, as a result, improves anomaly classification accuracy.

- The study implies a two-stream architecture for anomaly classification. The spatial stream uses a pre-trained CNN model for feature extraction, whereas the temporal stream employs an innovative approach known as Stacked Grayscale Image (SG3I). SG3I effectively lowers the computational costs associated with optical flow computation while accurately capturing short-term temporal characteristics. The extracted features from both streams are concatenated and fed into a Gated Recurrent Unit (GRU) layer, which allows the model to learn and exploit long-term dependencies.
- Experiments show that the super-resolution model improves classification accuracy by 3.7% when compared to traditional bicubic interpolation methods. When combined with the encoder–bank–decoder super-resolution model, the classification model achieves an impressive accuracy of 92.28%, an F1-score of 69.29%, and a low false positive rate of 4.41%.

To sum up, this research not only identifies and understands the difficulties that are associated with surveillance footage, but it also introduces novel approaches to deal with those difficulties. The end result of these efforts is observable improvements in performance and accuracy for the classification of anomalies in surveillance videos.

2. Related Work

Video anomaly detection is critical in the domain of surveillance systems, as it addresses the need to identify anomalous segments within video streams. Over time, two major streams of methodologies have emerged for this critical task: handcrafted approaches and deep-learning-based methods. The former employs manual feature engineering techniques such as STIP, SIFT-3D, and optical flow histograms, whereas the latter makes use of the power of deep neural networks such as VGG and ResNet to process spatiotemporal data efficiently. The introduction of two-stream Convolutional Neural Networks (CNNs) for improved activity recognition and novel approaches to modeling long-term temporal dependencies are notable advancements. The literature includes a wide range of deep learning models, from ConvLSTM to attention-based architectures, all of which contribute to the improvement of anomaly detection in videos. Furthermore, weakly supervised techniques, generative models, and recent efforts to address anomalies in low-resolution videos have significantly expanded the scope of this evolving field. In the midst of these advances, our research focuses on a novel problem: detecting anomalies in multi-class scenarios in low-quality surveillance videos. We present a unified methodology that combines novel super-resolution techniques with a two-stream architecture, providing a comprehensive solution to the complexities of real-world surveillance scenarios.

Manual feature engineering methods such as STIP, SIFT-3D, and optical flow histograms involve human intervention [26,27]. While insightful, improved dense trajectory approaches like the one by Wang et al. [28] surpass earlier handcrafted techniques. The advent of deep learning has revolutionized video anomaly identification, with networks like VGG and ResNet efficiently processing spatiotemporal data in videos [29,30]. Noteworthy in this domain is the introduction of two-stream Convolutional Neural Networks (CNNs), which combine spatial and temporal inputs for improved activity recognition [31,32].

Advancements in modeling long-term temporal dependencies have been achieved through techniques like temporal segment networks and 3D convolutional filters. Wang et al. [33] introduced a temporal segment network that exhibited robust performance on benchmark datasets. The C3D method by Tran et al. [34] addressed challenges in modeling temporal information and inspired subsequent work by Maqsood et al. [35] for anomaly classification. Among deep-learning-based models, significant strides have been made, particularly in domains involving nonlinear, high-dimensional data. Luo et al. [36] proposed a Convolutional Long Short-Term Memory (ConvLSTM) model for encoding video frames and identifying anomalies. Ullah et al. [37] introduced a Convolution-Block-Attention-based LSTM model that enhances spatial information accuracy. Riaz et al. [38] combined human posture estimation with a densely connected fully Convolutional Neural Network (CNN) for anomaly identification. Hasan et al. [1] utilized a recurrent neural net-

work (RNN) and a convolutional autoencoder for anomaly detection, while Liu et al. [39] integrated temporal and spatial detectors for anomaly identification.

Weakly supervised techniques, including C3D and MIL, have been employed for anomaly detection. Sultani et al. [2] combined weak video labels with Multiple Instance Learning (MIL). Landi et al. [40] used a coordinate-based regression model for tube extraction. Generative models like GANs have been explored, with Sabokroul et al. [41] training GANs for visual anomaly detection. BatchNorm into Weakly Supervised Video Anomaly Detection (BN-WVAD) [42] has been used to capitalize on the statistical insight that temporal features of abnormal events often behave as outliers; BN-WVAD leverages the Divergence of Feature from Mean vector (DFM) function from BatchNorm. This DFM criterion serves as a robust abnormality indicator and identifies potential abnormal snippets in videos. It enhances anomaly recognition, proves to be more resistant to label noise, and provides an additional anomaly score to refine predictions from classifiers that are sensitive to noisy labels. In [43], a Temporal Context Aggregation (TCA) module for efficient context modeling and a Prompt-Enhanced Learning (PEL) module for enhanced semantic discriminability are demonstrated. The TCA module captures complete contextual information, while the PEL module incorporates semantic priors using knowledge-based prompts to improve discriminative capacity and maintain separability between anomaly sub-classes. Additionally, a Score Smoothing (SS) module is introduced in the testing phase to reduce false alarms. In [44], a U-Net-like structure is implemented to effectively capture both local and global temporal dependencies in a unified manner. The encoder hierarchically learns global dependencies on top of local ones, and the decoder propagates this global information back to the segment level for classification.

Recent research has focused on addressing anomalies in extremely low-resolution videos [25,45–47]. Techniques such as Inverse Super-Resolution (ISR), initially introduced by Ryoo et al. [45], aim to identify optimal image modifications for extracting additional information from low-resolution images. Additionally, multi-Siamese loss functions have been proposed to maximize data utilization from a collection of low-resolution images. Chen et al. [46] developed a semi-coupled two-stream network that leverages high-resolution images to assist with training a low-resolution network. Xu et al. [47] demonstrated that using high-resolution images improves low-resolution recognition by incorporating a two-stream neural network architecture that takes high-resolution images as inputs. Their approach, sharing convolutional filters between low- and high-resolution networks, significantly enhanced performance. In addition, Demir et al. [48] proposed the TinyVIRAT dataset for natural low-resolution videos and presented a gradual generative technique for enhancing the quality of low-resolution events. Super-resolution techniques have also found success in various applications such as low-resolution face verification, small object detection, person re-identification, and activity recognition [49–52]. For instance, Ataer et al. [50] introduced an identity-preserving super-resolution approach for face verification at very low resolutions, and Bai et al. [51] developed a multitask generative adversarial network for small object detection.

In summary, the field of video anomaly detection has witnessed diverse advancements, from Bayesian deep learning to convolutional models, recurrent neural networks, and spatial–temporal graph attention networks. Our study addresses the challenge of detecting anomalies in multi-class scenarios within low-quality surveillance videos and showcases improved classification performance compared to interpolation-based strategies. The integration of novel super-resolution techniques and a two-stream architecture forms the backbone of our methodology and contributes to the evolution of video anomaly recognition in complex real-world scenarios. While the literature review reflects significant progress in video anomaly detection, there is a significant research gap that our study seeks to fill. Existing approaches have primarily focused on either high-quality video scenarios or have addressed anomalies in a binary manner, both of which are insufficient for real-world applications. The combination of novel super-resolution techniques and a two-stream architecture, as proposed in our methodology, represents a novel approach to closing this

gap. Our research contributes to the evolving landscape of video anomaly recognition by providing a tailored solution to the complexities of multi-class scenarios and low-quality surveillance videos within 5G and IoT environments.

3. Materials and Methods

The effectiveness of anomaly detection in surveillance videos is inextricably linked to the quality of the input data. In this methodology, we address the challenges posed by low-quality surveillance videos; we focus on issues such as poor lighting and spatial resolution. Our method combines advanced video resizing techniques with deep-learning-based super-resolution methods to improve the overall quality of video streams. The initial stages of our methodology include a meticulous video resizing process in which we experiment with various interpolation methods to upscale low-resolution videos. We then present a novel video super-resolution strategy that takes advantage of GLEAN, a framework that uses Generative Adversarial Networks (GANs) for latent feature extraction. Unlike traditional GAN-based models, our implementation uses a streamlined process that requires only one forward pass to generate high-resolution images. The use of a StyleGAN, which has been fine-tuned on a dataset containing both low- and high-resolution representations of surveillance video frames, is critical to our super-resolution strategy. This pre-trained StyleGAN acts as a latent feature bank by providing rich priors for creating realistic, high-quality, high-resolution videos. The proposed framework "TempoFuseNet" is presented in Figure 1, and the specifics of all stages are discussed, including the dataset, pseudocode for the super-resolution algorithm, and an explanation of our two-stream architecture for anomaly classification. The goal is not only to improve the spatial resolution of surveillance videos but also to provide a solid framework for accurately detecting anomalies in challenging real-world scenarios within 5G and IoT environments.

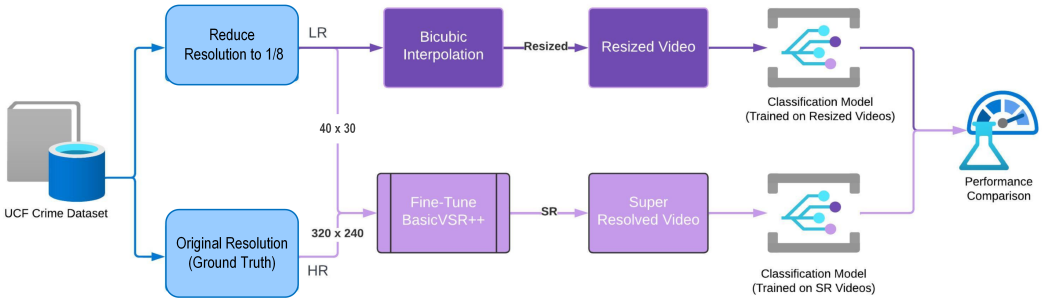


Figure 1. Proposed framework (TempoFuseNet) for anomaly classification for low-resolution videos.

3.1. Dataset

In order to perform classification learning to classify surveillance videos into one of several classes of anomalies, a labeled dataset of videos is required. Various datasets are used by the research community to demonstrate anomaly detection in surveillance videos, and each of these datasets has its own characteristics [2,39,53,54]. This study is based on the UCF-Crime dataset [2], which is modified to make it more useful for the demonstration of anomaly classification for low-quality surveillance videos.

There are 128 hours of surveillance footage in the original UCF-Crime dataset. The dataset includes 1900 complete and unfiltered surveillance videos from the real world, along with thirteen actual anomalies such as assault, arrest, abuse, arson, burglary, fighting, shooting, explosion, road accident, vandalism, robbery, and shoplifting. These anomalies were included in the dataset due to their possible impact on the safety of the general public. We meticulously curated the dataset to address class imbalance by retaining a standardized set of 50 videos per class to ensure the relevance and practicality of our study. Because of this deliberate selection process, classes with insufficient representation were excluded, resulting in a focused dataset with eight distinct categories: assault, arrest, abuse, arson, burglary, fighting, explosion, and normal. This strategic enhancement to the UCF-Crime

dataset ensures a balanced and representative collection, which improves the precision and applicability of our experimental results. All videos in each class have the same spatial resolution of 320 × 240 pixels, which contributes to the consistency and reliability of our analytical framework.

Data Preparation

In order to perform learning on low-quality videos, the original videos are downsampled by eight times to obtain a low-resolution version of the original videos. The video resolution after downsampling is 40 × 30 pixels . Downsampling is performed by using bilinear interpolation (refer to Equation (1)), in which the target image pixels are obtained by performing linear interpolation in both the horizontal and vertical directions.

Let $LR(x', y')$ be the low-resolution pixel values at coordinates (x', y'), and let $HR(x, y)$ be the high-resolution pixel values at coordinates (x, y). The downsampling operation can be expressed as:

$$LR(x', y') = f(HR(x, y)) \tag{1}$$

where

$$x' = [\, x/8], \quad y' = [\, y/8] \tag{2}$$

This stage results in two sets of data: one containing high-resolution (HR) videos that are the ground truth data, and the other has low-resolution (LR) videos, which are a downsampled version of the data and will be used for classification modeling.

3.2. Video Upscaling

Video resizing is the most commonly used operation to change the resolution of a video to match the requirements of the input layer of a convolutional neural network. There are various algorithms that can be used to perform the operation of video upscaling, and the most common are nearest neighbor interpolation, bilinear interpolation, bicubic interpolation, and Lanczos interpolation [55]. Among these methods, nearest neighbor is the fastest, and Lanczos is the slowest and most complex. Their upscaling performance is similarly related, but we used bicubic interpolation in our implementation due to its acceptable performance in terms of speed and upscaling quality. In order to perform bicubic interpolation for video scaling, we the used Libswscale library from the FFmpeg 4.2.1 package. The Libswscale library, which is developed in C and is part of the FFmpeg multimedia framework, includes highly optimized functions for scaling, colorspace conversion, and pixel format transformations.

3.3. Video Super Resolution

To obtain high-resolution videos, this study use a deep-learning-based video super-resolution approach as an effective strategy for overcoming technical limitations associated with low-quality videos: particularly, poor lighting and low spatial resolution. The proposed implementation employs GLEAN [56]: a framework that uses a Generative Adversarial Network (GAN) as a latent bank to extract rich and diverse priors from a pre-trained GAN model. Unlike traditional GAN-based methods, which involve adversarial loss and costly optimization through GAN inversion, our approach uses a single forward pass to generate high-resolution images.

To overcome poor lighting and low spatial resolution, a StyleGAN [57] is used in our implementation. The StyleGAN, fine-tuned on a dataset of surveillance videos with both low- and high-resolution representations of each frame, serves as a pre-trained latent feature bank. This latent feature bank functions similarly to a dictionary, but its distinct advantage is its nearly infinite feature bank, which provides superior priors for generating realistic high-resolution videos. Furthermore, our encoder–bank–decoder formulation, illustrated in Figures 2 and 3, is crucial for obtaining super-resolution images. Notably, the encoder accepts an input resolution of 40 × 30 pixels and outputs 320 × 240 pixels, demonstrating its ability to handle low-spatial-resolution scenarios. The latent feature bank,

which is powered by the pre-trained StyleGAN, ensures that the generated high-resolution videos retain realism and fidelity even in challenging lighting conditions.

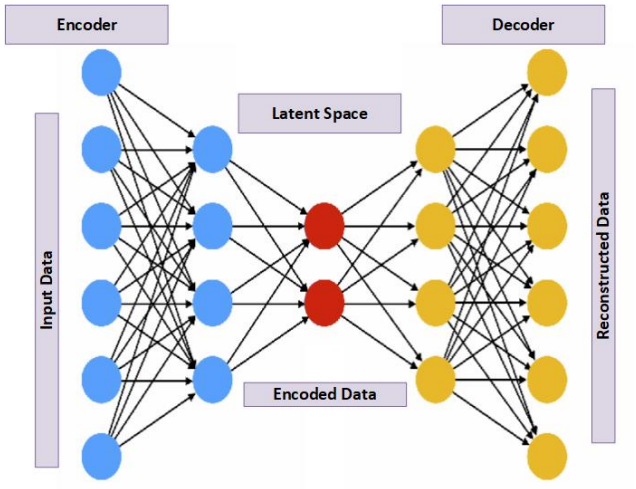


Figure 2. Encoder–bank–decoder representation [58].

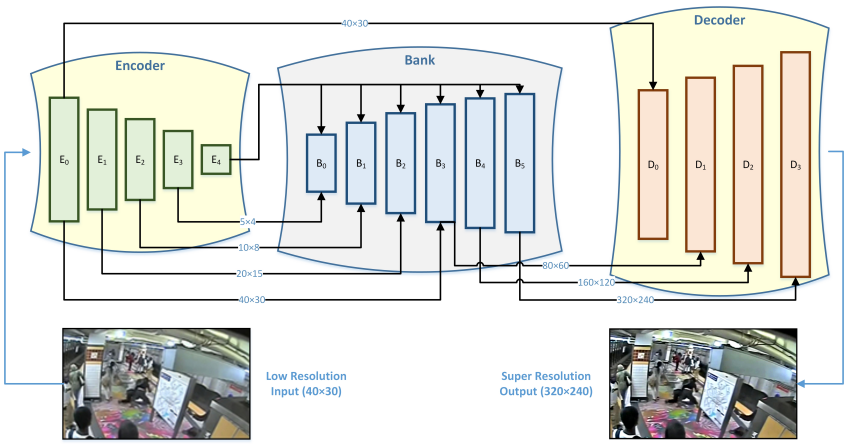


Figure 3. Video super-resolution framework (encoder–bank–decoder) based on pre-trained StyleGAN.

In order to obtain high-resolution videos apart from interpolation-based upscaling techniques, deep-learning-based video super resolution is an attractive approach. Generative Adversarial Networks (GANs) built using neural networks have shown excellent performance in video generation, enhancement, and super resolution, among other tasks. GLEAN [56] is an approach that uses a GAN-based model as a latent bank to obtain rich and diverse priors from pre-trained GAN. Unlike existing GAN-based approaches that generate realistic outputs through adversarial loss and the use of expensive optimization through GAN inversion, this approach uses a single forward pass to generate a high-resolution image. In this implementation, we used a StyleGAN [57] and fine-tuned it on a dataset of surveillance videos containing low-resolution and high-resolution representations of each frame.

Super-resolution images are obtained from low-resolution images by using an encoder–bank–decoder formulation. The latent features bank acts like a dictionary as in traditional approaches but differs in the sense that dictionaries contain a finite feature bank, whereas a GAN contains a practically infinite feature bank, making it a superior prior. The architecture of encoder–bank–decoder used in this implementation is provided in Figure 3. Note that the encoder accepts an input resolution of 40×30 pixels and provides an output of 320×240 pixels. The bank is a pre-trained StyleGAN that acts as a latent feature bank to provide realistic high-resolution videos.

3.4. Upscaling Performance

As discussed, there are various interpolation-based approaches that can be used for frame-by-frame video upscaling. The performances of four interpolation-based upscaling approaches along with the ground truth and the super-resolution image obtained by our implementation of GLEAN are provided in Figure 4. The results are provided for a single frame from a surveillance video belonging to the "fighting" class. Nearest neighbor and bilinear interpolation are the simplest and fastest methods to upscale an image but create the lowest-quality results. The difference between them is that bilinear interpolation provides a smoother image by using blurring, whereas nearest neighbor provides a boxing effect, and the choice of which one to use mainly depends on the intended application. Lanczos and bicubic are the next level of quality for upscaling and involve greater computational complexity. Lanczos has better detail preservation and a sharper appearance, while bicubic interpolation provides a smoother appearance. Because the targeted scenario for super resolution involves videos with a large number of frames, we use bicubic interpolation (Equation (3)) to upscale the video, which allows for a thorough comparison to the super-resolution videos.

$$I'(x', y') = \sum_{i=-1}^{2} \sum_{j=-1}^{2} I(x+i, y+j) \cdot K(x'-x+i) \cdot K(y'-y+j) \tag{3}$$

A super-resolution image produced using the modified GLEAN model is of much higher quality compared to its counterparts in terms of preservation of details and reconstruction of the structure. The modified GLEAN model includes improved architectural features and training strategies that allow for more effective detail preservation during the upscaling process. This entails a more sophisticated latent space representation or a fine-tuned generator network, which allow the model to capture and reproduce intricate details in the low-resolution input. The modified GLEAN model's superior quality of super-resolution images results from its advanced architecture and training strategies, which enable effective preservation of details and accurate reconstruction of complex structures when compared to other methods. To extract f_i features (Equations (4) and (5)) from a low-resolution image, we employed E_i sequence operations followed by Convolutional layers and fully connected layers to generate a matrix C of representative features.

$$f_i = E_i(f_{i-1}), \text{ for } i \in \{1, \ldots, N\} \tag{4}$$

$$C = E_{N+1}(f_N) \tag{5}$$

Moreover, it is evident from Figure 4 that a super-resolution image has higher overall contrast as compared to the ground truth image, which is due to the use of the latent feature bank containing a pre-trained StyleGAN.

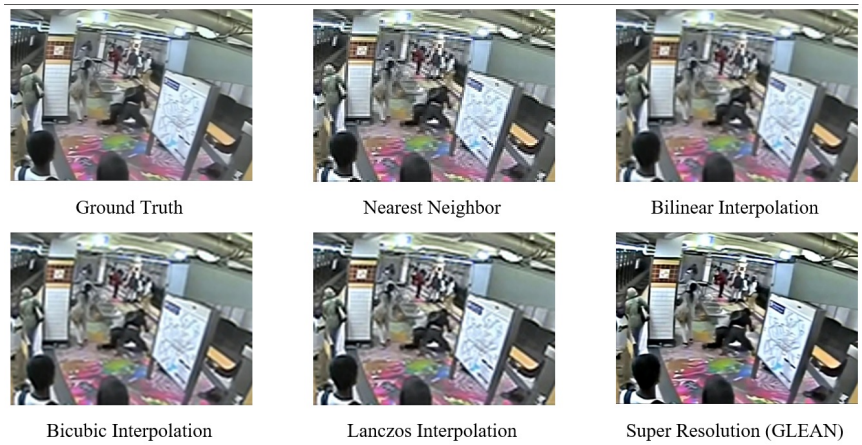


Figure 4. Video frame upscaling results for fighting scene.

Algorithms 1 and 2 are simplified pseudocode of the proposed "TempoFuseNet", with a focus on video super resolution using the modified GLEAN framework and anomaly classification using the two-stream architecture. This pseudocode is intended to provide an algorithmic and high-level representation of anomaly classification for real-world scenarios in 5G and IoT environments.

Algorithm 1: Video Super Resolution with GLEAN

```
Data: low_resolution_video
Result: super_resolved_video
GLEAN_model = initialize_GLEAN_with_StyleGAN();
foreach frame in low_resolution_video do
    super_resolved_frame = GLEAN_model.forward_pass(frame);
    Save super_resolved_frame;
return super_resolved_video;
```

Algorithm 2: Anomaly Classification with Two-Stream Architecture

```
Data: video_frames
Result: final_classification
for i to len(video_frames) 3 do
    spatial_features = extract_spatial_features(video_frames[i]);
    spatial_predictions = ResNet50_predict(spatial_features);
    Save spatial_predictions;
for frame to video_frames do
    SG3I_frame = convert_RGB_to_SG3I(frame);
    temporal_features = ResNet50_predict(SG3I_frame);
    Save temporal_features;
concatenated_features = concatenate(spatial_predictions, temporal_features);
temporal_model_output = apply_GRU(concatenated_features);
final_classification = Dense(temporal_model_output);
return final_classification;
```

3.5. Anomaly Classification

To perform anomaly classification, we used a two-stream architecture. Contrary to existing approaches that rely on the optical flow for one stream and the RGB image for the other stream, we used a simple but effective strategy that eliminates the need for expensive optical flow computation. The proposed two-stream architecture is depicted in Figure 5, whereas the details of both the spatial and temporal streams as well as late temporal modeling are provided later in this section.

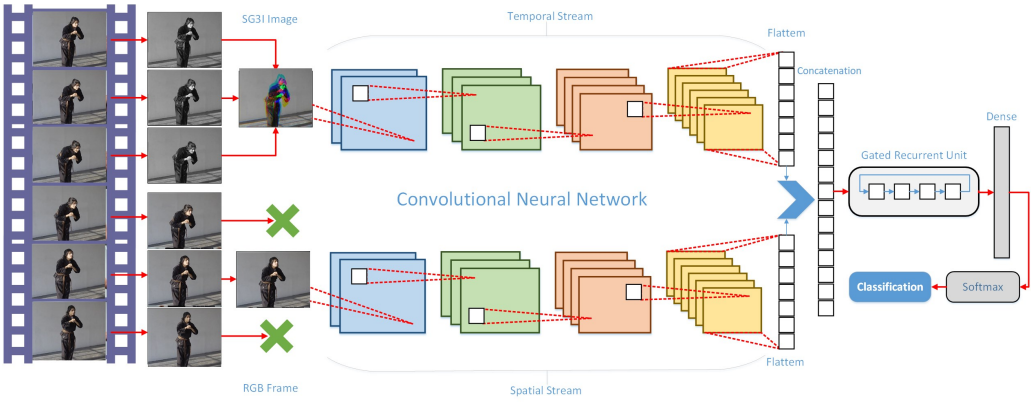


Figure 5. Proposed two-stream architecture for spatiotemporal learning for anomaly classification.

3.5.1. Spatial Stream

The spatial stream consists of a pre-trained CNN model base with its classification and dense layers are removed. The network performs prediction on an individual frame basis, and every third video frame is provided to the CNN model to match the predicted computational performance of the temporal stream. The spatial stream uses a ResNet50 model that effectively acts as a feature extractor from the RGB images obtained from the video stream.

3.5.2. Temporal Stream

In order to perform temporal learning without incurring a high computational load, we made use of Stacked Grayscale 3-channel Image (SG3I) (Equation (6)) [59].

$$SG3I(x, y) = R(x, y), G(x, y), B(x, y) \tag{6}$$

Here, SG3I(x, y) represents the function notation for the SG3I value at pixel coordinates (x, y), and $R(x, y), G(x, y), B(x, y)$ represents the intensity value at the same pixel coordinates. SG3I relies on the simple idea of combining multiple frames of video into single frame. The objective is achieved by converting the RGB frames into grayscale images. These grayscale images are combined to form a single 3-channel RGB image, and then, combining the three grayscale images forms the SG3I image, which acts like a single RGB video frame. The frames are selected in sequential order, and each subsequent grayscale frame is fitted to the red, green, and blue channels to yield a single RGB image. This new image is fed to the same pre-trained CNN model as the spatial stream, which serve two purposes: the SG3I image preserves the short temporal characteristics, and the grayscale conversion lets the model focus more on motion-related features.

3.5.3. Late Temporal Modeling

The features extracted from both the spatial and temporal streams are flattened and concatenated to perform feature fusion. In order to learn the long-term temporal characteristics of a video, late temporal modeling is performed from a concatenated feature set. Long Short-Term Memory (LSTM), bidirectional-LSTM, and Gated Recurrent Units (GRUs) are the three modeling approaches that are experimented with, and it is observed that GRUs provide the best temporal modeling characteristics, with a slight margin over LSTMs and bi-LSTMs. A possible explanation for the better performance of GRUs over LSTMs is the smaller size of the training dataset necessary to train a GRU. The GRU is followed by a dense layer and classification of the video into one of eight classes.

4. Experiments

4.1. Experimental Setup

Our experimental setup is intended to address the challenges posed by poor lighting and low spatial resolution in order to comprehensively assess the performance of anomaly detection in low-quality surveillance videos. The trimmed UCF-Crime dataset, which includes eight anomaly identification classes, is used in two different types of experiments.

The Trimmed UCF-Crime dataset has original video dimensions of 320×240. To simulate real-world scenarios with poor lighting and low spatial resolution, we intentionally reduced the spatial resolution by a factor of eight, resulting in low-resolution videos with dimensions of 40×30. It is important to note that this intentional downsampling is only for experimental purposes and is not a component of the proposed anomaly detection system. Moreover, we recognize that the term "low quality" can be broad: our research focuses on a specific aspect, low spatial resolution, to evaluate the robustness of our proposed methodology under these conditions.

In the first experiment, the downscaled dataset is upscaled to its original spatial resolution using bicubic interpolation. This experiment allows us to assess the performance of our proposed anomaly classifier under standard upscaling conditions and serves as the

baseline for comparison. In the second experiment, we use an advanced GLEAN-based model for super resolution. This model upscales low-resolution videos, resulting in super-resolved video frames. These frames are then used for classification modeling with our proposed anomaly classifier. This experiment addresses the issue of low spatial resolution by employing advanced super-resolution techniques. The experiments are carried out in TensorFlow with the Keras 2.4.0 backend on a Windows 10 machine. Table 1 shows the detailed system setup used for the experiments. During the model's training phase, a random search is used to select specific hyperparameters. To ensure the best model performance, we use the Adam optimizer with a piecewise learning rate. If no progress is seen after minimizing the learning rate for three consecutive validation checks, the training is stopped. Table 2 summarizes the hyperparameter tuning process and offers insights into optimizing different parameters for the best results.

Table 1. System specifications.

#	Type	Specifications
1	System	Dell Precision T5600
2	CPU	2× Intel® Xeon® Processor E5-2687W
3	RAM	32GB DDR3
4	GPU	GeForce RTX 2070
5	GPU Memory	8GB GDDR6
6	CUDA Cores	2304
7	Storage	512GB SSD

This experimental setup was designed to simulate and effectively address the technical limitations associated with poor lighting and low spatial resolution in real-world surveillance scenarios. This was to ensure a thorough evaluation of TempoFuseNet, our proposed anomaly identification framework.

Table 2. Training parameters used to train the model.

#	Training Parameter	Value
1	Optimizer	Adam
2	Initial Learning Rate	0.003
3	Learning Rate Schedule	Piecewise
4	Learning Rate Drop Factor	0.5
5	Gradient Decay Factor	0.9
6	L2 Regularization	0.0001
7	Max Epochs	100
8	Mini Batch Size	32
9	Loss Function	Categorical cross-entropy
10	Validation Check	Every epoch

4.2. Evaluation Method and Metrics

Model evaluation is a way to assess the skill of a prediction model, which is a classifier in our case. The model is trained and evaluated using holdout validation in which the data are partitioned into an 80:20 ratio with 80% of the data being used for model training and validation and 20% holdout data being used for model testing. The performance of both experiments is reported for the same train–test split of the data to make a fair comparison. Evaluation of the model's performance is made based on various performance metrics

obtained from the confusion matrix. The comparative performance for both models is also provided to assess the overall anomaly identification performance.

5. Results

Anomaly identification in surveillance videos is a difficult task, especially when using low-quality videos with poor spatial resolution and visual characteristics. Traditional methods, such as spatial interpolation, frequently result in limited improvement and can introduce undesirable artifacts. Alternatively, video super resolution, which improves spatial resolution, can be computationally expensive. This study addresses the challenges of low-quality videos using a video super-resolution approach based on StyleGAN priors. The StyleGAN improves not only the spatial resolution but also image sharpness and contrast. Unlike traditional video super-resolution methods, our approach selectively super-resolves frames that are relevant to anomaly identification, thereby improving computational efficiency. A two-stream architecture is used for classification modeling, which reduces the need for expensive optical flow computation. The RGB stream promotes spatial learning, whereas the SG3I stream emphasizes short-term temporal learning. Both streams use the same pre-trained CNN architecture, which has been fine-tuned for the dataset of interest. The learned features are concatenated and fed into a Gated Recurrent Unit (GRU) for long-term temporal modeling. The proposed approach effectively addresses the challenges posed by low-quality surveillance videos and delivers superior anomaly classification performance while minimizing the computational burden.

5.1. Classification Performance

5.1.1. Bicubic Interpolation of Videos

The classification performance of the upscaled images using bicubic interpolation is provided in the confusion matrix in Figure 6. It is to be noted that out of 50 videos in each class, 40 videos are used for model training, and 10 videos are used for model testing. The confusion matrix provides the actual number of videos classified into each category. Table 3 provides the performance metrics for each class as well as the macro-averaged value for all classes. Classification accuracy is usually regarded as the most important performance metric for anomaly classification, followed by the FPR. Moreover, the values of precision, recall (sensitivity), F1-score, specificity, FPR, and FNR are also reported for each class and are averaged for all classes.

5.1.2. Super-Resolution Videos

Like for the bicubicly interpolated videos, the classification performance for super-resolution videos is provided in the confusion matrix in Figure 7. The confusion matrix reports the classification performance based on 10 videos per anomaly class. Table 4 provides the performance metrics for each class as well as the macro-averaged value for all classes.

	Abuse	Arrest	Arson	Assault	Burglary	Explosion	Fight	Normal
Abuse	5	0	0	3	0	0	2	0
Arrest	0	5	0	0	0	0	3	2
Arson	0	0	5	0	0	4	0	2
Assault	3	0	0	5	0	0	2	0
Burglary	1	0	0	1	5	0	0	3
Explosion	0	0	3	0	0	6	0	1
Fight	2	0	0	2	0	0	6	0
Normal	0	0	1	0	2	0	0	7

Figure 6. Confusion matrix for bicubic interpolation of videos.

Table 3. Classification performance for bicubic interpolation of videos.

Class	Accuracy	Precision	Recall	F1-Score	Specificity	FPR	FNR
Abuse	86.42%	45.45%	50.00%	47.62%	91.55%	8.45%	50.00%
Arrest	93.83%	100.00%	50.00%	66.67%	100.00%	0.00%	50.00%
Arson	87.65%	55.56%	45.45%	50.00%	94.29%	5.71%	54.55%
Assault	86.42%	45.45%	50.00%	47.62%	91.55%	8.45%	50.00%
Burglary	91.36%	71.43%	50.00%	58.82%	97.18%	2.82%	50.00%
Explosion	90.12%	60.00%	60.00%	60.00%	94.37%	5.63%	40.00%
Fight	86.42%	46.15%	60.00%	52.17%	90.14%	9.86%	40.00%
Normal	86.42%	46.67%	70.00%	56.00%	88.73%	11.27%	30.00%
Macro-Average	88.58%	58.84%	54.43%	54.86%	93.48%	6.52%	45.57%

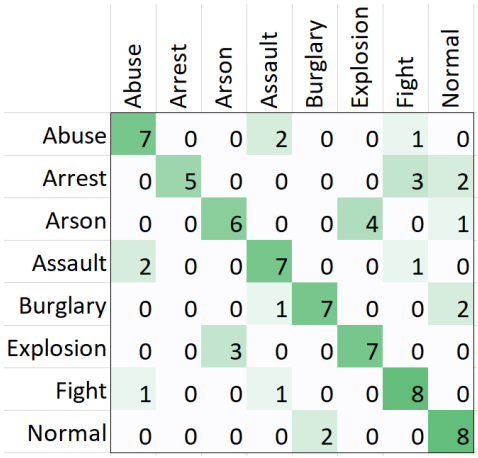


Figure 7. Confusion matrix for super-resolution videos.

Table 4. Classification performance for super-resolution videos.

Class	Accuracy	Precision	Recall	F1-Score	Specificity	FPR	FNR
Abuse	92.59%	70.00%	70.00%	70.00%	95.77%	4.23%	30.00%
Arrest	95.06%	100.00%	60.00%	75.00%	100.00%	0.00%	40.00%
Arson	90.12%	66.67%	54.55%	60.00%	95.71%	4.29%	45.45%
Assault	91.36%	63.64%	70.00%	66.67%	94.37%	5.63%	30.00%
Burglary	93.83%	77.78%	70.00%	73.68%	97.18%	2.82%	30.00%
Explosion	91.36%	63.64%	70.00%	66.67%	94.37%	5.63%	30.00%
Fight	92.59%	66.67%	80.00%	72.73%	94.37%	5.63%	20.00%
Normal	91.36%	61.54%	80.00%	69.57%	92.96%	7.04%	20.00%
Macro-Average	92.28%	71.24%	69.32%	69.29%	95.59%	4.41%	30.68%

5.2. Comparison with Existing Approaches

To validate the effectiveness of our proposed methodology, we conducted an extensive experimental variations. In addition to these experiments, we performed a comparative analysis between the TempoFuseNet framework and existing state-of-the-art approaches that center around multiclass anomaly classification, using the UCF-Crime dataset as our testing ground. In a similar context, Maqsood et al. [35] introduced a convolutional neural-network-based approach that initiates with video preprocessing to create 3D cubes through spatial augmentation. To streamline the analysis process, they employed a subset of the dataset: eliminating extraneous data and manually identifying atypical segments to ensure class balance. Subsequently, these 3D video cubes were fed into a convolutional neural network (CNN) to extract spatiotemporal features. Their analysis of the UCF-Crime dataset yielded a classification accuracy of 45% across fourteen distinct classes. In another study, Tiwari et al. [60] employed a fuzzy-rule-based approach for video summarization with the aim of addressing issues related to excessive data and high computational costs. Tiwari

et al. [60] achieved a classification accuracy of 53% in their classification experiment by leveraging a hybrid slow–fast neural network.

On the other hand, our study utilized a trimmed UCF-Crime dataset comprising eight classes and fifty videos. For the anomaly classification task, we applied a two-step approach: First, we upscaled the low-resolution (LR) videos using bicubic interpolation and an encoder–bank–decoder configuration for super resolution. The encoder and decoder played pivotal roles in downscaling and upscaling, while the bank was a pre-trained StyleGAN acting as a latent feature repository to enhance super-resolution performance based on feature priors. Our experiments encompassed both types of upscaled images, and the results were systematically compared in order to highlight the effectiveness of our super-resolution approach. Anomaly recognition was executed through a two-stream architecture wherein a pre-trained CNN model extracted features from RGB images in the spatial stream, and Stacked Grayscale 3-channel Images (SG3I) were used in the temporal stream, substantially reducing the computational load of optical flow computation while capturing short-term temporal characteristics. The features from both streams were concatenated and passed through a Gated Recurrent Unit (GRU) layer to capture long-term temporal characteristics. The output of the GRU layer was then processed through dense and softmax layers before reaching the final classification layer. Our proposed methodology, coupled with the encoder–bank–decoder super-resolution model, yielded remarkable results, achieving a classification accuracy of 92.28%, an F1-score of 69.29%, and a false positive rate of just 4.41%.

5.3. Comparison of Bicubic Interpolation and Super-Resolution Approaches

In order to perform a comparison of both approaches, a bar-chart is plotted, as shown in Figure 8, for seven classification evaluation metrics; the chart clearly shows the superiority of super resolution over bicubic interpolation to perform anomaly identification. It is to be noted that the reported scores for accuracy, precision, recall, F1-score, and specificity are higher for super-resolution videos in comparison to bicubic interpolation videos, which is desirous, as higher values for these metrics indicate good classification performance. On the other hand, FPR and FNR should be lower for a good classification system, and therefore, their values are lower for video super-resolution scenarios. A clear performance gap indicates that super-resolution-based anomaly detection models are very effective when the video stream is of low spatial resolution.

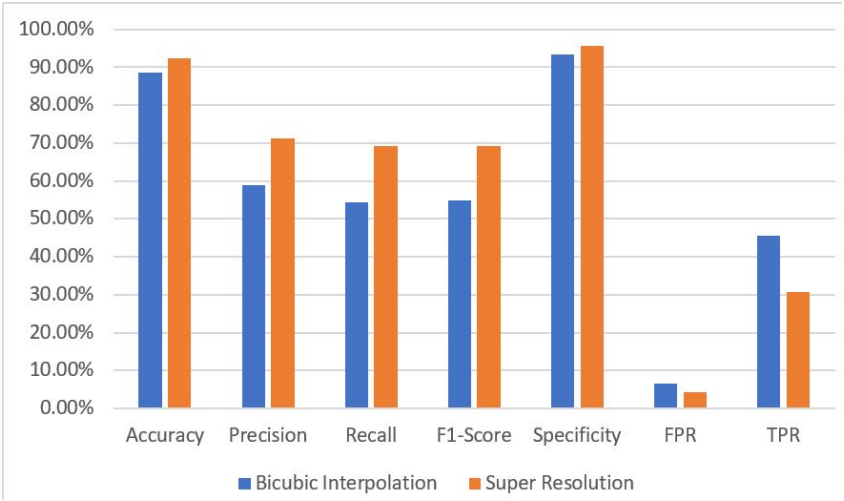


Figure 8. Comparison of bicubic interpolation and super-resolution approaches.

6. Conclusions

This study addressed the challenge of multi-class anomaly identification using low-quality surveillance videos within 5G and IoT environments. By conducting experiments on the trimmed UCF-crime dataset, the videos were downscaled to 1/8 resolution and then

upscaled using bicubic interpolation and super-resolution techniques. The TempoFuseNet framework employed a two-stream architecture that was followed by GRU for long-term temporal modeling. The experimental results showcased remarkable performance, with a classification accuracy of 92.28%, F1-score of 69.29%, and false positive rate of 4.41%. Moreover, the integration of super resolution in the anomaly classifier yielded substantial enhancements over the videos upscaled using bicubic interpolation. Specifically, the super-resolution-based approach achieved a 3.7% improvement in accuracy, a significant 14.34% boost in the F1-score, and a commendable 2.11% reduction in the false positive rate. Hence, TempoFuseNet outperforms existing state-of-the-art methods in multiclass classification performance and effectively addresses the technical limitations caused by low-quality videos, making it a robust solution for real-world surveillance scenarios, particularly in 5G and IoT environments.

Future Work

This study makes significant progress in improving video quality and anomaly detection in surveillance scenarios. However, future research could focus on integrating real-time processing capabilities and on investigating methods for automatically fine-tuning the model in response to changes to the lighting, spatial characteristics, or other dynamic factors. Moreover, integrating multi-modal data sources, such as contextual information, could improve anomaly detection accuracy and broaden the system's applicability in a variety of surveillance scenarios.

Author Contributions: Conceptualization, G.S. and U.I.B.; methodology, G.S.; software, G.S.; validation, G.S., U.I.B., R.H.R. and F.Z.; formal analysis, G.S. and U.I.B.; investigation, G.S., U.I.B., R.H.R. and F.Z.; resources, G.S.; data curation, G.S., U.I.B. and R.H.R.; writing—original draft preparation, G.S., U.I.B., R.H.R. and F.Z.; writing—review and editing, G.S., U.I.B., R.H.R. and F.Z.; visualization, G.S.; supervision, U.I.B. and R.H.R.; project administration, G.S., U.I.B. and R.H.R.; funding acquisition, F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Natural Science Foundation of China with award number 62372409, and by the Ministry of Science and Technology with award number DL2023147001L.

Data Availability Statement: The dataset used in this study for conducting experiments for Anomaly Recognition is publicly available at: UCF-CRIME. Retrieved September 2023, https://www.crcv.ucf.edu/projects/real-world/ (accessed on 8 January 2024).

Acknowledgments: This study acknowledges partial support from the National Center of Big Data and Cloud Computing (NCBC) and HEC of Pakistan.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could appear to have influence the work reported in this paper.

References

1. Hasan, M.; Choi, J.; Neumann, J.; Roy-Chowdhury, A.K.; Davis, L.S. Learning temporal regularity in video sequences. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 733–742.
2. Sultani, W.; Chen, C.; Shah, M. Real-world anomaly detection in surveillance videos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6479–6488.
3. Zhong, J.X.; Li, N.; Kong, W.; Liu, S.; Li, T.H.; Li, G. Graph convolutional label noise cleaner: Train a plug-and-play action classifier for anomaly detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–17 June 2019; pp. 1237–1246.
4. Akhtar, M. Automated analysis of visual leaf shape features for plant classification. *Comput. Electron. Agric.* **2019**, *157*, 270–280.
5. Ahmad, N.; Asif, H.M.S.; Saleem, G.; Younus, M.U.; Anwar, S.; Anjum, M.R. Leaf image-based plant disease identification using color and texture features. *Wirel. Pers. Commun.* **2021**, *121*, 1139–1168. [CrossRef]
6. Aslam, M.A.; Wei, X.; Ahmed, N.; Saleem, G.; Amin, T.; Caixue, H. Vrl-iqa: Visual representation learning for image quality assessment. *IEEE Access* **2024**, *12*, 2458–2473. [CrossRef]
7. Ahmed, N.; Asif, H.M.S. Perceptual quality assessment of digital images using deep features. *Comput. Inform.* **2020**, *39*, 385–409. [CrossRef]

8. Ahmed, N.; Asif, H.M.S. Ensembling convolutional neural networks for perceptual image quality assessment. In Proceedings of the 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), Karachi, Pakistan, 14–15 December 2019; pp. 1–5.
9. Saleem, G.; Bajwa, U.I.; Raza, R.H.; Alqahtani, F.H.; Tolba, A.; Xia, F. Efficient anomaly recognition using surveillance videos. *PeerJ Comput. Sci.* **2022**, *8*, e1117. [CrossRef] [PubMed]
10. Elharrouss, O.; Almaadeed, N.; Al-Maadeed, S. A review of video surveillance systems. *J. Vis. Commun. Image Represent.* **2021**, *77*, 103116. [CrossRef]
11. Duong, H.T.; Le, V.T.; Hoang, V.T. Deep Learning-Based Anomaly Detection in Video Surveillance: A Survey. *Sensors* **2023**, *23*, 5024. [CrossRef]
12. Ahmed, N.; Asif, H.M.S.; Khalid, H. Image quality assessment using a combination of hand-crafted and deep features. In *Proceedings of the Intelligent Technologies and Applications: Second International Conference, INTAP 2019, Bahawalpur, Pakistan, 6–8 November 2019*; Revised Selected Papers 2; Springer: Singapore, 2020; pp. 593–605.
13. Khalid, H.; Ali, M.; Ahmed, N. Gaussian process-based feature-enriched blind image quality assessment. *J. Vis. Commun. Image Represent.* **2021**, *77*, 103092. [CrossRef]
14. Ahmed, N.; Asif, S. BIQ2021: A large-scale blind image quality assessment database. *J. Electron. Imaging* **2022**, *31*, 053010. [CrossRef]
15. Ahmed, N.; Asif, H.M.S.; Saleem, G.; Younus, M.U. Image quality assessment for foliar disease identification (agropath). *J. Agric. Res.* **2021**, *59*, 177–186.
16. Ahmed, N.; Shahzad Asif, H.; Bhatti, A.R.; Khan, A. Deep ensembling for perceptual image quality assessment. *Soft Comput.* **2022**, *26*, 7601–7622. [CrossRef]
17. Ahmed, N.; Asif, H.M.S.; Khalid, H. PIQI: Perceptual image quality index based on ensemble of Gaussian process regression. *Multimed. Tools Appl.* **2021**, *80*, 15677–15700. [CrossRef]
18. Ullah, W.; Ullah, A.; Hussain, T.; Khan, Z.A.; Baik, S.W. An efficient anomaly recognition framework using an attention residual LSTM in surveillance videos. *Sensors* **2021**, *21*, 2811. [CrossRef]
19. Zhou, Y.; Du, X.; Wang, M.; Huo, S.; Zhang, Y.; Kung, S.Y. Cross-scale residual network: A general framework for image super-resolution, denoising, and deblocking. *IEEE Trans. Cybern.* **2021**, *52*, 5855–5867. [CrossRef]
20. Kwan, C.; Zhou, J.; Wang, Z.; Li, B. Efficient anomaly detection algorithms for summarizing low quality videos. In Proceedings of the Pattern Recognition and Tracking XXIX, Orlando, FL, USA, 15–19 April 2018; SPIE: Bellingham, WA, USA; Volume 10649, pp. 45–55.
21. Zhou, J.; Kwan, C. Anomaly detection in low quality traffic monitoring videos using optical flow. In Proceedings of the Pattern Recognition and Tracking XXIX, Orlando, FL, USA, 15–19 April 2018; SPIE: Bellingham, WA, USA, 2018; Volume 10649, pp. 122–132.
22. Lv, Z.; Wu, J.; Xie, S.; Gander, A.J. Video enhancement and super-resolution. In *Digital Image Enhancement and Reconstruction*; Elsevier: Amsterdam, The Netherlands, 2023; pp. 1–28.
23. Wu, J. Introduction to convolutional neural networks. *Natl. Key Lab Nov. Softw. Technol. Nanjing Univ. China* **2017**, *5*, 495.
24. Nguyen, T.N.; Meunier, J. Hybrid deep network for anomaly detection. *arXiv* **2019**, arXiv:1908.06347.
25. Ryoo, M.; Kim, K.; Yang, H. Extreme low resolution activity recognition with multi-siamese embedding learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; PKP Publishing Services Network; Volume 32.
26. Saleem, G.; Bajwa, U.I.; Raza, R.H. Toward human activity recognition: A survey. *Neural Comput. Appl.* **2023**, *35*, 4145–4182. [CrossRef]
27. Nayak, R.; Pati, U.C.; Das, S.K. A comprehensive review on deep learning-based methods for video anomaly detection. *Image Vis. Comput.* **2021**, *106*, 104078. [CrossRef]
28. Wang, H.; Schmid, C. Action recognition with improved trajectories. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, Australia, 1–8 December 2013; pp. 3551–3558.
29. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
30. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
31. Simonyan, K.; Zisserman, A. Two-stream convolutional networks for action recognition in videos. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 568–576.
32. Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1725–1732.
33. Wang, L.; Xiong, Y.; Wang, Z.; Qiao, Y.; Lin, D.; Tang, X.; Van Gool, L. Temporal segment networks for action recognition in videos. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 2740–2755. [CrossRef]
34. Tran, D.; Bourdev, L.; Fergus, R.; Torresani, L.; Paluri, M. Learning spatiotemporal features with 3d convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 4489–4497.
35. Maqsood, R.; Bajwa, U.I.; Saleem, G.; Raza, R.H.; Anwar, M.W. Anomaly recognition from surveillance videos using 3D convolution neural network. *Multimed. Tools Appl.* **2021**, *80*, 18693–18716. [CrossRef]

36. Luo, W.; Liu, W.; Gao, S. Remembering history with convolutional lstm for anomaly detection. In Proceedings of the 2017 IEEE International Conference on Multimedia and Expo (ICME), Hong Kong, China, 10–14 July 2017; pp. 439–444.
37. Ullah, M.; Mudassar Yamin, M.; Mohammed, A.; Daud Khan, S.; Ullah, H.; Alaya Cheikh, F. Attention-based LSTM network for action recognition in sports. *Electron. Imaging* **2021**, *2021*, 302–311. [CrossRef]
38. Riaz, H.; Uzair, M.; Ullah, H.; Ullah, M. Anomalous human action detection using a cascade of deep learning models. In Proceedings of the 2021 9th European Workshop on Visual Information Processing (EUVIP), Paris, France, 23–25 June 2021; pp. 1–5.
39. Liu, W.; Luo, W.; Lian, D.; Gao, S. Future frame prediction for anomaly detection—A new baseline. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6536–6545.
40. Landi, F.; Snoek, C.G.; Cucchiara, R. Anomaly locality in video surveillance. *arXiv* **2019**, arXiv:1901.10364.
41. Sabokrou, M.; Khalooei, M.; Fathy, M.; Adeli, E. Adversarially learned one-class classifier for novelty detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 3379–3388.
42. Zhou, Y.; Qu, Y.; Xu, X.; Shen, F.; Song, J.; Shen, H. BatchNorm-based Weakly Supervised Video Anomaly Detection. *arXiv* **2023**, arXiv:2311.15367.
43. Pu, Y.; Wu, X.; Wang, S. Learning Prompt-Enhanced Context Features for Weakly-Supervised Video Anomaly Detection. *arXiv* **2023**, arXiv:2306.14451.
44. Gan, K.Y.; Cheng, Y.T.; Tan, H.K.; Ng, H.F.; Leung, M.K.; Chuah, J.H. Contrastive-regularized U-Net for Video Anomaly Detection. *IEEE Access* **2023**, *11*, 36658–36671. [CrossRef]
45. Ryoo, M.; Rothrock, B.; Fleming, C.; Yang, H.J. Privacy-preserving human activity recognition from extreme low resolution. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
46. Chen, J.; Wu, J.; Konrad, J.; Ishwar, P. Semi-coupled two-stream fusion convnets for action recognition at extremely low resolutions. In Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), Santa Rosa, CA, USA, 24–31 March 2017; pp. 139–147.
47. Xu, M.; Sharghi, A.; Chen, X.; Crandall, D.J. Fully-coupled two-stream spatiotemporal networks for extremely low resolution action recognition. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 12–15 March 2018; pp. 1607–1615.
48. Demir, U.; Rawat, Y.S.; Shah, M. Tinyvirat: Low-resolution video action recognition. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 7387–7394.
49. Hou, W.; Wang, X.; Chouinard, J.Y.; Refaey, A. Physical layer authentication for mobile systems with time-varying carrier frequency offsets. *IEEE Trans. Commun.* **2014**, *62*, 1658–1667. [CrossRef]
50. Ataer-Cansizoglu, E.; Jones, M.; Zhang, Z.; Sullivan, A. Verification of very low-resolution faces using an identity-preserving deep face super-resolution network. *arXiv* **2019**, arXiv:1903.10974.
51. Bai, Y.; Zhang, Y.; Ding, M.; Ghanem, B. Sod-mtgan: Small object detection via multi-task generative adversarial network. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 206–221.
52. Wang, Z.; Ye, M.; Yang, F.; Bai, X.; Satoh, S. Cascaded SR-GAN for scale-adaptive low resolution person re-identification. In Proceedings of the IJCAI, Stockholm, Sweden, 13–19 July 2018; Volume 1, p. 4.
53. Lu, C.; Shi, J.; Jia, J. Abnormal event detection at 150 fps in matlab. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, Australia, 1–8 December 2013; pp. 2720–2727.
54. Mahadevan, V.; Li, W.; Bhalodia, V.; Vasconcelos, N. Anomaly detection in crowded scenes. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Beijing, China, 24–28 October 2010; pp. 1975–1981.
55. Han, D. Comparison of commonly used image interpolation methods. In Proceedings of the Conference of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013), Los Angeles, CA, USA, 1–2 July 2013; Atlantis Press: Amsterdam, The Netherlands, 2013; pp. 1556–1559.
56. Chan, K.C.; Xu, X.; Wang, X.; Gu, J.; Loy, C.C. GLEAN: Generative latent bank for image super-resolution and beyond. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *45*, 3154–3168. [CrossRef]
57. Karras, T.; Laine, S.; Aittala, M.; Hellsten, J.; Lehtinen, J.; Aila, T. Analyzing and improving the image quality of stylegan. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 8110–8119.
58. Barkhordar, E.; Shirali-Shahreza, M.H.; Sadeghi, H.R. Clustering of Bank Customers using LSTM-based encoder-decoder and Dynamic Time Warping. *arXiv* **2021**, arXiv:2110.11769.
59. Kim, J.H.; Won, C.S. Action recognition in videos using pre-trained 2D convolutional neural networks. *IEEE Access* **2020**, *8*, 60179–60188. [CrossRef]
60. Tiwari, A.; Chaudhury, S.; Singh, S.; Saurav, S. Video Classification using SlowFast Network via Fuzzy rule. In Proceedings of the 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Luxembourg, 11–14 June 2021; pp. 1–6.

future internet

MDPI

Article

Intelligent Resource Orchestration for 5G Edge Infrastructures

Rafael Moreno-Vozmediano [1,*,†], Rubén S. Montero [1,2,†], Eduardo Huedo [1,†] and Ignacio M. Llorente [2,†]

[1] Faculty of Computer Science, Complutense University of Madrid, 28040 Madrid, Spain; rubensm@ucm.es (R.S.M.); ehuedo@ucm.es (E.H.)
[2] OpenNebula Systems, Paseo del Club Deportivo 1, Pozuelo de Alarcón, 28223 Madrid, Spain; imllorente@opennebula.io
* Correspondence: rmoreno@ucm.es
† These authors contributed equally to this work.

Abstract: The adoption of edge infrastructure in 5G environments stands out as a transformative technology aimed at meeting the increasing demands of latency-sensitive and data-intensive applications. This research paper presents a comprehensive study on the intelligent orchestration of 5G edge computing infrastructures. The proposed Smart 5G Edge-Cloud Management Architecture, built upon an OpenNebula foundation, incorporates a ONEedge5G experimental component, which offers intelligent workload forecasting and infrastructure orchestration and automation capabilities, for optimal allocation of virtual resources across diverse edge locations. The research evaluated different forecasting models, based both on traditional statistical techniques and machine learning techniques, comparing their accuracy in CPU usage prediction for a dataset of virtual machines (VMs). Additionally, an integer linear programming formulation was proposed to solve the optimization problem of mapping VMs to physical servers in distributed edge infrastructure. Different optimization criteria such as minimizing server usage, load balancing, and reducing latency violations were considered, along with mapping constraints. Comprehensive tests and experiments were conducted to evaluate the efficacy of the proposed architecture.

Keywords: 5G edge infrastructures; intelligent edge orchestration; workload forecasting; resource allocation and optimization; machine learning; integer linear programming

Citation: Moreno-Vozmediano, R.; Montero, R.S.; Huedo, E.; Llorente, I.M. Intelligent Resource Orchestration for 5G Edge Infrastructures. *Future Internet* **2024**, *16*, 103. https://doi.org/10.3390/fi16030103

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 16 February 2024
Revised: 11 March 2024
Accepted: 16 March 2024
Published: 19 March 2024

1. Introduction

The evolution of 5G technology has ushered in a new era of connectivity, offering enhanced capabilities that extend beyond traditional communication paradigms. Advanced 5G networks facilitate a diverse range of applications, characterized by their complexity, computational demand, and low-latency requirements. These applications, designed to leverage the capabilities of advanced 5G, exhibit a hybrid profile, relying on resources spanning the spectrum from data centers to the cloud to the edge.

In particular, the deployment of edge infrastructure in 5G environments is crucial to addressing the unique challenges posed by latency-sensitive and data-intensive applications [1,2]. The next generation of applications, encompassing smart IoT applications, real-time analytics, machine learning applications, and more, require intelligent orchestration of resources in the edge domain, where computational tasks are strategically placed closer to data sources to minimize latency and enhance overall system performance.

Orchestrating edge infrastructure in 5G environments is not without its complexities. The heterogeneous nature of edge devices, coupled with the dynamic and resource-constrained characteristics of these environments, presents challenges for effective resource management. Unlike traditional cloud-centric models, the edge requires a nuanced approach that considers factors such as reliability, security, data protection, and energy efficiency. In light of these challenges, the need for intelligent orchestration becomes paramount. An orchestration system driven by artificial intelligence (AI) and machine

learning (ML) techniques can dynamically allocate and coordinate resources across a distributed edge infrastructure.

In the current cloud market, various vendors and tools claim to employ intelligent techniques for resource management and monitoring. For instance, VMware implements a predictive distributed resource scheduler (DRS) [3] for cloud platforms, which forecasts future demand and preemptively addresses potential hot spots, by reallocating workloads well in advance of any contention. OPNI [4], an open source project from SUSE, is another noteworthy example aimed at enhancing observability, monitoring, and logging in Kubernetes-based clusters. It offers a range of AIOps tools for detecting log anomalies, identifying root causes, and spotting metric irregularities. Google Active Assist [5] is another tool that aims to provide intelligent solutions for improving cloud operations, by offering recommendations for cost reduction, performance enhancement, security improvement, and sustainability. Generally, these prediction and optimization techniques deployed by cloud stakeholders are tailored for centralized cloud platforms and often comprise simple, proprietary solutions that may not be adaptable to highly distributed 5G edge environments. On the other hand, recent initiatives like OpenNebula OneEdge [6] enable the deployment and management of geo-distributed edge/cloud infrastructures, leveraging resources from various public cloud and edge infrastructure providers. However, these tools are still nascent and primarily offer basic management functionalities rather than advanced, intelligent orchestration capabilities for optimizing the deployment of large scale edge infrastructures.

In this work, we propose a pioneering Smart 5G Edge-Cloud Management Architecture, which seeks to expand the existing edge management platforms by integrating intelligent orchestration capabilities. This integration aims to automate and optimize the provisioning and deployment of geographically distributed 5G edge infrastructures. This architecture, built upon the foundation of OpenNebula [7,8], will integrate cutting-edge experimental components under development in the ONEedge5G project, as shown in Figure 1. ONEedge5G aims to enable efficient capacity planning, provisioning, and risk prevention in geographically distributed edge infrastructures and applications within the context of advanced 5G networks. This is achieved through the characterization and monitoring of edge infrastructures and virtual applications, prediction of the state of the data center–cloud–edge continuum, and programmatic intervention based on these predictions.

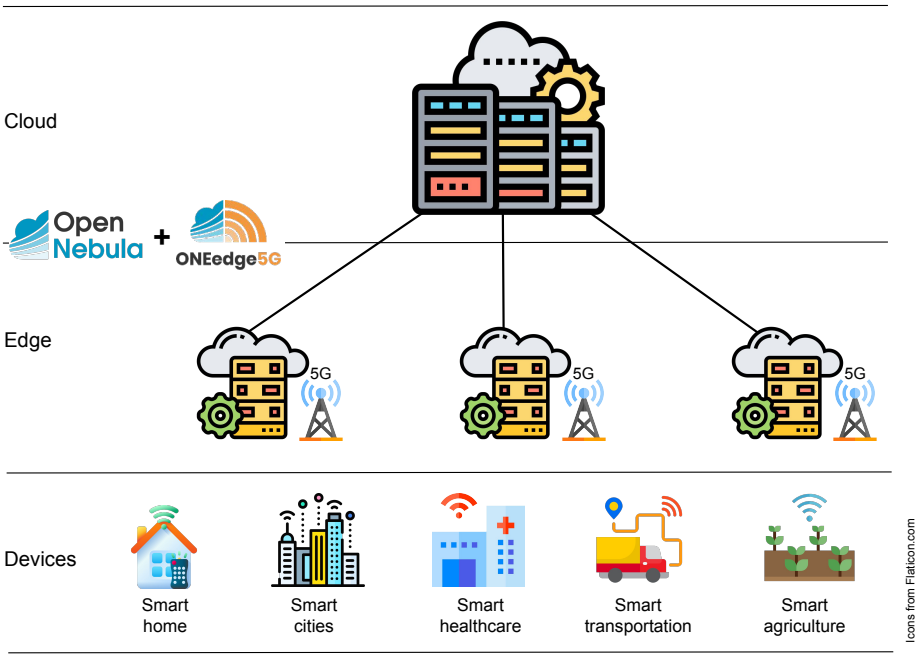


Figure 1. OpenNebula + ONEedge5G for intelligent orchestration of multiple 5G edge locations.

In its current developmental phase, ONEedge5G integrates diverse predictive intelligence mechanisms for workload forecasting. It also embeds various optimization algorithms to ensure optimal resource allocation across multiple edge locations. This paper introduces and assesses these intelligent techniques, demonstrating their effectiveness for improving the management and performance of distributed edge infrastructures.

To evaluate the efficacy of our Smart 5G Edge-Cloud Management Architecture, we conducted comprehensive tests and experiments. Through rigorous analysis, we assessed the capabilities of a ONEedge5G prototype in accurate workload forecasting and optimization. By presenting the results of these experiments, we demonstrate the benefits and efficiency gains achieved by leveraging intelligent prediction and optimization techniques in 5G edge management.

The remainder of this paper is structured as follows: Section 2 analyzes the advantages and challenges of edge computing on 5G networks. Section 3 discusses related works. Section 4 presents the design and main components of the Smart 5G Edge-Cloud Management Architecture. Section 5 summarizes the time-series forecasting models employed for workload prediction. Section 6 introduces the mathematical models utilized in the integer linear programming (ILP) formulation for edge resource optimization. Section 7 demonstrates the virtual resource CPU usage forecasting results for the different datasets and the resource optimization outcomes for the various objective functions and constraints. Finally, Section 8 summarizes the conclusions of this study and suggests potential directions for future research.

2. Edge Computing and Advanced 5G Networks

Advanced 5G networks bring a multitude of advantages [9], including significantly faster data speeds, through enhanced mobile broadband (eMBB), ultra-reliable and low-latency communication (URLLC), and support for a large number of connected devices with massive machine-type communication (mMTC). The implementation of network slicing allows customizable network services, tailoring offerings to specific requirements, while technologies like beamforming and MIMO contribute to improved network coverage and efficiency. Deploying edge computing infrastructures on advanced 5G networks offers several key benefits. First, it significantly reduces latency by processing data closer to the source, ensuring quicker response times for applications. This is particularly crucial for real-time applications like augmented reality and autonomous vehicles. Second, edge computing enhances energy efficiency by minimizing data transmission between central clouds and end devices, contributing to a more sustainable network. Third, the proximity of computational resources at the edge ensures improved application performance, particularly for latency-sensitive tasks. The combination of these technologies, along with the growth of the Internet of Things (IoT), has given rise to the emergence of new computing paradigms, such as multi-access edge computing (MEC) [10], which is aimed at extending cloud computing capabilities to the edge of the radio access network, hence providing real-time, high-bandwidth, and low-latency access to radio network resources.

According to [11], the main objectives of edge computing in 5G environments are the following: (1) improving data management, in order to handle the large amounts of real-time delay-sensitive data generated by user equipments (UEs); (2) improving quality of service (QoS) to meet diverse QoS requirements, thereby improving the quality of experience (QoE) for applications that demand low latency and high bandwidth; (3) predicting network demand, which involves estimating the network resources required to cater to local proximity network (or user) demand, and subsequently providing optimal resource allocation; (4) managing location awareness, to enable geographically distributed edge servers to infer their own locations and track the location of UEs to support location-based services; and (5) improving resource management, in order to optimize network resource utilization for network performance enhancement in the edge cloud, acknowledging the challenges of catering to diverse applications, user requirements, and varying demands with limited resources compared to the central cloud.

Focusing on the last objective, efficient resource management and orchestration in 5G edge computing [12,13] are crucial as they not only contribute to latency reduction by strategically deploying computing resources, minimizing the time for data processing and enhancing application responsiveness, but also play a pivotal role in optimizing energy consumption through dynamic allocation based on demand, thereby reducing the environmental footprint. Furthermore, proper resource management is essential for performance optimization, preventing bottlenecks, ensuring balanced workload distribution, and maintaining consistent application performance, all of which collectively enhance the overall user experience in these advanced network environments. In this context, AI and ML techniques prove instrumental in addressing these challenges [14–18]. For example, through predictive analytics and forecasting, ML algorithms can anticipate future demands, enabling proactive resource allocation and strategic deployment for minimized data processing times and reduced latency. AI-driven dynamic allocation can optimize resource utilization and, consequently, energy consumption by adapting to real-time demand patterns. Additionally, ML models, employing clustering and anomaly detection, can ensure consistent application performance by preventing bottlenecks and optimizing workload distribution.

In this article, we specifically address the issue of intelligent resource orchestration in distributed edge computing infrastructures by integrating forecasting techniques for predicting resource utilization [19,20] and optimization techniques for optimal resource allocation [21,22]. Resource utilization forecasting leverages historical and real-time data to predict future resource demands accurately. By employing both traditional statistical techniques and AI-based ML techniques [23,24], these forecasts enable proactive resource allocation, mitigating the risk of resource shortages or over-provisioning. Furthermore, forecasting techniques facilitate capacity planning, allowing organizations to scale their resources dynamically based on anticipated demands. Optimization techniques also play a vital role in orchestrating resources across multiple edge locations. These techniques employ mathematical models such as ILP and heuristic algorithms [25,26] to determine the optimal mapping of virtual resources (VMs or containers) onto physical servers. By considering various factors including proximity, resource availability, and application requirements, these techniques ensure efficient utilization of resources and minimize resource fragmentation.

3. Related Work

The literature has extensively explored the application of artificial intelligence (AI) techniques, including evolutionary algorithms and machine learning (ML) algorithms, to address diverse prediction and optimization challenges in both cloud and edge environments. A recent study [27] offered a comprehensive review of machine-learning-based solutions for resource management in cloud computing. This review encompassed areas such as workload estimation, task scheduling, virtual machine (VM) consolidation, resource optimization, and energy efficiency techniques. Additionally, a recent book [28] compiled various research works that considered optimal resource allocation, energy efficiency, and predictive models in cloud computing. These works leveraged a range of ML techniques, including deep learning and neural networks. For edge computing platforms, surveys such as [29] have analyzed different machine and deep learning techniques for resource allocation in multi-access edge computing. Similarly, ref. [30] provided a review of task allocation and optimization techniques in edge computing, covering centralized, decentralized, hybrid, and machine learning algorithms.

If we focus on workload prediction, we see this is an essential technique in cloud computing environments, as it enables providers to effectively manage and allocate cloud resources, save on infrastructure costs, implement auto-scaling policies, and ensure compliance with service-level agreements (SLAs) with users. Workload prediction can be performed at application or infrastructure level. Application-level techniques [31,32] involve predicting a metric related to the application demand (e.g., requests per second or task arrival rate) to anticipate the optimal amount of resources needed to meet that

demand. On the other hand, infrastructure-level techniques [33,34] are based on predicting one or more resource usage metrics (such as CPU, memory, disk, or network) and making advanced decisions about the optimal amount and size of virtual or physical resources to provision to avoid overload or over-sizing situations. Another interesting piece of research in this area is [35], which presented a taxonomy of the various real-world metrics used to evaluate the performance of cloud, fog, and edge computing environments.

The most common workload prediction techniques used in cloud computing are based on time-series prediction methods [19,24], which involve collecting historical data of the target metric (e.g., the historical CPU usage of a VM or group of VMs) and forecasting future values of that metric for a certain time horizon. There are many different methods for modeling and predicting the time-series used in cloud computing, many of them based on classical techniques such as linear regression [36], Bayesian models [37], or ARIMA statistical methods [38]. The main advantage of these models is their flexibility and simplicity in representing different types of time-series, making them quick and easy to use. However, they have a significant limitation in their linear behavior, making them inadequate in some practical situations.

More recently, different methods have been proposed for time-series prediction based on machine learning and deep learning models [34,39,40] using artificial neural networks that have inherent non-linear modeling capabilities. One of the most common ML models applied to time-series is the long short-term memory (LSTM) neural network [23,41,42], which overcomes the problem of vanishing gradients associated with other neural networks. However, these methods also have several drawbacks. The first is that the training and prediction times of the neural network can be quite high (several minutes, or even hours), making them unfeasible in certain situations. The second problem is that the quality of predictions of neural-network-based methods depends heavily on correct selection of the model's hyperparameters, which can vary depending on the input data, meaning that adjusting these hyperparameters poses a serious challenge, even for expert analysts.

In absolute terms, it is not possible to claim that one prediction method is better than another, as their behavior will depend on the specific use case, the profile of the input data, the correct tuning of each model's hyperparameters, and the use or non-use of covariables that may correlate with the variable being predicted. In this context, research in this field involves exploring and comparing different prediction methods for each case, improving existing methods, and combining different techniques by proposing new hybrid or adaptive methods that allow for the most accurate forecasts possible. Likewise, applying these prediction techniques to other emerging environments such as highly geo-distributed edge/cloud environments, IoT environments, and server-less environments also represents a significant challenge.

On the other hand, the optimal allocation of resources in cloud computing is an important problem that must be addressed to ensure efficient use of resources and to meet SLAs agreed with users. The goal is to allocate resources (e.g., VMs to physical hosts) in a way that maximizes infrastructure utilization, subject to certain performance or application response time requirements.

There are different optimization techniques used to solve this problem. One of the most commonly used techniques is linear programming, which finds the optimal solution to a linear function subject to a set of linear constraints. This technique is very useful for problems involving a large number of variables and constraints. For example, ref. [43] was a pioneering work in cloud brokering that used ILP to optimize the cost of a virtual infrastructure deployed on a set of cloud providers. Subsequently, this research was expanded upon in [44], which addressed dynamic cloud scenarios and incorporated diverse optimization criteria such as cost or performance. The authors in [45,46] presented MALLOOVIA, a multi-application load-level-based optimization for virtual machine allocation. MALLOOVIA formulates an optimization problem based on ILP and takes the levels of performance to must be reached by a set of applications as input, and generates a VM allocation to support the performance requirements of all applications as output.The authors

in [47,48] provided an approach for supporting the deployment of microservices in multi-cloud environments, focusing on the quality of monitoring and adopting a multi-objective mixed integer linear optimization problem, in order to find the optimal deployment satisfying all the constraints and maximizing the quality of monitored data, while minimizing costs. Some recent works have also used ILP optimization for resource allocation in edge computing infrastructures. For example, ref. [49] formulated an ILP model to minimize the access delay of mobile users' requests, in order to improve the efficiency of edge server and service entity placement. The authors in [50] focused on the joint optimization problem of edge sever placement and virtual machine placement. The optimization models proposed, which take into account the network load and the edge server load, are based on ILP and mixed integer programming.

Other optimization approaches that we can find in the literature are heuristic techniques based on bio-inspired algorithms [26,51–54], such as genetic algorithms, particle swarm optimization), and ant colony optimization, among others. These algorithms allow users to find suboptimal solutions for optimization problems that are too complex to solve exactly. Genetic algorithms, for example, are inspired by natural selection and biological evolution to find optimal solutions. Reinforcement learning is another technique that has been successfully used in resource management and allocation in cloud computing [55–57]. This technique is based on a learning model in which an agent interacts with an environment and receives a reward or punishment based on its actions. Through experience, the agent learns to make the optimal decisions that maximize the expected reward.

Each technique has its advantages and disadvantages, and the choice of the most appropriate technique will depend on the specific characteristics of the problem to be solved. In general, linear programming is more suitable for well-structured problems with a limited number of variables and constraints. Metaheuristic algorithms are more suitable for more complex problems with a large number of variables and constraints. Reinforcement learning, on the other hand, is more suitable for problems involving uncertain dynamic environments. Sometimes it will also be necessary to address multi-objective problems where it is necessary to optimize more than one objective function subject to certain constraints. Research in this field involves exploring, analyzing, and comparing different optimization techniques adapted to each problem and use case, including the treatment of both single and multi-objective problems, and the possibility of combining different techniques through the proposal of new hybrid optimization techniques. Furthermore, the application of these optimization techniques to other emerging environments such as highly geo-distributed edge/cloud environments, IoT environments, and serverless environments also represents an important challenge.

In the above research review, we found many studies focusing on predicting loads and optimizing resources in centralized clouds or simple edge infrastructures, using various machine learning techniques. However, there has been limited exploration or implementation of these techniques in highly distributed 5G edge environments within actual cloud/edge infrastructure managers. The Smart 5G Edge-Cloud Management Architecture proposed in this study aims to address this gap. It intends to analyze, enhance, and expand AI-based prediction and optimization methods for large-scale 5G edge infrastructures. Additionally, it plans to integrate these methods with existing edge management platforms like OpenNebula. This integration will enable automated and optimized provisioning and deployment of geo-distributed 5G edge infrastructures.

4. Proposed Architecture

Below, we present a Smart 5G Edge-Cloud Management Architecture built upon the foundation of OpenNebula for the orchestration and management of cloud-edge infrastructures.

4.1. Smart 5G Edge-Cloud Management Architecture

The design of the new Smart 5G Edge-Cloud Management Architecture is shown in Figure 2. In this context, it is important to note the difference between management

(implemented by OpenNebula) and orchestration (implemented by the new components of the ONEedge5G project). While management just involves the capacity for managing the lifecycle of resources (physical or virtual) and performing basic actions regarding these resources, orchestration involves intelligent and automated provision, configuration, and coordination of resources, keeping track of the state of resources and reacting to events, and making optimal decisions about, for example, scheduling, placement, migration, or consolidation, based on different optimization criteria. The main components of this architecture are the following:

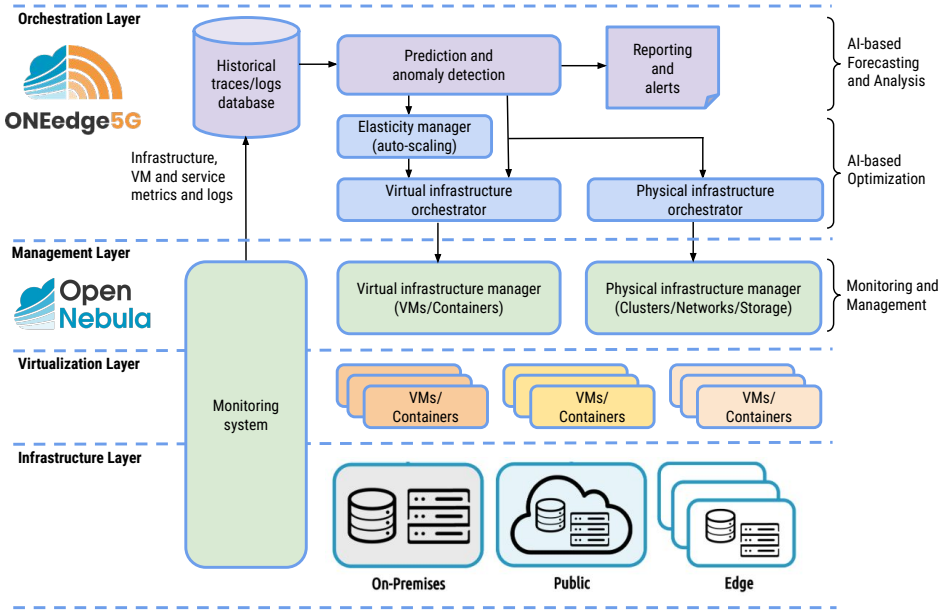


Figure 2. Smart 5G Edge-Cloud Management Architecture.

- The monitoring system is responsible for collecting different infrastructure-level metrics (e.g., CPU, memory, network I/O, or energy consumption) and logs from virtual and physical infrastructure and/or service-level metrics (e.g., response time or requests/s) and logs from deployed applications.
- The historical trace/logs database stores the historical values of these metrics and logs.
- The prediction and anomaly-detection system implements different AI-based algorithms in order to predict future infrastructure load (e.g., CPU or memory usage for a virtual or physical resource), to forecast application workloads (e.g., request/s for an application), or to detect or predict anomalies (e.g., system failures, service interruptions, or performance slow down).
- The reporting and alerting system is used to configure different metric-based alerting policies and reporting filters, in order to obtain valuable information, warnings, and recommendations from the historical traces/logs and from the information provided by the prediction and anomaly-detection system.
- The elasticity manager implements different horizontal or vertical auto-scaling mechanisms to provide service elasticity, including AI-based proactive autoscaling based on application workload predictions.
- The virtual infrastructure orchestrator is responsible for making automated decisions about the best possible allocation of virtual resources (VMs/containers) to physical servers, which can be located in different cloud regions and edge zones. It can implement different AI-based virtual resource allocation and migration strategies, based on predictions, and using different optimization criteria such as cost, energy consumption, and application performance.
- The physical infrastructure orchestrator is responsible for making automated decisions about deploying or shutting down physical (bare-metal) servers or clusters in different

cloud and edge locations. It can implement different AI-based server placement and consolidation strategies based on predictions, to decide when and where a new cluster must be deployed and its optimal size, including dynamic re-dimensioning to adapt the cluster size to actual or expected demand.

- The physical and virtual infrastructure managers provide the interfaces, drivers, and mechanisms necessary to manage the entire lifecycle of virtual and physical resources, as well as performing different actions (e.g., deploy, migrate, suspend, resume, or shutdown) for these resources, according to user commands or orchestrator decisions.

It is important to note that ONEedge5G is an ongoing experimental project, and therefore many of its components are still under development. This research work centers around two main elements within its architecture: a virtual resource CPU usage prediction system integrated within the prediction and anomaly detection module, and a virtual-to-physical resource mapping system that utilizes optimization techniques, forming an integral part of the virtual infrastructure orchestrator. Additionally, a historical trace database system has been implemented leveraging Prometheus [58], which plays a critical role in providing historical traces of virtual resource CPU usage to support the prediction system.

4.2. Intelligent Orchestration of Virtual Resources on 5G Edge Infrastructures

The problem addressed in this work focuses on the intelligent orchestration of virtual resources (VMs or containers) in 5G edge infrastructures. As depicted in Figure 3, the scenario under consideration involves multiple bare-metal clusters or servers located in different 5G edge locations. These edge clusters are managed by a centralized instance of the OpenNebula cloud manager. From the perspective of OpenNebula, these clusters form a uniform physical resource pool, where virtual resources can be dynamically deployed on an on-demand basis.

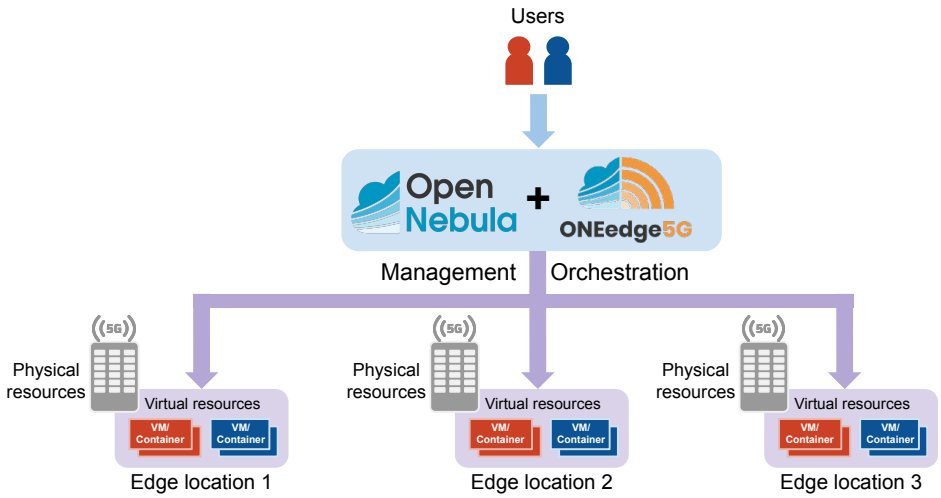


Figure 3. Distributed edge infrastructure.

Intelligent orchestration of this infrastructure entails finding an optimal allocation of virtual resources to the available physical servers within the edge infrastructure, while satisfying specific criteria. Various optimization criteria can be considered, such as minimizing the number of servers in use by consolidating virtual resources onto the fewest possible servers to reduce energy consumption, balancing the load of different servers to prevent overloading and CPU contention, or optimizing latency by selecting the closest edge server based on specific proximity criteria. In addition, the orchestration system can also deal with different constraints, including a limit on the maximum number of hosts used, or a limit on the number of violations of the proximity criteria.

To address these challenges, the ONEedge5G module incorporates various prediction and optimization mechanisms. Prediction mechanisms leverage historical data to forecast

the load of different virtual resources for the upcoming allocation period. This load can be quantified in terms of virtual resource consumption, such as CPU, memory, and/or bandwidth usage. Meanwhile, optimization techniques utilize the aforementioned predictions to determine the optimal allocation of virtual resources to physical servers based on specific optimization criteria (e.g., minimizing the number of servers, achieving server load balancing, or optimizing latency) and constraints.

5. Virtual Resource Load Forecasting

As mentioned in Section 3, workload prediction in cloud computing can be conducted at the application level or the infrastructure level. In this study, we focus on utilizing infrastructure-level techniques, which rely on predicting one or more resource usage metrics, such as CPU, memory, disk, or network utilization. Specifically, we employ and compare different time-series forecasting methods to predict virtual resource CPU usage.

To accomplish this, it was imperative to gather historical data on CPU usage from various virtual resources over a specified time period. For each virtual resource trace, the historical data were divided into two datasets: one for training and another for testing. The training data were then employed to train various time-series forecasting models, while the test data were used to evaluate the accuracy of the predictions using appropriate error metrics.

5.1. Time Series Forecasting Models

In this study, we implemented a range of forecasting models using the Darts Python library [59]. This library offers a wide array of models, including both classical approaches like ARIMA and sophisticated deep neural networks. Leveraging the capabilities of the Darts library, we implemented the following forecasting models:

- Naive seasonal. This is a simple baseline model for univariate time-series forecasting [60] that always predicts the value of K time steps ago. When $K = 1$, this model predicts the last value of the training set. When $K > 1$, it repeats the last K values of the training set.
- ARIMA (auto-regressive integrated moving average). The ARIMA model [61] is a form of regression analysis that assesses the relationship between a dependent variable and other changing variables. Its objective is to predict future values of a time-series by examining differences between values in the series, rather than the actual values themselves. In this study, we utilized the Auto-ARIMA model [62] offered by the Darts library, which automatically determines the optimal parameters for an ARIMA model.
- Bayesian regression. Bayesian regression [63] is a type of conditional modeling in which the mean of one variable is described as a linear combination of other variables. One of the most useful types of Bayesian regression is Bayesian ridge regression, which estimates a probabilistic model of the regression problem. The Darts model used in our experiments is based on the Scikit-Learn implementation of Bayesian ridge regression [64].
- Facebook (FB) Prophet. FB Prophet [65] is a forecasting package developed by Facebook's data science research team. Its objective is to provide users with a powerful and user-friendly tool for forecasting business results, without requiring expertise in time-series analysis. The underlying algorithm is a generalized additive regression model consisting of four main components: a piecewise linear or logistic growth curve trend, a yearly seasonal component modeled using Fourier series, a weekly seasonal component using dummy variables, and user-provided important holidays.
- Recurrent neural networks (RNN) based on LSTM (long short-term memory). LSTM [66] are a specialized type of RNN used in deep learning. They address the vanishing gradient problem of traditional RNNs by incorporating memory cells and gating mechanisms. These mechanisms enable LSTM networks to selectively remember or forget information over time, making them well-suited for tasks requiring the understanding of long-range dependencies in sequential data.

- Neural basis expansion analysis time-series forecasting (N-BEATS). N-BEATS is a neural network that was architecture initially described in [67]. The primary objective of N-BEATS is to address the univariate time-series point forecasting problem using deep learning techniques. In the N-BEATS implementation provided by Darts, the original architecture is adapted to handle multivariate time-series by flattening the source data into a one-dimensional series.

By exploring and comparing these forecasting models, we aimed to identify the most accurate and reliable approach for predicting virtual resource CPU usage in our infrastructure.

5.2. Forecast Accuracy Measures

Different error metrics can be used to evaluate the accuracy of forecasting models. Some of the most common error measures are the following:

- Mean absolute error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{1}$$

- Mean squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

- Root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

where y_i are the actual observations (e.g., CPU usage), $\hat{y}_i$ are the predictions made by the forecasting model, and n is the number of observations.

In this work, we used the RMSE metric to compare the different forecasting techniques. RMSE represents the standard deviation of the residuals (prediction errors), and it is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data are around the line of best fit.

6. Edge Resource Optimization

6.1. Problem Statement

The scenario considered in this work consists of a distributed 5G edge infrastructure composed of a set of physical servers, one per edge location, each server with a specific computing capacity (measured in the number of available cores) and a certain memory capacity (measured in MB of available RAM). Our objective is to deploy a set of virtual resources (VMs or containers) in this infrastructure, where each virtual resource has its own computing requirements (number of assigned cores), memory requirements (amount of assigned RAM in MB), and an estimated (forecasted) percentage of CPU usage for each time slot. Furthermore, to model latency in our system, we assume that each virtual resource is designated with a preferred edge location based on proximity criteria that consider the distance between the edge location and the end users it serves. If the virtual resource is not assigned to the preferred edge location, this is considered a latency violation. In our proposed model, we do not measure the exact time penalty caused by these violations but instead focus on counting the total number of virtual resources affected by latency violations.

The problem at hand involves finding the optimal mapping of virtual resources (VMs or containers) to edge servers for a given time period (in our case, the mapping is performed once a day). As outlined in Section 2, deploying edge computing infrastructures on 5G

networks presents new challenges related to reducing latency, minimizing energy consumption, and distributing workloads across geo-distributed edge nodes in various 5G locations (such as 5G base stations). This study tackles these challenges by considering different objective functions, such as minimizing the total number of cores utilized in the physical infrastructure in order to reduce energy consumption, optimizing load balancing among the different servers to improve workload distribution, and minimizing the number of virtual resources affected by latency violations to reduce the observed latency for end users and devices. Furthermore, we must account for some mapping constraints. For instance, it is essential to ensure that the aggregate CPU usage of all virtual resources assigned to a physical server should never exceed the server's capacity (number of available cores) in any given time slot. Additionally, there may exist optional constraints, such as imposing an upper limit on the utilization of resources (e.g., cores) within the edge infrastructure or establishing a limit on the number of latency violations.

6.2. Problem Formulation

The proposed solution for the previously stated problem is based on an ILP formulation. As an overview, the inputs for the model are the number of physical servers available at the 5G edge infrastructure, the capacity of these servers in terms of CPUs (number of cores available), the set of virtual resources (VMs or containers) to be deployed, the capacity allocated to these virtual resources in terms of CPUs (number of cores assigned to the virtual resource), and the estimated percentage of CPU usage per time interval of these virtual resources. These estimations are obtained using the forecasting methods detailed in Section 5.

The output is the mapping of virtual resources to physical servers; that is, which virtual resources should be deployed to each physical server, and which physical servers are used for this deployment. Three different integer linear programming problems are proposed, with three different objective functions.

The following subsections provide a detailed formulation of the problems.

6.2.1. Inputs

Let $\{v_1, v_2, \ldots, v_n\}$ be the set of virtual resources to be mapped to the edge infrastructure, and let $\{s_1, s_2, \ldots, s_m\}$ be the set of physical servers available at this infrastructure, assuming one server per edge location. Virtual resources and servers are characterized by the following parameters (inputs of the model):

- V_i^c is the number of cores assigned to virtual resource v_i, $\forall i = 1, \ldots, n$.
- $V_{i,t}^u$ is the estimated (predicted) CPU usage (%) of v_i at time interval t, $\forall i = 1, \ldots, n$, $\forall t = 0, \ldots, 23$.
- S_j^c is the number of cores available on server s_j, $\forall j = 1, \ldots, m$.
- $Pref_{ij}$ is the preferred edge location (or preferred server) designated for each virtual resource, as defined by the following binary input parameter:

$$Pref_{ij} = \begin{cases} 1 & \text{if } s_j \text{ is the preferred server designated for virtual resource } v_i \\ 0 & \text{Otherwise} \end{cases}$$

6.2.2. Outputs

The output of the model is a mapping of virtual resources to edge servers for the next full day, which can be defined using the two following decision variables:

- $$X_{ij} = \begin{cases} 1 & \text{if, for the next full day, the virtual resource } v_i \text{ is allocated to edge server } s_j \\ 0 & \text{Otherwise} \end{cases}$$
- $$Y_j = \begin{cases} 1 & \text{if edge server } s_j \text{ is used for the next day's allocation} \\ 0 & \text{Otherwise} \end{cases}$$

6.2.3. Objective Functions

The goal is to find the optimal mapping of virtual resources to edge servers for the next day ($X_{ij}, Y_j \quad \forall i = 1, \ldots, n, \quad \forall j = 1, \ldots, m$) that meets particular objective functions. We explore three different objective functions, as follows:

1. To minimize the total number of cores used in the edge infrastructure, which can be formulated as follows:

$$\text{Minimize} \quad Total\_cores \tag{4}$$

where

$$Total\_cores = \sum_{j=1}^{m} S_j^c \cdot Y_j \tag{5}$$

In the particular scenario where all servers are homogeneous and have an equal number of cores, i.e., $S_j^c = S^c, \forall j = 1, \ldots, m$, this objective function is equivalent to minimizing the number of servers in use, which can be formulated as follows:

$$\text{Minimize} \quad Total\_servers \tag{6}$$

where

$$Total\_servers = \frac{Total\_cores}{S^c} = \sum_{j=1}^{m} Y_j \quad \text{if} \quad S_j^c = S^c, \forall j = 1, \ldots, m \tag{7}$$

2. To balance the average load of all the cores used in the edge infrastructure. To formulate this objective function, we first define the average (daily) load per core of an edge server, S_j^{load}, as follows:

$$S_j^{load} = \frac{\sum_{i=1}^{n} \sum_{t=0}^{23} X_{ij} \cdot V_i^c \cdot V_{i,t}^u}{24 \cdot S_j^c}, \quad \forall j = 1, \ldots, m \tag{8}$$

Then, the load balancing objective function can be expressed as follows:

$$\text{Minimize} \quad Max\_load \tag{9}$$

where

$$Max\_load = max\{S_j^{load}, \quad \forall j = 1, \ldots, m\} \tag{10}$$

It is worth noting that the maximum function (max) is not linear. Therefore, in order to incorporate it into the ILP formulation, it is necessary to transform it into one or more linear expressions. This can be achieved by re-formulating the Max_load function as follows:

$$Max\_load >= S_j^{load} \quad \forall j = 1, \ldots, m \tag{11}$$

3. To minimize the number of virtual resources affected by latency violations, regarding the preferred location of each virtual resource, $Pref_{ij}$, and the actual location selected for this virtual resource, X_{ij}, which can be formulated as follows:

$$\text{Minimize} \quad Total\_latency\_violations \tag{12}$$

where

$$Total\_latency\_violations = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} |Pref_{ij} - X_{ij}|}{2} \tag{13}$$

As in the previous case, the absolute value function is also non-linear. Therefore, it is necessary to transform it into a linear expression using the following auxiliary variable, D_{ij}:

$$\begin{aligned} D_{ij} >= Pref_{ij} - X_{ij} \quad \forall i = 1, \ldots, n \quad \forall j = 1, \ldots, m \\ D_{ij} >= -(Pref_{ij} - X_{ij}) \quad \forall i = 1, \ldots, n \quad \forall j = 1, \ldots, m \\ D_{ij} >= 0 \quad \forall i = 1, \ldots, n \quad \forall j = 1, \ldots, m \end{aligned} \tag{14}$$

Then, the *Total_latency_violations* function can be re-formulated as follows:

$$Total\_latency\_violations = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} D_{ij}}{2} \tag{15}$$

6.2.4. Constraints

Each of these objective functions is subject to a set of strict constraints and some optional constraints.

(a) Strict constraints

- Each virtual resource must be allocated to exactly one edge server:

$$\sum_{j=1}^{m} X_{ij} = 1 \quad \forall i = 1, \ldots, n \tag{16}$$

- The estimated CPU usage of all the virtual resources allocated to a server cannot exceed, within a certain threshold α (with $\alpha \in [0, 1]$), the maximum capacity (number of cores) available on that server:

$$\sum_{i=1}^{n} X_{ij} \cdot V_i^c \cdot V_{i,t}^u \leq \alpha \cdot S_j^c \cdot Y_j \quad \forall j = 1, \ldots, m \quad \forall t = 0, \ldots, 23 \tag{17}$$

The analyst has the flexibility to select the value of the α threshold, which serves to mitigate the discrepancy between the estimated CPU usage and the actual CPU usage of the virtual resources and prevent overloading of the physical servers.

(b) Optional constraints

- We can set a limit on the maximum number of cores used in the edge infrastructure:

$$Total\_cores = \sum_{j=1}^{m} S_j^c \cdot Y_j \leq core\_limit \tag{18}$$

- We can set a limit on the maximum number of latency violations allowed:

$$\begin{aligned} Total\_latency\_violations = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} |Pref_{ij} - X_{ij}|}{2} \\ \leq latency\_violation\_limit \end{aligned} \tag{19}$$

It is necessary to note that, once again, the *Total_latency_violations* function must be reformulated, as shown in Equations (14) and (15), to transform the absolute value function into a linear expression.

It is important to note that this study primarily focuses on CPU usage for virtual resources. However, it is worth mentioning that similar constraints and considerations can be applied to other system resources, such as memory, bandwidth, and disk consumption.

7. Results

This section showcases the prediction and allocation results achieved with the various forecasting models, employing different optimization functions and constraints. All experiments were performed on an on-premises server equipped with a 2.3 GHz 8-core Intel Core i9 processor and 32 GB of RAM. This configuration represents a standard setup for executing a front-end instance of a cloud orchestrator.

7.1. Traces

The workload traces needed to feed the different forecasting and optimization algorithms can be obtained from the different sources, including public dataset repositories (e.g., Google Workload Cluster Traces [68], the Azure Public Datasets [69], the Grid Workload Archive [70], or the Alibaba Cluster Trace Program [71], among others), production traces obtained from some real companies, or synthetic traces generated by certain workload generator applications (e.g., Predator [72], Locust [73], etc.). Using public datasets has several advantages compared to the alternatives. Public datasets are typically curated, cleaned, and anonymized to ensure privacy and security; furthermore, they facilitate fair comparisons and reproducibility of experiments. On the other hand, accessing production traces from real companies can be challenging due to confidentiality concerns, legal restrictions, and the need for collaboration or data-sharing agreements. Synthetic traces generated by workload generators may not accurately represent real-world workload characteristics and patterns. Since there are no publicly available workloads with sufficient representation of distributed edge infrastructures, as noted in previous studies [74,75], in this work, we opted to use two well-known cloud VM traces from public datasets. Specifically, we utilized data from the Azure Public Datasets [76] and the GWA-T-12 Bitbrains traces sourced from the Grid Workload Archive [77].

The VM traces available in the Azure Public Datasets encompass a representative subset of the first-party Azure VM workload within a specific geographical region. These first-party workloads consist of both internal VMs utilized for research/development and infrastructure management, as well as first-party services provided to third-party customers, such as for communication, gaming, and data management. The time-series data derived from the Azure VM trace V1 span a duration of 30 days, commencing from 16 November 2016, and concluding on 16 February 2017. For each VM, this trace records the capacity provisioned for this VM in terms of its cores, memory, and disk allocations. Additionally, it collects CPU usage data reported every five minutes.

The GWA-T-12 Bitbrains dataset includes performance metrics obtained from approximately one thousand VMs within a distributed data center operated by Bitbrains. Bitbrains specializes in managed hosting and business computation services for enterprises, catering to prominent customers such as major banks (ING), credit card operators (ICS), insurers (Aegon), and more. The time-series data recorded in the Bitbrains traces were collected at 5 min intervals, spanning a duration of 30 days from 12 August 2013, to 11 September 2013. For each VM, this trace records the capacity provisioned for this VM in terms of number of CPU cores, CPU MHz, and memory allocations. Additionally, it collects data about CPU usage, memory usage, disk read/write throughput, and network input/output throughput, reported every five minutes.

7.2. Forecasting Results

We utilized the aforementioned traces from Bitbrains and Azure to forecast CPU usage using the various time-series forecasting models described in Section 5. These models were implemented in Python using the Darts library. In both datasets, VM traces were collected every five minutes over a 30-day period. We grouped these traces into hourly time-series, by computing the maximum CPU usage within each hour. Our objective was to predict CPU usage for the next 24 h period based on historical data. Before using them, both sets were normalized to the interval $[0, 1]$. The predicted CPU usage for the last 24 h period was then used as input for the optimization models, allowing us to achieve an optimal

allocation of VMs to physical servers. The results of these allocations are presented in Section 7.3.

7.2.1. Bitbrains Trace Forecasting

The Bitbrains dataset contains traces for 1250 VMs. We classified these VMs into four groups:

- Incomplete traces (252): Only VM traces that include data for all the time intervals of the complete 30-day period were selected. Incomplete traces were excluded.
- Low CPU usage traces (875): VM traces with low CPU usage (averaging under 10%) were discarded when applying the forecasting models. For these traces, we assumed the expected CPU usage for the next 24 h period could be computed simply as the average CPU usage from the historical dataset.
- Unpredictable traces (81): Many VM traces exhibit unpredictable CPU usage patterns, including variable length periods of 0% CPU usage and variable length periods of nearly 100% CPU usage. For these unpredictable traces, we assumed that the expected CPU usage for the next 24 h period was always 100%. This prevented potential under-provisioning situations that could lead to degradation of CPU performance.
- Predictable traces (42): These are the VM traces that did not fall into any of the groups above. There are a total of 42 predictable traces in the Bitbrains dataset. The different forecasting models were exclusively applied to this trace group.

We ran and compared six different forecasting models:

- Naïve seasonal model, with K = 24 h
- Bayesian regression model, using a Ridge-type regressor to predict the target time-series from lagged values, using a lag period of 24 h
- Facebook Prophet model, without specific parameters (default values were used)
- Auto-ARIMA model, without specific parameters (default values were used)
- RNN model based on LSTM, with the following parameters:
  - input_chunk_length = 24
  - output_chunk_length = 24
  - hidden_dim = 10
  - n_rnn_layers = 1
  - batch_size = 32
  - epochs = 50 For the remaining RNN-LSTM parameters, the default values were used.
- N-BEATS model, with the following parameters:
  - input_chunk_length = 24
  - output_chunk_length = 24
  - epochs = 50 For the remaining N-BEATS parameters, the default values were used.

Due to space limitations, it is impossible to display the charts comparing the actual and predicted CPU usage values for the 24 h testing period across all combinations of VMs and prediction models. Therefore, we only present a sample of four selected VMs, which are displayed in Figure 4.

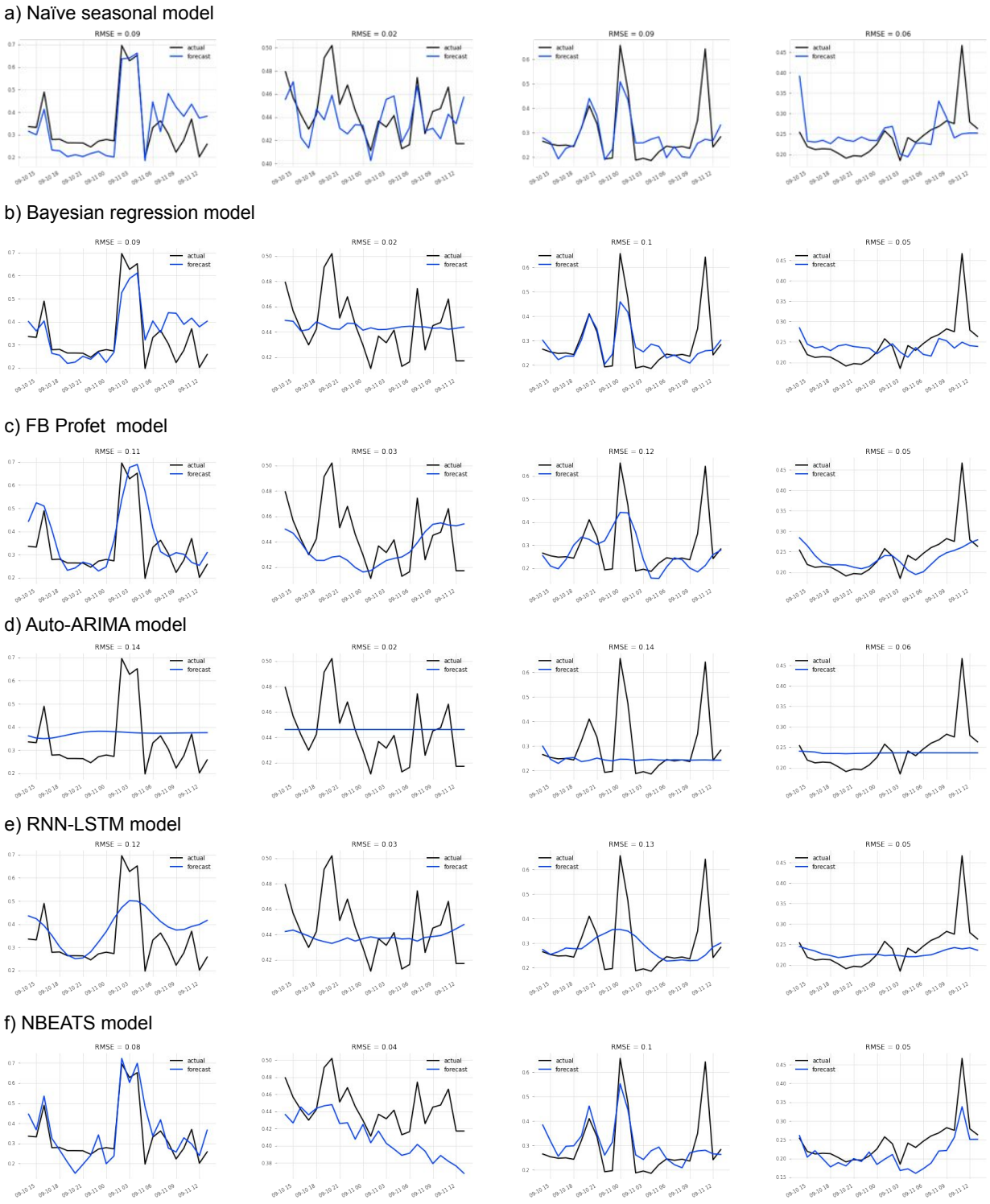


Figure 4. Forecasting results for a subset of Bitbrains VM traces (% CPU usage).

The graphs presented in Figure 5 illustrate the prediction accuracy (RMSE) achieved by the different models when forecasting CPU usage for the 42 VMs considered in this dataset. It is evident that certain forecasting methods outperformed others depending on the trace. On average, the Bayesian regression and neural network-based models (RNN-LSTM and N-BEATS) exhibited lower RMSE values compared to the other models.

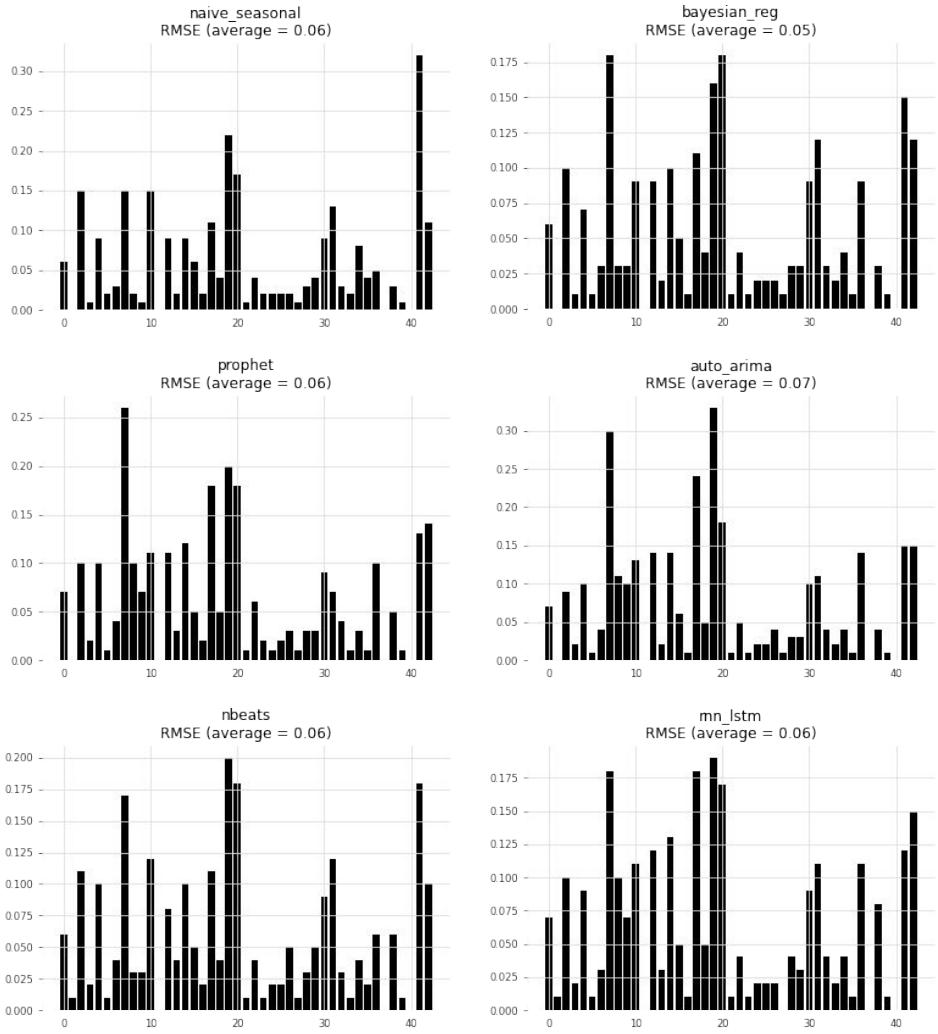


Figure 5. Prediction accuracy (RMSE) of various time-series forecasting models applied to Bitbrains VM traces.

This observation is further supported by Table 1, which summarizes the average RMSE and execution time (including fit and predict functions for all selected VMs) for each prediction model. Notably, Bayesian regression demonstrated the highest level of accuracy. In terms of execution times, neural network models (RNN-LSTM and N-BEATS) required more computational time for model fitting, but they did not yield an improved accuracy compared to Bayesian regression.

Table 1. Summary of prediction accuracy (average RMSE) for Bitbrains traces.

Model	RMSE (Avg.)	Exec. Time (Fit + Predict)
Naive seasonal	0.0613	33 s
Bayesian Regression	0.0536	36 s
FB Prophet	0.0627	75 s
Auto-ARIMA	0.0731	7 min
RNN-LSTM	0.0619	23 min
N-BEATS	0.0580	1 h
Adaptive (oracle)	0.0455	–
Adaptive (realistic)	0.0584	–

In addition to the six tested forecasting models, we proposed the utilization of an adaptive model that combines all of the analyzed models, as shown in Table 1. The un-

derlying concept of this adaptive approach is to execute all six forecasting models once a day, and to select for each specific VM trace the model that performs best. The challenge lies in the fact that it is impossible to know in advance which model will yield the best results for the upcoming day. The oracle adaptive model assumes that we have advance knowledge of the best-performing model for each individual VM trace on the following day. We would then select that particular model to predict the hourly CPU usage for that VM. However, in practice, this oracle selection is unfeasible. Therefore, we proposed a second realistic adaptive model, which compares the forecasts obtained from the different models on the day before and selects the model that performed best for making predictions for the next day. It is evident that the oracle adaptive model outperformed each of the individual models in terms of prediction accuracy. However, on average, the realistic adaptive model did not improve on the predictions obtained by the Bayesian regression model.

7.2.2. Azure Trace Forecasting

The Azure dataset considered in this work contained traces for 28,858 VMs. We categorized these VMs into three groups:

- Incomplete traces (5): Only traces that included data for all the time intervals of the complete 30-day period were selected. Incomplete traces were excluded.
- Low CPU usage traces (25,218): Traces with low CPU usage (averaging below 10%) were excluded from the application of forecasting models. For these traces, we assumed the expected CPU usage for the next 24 h period could be computed as the average CPU usage from the historical dataset.
- Predictable traces (3635): These traces did not fall into either of the two previous groups. In the Azure dataset, there were a total of 3635 predictable traces. The different forecasting models were exclusively applied to this group of traces.

We ran and compared five forecasting models (Naïve seasonal, Bayesian regression, FB prophet, Auto-ARIMA, and RNN-LSTM) with the same parameters as in the Bitbrains case. The N-BEATS model was not considered for forecasting Azure traces due to its extended execution time for all 3635 VM traces, exceeding three days.

Prediction results for this dataset are summarized in Figure 6, which shows the prediction accuracy (RMSE) achieved by the different models when forecasting CPU usage for the VM traces considered in this dataset. Table 2 displays the average RMSE and execution time (including fit and predict functions for all selected VMs) for each prediction model. As observed, the Bayesian regression model once again demonstrated the highest average accuracy. Regarding execution times, models based on neural networks (RNN-LSTM) required significantly more computational time for model fitting but they did not enhance the accuracy compared to the Bayesian regression. In addition to the five tested forecasting models, we also implemented the two previously explained adaptive methods (oracle and realistic). In this scenario, it can be observed that the realistic adaptive model marginally improved on the average RMSE of the predictions obtained by the Bayesian regression model.

Based on these results, we can conclude that the Bayesian regression model was sufficiently effective in generating accurate CPU usage forecasts, while also requiring a reasonably low execution time. Therefore, the outcomes generated by this model were selected as inputs for the optimization algorithms.

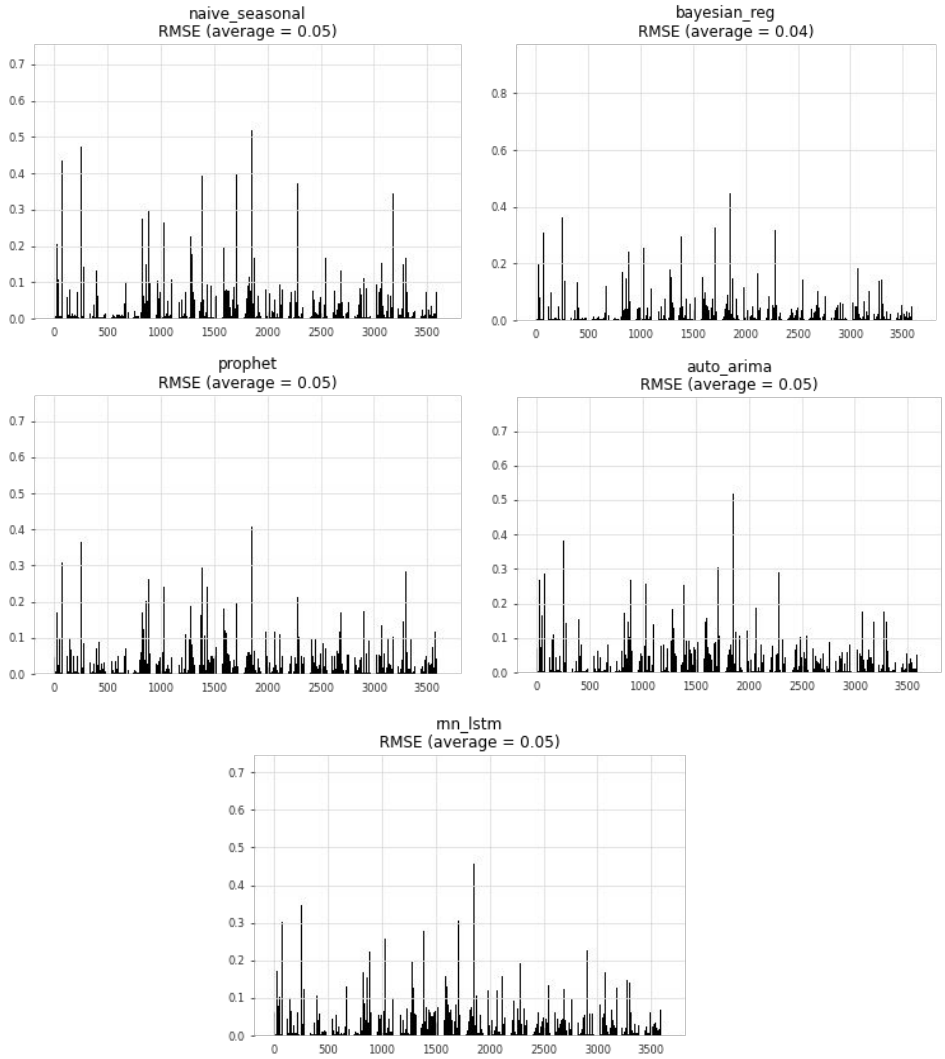


Figure 6. Prediction accuracy (RMSE) of various time-series forecasting models applied to Azure VM traces.

Table 2. Summary of prediction accuracy (average RMSE) for Azure traces.

Model	RMSE (Avg.)	Exec. Time (Fit + Predict)
Naive seasonal (K = 24)	0.0498	17 min.
Bayesian Regression	0.0419	21 min.
FB Prophet	0.0504	40 min.
Auto-ARIMA	0.0508	5.9 h
RNN-LSTM	0.0491	12.7 h
Adaptive (oracle)	0.0314	–
Adaptive (realistic)	0.0414	–

7.3. Resource Optimization Results

As stated previously, the optimization problem addressed in this work consists of finding the optimal mapping of virtual resources (VMs or containers) to edge servers using different optimization criteria and constraints. The solution proposed for this problem ise based on the ILP formulation, as shown in Section 6. These kinds of problems can be effectively solved using standard solvers for linear programming, such as COIN CBC [78], CPLEX [79], and SCIP [80]. In particular, our implementation was built upon the OR Tools library for Python [81], leveraging the SCIP solver integrated within the Pywraplp module [82].

Our experimental test bed consisted of 500 VMs randomly chosen from the Azure dataset, which had to be distributed across 12 different 5G edge locations. Each edge location housed a physical server equipped with 64 cores and 192 GB of RAM. The parameters defining each VM included the assigned number of cores, the predicted CPU usage percentage for the next 24 h period (based on hourly predictions derived from the Bayesian regression model), and the preferred edge location. These preferred edge locations were also chosen randomly, following the distribution illustrated in Figure 7.

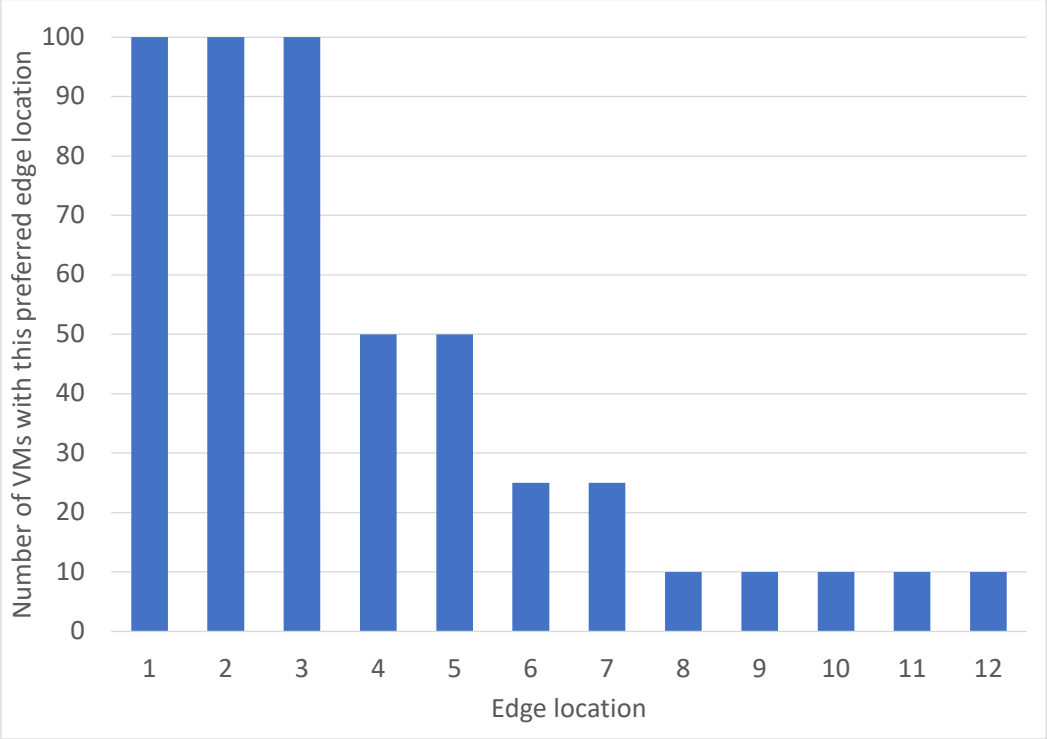


Figure 7. Distribution of VMs according to the preferred edge location.

7.3.1. Comparison of Different Objective Functions without Optional Constraints

The initial experiments involved a comparison of the optimal allocation obtained with the three different objective functions: minimization of the number of servers in use, Equation (6); minimization of the total number of latency violations, Equation (12); and optimization of load balance among servers, Equation (9). In these experiments, these objective functions were used without any optional constraints (only strict constraints were considered, as explained in Section 6.2.4). It is important to note that since all edge servers have similar hardware configurations, the objective function that minimizes the total number of servers in use is equivalent to minimizing the total number of cores, Equation (7). We also compared these objective functions with the solution obtained without any optimization (i.e., allocating each VM to its preferred edge location). The results of these experiments are shown in Table 3 and Figure 8.

Table 3 provides a summary of the allocation achieved for each objective function (cases 1, 2, and 3) and the solution obtained without optimization (case 4). In case 4, each VM was simply allocated to its preferred edge location. The table presents the total number of servers in use, the total number of latency violations, and the execution time of the ILP solver for each solution. On the other hand, Figure 8 depicts the estimated average daily load per core of each server based on the VM CPU usage predictions used in the optimization model, Equation (8), as well as the actual average daily load of each server (similar to Equation (8), but using real VM CPU usage values instead of predicted values). Additionally, the figure illustrates the maximum hourly actual load, representing the worst-case scenario (maximum peak load) across all servers and time intervals.

Table 3. Allocation results for different objective functions (without optional constraints).

Case	Objective Function	Total Servers	Total Lat. Violations	Solver Exec. Time
1	Number of servers	9	389	0.9 s
2	Latency violations	12	29	1.0 s
3	Load balance	12	432	1.4 s
4	None	12	0	-

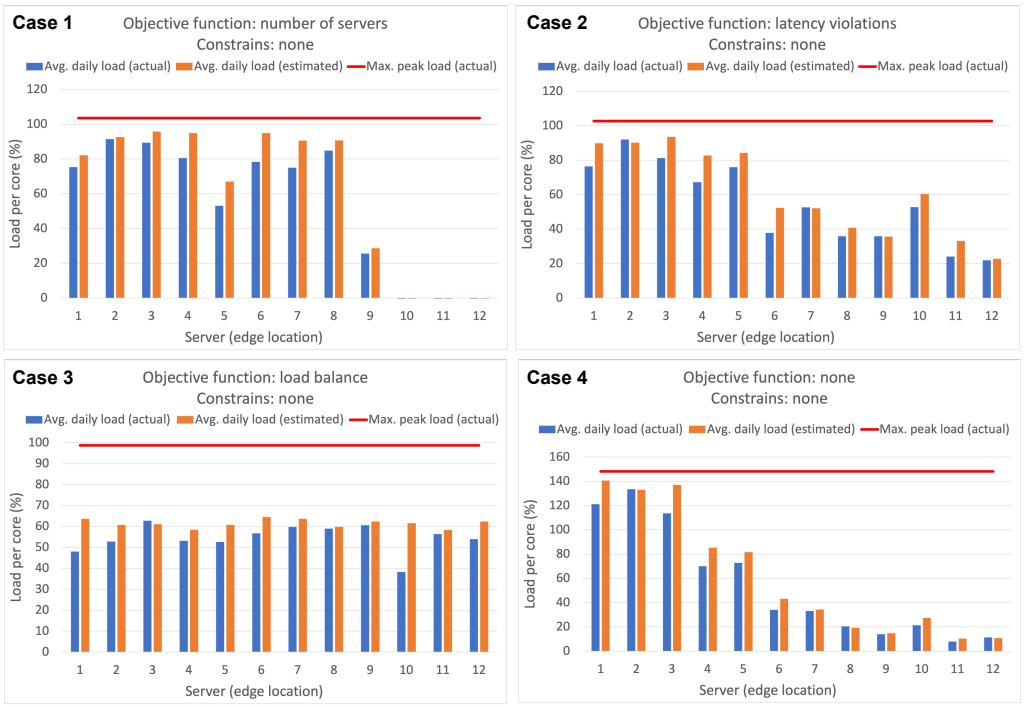


Figure 8. Estimated vs. actual load.

As observed in Table 3, allocating VMs to their preferred edge locations without any optimization (case 4) resulted in no latency violations. However, this approach led to significant server overloading, as depicted in Figure 8 (case 4). Servers 1, 2, and 3 exhibited actual average loads exceeding 100%, with a maximum peak load surpassing 140%. Upon implementing optimization (cases 1, 2, and 3), regardless of the chosen objective function, the strict constraint specified in Equation (17) (with $\alpha = 1$) effectively prevented overloading. Consequently, the actual average load remained below 100% for all three cases and the maximum peak load approached 100%.

When comparing the results of the different objective functions, optimizing the number of latency violations (case 2) allocated only 29 out of 500 VMs to an edge location different from their preferred one, utilizing the 12 available servers. On the other hand, when minimizing the number of servers in use (case 1), the solution obtained utilized only nine servers instead of 12. However, this resulted in a significant number of latency violations, with 389 out of 500 VMs affected. These minimum values (29 latency violations and nine servers) represent the global minima for these two optimization problems. Hence, solutions with fewer than nine servers or fewer than 29 latency violations are not considered feasible. Regarding the load profiles in Figure 8, for these two cases, although the average daily load per core did not exceed 100% for any server, there was a noticeable imbalance in the load across the different servers, and the peak load slightly exceeded 100%. The load balancing objective function (case 3) addressed this issue by utilizing all 12 available servers and incurring a larger number of latency violations (432). However, it achieved a better distribution of the load among the different servers and successfully avoided overloading in all time slots. Finally, regarding the execution times of the ILP solver, it can be observed that,

for the given problem size (500 VMs and 12 servers with 64 cores per server), the optimal solution could be reached in less than two seconds in all cases.

Another noteworthy observation from Figure 8 is the comparison between the estimated and actual average loads. The disparities between them stemmed from the prediction errors of the Bayesian regression model employed to forecast the CPU usage of the various VMs for the upcoming 24 h period. These prediction errors led to an estimation discrepancy in the server load, typically ranging from 1% to 30%. However, all the optimization algorithms (cases 1, 2, and 3) demonstrated the capability to achieve an optimal solution with an average daily load per server below 100%.

Next, we analyze how the different objective functions performed under various optional constraints.

7.3.2. Minimization of Number of Servers with Latency Violation Constraints

In this subsection, we analyze the objective function that minimized the number of servers, with different limits on the number of latency violations allowed. Table 4 and Figure 9 summarize the results of these experiments.

As observed in Table 4, when the limit on latency violations approached the global minimum (29 latency violations), it became necessary to utilize all 12 available servers. However, by relaxing the latency violation constraint, the number of servers in use could be reduced. For a limit of 60 latency violations, it was possible to achieve a solution with the minimum number of servers (9 servers). Increasing the latency violations limit beyond 60 did not lead to any significant improvement in solution quality. Thus, we can conclude that a solution with nine servers and around 60 latency violations represents a favorable trade-off between both variables. Regarding the solver's execution times, we observed that the introduction of constraints increased the time required to achieve an optimal solution. However, these times remained below 20 s in all cases.

Figure 9 displays the load profiles of these solutions, showcasing only the maximum and minimum values of the actual average daily load, as well as the maximum peak load for simplicity. As observed, a significant disparity existed between the maximum and minimum values of the average daily load. The maximum value was close to 100%, while the minimum value fell below 40%. This imbalance in the load distribution among the different servers is significant and can result in the overloading of certain servers during specific time slots. This is evident in the graph displaying the maximum peak load, which exceeded 100% in all cases.

Table 4. Allocation results for the 'number of servers' objective function under different latency violation constraints.

Limit on Latency Violations	Total Servers Used	Total Latency Violations	Solver Exec. Time
30	12	30	3.2 s
40	11	40	10.7 s
50	10	50	18.6 s
60	9	60	10.6 s
None	9	389	0.9 s

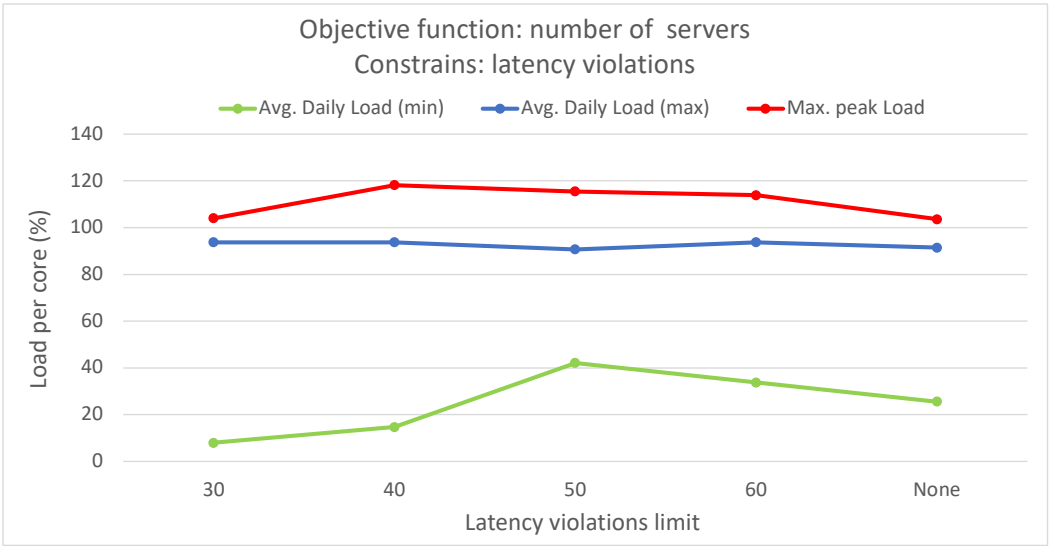


Figure 9. Actual load per core for the 'number of servers' objective function under different latency violation constraints.

7.3.3. Minimization of Total Latency Violations with Number of Server Constraints

Now, we analyze the objective function that minimized the number of latency violations, with different limits on the number of servers. Table 5 and Figure 10 summarize the results of these experiments. As observed, when the limit on the number of servers was increased from 9 to 12, the optimization algorithm yielded solutions with highly satisfactory numbers of latency violations, ranging from 59 to 29 (the global minimum). Therefore, we can further refine the conclusion from the prior case and state that one of the solutions with the best trade-off between the number of servers and latency violations was the one with 9 servers and 59 latency violations (which represented less than 12% of the total number of VMs). We can observe that the solver's execution time remained below 10 seconds in all cases.

Regarding the load profiles of these solutions (Figure 10), they exhibited similarities to the previous case, displaying a significant imbalance in load distribution, with a maximum peak load exceeding 100% in all the cases.

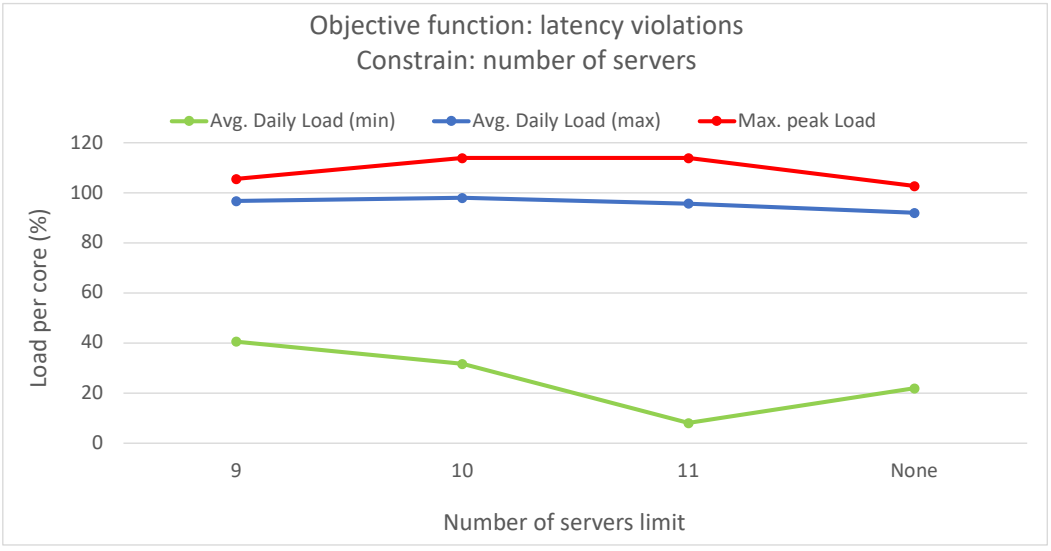


Figure 10. Actual load per core for the 'latency violations' objective function under different server constraints.

Table 5. Allocation results for the 'latency violations' objective function under different server constraints.

Limit on Number of Servers	Total Servers Used	Total Latency Violations	Solver Exec. Time
9	9	59	8.9 s
10	10	50	7.1 s
11	11	40	3.0 s
12	12	29	1.0 s

7.3.4. Optimization of Load Balance with Latency Violation and Number of Server Constraints

In this subsection, we analyze the objective function that optimizes the load balance using both constraints in the number of servers and the number of latency violations. We carried out two sets of experiments, the first with the maximum number of servers (12 servers) and different latency violation constraints (Figure 11), and the second with the minimum number of servers (nine servers) and different latency violation constraints (Figure 12). In this case, we only show the load profile graphs, since the number of servers and number of latency violations values were implicit in each experiment.

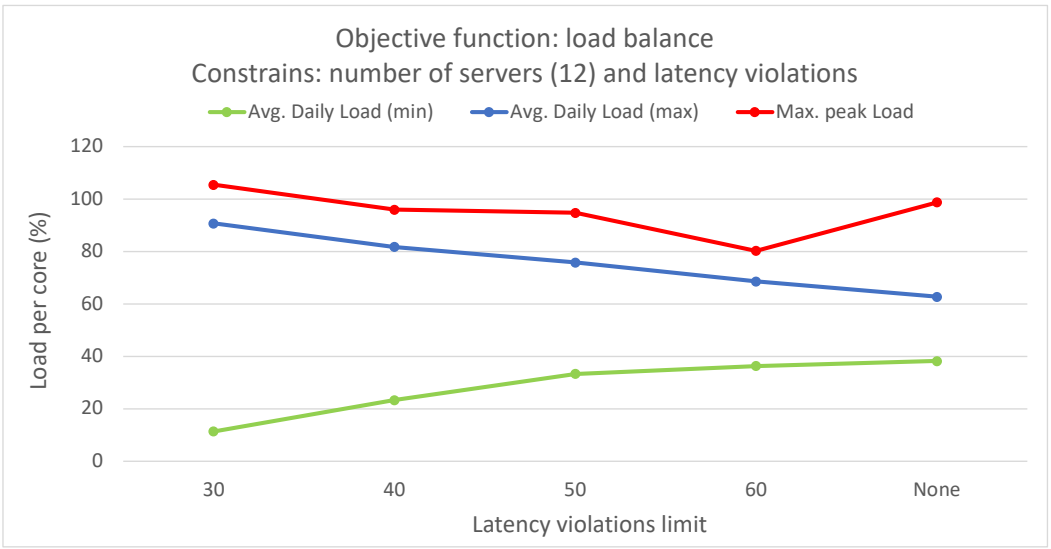


Figure 11. Actual load per core for the 'load balancing' objective function under different latency violation constrains and a server limit of 12.

When utilizing 12 servers, we could observe a convergence between the maximum and minimum values of the average daily load as we relaxed the constraint on latency violations. This indicates an improved load distribution among the servers. Furthermore, with a latency violation limit above 40, the maximum peak load remained below 100%, indicating the absence of server overloading in these scenarios. The solver's execution time stayed below two seconds in all experiments.

On the other hand, if we limit the number of servers to the minimum value of nine, each server has to support a higher load, leaving less room for load balance improvement. However, by establishing a latency violation limit of around 80–90, we can slightly reduce the gap between the maximum and minimum average daily load. This helps to avoid overloading, keeping the maximum peak load at around 100%. In some of these experiments, the solver took longer to converge to an optimal solution, with execution times ranging from 3 to 120 s.

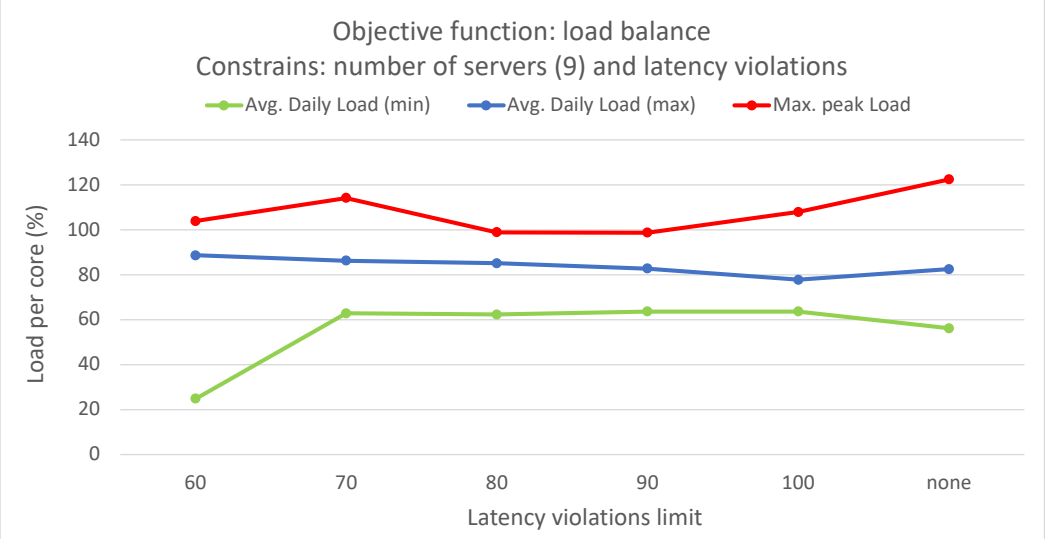


Figure 12. Actual load per core for the 'load balancing' objective function under different latency violation constrains and a server limit of 9.

7.3.5. Sensitivity Analysis

The final experiment in this study involved conducting a sensitivity analysis to examine the impact of the CPU usage prediction errors in the allocations results. The CPU usage percentage values used in the previous experiments were based on the predictions made by the Bayesian regression model shown in Section 7.2, without considering the prediction intervals. These prediction intervals represent the range of values within which the actual observation is expected to fall with a certain level of confidence and are typically symmetrically centered around the forecasted value, incorporating a designated error margin. In this analysis, we established a conservative prediction interval using an error margin of $\pm 10\%$, which was calculated as twice the rounded-up value of the RMSE for these predictions.

We compared the allocation results for the objective function of minimizing server count, with no additional constraints, using four sets of CPU usage values: the actual CPU usage values (ideal case, as these values cannot be known in advance in a real scenario); the mean value of the prediction interval; the upper bound of the prediction interval (i.e., mean predicted values plus a 10% error); and the lower bound of the prediction interval (i.e., mean predicted values minus a 10% error). Table 6 displays the total number of servers allocated in each scenario. As observed, using the actual CPU usage values allowed for an allocation solution with only eight servers, comparable to the solution obtained when using the lower bound values of the prediction intervals. On the other hand, allocation based on the mean predicted values resulted in a total of nine servers, while allocating based on the upper bound values of the prediction intervals requires ten servers.

Table 6. Sensitivity analysis for the 'number of servers' objective function with no optional constraints.

CPU Usage Values Used for the Optimization	Total Servers Used
Actual values	8
Predicted values (lower bound)	8
Predicted values (mean)	9
Predicted values (upper bound)	10

Figure 13 displays the load profiles for the four scenarios. It is evident that when using the actual CPU usage values (ideal case), the optimization algorithm returned an optimal solution with the minimum number of servers and a load per core that never exceeded 100%. Solutions based on mean values of the prediction intervals, and upper bound values, also exhibited minimal overloading, but using a higher number of servers. Conversely, the allocation based on lower bound values of the prediction intervals resulted in a minimum number of servers, but increased overloading, with a maximum peak load of 115%. In conclusion, the sensitivity analysis indicated that while the best solution in terms of server count and load profile was obtained using the actual CPU usage values, this is unfeasible in a real scenario. The solution utilizing the mean values of the prediction intervals offered the best trade-off between the total number of servers used and limited overloading.

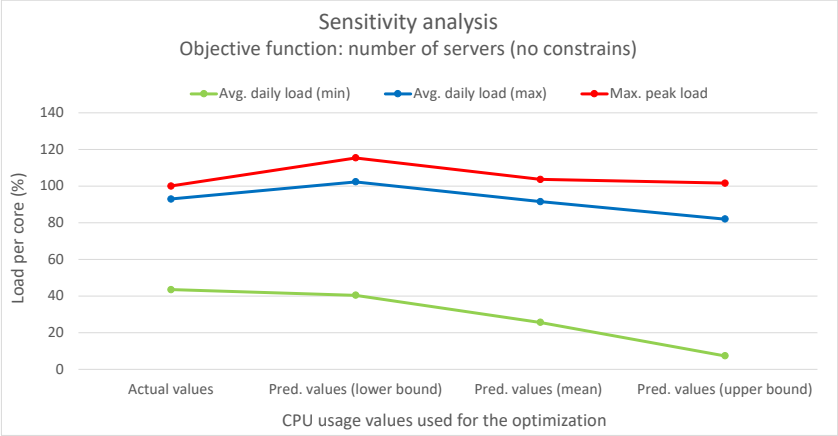


Figure 13. Sensitivity analysis results (actual load per core).

8. Conclusions and Future Work

This work introduced a novel Smart 5G Edge-Cloud Management Architecture based on OpenNebula. The proposed architecture incorporates experimental components from the ONEedge5G project which, in its current developmental phase, will incorporate predictive intelligence mechanisms for CPU utilization forecasting and optimization algorithms for the optimal allocation of virtual resources (VMs or containers) on geographically distributed 5G edge infrastructures.

This study emphasized infrastructure-level techniques for CPU usage forecasting, employing different statistical and ML time-series forecasting methods. Bayesian regression demonstrated the highest accuracy among the methods evaluated. The optimization problem addressed involved finding the optimal mapping of virtual resources to edge servers using different criteria and constraints. An ILP formulation was proposed for solving this problem. The scenario included a distributed edge infrastructure with physical servers, each with specific computing and memory capacities. Virtual resources with their computing requirements, as well as preferred edge locations based on proximity criteria, were deployed in the infrastructure. The results showed that optimizing different objective functions, such as minimizing the number of servers, reducing latency violations, or balancing server loads, led to improved management of the infrastructure. Allocating virtual resources based on their preferred edge locations without optimization resulted in no latency violations but severe server overloading. However, optimization algorithms successfully prevented overloading, while maintaining an average daily load per server below 100%.

By merging various optimization criteria and constraints, such as the number of servers in use and the number of latency violations, different optimized solutions can be obtained. For instance, one approach is to minimize latency violations to a minimum of 29 by utilizing all available servers, while another option is to reduce the number of servers to nine with only 59 latency violations. However, these solutions introduce significant

imbalances in load distribution among servers and may result in overloading on certain servers and time slots. To address this issue, incorporating the load balancing objective function proves effective, as it can achieve solutions with a limited number of latency violations, while improving load distribution and preventing overloading.

In our future work, we have various plans to enhance the prediction and optimization models. First, we aim to incorporate additional hardware metrics, including memory, bandwidth, and disk usage, to explore their potential correlation and integrate them into the mathematical models for optimization. Additionally, we plan to propose new optimization criteria based on the previous metrics, as well as integrating different objective functions using various multi-objective approaches. We also intend to investigate alternative optimization techniques, such as bio-inspired algorithms or reinforcement learning algorithms, to further improve the efficiency of the system. Another aspect not addressed in this study but worthy of consideration in future research is the potential utilization of nested virtualization, where containers are not run directly on bare-metal servers but within VMs. In such scenarios, two levels of allocation should be addressed: containers to VMs, and VMs to physical servers. Finally, expanding the capabilities of the ONEedge5G modules is on our agenda, encompassing functionality for capacity planning, prediction, and anomaly detection, as well as proactive auto-scaling mechanisms to facilitate elasticity management. These advancements will contribute to the comprehensive development of our Smart 5G Edge-Cloud Management Architecture and enable more robust and adaptive management of distributed 5G edge infrastructures.

Author Contributions: All authors participated in the definition of the architecture. R.M.-V. conceived the study, coordinated the research, conducted the experimental section, and drafted the manuscript. E.H., R.S.M. and I.M.L. also participated in the definition of the experimental scenarios and helped to refine the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Spanish Ministry for Digital Transformation and Civil Service through the UNICO I+D 6G Program, Project OneEdge5G-Intelligence and Automation for the Operation of Distributed Edge Systems on 5G Advanced Infrastructures (TSI-064200-2023-1), co-funded by the European Union—NextGenerationEU through the Recovery and Resilience Facility (RRF).

Data Availability Statement: Datasets used in this work are publicly available at the following links: Azure public datasets: https://github.com/Azure/AzurePublicDataset/blob/master/AzurePublicDatasetV1.md (accessed on 25 January 2024); GWA-T-12-Bitbrains datasets: http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains (accessed on 25 January 2024).

Conflicts of Interest: Ignacio M. Llorente and Rubén S. Montero are, respectively, an employee and an external collaborator of OpenNebula Systems company. The author declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AR	Autoregression
ARIMA	Auto-Regressive Integrated Moving Average
eMBB	Enhanced Mobile Broadband
FB	Facebook
AI	Artificial Intelligence
ILP	Integer Linear Programming
IoT	Internet of Things
LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
MEC	Multi-access Edge Computing
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning

mMTC	Massive Machine Type Communication
MSE	Mean Squared Error
N-BEATS	Neural Basis Expansion Analysis for Time Series
QoE	Quality of Experience
QoS	Quality of Service
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
SLAs	Service Level Agreements
UE	User Equipment
URLLC	Ultra-Reliable and Low-Latency Communication
VM	Virtual Machine

References

1. Huedo, E.; Montero, R.S.; Moreno-Vozmediano, R.; Vázquez, C.; Holer, V.; Llorente, I.M. Opportunistic Deployment of Distributed Edge Clouds for Latency-Critical Applications. *J. Grid Comput.* **2021**, *19*, 2. [CrossRef]
2. Aral, A.; Brandic, I.; Uriarte, R.B.; Nicola, R.D.; Scoca, V. Addressing Application Latency Requirements through Edge Scheduling. *J. Grid Comput.* **2019**, *17*, 677–698. [CrossRef]
3. Overbeek, D. Predictive DRS. Available online: https://blogs.vmware.com/management/2016/11/predictive-drs.html (accessed on 26 February 2024).
4. Kralicky, J. Opni-Multi-Cluster Observability with AIOps. Available online: https://opni.io/ (accessed on 26 February 2024).
5. Google Developers. Google Active Assist. Available online: https://cloud.google.com/recommender/docs/whatis-activeassist (accessed on 26 February 2024).
6. OpenNebula Systems. ONEedge: An On-Demand Software-Defined Edge Computing Solution. Available online: https://oneedge.io/ (accessed on 26 February 2024).
7. Milojičić, D.; Llorente, I.M.; Montero, R.S. OpenNebula: A Cloud Management Tool. *IEEE Internet Comput.* **2011**, *15*, 11–14. [CrossRef]
8. OpenNebula Systems. OpenNebula: The Open Source Cloud & Edge Computing Platform. Available online: https://opennebula.io (accessed on 25 January 2024).
9. Liu, B.; Zhu, P.; Li, J.; Wang, D.; You, X. Energy-Efficient Optimization in Distributed Massive MIMO Systems for Slicing eMBB and URLLC Services. *IEEE Trans. Veh. Technol.* **2023**, *72*, 10473–10487. [CrossRef]
10. Filali, A.; Abouaomar, A.; Cherkaoui, S.; Kobbane, A.; Guizani, M. Multi-Access Edge Computing: A Survey. *IEEE Access* **2020**, *8*, 197017–197046. [CrossRef]
11. Hassan, N.; Yau, K.L.A.; Wu, C. Edge Computing in 5G: A Review. *IEEE Access* **2019**, *7*, 127276–127289. [CrossRef]
12. Raeisi-Varzaneh, M.; Dakkak, O.; Habbal, A.; Kim, B.S. Resource Scheduling in Edge Computing: Architecture, Taxonomy, Open Issues and Future Research Directions. *IEEE Access* **2023**, *11*, 25329–25350. [CrossRef]
13. Sarah, A.; Nencioni, G.; Khan, M.M.I. Resource Allocation in Multi-access Edge Computing for 5G-and-beyond networks. *Comput. Netw.* **2023**, *227*, 109720. [CrossRef]
14. Dai, Y.; Xu, D.; Zhang, K.; Lu, Y.; Maharjan, S.; Zhang, Y. Deep Reinforcement Learning for Edge Computing and Resource Allocation in 5G Beyond. In Proceedings of the 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 16–19 October 2019; pp. 866–870. [CrossRef]
15. Chen, M.; Miao, Y.; Gharavi, H.; Hu, L.; Humar, I. Intelligent Traffic Adaptive Resource Allocation for Edge Computing-Based 5G Networks. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *6*, 499–508. [CrossRef]
16. Wang, S.; Chen, M.; Liu, X.; Yin, C.; Cui, S.; Vincent Poor, H. A Machine Learning Approach for Task and Resource Allocation in Mobile-Edge Computing-Based Networks. *IEEE Internet Things J.* **2021**, *8*, 1358–1372. [CrossRef]
17. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Emerg. Top. Comput.* **2021**, *9*, 1529–1541. [CrossRef]
18. Šlapak, E.; Gazda, J.; Guo, W.; Maksymyuk, T.; Dohler, M. Cost-Effective Resource Allocation for Multitier Mobile Edge Computing in 5G Mobile Networks. *IEEE Access* **2021**, *9*, 28658–28672. [CrossRef]
19. Masdari, M.; Khoshnevis, A. A survey and classification of the workload forecasting methods in cloud computing. *Clust. Comput.* **2020**, *23*, 2399–2424. [CrossRef]
20. Yadav, A.; Kushwaha, S.; Gupta, J.; Saxena, D.; Singh, A.K. A Survey of the Workload Forecasting Methods in Cloud Computing. In Proceedings of the 3rd International Conference on Machine Learning, Advances in Computing, Renewable Energy and Communication (Lecture Notes in Electrical Engineering), Singapore, 10–11 December 2022; pp. 539–547. [CrossRef]
21. Rawat, P.S.; Gupta, P. Application of Optimization Techniques in Cloud Resource Management. In *Bio-Inspired Optimization in Fog and Edge Computing Environments*; Auerbach Publications: New York, NY, USA, 2022; pp. 73–90. [CrossRef]
22. Ghobaei-Arani, M.; Souri, A.; Rahmanian, A.A. Resource Management Approaches in Fog Computing: A Comprehensive Review. *J. Grid Comput.* **2019**, *18*, 1–42. [CrossRef]

23. Gao, J.; Wang, H.; Shen, H. Machine Learning Based Workload Prediction in Cloud Computing. In Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN), Honolulu, HI, USA, 3–6 August 2020. [CrossRef]
24. Devi, K.L.; Valli, S. Time series-based workload prediction using the statistical hybrid model for the cloud environment. *Computing* **2023**, *105*, 353–374. [CrossRef]
25. Rezvani, M.; Akbari, M.K.; Javadi, B. Resource Allocation in Cloud Computing Environments Based on Integer Linear Programming. *Comput. J.* **2014**, *58*, 300–314. [CrossRef]
26. Lu, J.; Zhao, W.; Zhu, H.; Li, J.; Cheng, Z.; Xiao, G. Optimal machine placement based on improved genetic algorithm in cloud computing. *J. Supercomput.* **2021**, *78*, 3448–3476. [CrossRef]
27. Khan, T.; Tian, W.; Zhou, G.; Ilager, S.; Gong, M.; Buyya, R. Machine learning (ML)-centric resource management in cloud computing: A review and future directions. *J. Netw. Comput. Appl.* **2022**, *204*, 103405. [CrossRef]
28. Gupta, P.; Goyal, M.K.; Chakraborty, S.; Elngar, A.A. *Machine Learning and Optimization Models for Optimization in Cloud*; Chapman and Hall/CRC: New York, NY, USA, 2022. [CrossRef]
29. Djigal, H.; Xu, J.; Liu, L.; Zhang, Y. Machine and Deep Learning for Resource Allocation in Multi-Access Edge Computing: A Survey. *IEEE Commun. Surv. Tutor.* **2022**, *24*, 2449–2494. [CrossRef]
30. Patsias, V.; Amanatidis, P.; Karampatzakis, D.; Lagkas, T.; Michalakopoulou, K.; Nikitas, A. Task Allocation Methods and Optimization Techniques in Edge Computing: A Systematic Review of the Literature. *Future Internet* **2023**, *15*, 254. [CrossRef]
31. Liang, Q.; Zhang, J.; Zhang, Y.H.; Liang, J.M. The placement method of resources and applications based on request prediction in cloud data center. *Inf. Sci.* **2014**, *279*, 735–745. [CrossRef]
32. Moreno-Vozmediano, R.; Montero, R.S.; Huedo, E.; Llorente, I.M. Efficient resource provisioning for elastic Cloud services based on machine learning techniques. *J. Cloud Comput.* **2019**, *8*, 1–18. [CrossRef]
33. Mehmood, T.; Latif, S.; Malik, S. Prediction Of Cloud Computing Resource Utilization. In Proceedings of the 15th International Conference on Smart Cities: Improving Quality of Life Using ICT & IoT (HONET-ICT), Islamabad, Pakistan, 8–10 October 2018. [CrossRef]
34. Al-Asaly, M.S.; Bencherif, M.A.; Alsanad, A.; Hassan, M.M. A deep learning-based resource usage prediction model for resource provisioning in an autonomic cloud computing environment. *Neural Comput. Appl.* **2021**, *34*, 10211–10228. [CrossRef]
35. Aslanpour, M.S.; Gill, S.S.; Toosi, A.N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet Things* **2020**, *12*, 100273. [CrossRef]
36. Yang, J.; Liu, C.; Shang, Y.; Mao, Z.; Chen, J. Workload Predicting-Based Automatic Scaling in Service Clouds. In Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, Santa Clara, CA, USA, 28 June–3 July 2013. [CrossRef]
37. Di, S.; Kondo, D.; Cirne, W. Host load prediction in a Google compute cloud with a Bayesian model. In Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 24–29 June 2012. [CrossRef]
38. Calheiros, R.N.; Masoumi, E.; Ranjan, R.; Buyya, R. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Trans. Cloud Comput.* **2015**, *3*, 449–458. [CrossRef]
39. Khan, T.; Tian, W.; Ilager, S.; Buyya, R. Workload forecasting and energy state estimation in cloud data centres: ML-centric approach. *Future Gener. Comput. Syst.* **2022**, *128*, 320–332. [CrossRef]
40. Nawrocki, P.; Osypanka, P. Cloud Resource Demand Prediction using Machine Learning in the Context of QoS Parameters. *J. Grid Comput.* **2021**, *19*, 20. [CrossRef]
41. Bi, J.; Li, S.; Yuan, H.; Zhou, M. Integrated deep learning method for workload and resource prediction in cloud systems. *Neurocomputing* **2021**, *424*, 35–48. [CrossRef]
42. Chen, L.; Zhang, W.; Ye, H. Accurate workload prediction for edge data centers: Savitzky-Golay filter, CNN and BiLSTM with attention mechanism. *Appl. Intell.* **2022**, *52*, 13027–13042. [CrossRef]
43. Tordsson, J.; Montero, R.S.; Moreno-Vozmediano, R.; Llorente, I.M. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Gener. Comput. Syst.* **2012**, *28*, 358–367. [CrossRef]
44. Lucas-Simarro, J.L.; Moreno-Vozmediano, R.; Montero, R.S.; Llorente, I.M. Scheduling strategies for optimal service deployment across multiple clouds. *Future Gener. Comput. Syst.* **2013**, *29*, 1431–1441. [CrossRef]
45. Entrialgo, J.; Díaz, J.L.; García, J.; García, M.; García, D.F. Cost Minimization of Virtual Machine Allocation in Public Clouds Considering Multiple Applications. In *Lecture Notes in Computer Science*; Springer International Publishing: Cham, Switzerland, 2017; pp. 147–161. [CrossRef]
46. Entrialgo, J.; García, M.; Díaz, J.L.; García, J.; García, D.F. Modelling and simulation for cost optimization and performance analysis of transactional applications in hybrid clouds. *Simul. Model. Pract. Theory* **2021**, *109*, 102311. [CrossRef]
47. Fadda, E.; Plebani, P.; Vitali, M. Optimizing Monitorability of Multi-cloud Applications. In *Lecture Notes in Computer Science*; Springer International Publishing: Cham, Switzerland, 2016; pp. 411–426. [CrossRef]
48. Fadda, E.; Plebani, P.; Vitali, M. Monitoring-Aware Optimal Deployment for Applications Based on Microservices. *IEEE Trans. Serv. Comput.* **2021**, *14*, 1849–1863. [CrossRef]
49. Gong, Y. Optimal Edge Server and Service Placement in Mobile Edge Computing. In Proceedings of the 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 11–13 December 2020; Volume 9, pp. 688–691. [CrossRef]

50. Takeda, A.; Kimura, T.; Hirata, K. Joint optimization of edge server and virtual machine placement in edge computing environments. In Proceedings of the 2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Auckland, New Zealand, 7–10 December 2020; pp.1545–1548.
51. Abbasi, M.; Pasand, E.M.; Khosravi, M.R. Workload Allocation in IoT-Fog-Cloud Architecture Using a Multi-Objective Genetic Algorithm. *J. Grid Comput.* **2020**, *18*, 43–56. [CrossRef]
52. Devarasetty, P.; Reddy, S. Genetic algorithm for quality of service based resource allocation in cloud computing. *Evol. Intell.* **2019**, *14*, 381–387. [CrossRef]
53. Xia, W.; Shen, L. Joint Resource Allocation at Edge Cloud Based on Ant Colony Optimization and Genetic Algorithm. *Wirel. Pers. Commun.* **2021**, *117*, 355–386. [CrossRef]
54. Pradhan, A.; Bisoy, S.K.; Das, A. A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 4888–4901. [CrossRef]
55. Wang, B.; Liu, F.; Lin, W. Energy-efficient VM scheduling based on deep reinforcement learning. *Future Gener. Comput. Syst.* **2021**, *125*, 616–628. [CrossRef]
56. Jiang, Y.; Kodialam, M.; Lakshman, T.V.; Mukherjee, S.; Tassiulas, L. Resource Allocation in Data Centers Using Fast Reinforcement Learning Algorithms. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4576–4588. [CrossRef]
57. Hummaida, A.R.; Paton, N.W.; Sakellariou, R. Scalable Virtual Machine Migration using Reinforcement Learning. *J. Grid Comput.* **2022**, *20*, 15. [CrossRef]
58. Prometheus authors. Prometheus: From Metrics to Insight. Available online: https://prometheus.io (accessed on 25 January 2024).
59. Unit8 SA. Darts: Time Series Made Easy in Python. Available online: https://unit8co.github.io/darts (accessed on 25 January 2024).
60. Unit8 SA. Darts Baseline Models. Available online: https://unit8co.github.io/darts/generated_api/darts.models.forecasting.baselines.html (accessed on 25 January 2024).
61. Box, G.E.P.; Jenkins, G.M.; Reinsel, G.C. *Time Series Analysis*; Wiley: Hoboken, NY, USA, 2008. [CrossRef]
62. Smith, T.G. Alkaline-ML API Reference (pdmarima.arima.auto-arima). Available online: https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html (accessed on 25 January 2024).
63. Box, G.E.; Tiao, G.C. *Bayesian Inference in Statistical Analysis*; John Wiley & Sons, Inc.: New York, NY, USA, 1992. [CrossRef]
64. Scikit-Learn Developers. Bayesian Ridge Regression. Available online: https://scikit-learn.org/0.24/modules/linear_model.html#bayesian-ridge-regression (accessed on 25 January 2024).
65. Facebook Open Source. Prophet: Forecasting at Scale. Available online: https://facebook.github.io/prophet (accessed on 25 January 2024).
66. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
67. Oreshkin, B.N.; Carpov, D.; Chapados, N.; Bengio, Y. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26 April–1 May 2020.
68. Wilkes, J. The Google Workload Cluster Traces. Available online: https://github.com/google/cluster-data (accessed on 25 January 2024).
69. Cortez, E. The Azure Public Dataset. Available online: https://github.com/Azure/AzurePublicDataset (accessed on 25 January 2024).
70. Delft University of Technology. The Grid Workloads Archive: GWA-T-12 Bitbrains. Available online: http://gwa.ewi.tudelft.nl (accessed on 25 January 2024).
71. Ding, H. The Alibaba Cluster Trace Program. Available online: https://github.com/alibaba/clusterdata (accessed on 25 January 2024).
72. Nudler, E. Predator: Distributed Open-Source Performance Testing Platform for APIs. Available online: https://predator.dev (accessed on 25 January 2024).
73. Heyman, J.; Byström, C.; Hamrén, J.; Heyman, H. Locust: A modern load testing framework. Available online: https://locust.io (accessed on 25 January 2024).
74. Kolosov, O.; Yadgar, G.; Maheshwari, S.; Soljanin, E. Benchmarking in The Dark: On the Absence of Comprehensive Edge Datasets. In Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20). USENIX Association, Santa Clara, CA, USA, 25–26 June 2020.
75. Tocze, K.; Schmitt, N.; Kargen, U.; Aral, A.; Brandic, I. Edge Workload Trace Gathering and Analysis for Benchmarking. In Proceedings of the 2022 IEEE 6th International Conference on Fog and Edge Computing (ICFEC), Messina, Italy, 18–19 May 2022. [CrossRef]
76. Cortez, E.; Bonde, A.; Muzio, A.; Russinovich, M.; Fontoura, M.; Bianchini, R. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In Proceedings of the SOSP'17: 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017; pp. 153–167. [CrossRef]
77. Shen, S.; Beek, V.V.; Iosup, A. Statistical Characterization of Business-Critical Workloads Hosted in Cloud Datacenters. In Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4–7 May 2015; pp. 465–474. [CrossRef]

78. Matthew, S.; Stefan, V.; Lou, H.; John, F.; Miles, L.; Ted, R.; Haroldo, G.S.; Samuel, B.; Bjarni, K.; Bruno, P.; et al. coin-or/Cbc: Release Releases/2.10.8. Available online: https://zenodo.org/records/6522795 (accessed on 25 January 2024).
79. Cplex IBM ILOG. V12.1: User's Manual for CPLEX. *Int. Bus. Mach. Corp.* **2009**, *46*, 157.
80. Bestuzheva, K.; Besançon, M.; Chen, W.K.; Chmiela, A.; Donkiewicz, T.; van Doornmalen, J.; Eifler, L.; Gaul, O.; Gamrath, G.; Gleixner, A.; et al. The SCIP Optimization Suite 8.0. *arXiv* **2021**, arXiv:2112.08872.
81. Google for Developers. Get Started with OR-Tools for Python. Available online: https://developers.google.com/optimization/introduction/python (accessed on 25 January 2024).
82. Google for Developers. OR-Tools Python Reference: Linear Solver (Pywraplp Module). Available online: https://developers.google.com/optimization/reference/python/linear_solver/pywraplp (accessed on 25 January 2024).

future internet

MDPI

Article

SRv6-Based Edge Service Continuity in 5G Mobile Networks †

Laura Lemmi [1], Carlo Puliafito [1], Antonio Virdis [1,*] and Enzo Mingozzi [1,2]

[1] Department of Information Engineering, University of Pisa, 55022 Pisa, Italy; laura.lemmi@phd.unipi.it (L.L.); carlo.puliafito@unipi.it (C.P.); enzo.mingozzi@unipi.it (E.M.)
[2] Department of Information Engineering, University of Florence, 50139 Florence, Italy
* Correspondence: antonio.virdis@unipi.it
† This paper is an extended version of our paper published in IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN 2023), "Ensuring Lossless Workload Migration at the Edge with SRv6", Dresden, Germany, 7–9 November 2023.

Abstract: Ensuring compliance with the stringent latency requirements of edge services requires close cooperation between the network and computing components. Within mobile 5G networks, the nomadic behavior of users may impact the performance of edge services, prompting the need for workload migration techniques. These techniques allow services to follow users by moving between edge nodes. This paper introduces an innovative approach for edge service continuity by integrating Segment Routing over IPv6 (SRv6) into the 5G core data plane alongside the ETSI multi-access edge computing (MEC) architecture. Our approach maintains compatibility with non-SRv6 5G network components. We use SRv6 for packet steering and Software-Defined Networking (SDN) for dynamic network configuration. Leveraging the SRv6 Network Programming paradigm, we achieve lossless workload migration by implementing a packet buffer as a virtual network function. Our buffer may be dynamically allocated and configured within the network. We test our proposed solution on a small-scale testbed consisting of an Open Network Operating System (ONOS) SDN controller and a core network made of P4 BMv2 switches, emulated using Mininet. A comparison with a non-SRv6 alternative that uses IPv6 routing shows the higher scalability and flexibility of our approach in terms of the number of rules to be installed and time required for configuration.

Keywords: 5G; edge computing; SRv6; service continuity; SDN; ETSI MEC

Citation: Lemmi, L.; Puliafito, C.; Virdis, A.; Mingozzi, E. SRv6-Based Edge Service Continuity in 5G Mobile Networks. *Future Internet* **2024**, *16*, 138. https://doi.org/10.3390/fi16040138

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 22 March 2024
Revised: 14 April 2024
Accepted: 17 April 2024
Published: 19 April 2024

1. Introduction

Emerging applications demand efficient network infrastructure and protocols, as well as the collaboration of network and computing elements. 5G network technology meets these requirements, as it features qualities such as programmability, low latency, high bandwidth, network, and context awareness [1]. 5G expands the previous generation capabilities, defining new application areas: Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and Massive Machine-Type Communications (mMTC) [2,3].

5G is a technology that relies on small cells, which have low-power and short-range transmissions. This feature enables higher bandwidth and reliability, but also leads to more frequent handovers due to the limited coverage [4,5]. The ability of the client to move seamlessly is, in fact, a fundamental feature of 5G networks.

Edge computing is a key concept for 5G applications that require very low latency, such as augmented reality, virtual reality, autonomous vehicles, and the Internet of Things (IoT) [6]. Edge computing is characterized by its wide and pervasive distribution across geographic areas, close to the end user. The proximity of computing facilities is the key factor that enables ultra-low latency, high bandwidth, and security. Edge services, i.e., those services running in edge data centers, may then require migration across edge sites to support user mobility. When the user changes her/his location, the edge site that hosts

the user-assigned edge service may become too distant and cause higher latency. In this situation, another edge data center may be closer. The edge service should "follow" the client to preserve the Quality of Service (QoS). If services are stateful, moving the service along with its internal state is required. In this context, one way to achieve this mechanism is to wrap the service in virtual machines or containers and migrate the instance from the original edge site to a target one using existing methods to smoothly transfer the whole package [7].

The European Telecommunications Standards Institute Industry Specification Group (ETSI ISG) worked in the last decade to create a standardized, open, multi-vendor, and multi-access edge computing environment. The ETSI specifies the elements required to enable applications to be hosted in a multi-vendor multi-access edge computing (MEC) environment [8]. The MEC architecture provides cloud computing capabilities and a distributed environment at the edge of the network. The system consists of MEC Hosts dislocated at the edge of the network and connected through the underlay communication infrastructure. It provides advantages in terms of delay, network traffic, and data localization. The ETSI MEC has been identified as the key technology for supporting low-latency edge services in 5G networks. The ETSI MEC architecture provides a system orchestration mechanism that enables the management of service instances and their dynamic association with the user equipment (UE) [9]. The orchestration oversees the selection of the best MEC Host where a service instance should run in order to guarantee the application requirements' satisfaction. It is also in charge of triggering the proper network configuration through the 5G core (5GC) in a centralized manner. The ETSI MEC environment also supports the service continuity required by user mobility in 5G networks. The MEC system, based on the information exchanged with the 5GC network entities, verifies which is, at any time, the best MEC Host for an edge service. To guarantee the latency constraints, instance migration may be triggered [10]. MEC can guarantee service continuity by employing various techniques such as DNS and device–application interface solutions. Most of the proposed methods, however, require the active involvement of the client [11].

In line with the 5G requirements for a network that is highly scalable, maintainable, and programmable [12], Segment Routing over IPv6 (SRv6) has been recently proposed as the new data plane for 5GC [13,14]. SRv6 provides flexibility, programmability, scalability, and has great potential for further development, which are essential features of 5G networks. SRv6 Network Programming allows integrating functions, implemented as virtual network functions (VNFs), as part of the network node packet processing. SRv6 also facilitates the dynamic reconfiguration of the network, enabling seamless service continuity when edge services are migrated to follow their mobile 5G users. Enabling seamless service continuity means letting the client reach the edge service in its new location, while guaranteeing qualitative and quantitative requirements. For what concerns qualitative aspects, the service must be reached transparently to the client application logic, which is achieved through SRv6 by keeping the same IP address for the migrated service. Moreover, no packets should be lost during the service migration process. With respect to quantitative requirements, the application should not suffer from any significant impairment in terms of, for example, experiencing latency or throughput. In addition, SRv6 simplifies the network protocol stack in 5G scenarios. Finally, SRv6 supports the centralized control plane [15], allowing an easy integration with the 5G-MEC environment.

Let us consider the scenario of Figure 1, where MEC Hosts are deployed in proximity to the gNBs to run latency-constrained services. A UE can move and access the network through a different gNB. After the handover, the MEC system may decide to relocate the edge service associated with the moved user to a different MEC Host. Subsequently, a network reconfiguration is required to allow the UE to seamlessly reach the edge service in its new location. In this context, we propose a solution for lossless workload migration, based on buffering implemented as a VNF that is dynamically allocated in the network and by the network.

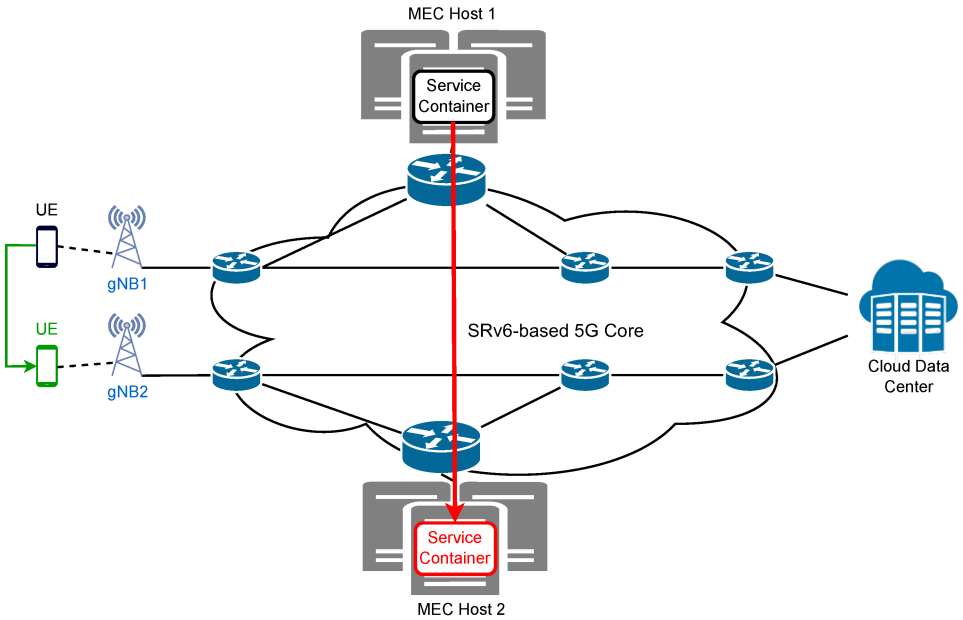


Figure 1. SRv6-based reference scenario.

Our approach relies on the integration between MEC and 5G, using an SRv6-based data plane, and on the strategic implementation of SRv6 Network Programming. Leveraging the MEC environment enables the allocation of edge services and their dynamic relocation to accommodate user mobility. The SRv6 Network Programming paradigm provides the capabilities required to ensure service continuity within the ETSI MEC framework through dynamic reconfiguration of the network, by means of an SDN controller, and through the integration of buffering as a VNF within the packet-processing pipeline.

A preliminary version of this work was presented in [16]. This work extends it by integrating our SRv6-based, service continuity solution into the 5GC data plane and the ETSI MEC architecture. Specifically, this work makes the following four-fold contribution:

1. It presents a converged MEC-based architecture for service continuity in 5G with an SRv6 programmable data plane.
2. It defines an SRv6 `End.Buffer` behavior for guaranteeing lossless workload migration in SRv6-compliant 5G networks.
3. To ensure compatibility with non-SRv6 5G network components, it introduces a new `End.M.GTP6.D.Buffer` behavior acting as a VNF at the border between such components and the SRv6 5GC network.
4. It describes a Proof-of-Concept (PoC) implementation of the proposed system based on P4 devices. We evaluated our solution over a limited-scale test environment and compared the performance against a non-SRv6 alternative that uses IPv6 routing.

The remainder of this paper is organized as follows. Section 2 introduces background content on SRv6 Network Programming. Next, Section 3 reports the related work in the field. Then, Section 4 presents our converged 5G and MEC architecture and related procedures, whereas Section 5 describes our SRv6-based migration solution that is integrated in the above 5G-MEC context. In Section 6, we detail the experimental analysis of the proposed solution. Finally, Section 7 concludes the paper.

2. Background on SRv6 Network Programming

2.1. SRv6 Architecture

Segment Routing (SR) is a source-routing-based tunneling method enabling packet steering through a network via a list of segments. It accommodates the Equal-Cost Multi-path (ECMP) aspect of IP and offers compatibility with both distributed and centralized control paradigms for segment construction and network optimization [17,18].

The data plane of SR architecture outlines the encoding of segments to be incorporated into a packet and defines the forwarding semantics for each segment, i.e., how a node should process a packet based on the segment. Two different implementations of the SR architecture are available: SR over MPLS and SRv6.

The control plane of SR architecture outlines the method for distributing segments across network nodes. Additionally, it establishes a mechanism for instructing nodes on how to apply the segment list to a packet or a flow.

Each segment is identified by a Segment IDentifier (SID), which is an MPLS label in the case of MPLS implementation or an IPv6 address in the case of SRv6 implementation. Segments are divided into Node SIDs, Adjacency SIDs, and Service SIDs. In the first type, the segment identifies a node, which means that a packet must be forwarded to that node through the shortest path. The Adjacency SID is a segment identifying the link over which the packet must be forwarded. These kinds of SIDs have local scope, limited to the node that processes it. Finally, the Service SID is a kind of segment identifying a function that must be executed on a node. That SID contains information on both the function and the node that must execute it. In this work, we utilize SRv6 implementation, which is currently the most commonly employed.

IPv6 architecture can embed SR functionalities through an extension header, called Segment Routing Header (SRH), which is a type of Routing Header [19]. Figure 2 depicts the SRv6 header structure. In SRv6, each segment is an IPv6 address, which represents an action(s) that a network node should execute, for example, forwarding the packet to a next hop or running a VNF. The SID currently designated as the packet's destination address is said to be the active segment. The SRH is composed of the list of segments, entered in reverse order, i.e., the last segment to be reached will be placed at index 0, while the first one at the last index. The SRH contains other fields, such as Last Entry, which states the index (starting from 0) of the last element in the segment list, i.e., the first one to be used, and Segment Left, which specifies the index of the active segment.

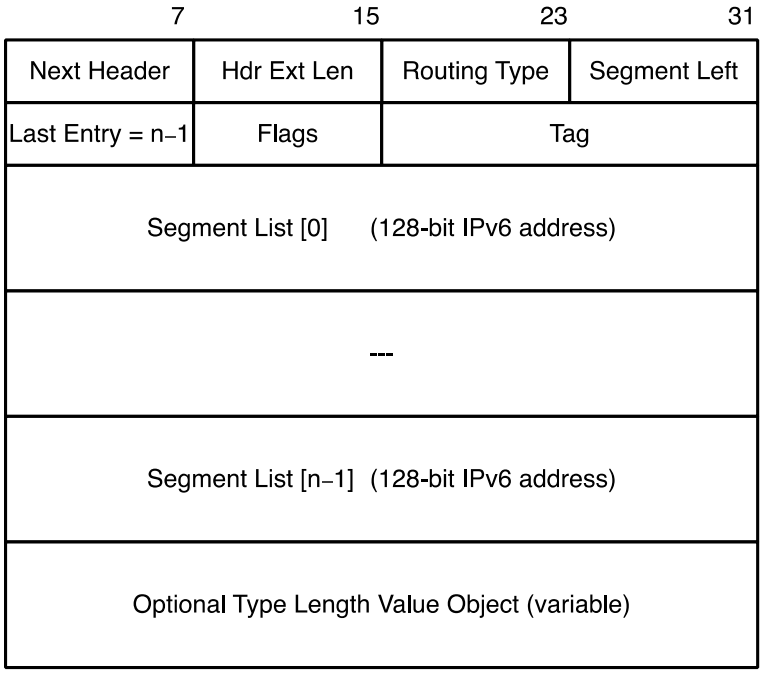


Figure 2. The SRv6 header structure.

The SR architecture comprises three types of nodes: source, endpoint, and transit [19]. The source node is the one in charge of embedding the SRH in the packet. As detailed later, the source node may either embed the SRH in the original IPv6 packet, called inline SRH, or encapsulate the original packet into an outer IPv6 header with the SRH extension,

called encapsulated SRH. In the case of encapsulation, the SRH may not be necessary. This happens when the number of segments in the list is equal to one. In this case, it is enough to insert that segment as the destination address of the outer IPv6 header. In the case of the inline SRH, the source node must modify the original IPv6 header fields as follows. It sets as the destination address the last segment of the SRH segment list. It increases the payload length by the SRH size. It sets as the Next Header the Routing Header. It also needs to set the Next Header field of the SRH to the Next Header field value of the original IPv6 packet. In the case of the encapsulated SRH, the source node must fill the outer IPv6 header fields as follows. It sets as the source address the current node address, and as the destination address, the last segment of the SRH segment list. Its Next Header field value must be equal to 34, which is the Routing Header. Finally, the payload length must be set to the SRH size plus the original IPv6 packet size. The transit node is any node that forwards a packet based on its IPv6 destination address, without processing the SRH. The packet destination address is not a node's locally configured SID. The transit node may even be non-SRv6 capable, but it must be IPv6 capable. Finally, the endpoint node is a node receiving a packet whose IPv6 destination address is a locally instantiated SID. It will process the SRH and apply the instruction coded into the received SID.

The source node is the one in charge of steering packets based on a policy, i.e., a list of segments, which are instructions [20]. The policy can be identified through an IPv6 address, called Binding SID (BSID). A given policy may be executed by different source nodes. Note, however, that each source node must use a different BSID from other source nodes, to indicate that same policy. When a source node receives a packet whose destination address matches with a locally instantiated BSID, it applies the corresponding policy. In other words, the source node inserts in the packet the list of segments associated with that policy [20–22]. A source node must instantiate a different BSID for each policy it wants to implement. The BSID is a key element in the SR architecture, providing scalability and network opacity. It also allows the exploitation of the same SR policy by different entities, because setting a specific BSID as the destination address is enough to obtain the execution of a policy.

2.2. SRv6 Network Programming

SRv6 Network Programming is a framework that allows the encoding of a sequence of instructions in an IPv6 packet. Each instruction is identified by an SID and is executed on the node that has instantiated that SID. Those instructions can range from simple packet forwarding to more complex functions [23]. Functions are called behaviors in the SRv6 Network Programming domain.

An SRv6 Service SID is a 128-bit address structured as LOC:FUNC:ARG. The number of bits composing each part are not pre-established; the only constraint is that the total length cannot exceed 128. LOC identifies the locator, i.e., the endpoint node, and it is the routable part. FUNC is an opaque identifier of the behavior locally instantiated by the node identified by LOC. It is bounded to the SID. Finally, the ARG field contains the behavior's arguments. The SID structure is usually defined by the SRv6 domain provider, which establishes the number of bits to be used for each part and assigns the locator to each node. Then, each node or a central entity can assign the bits related to the function and argument parts. In a distributed scenario, each SID must be advertised together with a codepoint, mapping the SID to a specific function. Codepoints and their meaning must be known by all the SRv6-enabled devices within the domain. In a centralized scenario, the SIDs will be configured and installed by the central controller, which also instructs the source node to enter a segment list.

In [23], a set of standard behaviors is defined; however, SRv6 Network Programming has the flexibility to support the association of any function with an SID, as shown in [24–26]. In what follows, we give an overview of the standard behaviors exploited in this work. In Section 2.2.1, we describe `H.Encaps.Red`, which is used to tunnel the packet through the 5GC, encapsulating it into an outer IPv6 header along with an additional SRv6 sub-header.

In Section 2.2.2, we illustrate `H.Insert`, which, instead, adds an SRv6 sub-header to the original IPv6 packet. Finally, in Section 2.2.3, we detail the `End` behavior, applied by each SRv6 endpoint.

2.2.1. H.Encaps.Red

This behavior is applied by the source node in the case of the SRH-encapsulated version. When the source node receives a packet whose destination address matches with a locally instantiated BSID associated with the `H.Encaps.Red` behavior, it does the following. Firstly, it encapsulates the packet into an outer IPv6 header. Then, if the segment list to be entered is more than one element long, it embeds the SRH into the IPv6 outer header. The first segment to be reached is not included in the SRH; instead, it is just set as the destination address. Based on that, the Segment Left field of the SRH will be the index of the last element in the list plus one, because the active segment is not included in the SRH. In addition, the header length is 128 bits shorter. If, instead, the segment list to be entered contains just one element, the SRH is not needed: the element will be placed in the destination address of the outer IPv6 header. Finally, the outer IPv6 header fields are properly set, and the packet is forwarded.

2.2.2. H.Insert

This behavior is applied by the source node in the case of the inline SRH version. When the source node receives a packet whose destination address matches with a locally instantiated SID associated with the `H.Insert` behavior, it does what follows. Firstly, it embeds the SRH into the IPv6 header. Then, it sets the last segment of the list as the packet destination address. Finally, the SRH and IPv6 header fields are properly set, and the packet is forwarded.

Differently from the `H.Encaps.Red`, where the final destination (e.g., the service address) is in the inner IPv6 header, in the `H.Insert` case, that address must be included in the segment list. The latter requires removing the SRH at the penultimate segment to make the procedure transparent or if the service is SRv6 unaware. The `H.Insert` behavior is less scalable compared to the `H.Encaps.Red` because the same SRv6 policy (i.e., list of segments) cannot be applied to multiple services. The reason is that in the case of `H.Insert`, the service address must be included in the SRH segment list. In the case of `H.Encaps.Red` instead, using an outer IPv6 header, the service address can be left in the destination address field of the inner IPv6 header. `H.Encaps.Red` allows the exploitation of the same SRv6 policy for all the packets that require the same list of segments.

2.2.3. End

The `End` behavior actions are performed by all the endpoints, and most of the custom behaviors are extensions of this one. When a node receives a packet whose destination address matches with its locally instantiated `End` SID, if the Segment Left field of the SRH is not zero, the node does the following actions. Firstly, it decreases by one the hop limit field of the outermost IPv6 header. Then, it decreases by one the Segment Left by one. Subsequently, it takes the new active segment and sets it as the IPv6 destination address. Finally, it forwards the packet. If, instead, the Segment Left field of the packet is zero, the SRH must be removed, using one of the available flavors. In this work, we exploit the Ultimate Segment Decapsulation (USD).

When a node applying the End behavior finds the Segment Left field equal to zero, it does the following. Firstly, it removes the outer IPv6 header. Then, it forwards the packet based on the inner IPv6 destination address.

3. Related Work

The service continuity problem with workload migration is addressed in the literature at different protocol stack layers [27]. The authors in [24] propose a three-layer solution to guarantee no loss when a virtual machine is migrated within the same cloud data center

to achieve load balancing. They exploit SRv6 to embed source and target servers into pre-established logical paths. They defined three SRv6 behaviors; however, only the routers directly connected with the virtual machine can apply them because they must be able to detect the virtual machine status. Moreover, the approach can be used only in a single data center because the routers must have knowledge of the local network. Finally, the packet buffer cannot be flexibly placed. The work in [24] cannot support service continuity during inter-data center migration, as it relies on local network awareness.

Another network-layer solution is provided by the work in [28], where the Locator/Identifier Separation Protocol (LISP) is used in conjunction with SRv6. The approach allows the managing of the path migration for the Service Function Chains (SFCs) in a distributed edge environment, with the aim of facilitating user mobility. The LISP is exploited to associate a service identifier with its location, introducing additional and unnecessary complexity. SRv6 is indeed a flexible framework able to inherently offer functionalities similar to those provided by the LISP, without requiring a separate mapping for a service locator/identifier. Some works in the literature have extended the SRv6 Network Programming model proposed in [23]. The authors in [25] introduce a new behavior to direct incoming traffic toward the egress node along the path ensuring the highest throughput. The authors in [26] proposed a new SRv6 behavior that enables in-network programming. The new behavior allows the execution of any possible function in eBPF by encoding the latter in a segment. Contributions to SRv6 Network Programming outlined by [25,26] are not applicable to ensuring service continuity during workload migration within an edge computing environment.

For what concerns solutions at the transport layer, the authors in [29] enhance the QUIC protocol on the server side to maintain connections after service relocation; however, they do not preserve the original IP address and therefore need to handle this aspect. Alternatively, service continuity can be ensured at the link layer, as demonstrated in [30,31], where VXLAN and NVGRE are, respectively, employed for this purpose. However, these solutions are not suitable in the distributed edge environment.

The service continuity problem has also been addressed at the application level. The ETSI MEC [11] proposes a solution, also exploited by [32], where the DNS and the client are involved in the service continuity procedure. The client, at each HTTP transaction, must open a new TCP connection and query the DNS to have the service location. In addition, the DNS must notify the client each time the service moves to a different MEC Host. The ETSI MEC [11] provides another solution at the application layer to guarantee service continuity. When a client is able to directly interact with the MEC system, it can be notified of the MEC application instance address by the MEO. This solution, however, requires modifying the client to make the interaction possible.

The authors in [33] propose a solution based on SDN to ensure service continuity in the 5G-MEC scenario. They guarantee the connection persistency by maintaining the original IP address after service relocation. The solution, however, considers a legacy 5G network, where the GTP-U encapsulation is used, and no SRv6 support is provided.

4. Architecture and Procedures for MEC-Based Mobile Services

In this section, we describe a converged 5G and MEC architecture that combines elements from both domains. Our focus is on how these components interact and enable efficient service delivery and their seamless migration at the network edge. We consider a modified version of the 5G MEC architecture proposed in [9]. Our architecture is shown in Figure 3. Using Figure 3 as a reference, the Radio Access Network (RAN) serves as the bridge between user equipments (UEs) and the cellular network. It encompasses base stations (gNBs in 5G terminology) and handles radio-resource management, modulation, and demodulation.

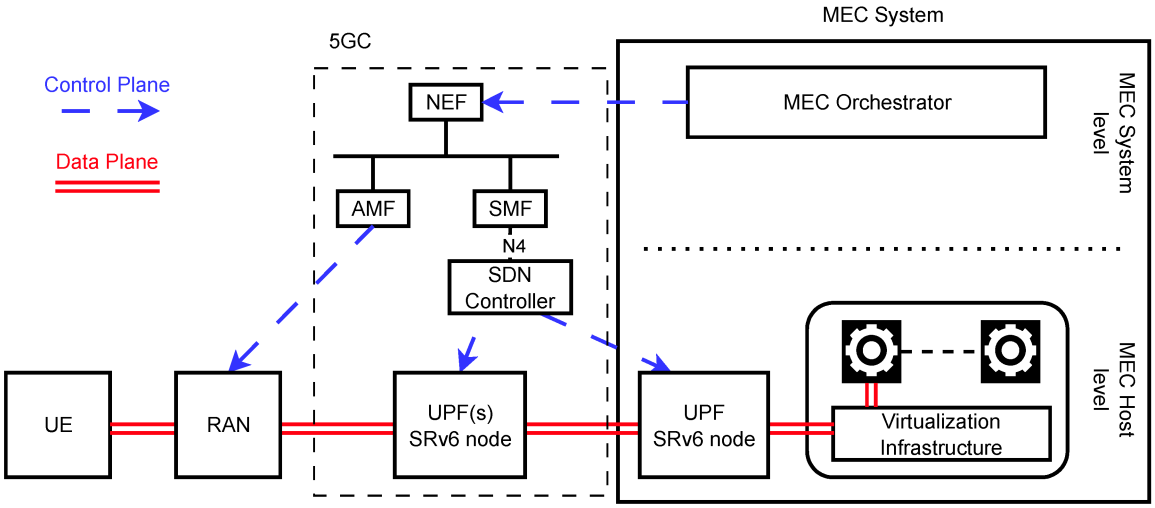


Figure 3. Converged 5G and MEC reference architecture.

The 5GC forms the central part of the 5G system, providing connectivity between the RAN and other data networks. The data plane (or user plane in 5G terminology) is composed of one or more User Plane Functions (UPFs) that process and forward user data to/from external data networks. UPFs can be deployed as part of the external data networks, such as an MEC deployment, although remaining logically a component of the 5GC. In current 5GC architectures, the communication between the RAN and the UPF, and among UPFs, is handled by the GPRS Tunneling Protocol User Plane (GTP-U). However, several studies and standardization efforts from both 3GPP and IETF are proposing SRv6 for substituting GTP-U toward a full-IP 5GC [13,14]. We follow this approach, and we consider the user plane to be composed of SRv6-capable UPFs. We note that additional SRv6 nodes can be deployed between UPFs, e.g., to embed virtualized network functions (VNFs) or for traffic engineering purposes [13]. In the following, we will refer to both these nodes and UPFs as SRv6 nodes. In Section 4.2, we will present the procedures related to configuring SRv6 nodes for forwarding IP packets through the data plane.

The 5GC control plane is composed of several functions controlling and configuring both the core and access network aspects. Each function offers one or more services to other functions in the networking through a service bus, following a so-called Service-Based Architecture (SBA).

From the perspective of this work, the functions of interest are the Access and Mobility Management Function (AMF), Session Management Function (SMF), and the Network Exposure Function (NEF). The AMF handles UE registration and session management and is responsible for user mobility management, authentication, and security. The SMF controls session establishment, modification, and termination, and is responsible for routing data between UEs and external networks. In the considered architecture, the AMF and SMF will be responsible for configuration, respectively, in the gNBs and in SRv6 nodes. As far as the IP packets forwarding is concerned, we will assume the SMF to configure it through a dedicated SDN controller. In more detail, as reported in [34], the SMF sees the SDN controller as a UPF. Communication between the SDN controller and the SMF is enabled via the N4 interface. The SDN controller, instead, configures the forwarding rules on SRv6 nodes. The utilization of an SDN controller allows us to decouple the SMF and the network devices, creating a hybrid SDN network wherein different protocols can be employed for device configuration [35]. Following this approach, the SMF is not required to interact with devices directly; instead, it only needs to communicate with the SDN controllers using a standard 3GPP interface. Finally, the NEF provides APIs for exposing network capabilities to external systems, e.g., letting them request network reconfiguration to the AMF or SMF.

In a typical deployment, both the 5G control plane components and the SDN controller should be replicated so that they do not constitute a single point of failure. They must provide high availability and they must ensure consistency.

Exploiting SDN controllers for routing rules configuration also facilitates the adaptability of the solution for hybrid networks. Each SDN controller is responsible for configuring its network components using network-specific protocols.

The MEC system acts as an external data network that is accessible through the 5GC. It is composed of several MEC Hosts, deployed at the network edge, and providing virtualized resources. MEC Hosts allow the deployment of MEC applications, which provide services for the end user, including, e.g., augmented reality, video analytics, and IoT applications. In the augmented reality domain, a typical scenario where a stateful service is run in edge servers is assisted surgery [36]. Considering the IoT, a mobile application can be used to remotely control sensors and actuators through a service running on an MEC Host. Video analytics applications can be exploited for autonomous driving [37].

Several MEC Hosts are managed by a logically centralized MEC Orchestrator (MEO), which manages the deployment, scaling, and lifecycle of MEC applications across multiple MEC Hosts. It optimizes resource utilization based on user demands and network conditions. In this work, we have not studied policies and algorithms for selecting the service instance in order to optimize the performance and resources. However, considering the growing adoption of a new paradigm, known as computing-aware traffic steering (CATS), in future work, we intend to study orchestration solution matching that new vision. In CATS, networking and computing capabilities and their dynamic status must be jointly taken into consideration to carry out the proper decision in terms of traffic steering and service instance selection [38]. The MEO decisions may require the 5GC to be configured accordingly. For this purpose, the MEC communicates with the AMF and SMF through the NEF.

The integration of the above components may introduce new vulnerabilities in the system, each one inherited by the added elements. In this respect, within the literature, there are several works that describe and solve the possible security threats of 5G, SRv6, MEC, and SDN controllers. The potential risks and typical attacks on SDN networks are comprehensively outlined in [39], which also presents solutions for protecting the network. The work in [40] presents the most common security issues in 5G networks, providing possible countermeasures as well. ETSI MEC vulnerabilities are described in [41], also illustrating the solutions available in the literature. Finally, the authors in [42] deal with security and risks in SRv6 networks.

4.1. IPv6 Addressing Convention and Reference Scenario

We now present a possible allocation of IPv6 addresses for the different entities, which we use throughout the paper. The used addressing convention has been inspired by [43].

The pool `C:0::/32` is reserved for services. The client is allocated the pool `A:0::/32`. The 5GC network devices are allocated addresses from the `B:0::/32` pool, as Table 1 shows. In more detail, the gNB addresses are assigned from `B:0:1::/48`, whereas the addresses of the SRv6 nodes are allocated from `B:0:0::/48`. As suggested by [23], the SID comprises three components: the locator, the function, and the arguments. For a given device, the locator remains consistent across all behaviors and is constructed from the node identifier, e.g., `B:0:0:1::/48` being the locator, each implemented behavior has an address whose first 48 bits are `B:0:0:1::`.

In Section 4.2 and in Section 5, we will employ four distinct behaviors. Two behaviors are taken from [23], whereas two new behaviors, `End.Buffer` and `End.M.GTP6.D.Buffer`, are defined in this work to support the lossless mobility in SRv6-based 5G networks. The End behavior, as defined in [23], is allocated the prefix `B:0:0:<nodeid>:E:/80` and requires no arguments.

Table 1. 5G core devices addressing space.

5G Core Locator or Behavior	Allocated SID Prefix
Locator	B:0:0:<nodeid>::/64
End behavior	B:0:0:<nodeid>:E::/80
End.Buffer behavior	B:0:0:<nodeid>:B::/80
End.M.GTP6.D BSID	B:0:0:<nodeid>:G000:<args>::/96
End.M.GTP6.D.Buffer BSID	B:0:0:<nodeid>:GB00:<args>::/96

Regarding the two new behaviors, the `End.Buffer`, which will be explained in Section 5.2.1, is denoted by `B:0:0:<nodeid>:B::/80`. The `End.M.GTP6.D` behavior, defined in [13] and identified by the BSID `B:0:0:<nodeid>:G000:<args>::/80`, requires arguments to establish the SRv6 policy, i.e., to delineate the list of segments to be encapsulated in the SRH. Finally, the `End.M.GTP6.D.Buffer` behavior, which will be described in Section 5.2.2, is identified by the BSID `B:0:0:<nodeid>:GB00:<args>::/80`. It requires arguments to identify the SRv6 policy, i.e., to understand the list of segments to be encapsulated in the SRH before executing the buffering function.

We conclude this section by illustrating a reference network topology, reported in Figure 4, which is used as a reference in Sections 4.2 and 5 for describing the allocation and migration procedures. A UE can be either a mobile device or an IoT device. A UE is connected to the base station gNB1, which accesses the 5GC network via SR Endpoint1. The service instance of the UE runs on MEC Host1 and is deployed in the form of containers. MEC Host1 is in turn connected to the 5GC through SR Endpoint3. The 5GC control plane, prompted by the MEO, configures the system's routing rules via the AMF and SMF. The SMF communicates with the SDN controller to implement rules in the SRv6 nodes.

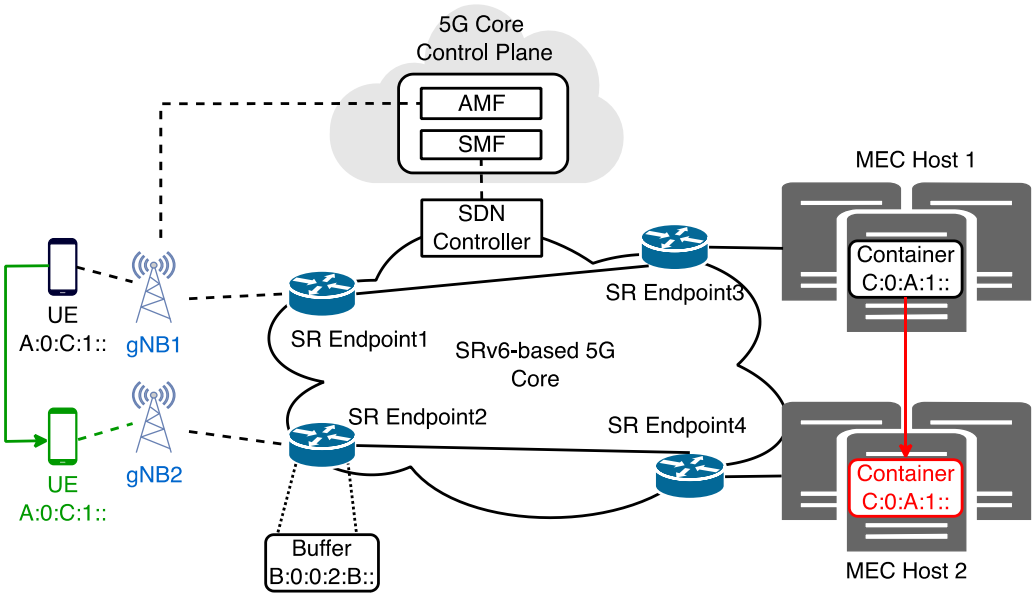


Figure 4. Proposed network topology.

Since the UE has mobility capabilities, a handover, e.g., from gNB1 to gNB2 could occur. Assuming a non-roaming handover, the client retains its original IPv6 address. The MEC architecture provides mechanisms to support service instance migration and information transfer between MEC Hosts [44]. Upon notification from the 5GC regarding the UE mobility, the MEC system can initiate the migration of the corresponding service from MEC Host1 to MEC Host2. The migration in this case is the procedure that allows a running container to be moved from one host to another [45]. Uplink packets traverse the

5GC via SR Endpoint2 and exit through SR Endpoint4. As soon as migration is triggered, a network update to reroute packets toward the new destination is enforced.

4.2. SRv6-Based Path Allocation

Based on the architecture presented so far, we now discuss the configuration of a 5GC network in the case of MEC-based services. The procedures are based on [13], which considers two operation modes: *Enhanced Mode* and *Enhanced Mode with Unchanged gNB Behavior*. The first mode belongs to a full SRv6-aware 5G core network, where the SRv6 domain is extended to the gNBs. The gNBs tunnel the packet across the 5GC network through an IPv6-SRv6 encapsulation. The second concerns the case of a 5G network with SRv6-unaware gNBs. This second mode guarantees the retro compatibility, by instructing the first SRv6 node of the 5GC network, called the Ingress Router, to remove the GTP-U encapsulation and add the SRv6 one. As anticipated in Section 4, GTP-U is a protocol that allows the establishment of a tunnel between the RAN and the 5GC to carry user information. A GTP-U tunnel is uniquely identified by a Tunnel Endpoint Identifier (TEID). Using Figure 4 as a reference, we describe the SRv6-based path allocation procedure in both modes.

The path allocation procedure is triggered by the first request for an edge service by a client. Initially, the MEC system instantiates a container to host the requested service, assigning it an IPv6 address from the appropriate pool. This IPv6 address is retained throughout the container's lifecycle. Hence, any subsequent migration does not alter its assigned address. Subsequently, the MEO triggers the AMF and the SMF to configure routing rules. Due to the characteristics of SRv6, rule configuration is confined to devices positioned at the network edge and any other potential endpoints. Core routers within the 5GC network are preconfigured with necessary rules to allow packet forwarding to SR endpoints. The SIDs format permits aggregation, thus enhancing scalability. It is worth noting that core routers may lack SRv6 capabilities and route the packets toward the endpoint using the IPv6 routing mechanism.

In both modes, the `H.Encaps.Red` behavior is employed. By encapsulating IPv6 packets within an outer IPv6 Header, a uniform SRv6 tunnel can be utilized for all packets necessitating the same path and functions within the 5GC network, increasing scalability. This approach is possible thanks to the exclusion of the container address from the segment list, with the container's IPv6 address placed as the destination address within the inner IPv6 packet. Instead, adopting the inline version of the SRH precludes this capability, as it includes the container's IPv6 address as part of the segments list. Moreover, the use of `H.Encaps.Red` instead of `H.Insert` offers higher security and efficiency, as it avoids the need for modifying the original IPv6 packet. However, our architecture also supports the `H.Insert` method, although in the Enhanced Mode with Unchanged gNB mode, an extension of the `End.M.GTP6.D` behavior is required.

4.2.1. Path Allocation Procedure in Enhanced Mode

The Enhanced Mode considers a full SRv6 5GC network, where the gNB1 undertakes the SRv6 encapsulation process. In such a configuration, the execution of the `End.M.GTP6.D` behavior by an SRv6 endpoint is not needed. The gNB1 must be configured by the AMF to insert the required list of segments to tunnel the packet through the 5GC network and reach the service container running in MEC Host1. We consider a scenario wherein, upon reaching the SR Endpoint3 node, the packet exits the core network and is delivered to the service container.

Figure 5 shows an exemplary packet flow in the data plane. The UE sends a packet to its service instance running on MEC Host 1. When the packet reaches gNB1, the latter applies the `H.Encaps.Red` behavior and forwards it. Given that the segment list consists only of a single segment, the inclusion of the SRH is unnecessary; specifying the destination address within the outer IPv6 header is enough. Subsequently, SR Endpoint3 applies the End behavior, removing the outer IPv6 header before forwarding the packet to the

designated container. A description of the control plane procedures required to configure both the gNB and the SR Endpoints can be found in Appendix A.

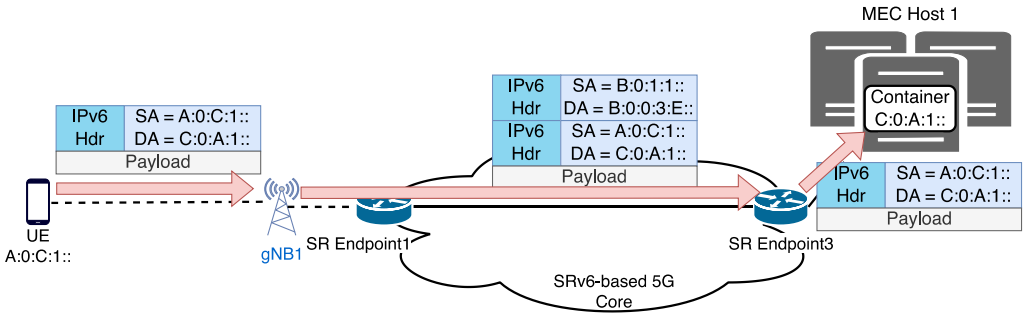


Figure 5. Data plane in Enhanced Mode path allocation.

4.2.2. Path Allocation Procedure in Enhanced Mode with Unchanged gNB Behavior

The Enhanced Mode with Unchanged gNB Behavior characterizes a 5G network configuration wherein the gNB lacks SRv6 capabilities and instead supports only the GTP-U encapsulation of IPv6 packets. The 5GC network can manage SRv6 tunneling. The Enhanced Mode with Unchanged gNB Behavior, as stated in [13], provides a mechanism for the interworking between a 5G legacy gNB and 5G SRv6-enabled core network.

As shown by Figure 6, gNB1 encapsulates the uplink packets into GTP-U. Therefore, SR Endpoint1 is required to substitute the GTP-U encapsulation with the SRv6 one, by applying the `End.M.GTP6.D` behavior.

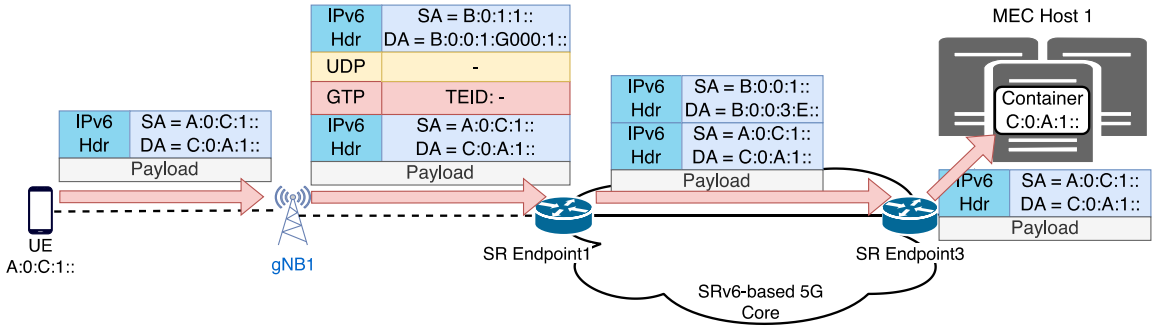


Figure 6. Data plane in Enhanced Mode with Unchanged gNB path allocation.

Figure 6 shows the data plane of the path allocation procedure. The UE sends a packet to its service instance running on MEC Host 1. Since gNB1 is not SRv6 capable, it tunnels the packet through GTP-U encapsulation. gNB1 is configured by the AMF to enter, as the destination address, `B:0:0:1:G000:1::`, so that SR Endpoint1 applies the correspondent policy. SR Endpoint1, upon receiving a packet whose destination address corresponds to `B:0:0:1:G000:1::`, which is a BSID, applies the correspondent SR policy, consisting of the `End.M.GTP6.D` behavior with a preinstalled list of segments. The BSID is an address that identifies, for a node, an SRv6 tunnel between itself and another endpoint. The SR policies can be preconfigured or installed during the path allocation procedure by the SDN controller. Again, since the list would be composed of just one segment, it is not required to add the SRH; placing the only segment in the destination field of the outer IPv6 header is enough. SR Endpoint1 sets as the source address of the outer IPv6 header its locator (`B:0:0:1::`). SR Endpoint3 applies the End behavior with the USD, removing the outer IPv6 header; then, it forwards the packet to the destination container. A description of the control plane procedures required to configure both the gNB and the SR Endpoints can be found in Appendix A.

5. SRv6-Based Lossless Migration

In this section, we first present service migration in the considered 5G-MEC scenario. Then, considering the two operation modes described in Section 4.2, two corresponding migration procedures are explained, providing the control plane and data plane details. The service migration procedure described must guarantee service continuity, i.e., service relocation must remain hidden from the client. Our solution enables making the client not aware of the location of the service, and we ensure that the latter always maintains the same IP address, regardless of the MEC Host where it is running. Moreover, we have to guarantee connection persistence, a lossless workload migration, and preservation of performance experienced by the application. The first one is the Enhanced Mode, i.e., a full 5GC network, whereas the second one is the Enhanced Mode with Unchanged gNB, i.e., the integration of a non-SRv6-enabled gNB with an SRv6 core network. Finally, we describe the `End.Buffer` and `End.M.GTP6.D.Buffer` behaviors, proposed for the Enhanced Mode and Enhanced Mode with Unchanged gNB, respectively.

5.1. Migration Procedure

The high-level idea behind our approach is the following. When the MEC system is notified of a client handover, the MEO may trigger a container migration to guarantee latency requirements. After relocation, the migrated service instance keeps its original IPv6 address, ensuring a transparent migration process for the UE, which remains unaware of the movement. Using Figure 4 again as a reference, we assume that after the client handover from gNB1 to gNB2, the service is migrated from MEC Host1 to MEC Host2. The migration transparency is provided through the reconfiguration of the SRv6 tunnel traversing the 5GC network. The only entities within the network affected by this migration are the ingress edge nodes responsible for encapsulating the packets into an IPv6-SRv6 header and the corresponding egress edge nodes dealing with the decapsulation and final delivery. When the container migrates, a buffering mechanism is required to hold packets during the downtime period [24]. That mechanism will be further explained in Section 5.2. The migration procedure in terms of path reconfiguration is different depending on the mode we are considering, i.e., whether the new gNB is SRv6 capable or not and will rely on different SRv6 behaviors. The first is the `End.Buffer` behavior. It is applied in the case of the full SRv6-based 5G core network. It includes the execution of the `End` behavior and the storing of the processed packet in a buffer. The other is the `End.M.GTP6.D.Buffer` behavior, applied by the SR Endpoint at the ingress of the 5GC network in the SRv6-unaware gNB mode. The behavior implements `End.M.GTP6.D` followed by the storing of the packet in a buffer. Those novel behaviors are extensions of the SRv6 Network Programming framework, introduced to obtain lossless workload migration by integrating the buffering mechanism into the SRH. Therefore, the buffer becomes part of SIDs processing.

Nevertheless, to guarantee no packets are lost during the process, the migration procedure must follow a strict order, reported in Figure 7. Firstly, the MEO selects the node where the buffer will be instantiated and proceeds with its allocation. The node can be either a simple SRv6 Endpoint or a UPF. In both modes, the network node will be in charge of handling the SRH. Then, the MEO triggers the update of the network devices through the SMF, AMF, and SDN controller. That rule update must be performed in reverse order, from the egress edge node (SR Endpoint4) to gNB2. The strict order is mandatory to guarantee lossless migration. Updating the rules on the devices closer to the destination first, on the one hand increases the number of on-the-fly packets, and on the other hand, it permits the recovery of them, drastically reducing the possibility of losses. The rule installed in gNB2 must include the transit through the packet buffer. After the configuration of SR Endpoint2, SR Endpoint3 must also be updated to redirect the on-the-fly packets toward the target MEC Host through the packet buffer. Subsequently, container migration can start. The container is initially checkpointed at source (MEC Host1). After that, its checkpoint image is transferred to the target MEC Host (MEC Host2), and it is finally restored. Upon restoration, the MEO can trigger the buffer flush. Specifically, it lets in-buffer packets be

forwarded toward the destination. At the same time, the routing rules must be updated again to steer new packets directly toward the destination without traversing the buffer.

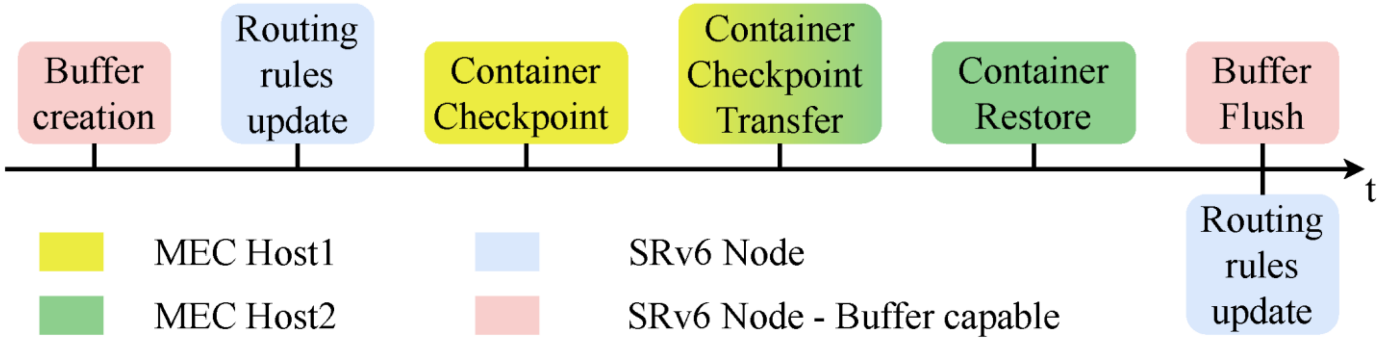


Figure 7. The proposed migration timeline.

The total time of the migration procedure is influenced by multiple factors, e.g., the SDN controller location, the topological distance between the source and target MEC Hosts, the network conditions, and the size of the service instance's checkpoint to be transferred. The container migration time is composed of three phases: the checkpoint, the transfer, and the restore. The work in [45] demonstrates that in a real-world scenario, the time required for container migration is the component that most contributes to the total delay, being of the order of tens of seconds in the worst case.

For linearity and completeness reasons, in the description of the two procedures, we place the buffer on the ingress edge node, SR Endpoint2. The rule update procedures are explained considering the buffer has already been instantiated.

5.1.1. Enhanced Mode

The Enhanced Mode, characterized by comprehensive SRv6 encapsulation, requires the configuration of routing rules only on gNB2, SR Endpoint4, and SR Endpoint2. gNB2 is in charge of encapsulating the packets in an IPv6-SRv6 header. During the migration interval, gNB2 must include in the segment list the SID identifying the `End.Buffer` behavior for SR Endpoint2.

Figure 8 shows the control plane procedure for path migration. The MEO configures gNB2 to apply the `H.Encaps.Red`. gNB2 includes within the SRH, the SR Endpoint4's End SID and sets the SR Endpoint2's `End.Buffer` SID as the destination address of the IPv6 outer header. Firstly, the MEO asks the SMF to activate the SDN controller for SR Endpoint4 configuration. The latter requires a rule to forward the packets to the designed container once decapsulated. Subsequently, the MEO prompts the AMF to configure the `H.Encaps.Red` encapsulation rule on gNB2, as just described. At the same time, the MEO asks the SMF to trigger the SDN controller for configuring the `H.Encaps.Red` rule on SR Endpoint2 to redirect the on-the-fly packets. The latter rule must include the SR Endpoint2's `End.Buffer` SID.

Once the container is restored on MEC Host2, a new network configuration is required. The buffer is not required anymore, and the packets can be forwarded directly to the service instance. The MEO, in this phase, can trigger all the following actions concurrently. The `H.Encaps.Red` rule must be updated on gNB2, to encapsulate the packet. At the same time, the MEO prompts the SMF and the SDN controller to update the backup rule on SR Endpoint3. Concurrently, the `H.Encaps.Red` rule on gNB1 can be removed. Finally, the MEO triggers the buffer flush.

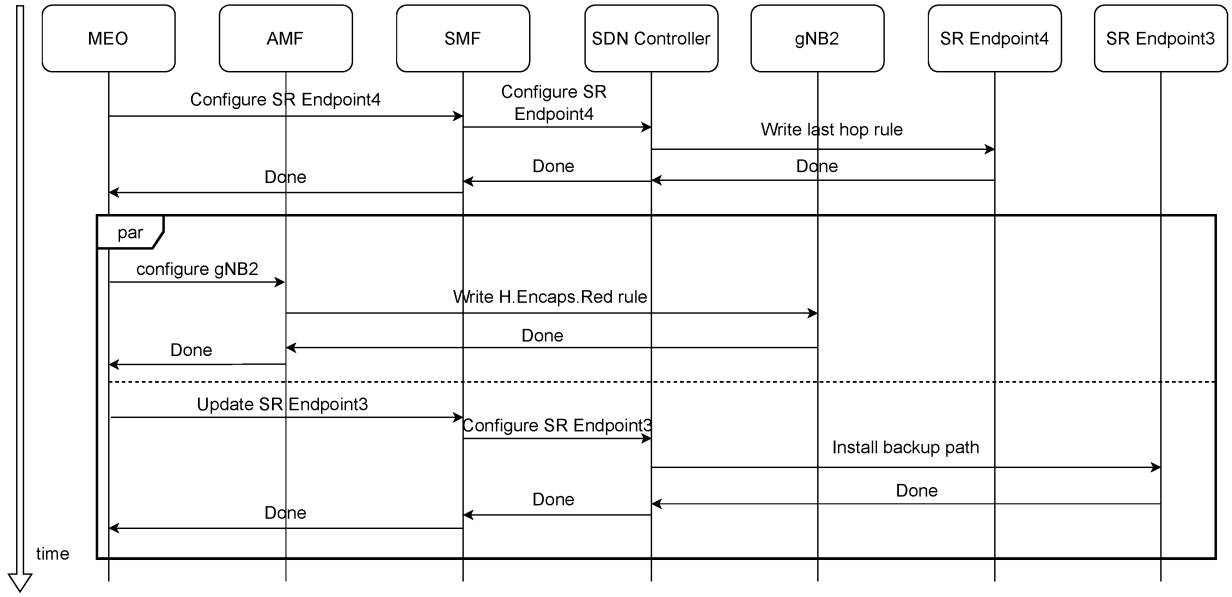


Figure 8. Control plane in Enhanced Mode path migration with buffering.

Figure 9 delineates a data plane example during service migration, which has been triggered by the MEO after the client handover. It illustrates the data plane downtime interval. Figure 9 shows the UE sending a packet to its service instance, whose transfer is in progress. When it reaches gNB2, the latter applies the `H.Encaps.Red` behavior and forwards the packet. gNB2 inserts the SRH; Figure 9 shows, in the SRH, the list of the segments (`B:0:0:4:E::`) and the index of the active segment (Segment Left field). The packet reaches SR Endpoint2, which implements the `End.Buffer` behavior. After that, the packet with the updated outer header is forwarded to SR Endpoint3. SR Endpoint3 applies the `End` behavior with the USD, removing the outer IPv6-SRv6 header. Then, it forwards the packet to the destination container.

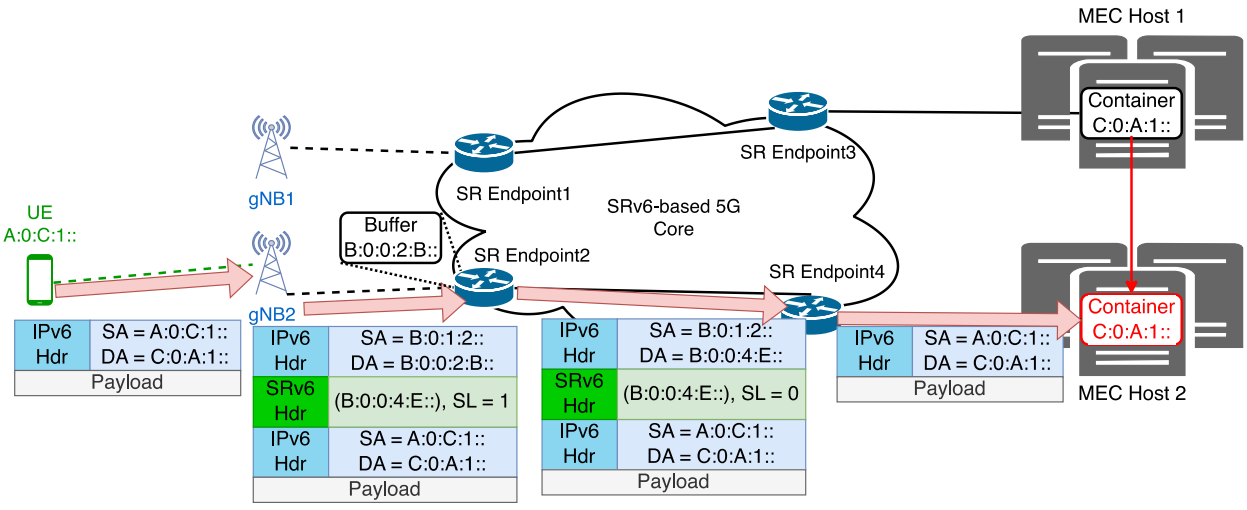


Figure 9. Data plane in Enhanced Mode path migration with buffering.

5.1.2. Enhanced Mode with Unchanged gNB GTP-U Behavior

To provide lossless migration in the interworking between GTP-U and SRv6, the gNB2 must set a BSID as the destination of the outer IPv6 header. This BSID identifies, within SR Endpoint2, a policy implementing the `End.M.GTP6.D.Buffer` behavior.

Figure 10 shows the control plane procedure for path migration. Firstly, the MEO asks the SMF to trigger the SDN controller for SR Endpoint4 configuration. The latter requires

a rule to forward the packets after the SRH decapsulation. Then, the policy to be applied by SR Endpoint2 is selected, and the correspondent IPv6 address and actions must be installed on SR Endpoint2, if not performed yet. After that, the MEO must configure gNB2 to insert the `B:0:0:2:GB00::` address, which is the SR Endpoint2's BSID identifying an `End.M.GTP6.D.Buffer` policy. The determination of the BSID to be used is based on both the function along the path to be executed and the tunnel which the packet needs to traverse. This means that, in the scenario where multiple flows enter and exit the core network through the same nodes and use the same buffer instance, a single BSID can be exploited. Concurrently, the MEO must trigger the configuration of a backup path on SR Endpoint3; the latter, to redirect on-the-fly packets, must apply the `H.Encaps.Red`. It includes the SR Endpoint4's End SID in the segment list and it sets, as the outer IPv6 destination, the SR Endpoint2's `End.Buffer` SID. Once the container is restored on MEC Host2, a rule must be updated again to remove the buffering phase. At this point, the buffer is not required anymore, so the packets can be forwarded directly to the designed instance. The MEO, in this phase, can trigger all the following actions concurrently. Given the policy to be applied by SR Endpoint2, the MEO requires the SMF to activate the SDN controller for configuring the former on SR Endpoint2. That policy, identified by address `B:0:0:2:G000::`, consists of the application of `End.M.GTP6.D` behavior to tunnel the packet toward the destination through SR Endpoint4. The MEO also prompts the update of the GTP-U encapsulation rule on gNB2. gNB2 must insert `B:0:0:2:G000::` as the outer IPv6 destination address. At the same time, the MEO triggers the SMF and the SDN controller to update the backup rule on SR Endpoint3. Moreover, the GTP-U rule on gNB1 can be removed. Finally, the MEO triggers the buffer flush.

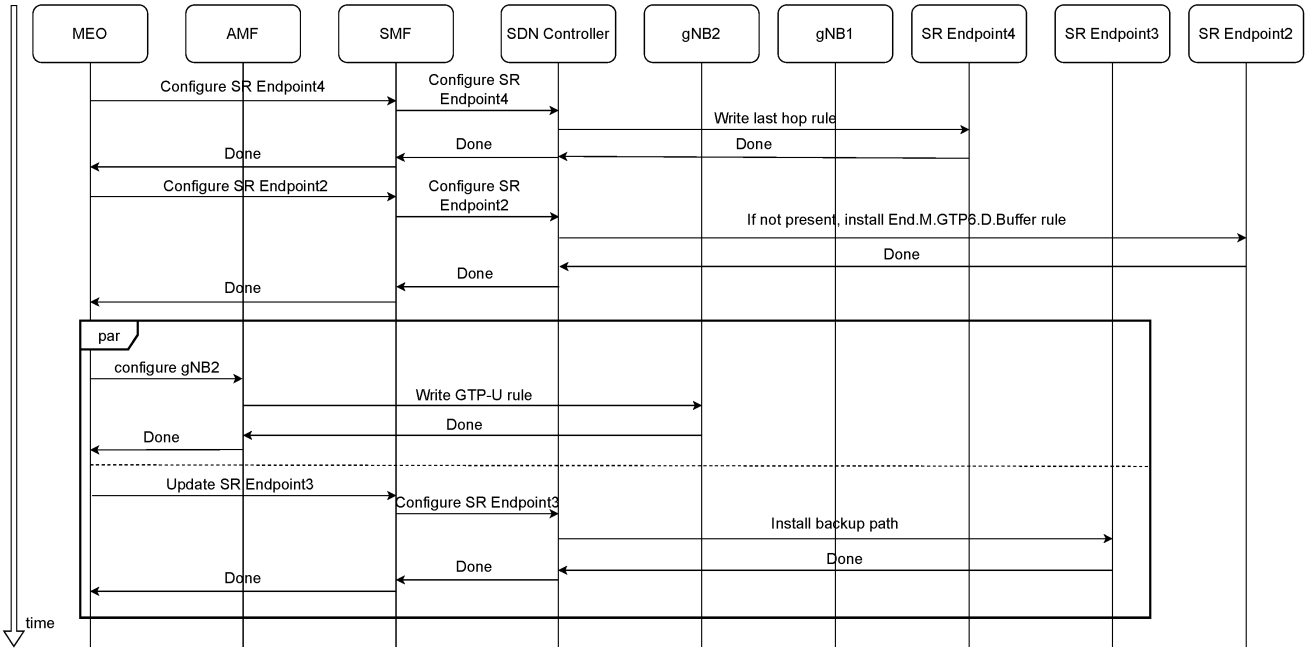


Figure 10. Control plane in Enhanced Mode with Unchanged gNB path migration with buffering.

Figure 11 shows the data plane example for path migration during the downtime. It illustrates the UE sending a packet to its service instance, while the transfer is in progress. When gNB2 receives the packet, it encapsulates that packet in GTP-U, by setting `B:0:0:2:GB00:2::` as the destination address. Once the packet reaches SR Endpoint2 with the latter `End.M.GTP6.D.Buffer` SID, the behavior is executed. Subsequently, the packet with the updated outer header is forwarded to SR Endpoint4. SR Endpoint4 applies the `End` behavior with the USD, removing the outer IPv6-SRv6 header. Then, it forwards the packet to the destination container.

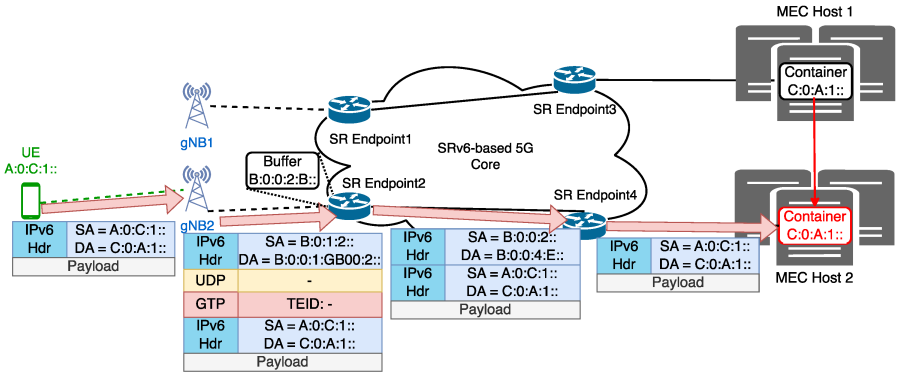


Figure 11. Data plane in Enhanced Mode with Unchanged gNB path migration with buffering.

5.2. Supporting Buffering and SRv6 Behavior

To guarantee service continuity in the case of workload migration, we leverage both standard SRv6 behaviors and newly defined behaviors. Concerning the standard behaviors, we exploit the `H.Encaps.Red`, the `End` with the USD defined in [23] and the `End.M.GTP6.D` defined in [13]. Taking advantage of the programmability and flexibility of SRv6, we extend the list of behaviors by the definition of custom ones: `End.Buffer` and `End.M.GTP6.D.Buffer`.

Both behaviors update the packet headers and store them in the buffer. When the packet flush is triggered, the stored packets are submitted to the IPv6 routing table hosted on the device. We highlight that the packet can be immediately forwarded after release from the buffer without the need for any additional modification. This approach speeds up packet delivery and reduces the overhead on the device. As a final remark, note that the packet buffer can be used to handle packets that are targeted at different services; all the information to reach the service is embedded in the packets.

5.2.1. END.Buffer Behavior

The `End.Buffer` behavior is an extension of the `End` behavior with the USD described in [23]. When a packet arrives to a node with a destination address matching a node's `End.Buffer` SID, it is first checked whether the current destination address of the outer IPv6 header is the last segment of the SRv6 list. In that case, the packet is decapsulated. If, instead, the SRH is still needed, the packet is updated as follows. The IPv6 outer header Hop Limit field and the Segment Left field of the SRH are decreased by 1. Finally, the IPv6 outer header destination address is updated with the current active segment. Whether the packet has been decapsulated or not, the packet is placed in the buffer, ready to be forwarded without further modifications once the buffer flush is triggered.

Figure 12 reports an example of the `End.Buffer` behavior, used in the mode described in Section 5.1.1. The incoming packet (on the left) has the `End.Buffer` SID of the SR Endpoint2 router as the destination address of the outer IPv6 Header. Firstly, the behavior modifies the SRH outer IPv6 header. It decreases the Segment Left field. Then, it sets the IPv6 destination address equal to the active segment. Subsequently, it decreases the Hop Limit header field. Finally, the resulting packet is placed in the buffer.

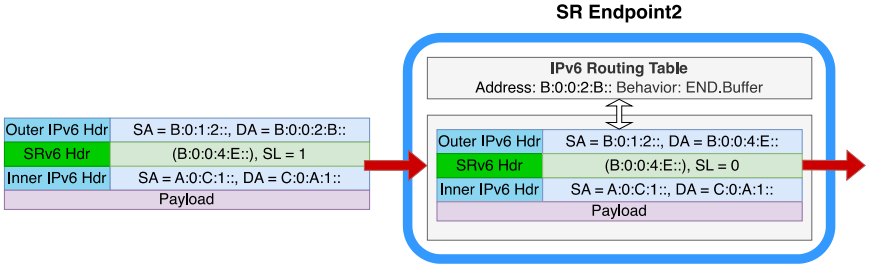


Figure 12. The End.Buffer behavior.

5.2.2. End.M.GTP6.D.Buffer

The `End.M.GTP6.D.Buffer` behavior is an extension of the `End.M.GTP6.D` behavior described in [13]. When a packet arrives to a node with a destination address matching one of the node's `End.M.GTP6.D` SIDs, firstly, the outer IPv6 headers are inspected to determine whether the subsequent headers correspond to UDP and GTP-U. After the verification, GTP-U, UDP, and the outer IPv6 headers are extracted. Next, the packet is encapsulated into a new outer IPv6 header with the SR as the extension header. The list of segments put in the SRH depends on the `End.M.GTP6.D` SID received, which identifies the policy. The SIDs' list included in the SRH is missing the first segment because `H.Encaps.Red` is used. Subsequently, the source address field of the outer Ipv6 header is set to the current node's locator. The destination address field instead is the first SID of the segment list that was not included in the SRH. Finally, the other Ipv6 header fields are properly set. At this point, the packet is placed in the buffer, ready to be forwarded without further modifications. Figure 13 reports an example of the `End.M.GTP6.D.Buffer` behavior in the mode described in Section 5.1.2. The incoming packet (on the left) is addressed to an `End.M.GTP6.D.Buffer` SID of the router SR Endpoint2. SR Endpoint2 applies the `End.M.GTP6.D.Buffer`, which embeds the following operations. Firstly, it removes the outer Ipv6 header and the subsequent UDP and GTP-U headers. Then, it encapsulates the packet into another Ipv6 outer header (`H.Encaps.Red`). The segment list would be composed of just one segment, so the addition of the SRH can be avoided; it is enough to set the `End` SID of the SR Endpoint4 node (`B:0:0:4:E::`) as the destination address of the outer IPv6. At this point, the packet can be placed in the buffer, ready to be forwarded when the buffer flush is triggered.

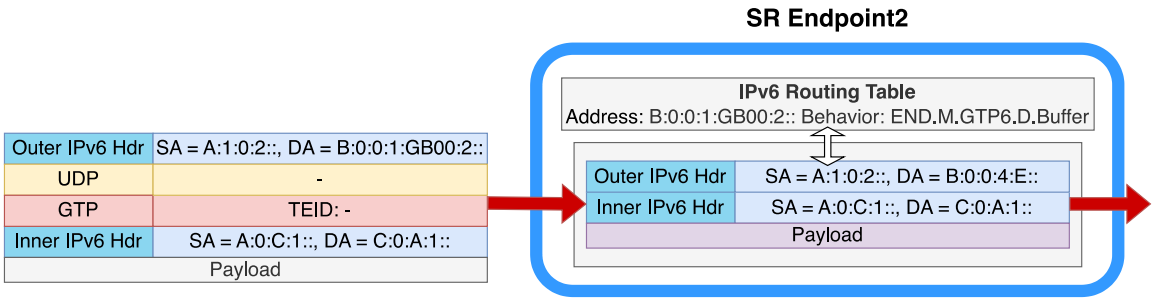


Figure 13. The End.M.GTP6.D.Buffer behavior.

6. Testbed and Performance Evaluation

In this section, we first present the implementation of our solution in the scenario of a full 5G network, where the gNB is responsible for the IPv6-SRv6 encapsulation. Then, we describe a performance evaluation of the proposed solution, comprising a scalability analysis, an evaluation of the performance in static conditions, and the results in a scenario with container migration.

6.1. Implementation and Testbed Deployment

The proposed solution was evaluated through a small-scale PoC, comprising an Open Network Operating System (ONOS) SDN controller, a simplified data plane composed of Stratum BMv2 switches emulated through Mininet, and three MEC Hosts running Docker as the virtualization technology.

6.1.1. Data Plane and ONOS SDN Controller

The ONOS project in [46] provides a $P4_{16}$ (P4 version dated to 2016) implementation of an SRv6 network. We extended the P4 code for the BMv2 switch architecture by implementing the `End.Buffer` behavior, which is used to handle packets that must be placed in the buffer during the migration procedure. In addition, we modified the table responsible for entering the SRH; in particular, we defined different actions based on the number of

segments to be included. The latter modification is required due to the P4's lack of support for loops.

Moreover, we implemented two ONOS applications, used, respectively, for interacting with the SRv6 nodes and higher-level services, i.e., the SMF/MEO in our architecture. The first one is an ONOS core application that interacts with the BMv2 switches. It injects the processing pipeline and fills their tables. In addition, it handles the packet in and packet out messages. The application is in charge of managing all the device tables, configuring both the standard tables and those related to SRv6 behaviors. Our contribution concerns the implementation of those functions in charge of managing the SRv6 tables and actions; in more detail, those for including the SRH with the segment list, those for implementing the `End` and `End.Buffer` behavior, and, finally, the action in charge of removing the SRH. The ONOS application, by default, configures all network devices with routing rules for each possible SID in the network. In order to exploit the scalability characteristic of SRv6, we modified the part of the code managing the IPv6 routing tables in such a way that the ONOS distributes only the routes for the edge nodes' SIDs.

The second application exposes custom REST APIs to external services. This application allows the interaction with the ONOS controller, which is needed to request the ONOS to inject a network configuration into the BMv2 switches. The requests supported by the application include the configuration of specific SR paths and IPv6 routing rules. The application is reachable through a URI, and the requests must contain a JSON object specifying the operations to be performed and their parameters, e.g., installing SRv6 paths requires the ONOS identifiers of the ingress and egress edge nodes. The application then parses the request and invokes the proper core app function.

The core network is emulated using Mininet [47]. We used different networks to evaluate the performance of our solution, all composed of Stratum BMv2 switches connected in leaf–spine topology. The implemented networks differ in the number of core layers. Each node at the border of the Mininet-emulated network needs to have at least one port attached to a different Mininet container interface. This is necessary to let such nodes exchange packets with external hosts, namely, clients and MEC Hosts.

We note that our implementation focuses only on emulating the data plane of the 5GC, leaving aside the RAN aspects. In this respect, our network includes simplified gNBs, which receive IPv6 packets from clients and encapsulate them using SRv6, following the Enhanced Mode. The implementation of the RAN is out of the scope of this work, which focuses more on managing the 5GC network in order to guarantee lossless service continuity.

6.1.2. Edge Service Migration

Edge services run as Docker containers. For what concerns their migration, we leveraged CRIU [48] and rsync [49] tools. CRIU, which stands for Checkpoint/Restore in Userspace, deals with the checkpoint of the container's status on its original host and its restoration on the target host. We exploited CRIU's `--tcp-established` option to guarantee TCP connection persistence after migration [50]. This allows CRIU to obtain the support of Linux Kernel to retrieve and restore the TCP connection status, i.e., it saves the socket state and restores it at the destination. In addition, before the connection is restored on the target host, the original container's IP address must be available. To guarantee the latter requirement, the same network namespace must be created on the target host. The rsync tool is used to transfer the container and connection checkpoints to the target host.

6.1.3. Testbed

We deployed a test environment comprising the six devices shown in Figure 14. Firstly, the Qotom mini-PC runs the system orchestrator and the Mininet container, which emulates the core networks of the BMv2 switches. Further information about the network topologies will be given in Section 6.2. Four additional Qotom mini-PCs are also used as follows: one is the client, while the others are MEC Hosts. Two of them are the source and target of the container migration, whereas the third one is used to run the buffer function in some of

the experiments. The last device is a PC server running the ONOS controller. The Qotom mini-PCs have a Quad-core Intel i7-4600U CPU, 8 GB RAM, Ubuntu 18.04; whereas the PC server has an Octa-core Intel i7-3770 CPU, 16 GB RAM, Ubuntu 18.04.

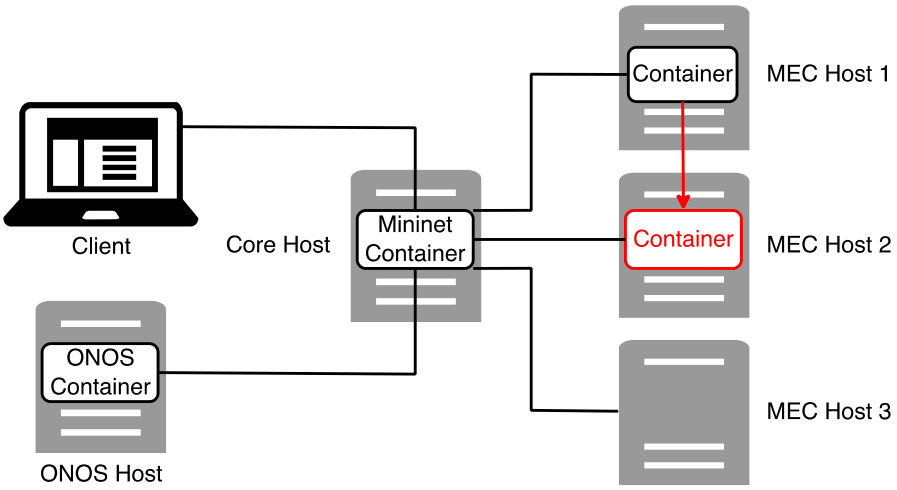


Figure 14. Proposed high-level testbed architecture.

The ONOS configures the nodes emulated in Mininet out-of-band. To simulate the in-band network with consistent network delays, we used Linux tc- netem. In particular, we set a 1 ms one-way delay for each traversed link in the core network.

6.2. Performance Evaluation

We evaluated our solution in terms of scalability and flexibility. Firstly, we present a scalability analysis where we compared our SRv6 solution against a non-SRv6 one in terms of the configuration times and number of rules to be installed. Subsequently, we show the performance of our solution, comparing it with a non-SRv6 one in terms of the round-trip time in static conditions. Finally, we illustrate the results of the analysis in dynamic conditions, i.e., when the MEO triggers container migration.

As stated in the testbed description, for our evaluation, we used different network topologies. All of them are leaf–spine topologies, which differ in terms of the number of core layers. Specifically, we used three different networks. The smallest one is composed of one core layer, for a total of one core node. Another network is composed of two core layers, with three nodes each. The biggest network is composed of four core layers, for a total of twelve core nodes.

6.2.1. Scalability Analysis

Firstly, we conducted measurements on the time required to install rules on network devices for both path allocation and path migration. In our setup, the ONOS sends write operations in a specific order. Operations directed toward a particular switch are not mixed with those for other switches. Furthermore, these operations are sent sequentially, and the ONOS does not wait for one operation to finish before sending the next one. We evaluated scalability across the network topologies explained above. Additionally, we compared our SRv6-based solution with a non-SRv6 one, relying on the standard IPv6 routing mechanism. In the latter mechanism, each router forwards packets solely based on the IPv6 address of either the client or the server.

Figure 15 demonstrates the consistent superiority of our solution over the non-SRv6 approach; the most significant disparities are observed in the case of the path allocation procedure. Specifically, under the non-SRv6 method, the configuration of all core network devices is necessary; thus, the effect of the number of core layers is evident. Our approach, instead, needs the configuration of only the ingress and egress edge nodes. However, the number of core layers influences the outcomes because its increase results in the ONOS

being topologically more distant from the egress edge device. The higher the distance between the ONOS and the target device, the higher the latency for the rule installation. Regarding path migration, discrepancies between the two methods are less pronounced, as a similar number of devices necessitate reconfiguration in both cases. The characteristics of the network and the positions of the source and target MEC Hosts with respect to the client impact the number of devices to be reconfigured in the non-SRv6 approach. As stated previously, the distance between the ONOS controller and the target devices impacts the rule installation time. Moreover, the ONOS controller sends the requests sequentially without waiting for a response. Based on that, the outcomes in Figure 15 are influenced by the order in which the ONOS sends the operations. We now analyze the worst-case scenario, i.e., where the ONOS sends the last write operations to the farthest device, which also has the longest processing time. Figure 16 shows the worst-case rules installation overhead across all the involved switches, decomposed to underlying the different contributions. Assuming J to be the number of network nodes in the system and the number of operations sent to the last node, we define the time at which the ONOS sends operation i to node j as s_{ij}. Then, we define r_{ij} as the time at which that operation enters its destination. Finally, d_{ij} is the processing time of the above-mentioned operation. We suppose that operations i and $i+1$ addressed to node j are separated, both at transmission and reception by an interval τ_{ij}. Subsequently, ρ_{ij} is the time required by node j to process operation i. To conclude, we suppose that, once the node terminates the execution of operation i, the subsequent operation is already available, as stated by the following formula:

$$\tau_{ij} \leq \rho_{ij} \, \forall \, i, \, j. \tag{1}$$

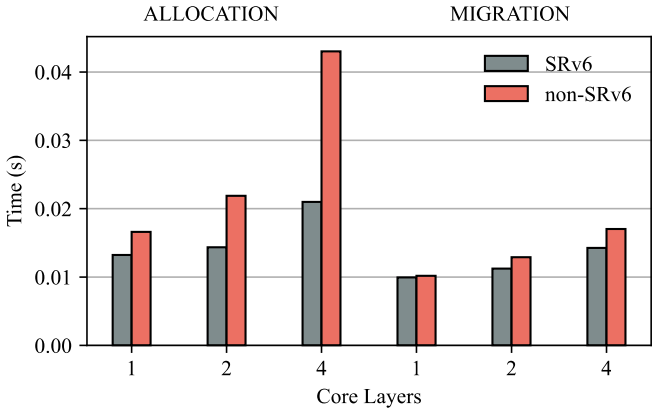


Figure 15. Network nodes installation overhead.

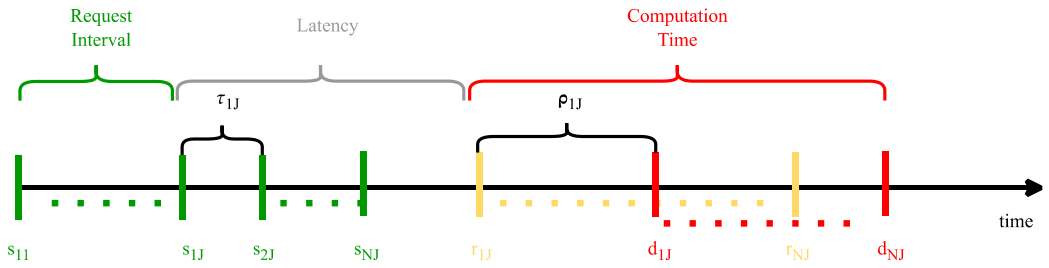


Figure 16. Worst-case network configuration timeline.

The decomposition of the rule installation overhead is also shown in Figure 17, where the worst-case scenario is considered. The first component, the green one, is the interval of the time between the dispatch of the first instruction to the first node and the dispatch of the first operation to the last node, called the request interval. Then, we have the latency, which is the time between the dispatch and delivery of the first instruction to the last node.

Finally, we have the computation time, which is the interval of the time required by the last and slowest node to satisfy all the received requests. Formally, this is defined as

$$c = \sum_{i=1}^{N} \rho_{ij}. \tag{2}$$

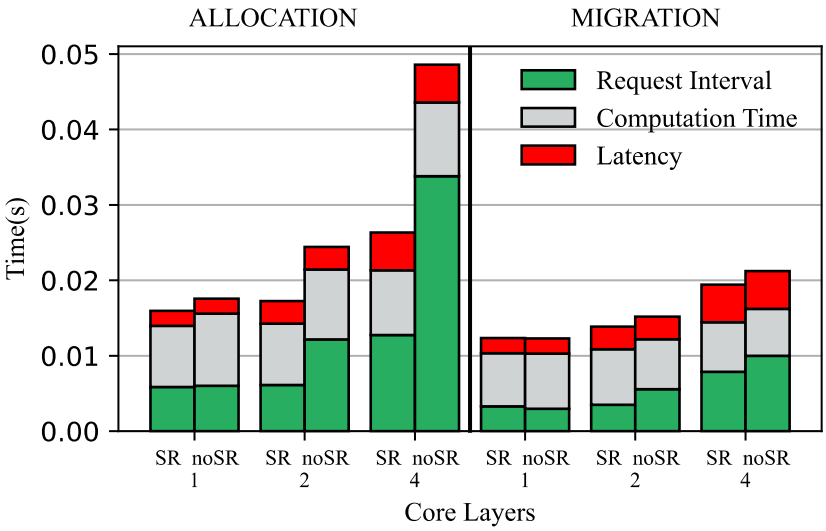


Figure 17. Worst-case network configuration overhead.

Figure 17 confirms that the most significant difference in terms of the rule installation time between the SRv6 and non-SRv6 approaches is given by the request interval. This is due to the fact that in the solution without SRv6, it is necessary to configure a much larger number of devices. The other components, transmission time, and processing time of a request are similar in both cases, as expected.

In the previous experiments, we analyzed the scalability in terms of the rule installation time for a single client–server pair. We now consider the scalability in terms of the number of write operations to be performed, within a scenario with multiple client–server pairs. In our SRv6-based solution, the ONOS must configure only the ingress and egress edge routers, maintaining the number of write operations constant despite the depth of the core network. We define C as the number of clients, ES as the number of edge nodes receiving SRv6 rules to be installed, and EI as the number of edge nodes receiving IPv6 rules to be written. The number of operations to be performed in the SRv6-based case, SR, for both the initial configuration and the migration update are the following:

$$SR = C \times (ES + EI). \tag{3}$$

In the case of the initial path allocation, both ES and EI are equal to 1. In the case of path migration, we have ES equal to 2 and EI equal to 1. As a result, we have a complexity in both cases of $\mathcal{O}(C)$.

Considering the non-SRv6 approach instead, all the core nodes must be configured in addition to the edge ones. The number of core layers in the latter case significantly impact the result. Exploiting the above variables and defining K as the number of core layers, the number of operations for the allocation procedure, $noSR_A$, are the following:

$$noSR_A = C \times (2 \times K + 2 \times EI). \tag{4}$$

The complexity of the path allocation procedure, using the non-SRv6 approach, is $\mathcal{O}(K \times C)$. For what concerns the path migration procedure, identifying the number of core devices that must receive a rule update as $\overline{K}$, the number of required write operations, $noSR_M$, are defined as

$$noSR_M = CU \times \left(\overline{K} + EI\right). \tag{5}$$

Therefore, the complexity of the path migration procedure in the non-SRv6 scenario is $\mathcal{O}(\overline{K} \times C)$.

6.2.2. Analysis in Static Conditions

We analyzed the performance in terms of the request–response time. We measured the request–response time of a TCP-based client–server application as in [24], consisting of a client that sends 100 B requests with a rate of 0.5 s, and a server sending an echo response. For the comparison, we considered the non-SRv6 approach as well as two different SRv6 flavors: the first, called *Loose SRv6*, introduces only the required segments, which is the egress edge node `End` SID. The second one, called *Hop-by-Hop SRv6*, includes in the SRH the segments of all the traversed devices within the core network. In Figure 18, we show the results of the request–response time over 50 repetitions when no migration is involved. As shown, the adoption of the Loose SRv6 approach does not worsen the performance with respect to the standard IPv6 routing mechanism. Hop-by-hop SRv6 flavor, instead, does not scale with the number of hops. What affects the performance, making the approach less efficient from a latency point of view, is the bigger SRH that must be processed by each node within the network. Therefore, if there are no reasons necessitating a hop-by-hop approach, such as in the case of traffic engineering, it is recommended to use the Loose SRv6 flavor.

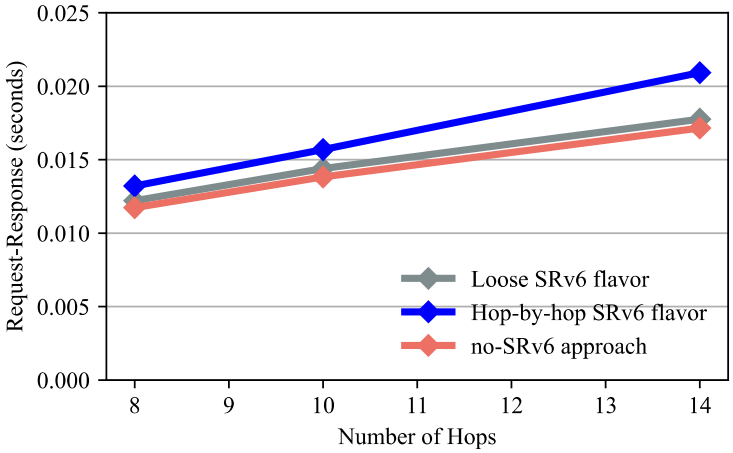


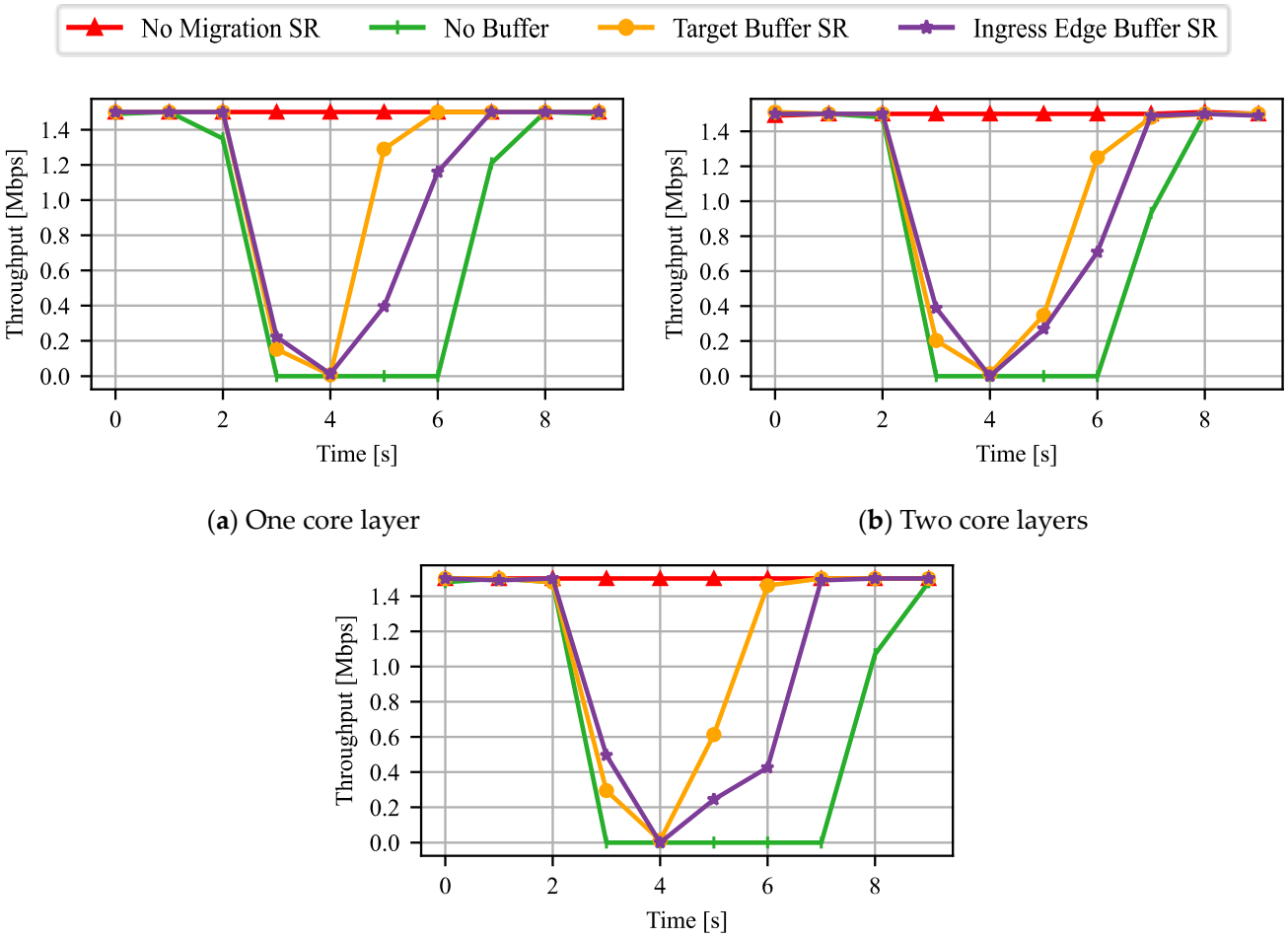
Figure 18. Mean request–response time.

6.2.3. Analysis in Dynamic Conditions

We evaluated our approach in time intervals within which the container is migrated. We considered four different scenarios. The first, *No Migration SR*, is the baseline, where the container is not migrated. The second, *No Buffer SR*, migrates the container without any packet buffer. The third, *Target Buffer SR*, uses a packet buffer instantiated on the UPF within the target MEC Host. The last, *Ingress Edge Buffer SR*, involves a packet buffer executed on the UPF placed within an MEC Host connected to the ingress edge node. In all the scenarios, SRv6 is used to steer the packets.

Figure 19 shows the throughput of an iperf3 application, measured at the server side, over a 10 s time interval including migration. The client sent 5 MB to the server with a bandwidth limited to 1.5 Mbps. The throughput is measured for each of the network topologies described previously. The considered server container required, on average, 3.19 s for migration. Figure 19 confirms the throughput stability of around 1.5 Mbps in static conditions and shows similar performances in the other three scenarios. They have a similar throughput drop during the container downtime, despite the considered topology. The main difference is the time required by the throughput to go back to its original stable value. The *No Buffer* scenario takes more compared to the buffering cases, and the difference increases with the network depth. The explanation for this outcome is based on the necessity to retransmit lost packets, requiring them to traverse the entire

network before reaching their target destination. To conclude, Figure 19 shows the slight superiority of the *Target Buffer* compared to the *Ingress Edge Buffer* despite the topologies.



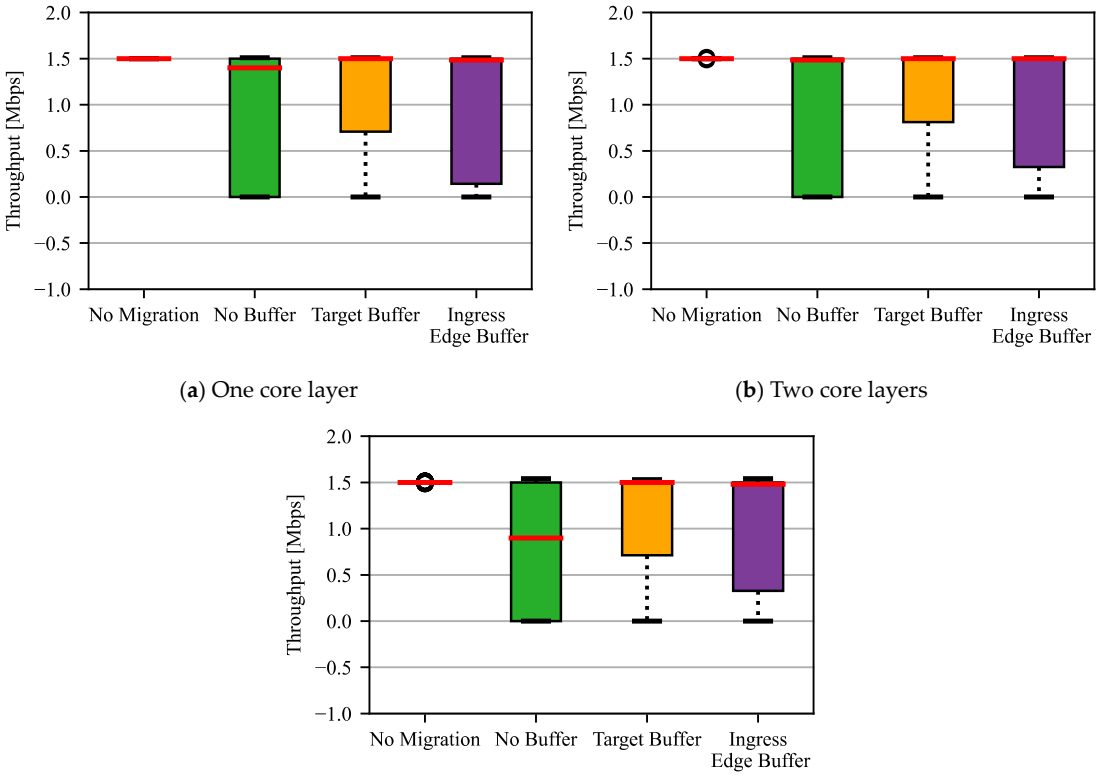
(**a**) One core layer

(**b**) Two core layers

(**c**) Four core layers

Figure 19. Throughput within a time interval containing migration.

Figure 20 shows the boxplot of the throughput in the four scenarios listed above, using the same network topologies. Data are taken from a 10 s time interval containing migration, and they have been aggregated. The boxplot graphs appear as expected, following the conclusions taken from those of Figure 19. In the *No Migration* scenario, the box is collapsed in a line, corresponding to the median. This is due to the throughput constant value of around 1.5 Mbps. The *No Buffer* case, instead, has the wider box, due to a larger number of values equal to zero and to the slower throughput increment after container restoration. The reason is the packet retransmission. The buffering cases have a better performance compared to the *No Buffer* one. Their throughput increases faster, with the *Target Buffer* exceeding the *Ingress Edge Buffer*. Finally, as shown in Figure 21, we analyzed the performance in terms of the request–response time in a dynamic scenario, using the TCP-based client–server application already exploited in Section 6.2.2. In this case, the container migration procedure lasts 3.28 s, on average. As shown, the *No Migration* scenario is the best, as expected. Concerning the migration cases, those approaches with buffer outperform the one without it, and they have a similar performance irrespective of the topology. The use of the buffer, enabling the prevention of packet losses, allows having the increase in the request–response time as close as possible to the container downtime, in the worst case.

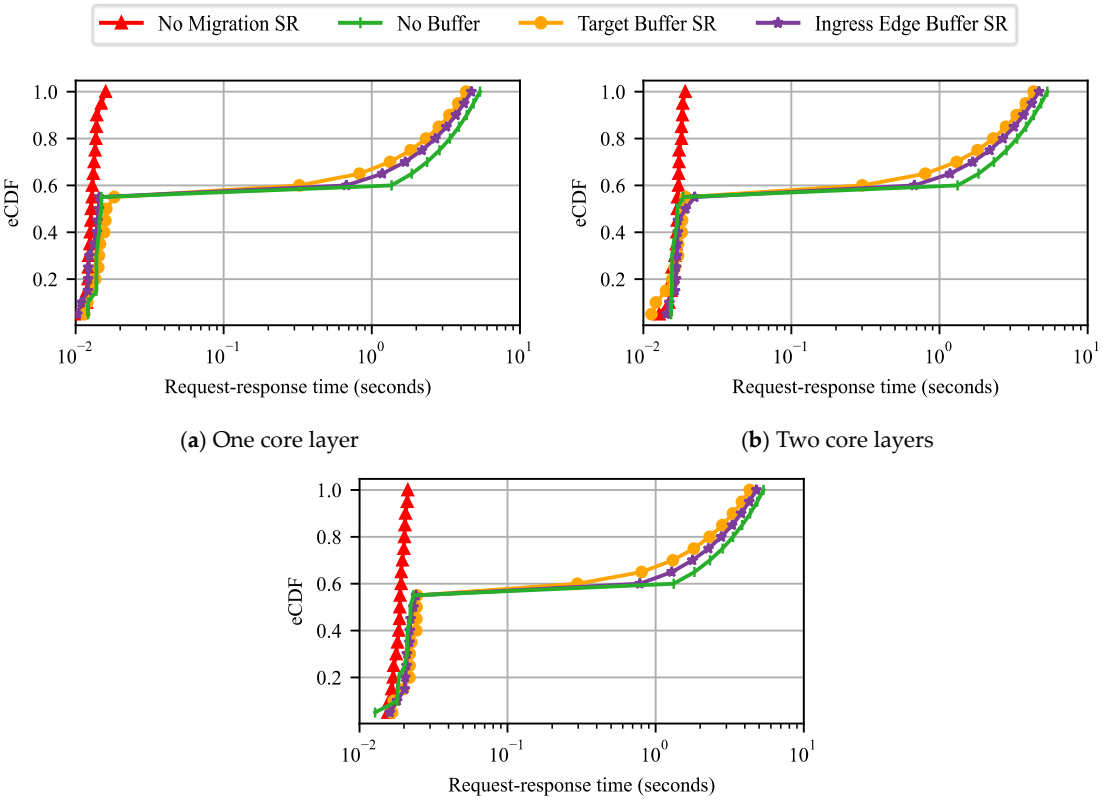


(**a**) One core layer

(**b**) Two core layers

(**c**) Four core layers

Figure 20. Boxplot of the throughput within a time interval containing migration.



(**a**) One core layer

(**b**) Two core layers

(**c**) Four core layers

Figure 21. eCDF of the request–response time.

The conducted experiments demonstrate that our proposed solution guarantees service continuity in the presence of edge service migration, as per the definition provided in Section 1. Specifically, we verified that our approach allows a client to transparently reach its edge service after migration by using the very same IP address and while leveraging the original TCP connection. Moreover, the graphs show how our solution allows the throughput to be the same before and after service migration. Furthermore, our buffer-based approach ensures the throughput increases more quickly than a solution without buffer, as in the former case, no packet is lost.

7. Conclusions

In the ETSI MEC environment, edge service migration may be required to support user mobility. Leveraging the SRv6-based implementation of the 5GC network, in this work, we proposed a solution that guarantees service continuity in the case of workload migration between MEC Hosts. Our method preserves the IPv6 service address, enabling relocation transparency for the client, achieved through SRv6 to steer packets to the new service location. We introduced new SRv6 behaviors to support service continuity in the 5G-MEC environment, namely, `End.Buffer` and `End.M.GTP6.D.Buffer`. Those behaviors allow being able to flexibly place the buffer within the network and integrate it as part of the SID processing. We set up a small-scale testbed environment to assess our proposed solution. The results outline that our approach has superior scalability compared to a non-SRv6 alternative relying on standard IPv6 routing. The configuration time for the SRv6 approach is at most 52% less than the non-SRv6 alternative. The number of rules to be installed for the SRv6 approach scale with the number of clients, while the non-SRv6 alternative scales with the number of clients multiplied by the number of core devices. Additionally, the experimental analysis illustrates that our system ensures lossless workload migration through the utilization of a packet buffer. The packet buffer allows the throughput's standard deviation to be 10% lower compared to the No Buffer solution for all the considered topologies.

As future work, we plan to extend the analysis by including the RAN in order to have a more complete view of the performance of our system. We also intend to study orchestration solutions following the CATS paradigm.

Author Contributions: Conceptualization, L.L., C.P., A.V. and E.M.; methodology, L.L., C.P., A.V. and E.M.; software, L.L.; writing, L.L. and A.V.; writing—review and editing, L.L., C.P., A.V. and E.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Italian Ministry of Education and Research (MUR) in the framework of the CrossLab and the FoReLab projects (Departments of Excellence), and by the European Union—NextGenerationEU—under the National Sustainable Mobility Center (CN00000023, Italian Ministry of University and Research Decree n. 1033—17/06/2022, Spoke 10), partnership on "Telecommunications of the Future" (PE0000001—program "RESTART"), and PRIN 2022 project TWINKLE (project ID: 20223N7WCJ and CUP: E53D23007770001).

Data Availability Statement: The data presented in this study are available in this article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Appendix A.1. Control Plane Path Allocation Procedure in Enhanced Mode Scenario

Figure A1 shows the control plane procedure for the path allocation. Firstly, the MEO asks the SMF to trigger the SDN controller for SR Endpoint3 configuration. SR Endpoint3, being the last device in the SR domain, requires a rule to forward the packets after the SRH decapsulation. Then, the MEO triggers the AMF to configure the `H.Encaps.Red` rule on gNB1, which is the first device of the SRv6 domain, in charge of encapsulating the original packet.

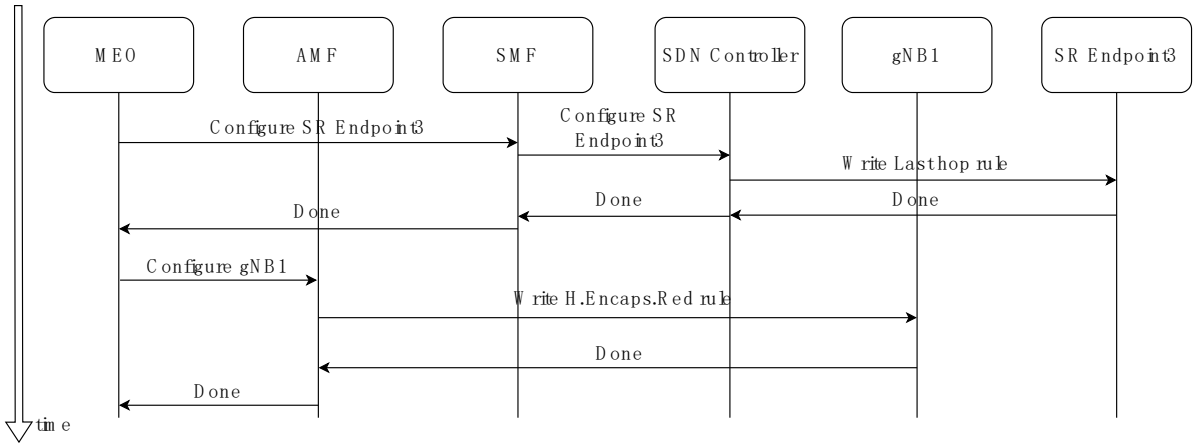


Figure A1. Control plane Enhanced Mode path allocation.

Appendix A.2. Control Plane Path Allocation Procedure in Enhanced Mode with Unchanged gNB Behavior

Figure A2 shows the control plane procedure for the path allocation. The MEO determines the appropriate SR policy to be enforced at SR Endpoint1. Based on that, it selects the corresponding BSID. For clarity, we assign label BSID1 to the selected BSID, which is an IPv6 address. This BSID serves as the designated destination address that gNB1 embeds within the IPv6 outer header to enforce the specified SR policy at Endpoint1. Firstly, the MEO asks the SMF to activate the SDN controller for the configuration of SR Endpoint3, as in the previous case. Then, the MEO instructs the SMF to trigger the SDN controller in configuring the `End.M.GTP6.D` rule on SR Endpoint1, if not installed yet, for the determined SR policy (BSID1). Finally, the MEO prompts the AMF to configure the GTP-U encapsulation rule on gNB1.

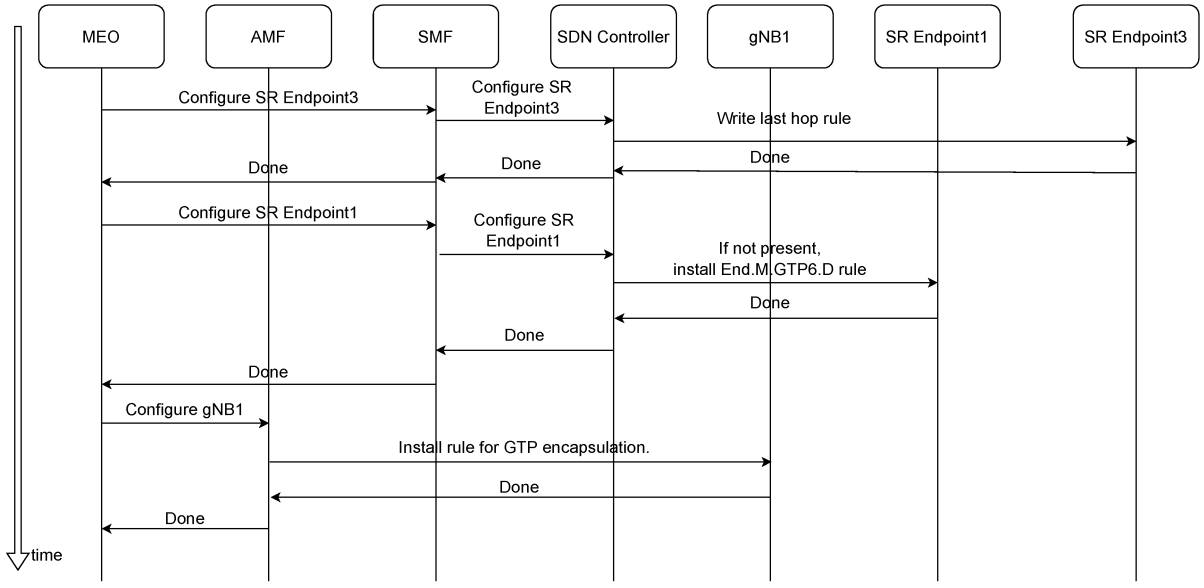


Figure A2. Control plane Enhanced Mode with Unchanged gNB path allocation.

References

1. Mitra, R.N.; Agrawal, D.P. 5G mobile technology: A survey. *ICT Express* **2015**, *1*, 132–137. [CrossRef]
2. Recommendation ITU-R M.2150-2. Detailed Specifications of the Terrestrial Radio Interfaces of International Mobile Telecommunications-2020 (IMT-2020). Available online: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2150-2-202312-I!!PDF-E.pdf (accessed on 19 March 2024).

3. Li, X.; Xie, W.; Hu, C. Research on 5G URLLC Standard and Key Technologies. In Proceedings of the 2022 3rd Information Communication Technologies Conference (ICTC), Nanjing, China, 6–8 May 2022; pp. 243–249.
4. Cicioğlu, M. Performance analysis of handover management in 5G small cells. *Comput. Stand. Interfaces* **2021**, *75*, 103502. [CrossRef]
5. Agiwal, M.; Roy, A.; Saxena, N. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1617–1655. [CrossRef]
6. ETSI. Mobile-Edge Computing (MEC); Service Scenarios. *Tech. Rep.* **2015**. Available online: https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf (accessed on 19 March 2024).
7. Wang, S.; Xu, J.; Zhang, N.; Liu, Y. A Survey on Service Migration in Mobile Edge Computing. *IEEE Access* **2018**, *6*, 23511–23528. [CrossRef]
8. ETSI. Multi-access Edge Computing (MEC); Framework and Reference Architecture. *Tech. Rep.* **2022**. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_MEC003v030101p.pdf (accessed on 19 March 2024).
9. Kekki, S.; Featherstone, W.; Fang, Y.; Kuure, P.; Li, A.; Ranjan, A.; Purkayastha, D.; Jiangping, F.; Frydman, D.; Verin, G.; et al. MEC in 5G Networks. *ETSI Tech. Rep.* **2018**. Available online: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf (accessed on 19 March 2024).
10. Barbarulo, F.; Puliafito, C.; Virdis, A.; Mingozzi, E. Extending ETSI MEC Towards Stateful Application Relocation Based on Container Migration. In Proceedings of the 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Belfast, UK, 14–17 June 2022; pp. 367–376.
11. Suzuki, M.; Miyasaka, T.; Purkayastha, D.; Fang, Y.; Huang, Q.; Zhu, J.; Burla, B.; Tong, X.; Druta, D.; Shen, J.; et al. Enhanced DNS Support towards Distributed MEC Environment. *ETSI Tech. Rep.* **2020**. Available online: https://www.etsi.org/images/files/ETSIWhitePapers/etsi-wp39-Enhanced-DNS-Support-towards-Distributed-MEC-Environment.pdf (accessed on 19 March 2024).
12. 3GPP. 5G; Service Requirements for the 5G System. *ETSI Tech. Spec.* **2021**. Available online: https://www.etsi.org/deliver/etsi_ts/122200_122299/122261/15.09.00_60/ts_122261v150900p.pdf (accessed on 19 March 2024).
13. Matsushima, S.; Filsfils, C.; Kohno, M.; Camarillo, P.; Voyer, D. Segment Routing over IPv6 for the Mobile User Plane. IETF 9433, 2023. Available online: https://datatracker.ietf.org/doc/rfc9433/ (accessed on 19 March 2024).
14. 3GPP. Technical Specification Group Core Network and Terminals; Study on User Plane Protocol in 5GC. TR 29.892, 2019. Available online: https://www.3gpp.org/ftp/Specs/archive/29_series/29.892/29892-g00.zip (accessed on 19 March 2024).
15. Ventre, P.L.; Tajiki, M.M.; Salsano, S.; Filsfils, C. SDN Architecture and Southbound APIs for IPv6 Segment Routing Enabled Wide Area Networks. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1378–1392. [CrossRef]
16. Lemmi, L.; Puliafito, C.; Virdis, A.; Mingozzi, E. Ensuring Lossless Workload Migration at the Edge with SRv6. In Proceedings of the 2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Dresden, Germany, 7–9 November 2023; pp. 53–58.
17. Filsfils, C.; Previdi, S.; Ginsberg, L.; Decraene, B.; Litkowski, S.; Shakir, R. Segment Routing Architecture. IETF 8402, 2018. Available online: https://datatracker.ietf.org/doc/html/rfc8402 (accessed on 19 March 2024).
18. Filsfils, C.; Nainar, N.K.; Pignataro, C.; Cardona, J.C.; Francois, P. The Segment Routing Architecture. In Proceedings of the IEEE GLOBECOM, San Diego, CA, USA, 6–10 December 2015; pp. 1–6.
19. Filsfils, C.; Dukes, D.; Previdi, S.; Leddi, J.; Matsushima, S.; Voyer, D. IPv6 Segment Routing Header (SRH). IETF RFC 8754, 2020. Available online: https://datatracker.ietf.org/doc/html/rfc8754 (accessed on 19 March 2024).
20. Filsfils, C.; Talaulikar, K.; Voyer, D.; Bogdanov, A.; Mattes, P. Segment Routing Policy Architecture. IETF 9256, 2022. Available online: https://datatracker.ietf.org/doc/rfc9256/ (accessed on 19 March 2024).
21. Yoo, H.; Byun, S.; Yang, S.; Ko, N. A Service Programmable Network Architecture based on SRv6. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19-21 October 2022; pp. 2068–2070.
22. Zhang, Y.; Cao, C.; Tang, X.; Pang, R.; Wang, S.; Wen, X. Programmable Service System Based on SIDaaS in Computing Power Network. In Proceedings of the 2022 5th International Conference on Hot Information-Centric Networking (HotICN), Guangzhou, China, 24–26 November 2022; pp. 67–71.
23. Filsfils, C.; Camarillo, P.; Leddy, J.; Voyer, D.; Matsushima, S.; Li, Z. Segment Routing over IPv6 (SRv6) Network Programming. IETF 8986, 2021. Available online: https://datatracker.ietf.org/doc/html/rfc8986 (accessed on 19 March 2024).
24. Desmouceaux, Y.; Townsley, M.; Clausen, T.H. Zero-Loss Virtual Machine Migration with IPv6 Segment Routing. In Proceedings of the 2018 14th International Conference on Network and Service Management (CNSM), Rome, Italy, 5–9 November 2018.
25. Polverini, M.; Aureli, D.; Cianfrani, A.; Lavacca, F.; Listanti, M. Enhancing the SRv6 Network Programming Model Through the Definition of the Maximize Throughput Behavior. In Proceedings of the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022.
26. Xhonneux, M.; Duchene, F.; Bonaventure, O. Leveraging eBPF for programmable network functions with IPv6 segment routing. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, Heraklion, Greece, 4–7 December 2018; pp. 67–72.
27. Zhang, F.; Liu, G.; Fu, X.; Yahyapour, R. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1206–1243. [CrossRef]

28. Sun, K.; Kim, Y. LISP-Based Integrated Control Plane Framework for Service Function Chaining in Distributed Edge Clouds. *IEEE Access* **2021**, *9*, 52944–52956. [CrossRef]
29. Puliafito, C.; Conforti, L.; Virdis, A.; Mingozzi, E. Server-side QUIC connection migration to support microservice deployment at the edge. *Pervasive Mob. Comput.* **2022**, *83*, 101580. [CrossRef]
30. Mahalingam, M.; Dutt, D.; Duda, K.; Agarwal, P.; Kreeger, L.; Sridhar, T.; Bursell, M.; Wright, C. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348, 2014. Available online: https://rfc-editor.org/rfc/rfc7348.txt (accessed on 18 March 2024).
31. Garg, P.; Wang, Y. NVGRE: Network Virtualization Using Generic Routing Encapsulation. RFC 7637, 2015. Available online: https://rfc-editor.org/rfc/rfc7637.txt (accessed on 18 March 2024).
32. Benjaponpitak, T.; Karakate, M.; Sripanidkulchai, K. Enabling live migration of containerized applications across clouds. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 2529–2538.
33. Fondo-Ferreiro, P.; Gil-Castiñeira, F.; González-Castaño, F.J.; Candal-Ventureira, D. A Software-Defined Networking Solution for Transparent Session and Service Continuity in Dynamic Multi-Access Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1401–1414. [CrossRef]
34. Royer, L.; Lavinal, E.; Chaput, E. Using SRv6 to access Edge Applications in 5G Networks. In Proceedings of the 2023 CoNEXT Student Workshop, Paris, France, 8 December 2023; pp. 23–24.
35. Mayer, A.; Loreti, P.; Bracciale, L.; Lungaroni, P.; Salsano, S.; Filsfils, C. Performance Monitoring with H^2: Hybrid Kernel/eBPF data plane for SRv6 based Hybrid SDN. *Comp. Net.* **2021**, *185*, 107705. [CrossRef]
36. Siriwardhana, Y.; Porambage, P.; Liyanage, M.; Ylianttila, M. A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1160–1192. [CrossRef]
37. Lin, J.; Yang, P.; Zhang, N.; Lyu, F.; Chen, X.; Yu, L. Low-Latency Edge Video Analytics for On-Road Perception of Autonomous Ground Vehicles. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1512–1523. [CrossRef]
38. Yao, K.; Trossen, D.; Boucadair, M.; Contreras, L.M.; Shi, H.; Li, Y.; Zhang, S.; An, Q. Computing-Aware Traffic Steering (CATS) Problem Statement, Use Cases, and Requirements. IETF CATS WG. Available online: https://datatracker.ietf.org/doc/draft-ietf-cats-usecases-requirements/ (accessed on 21 March 2024).
39. Chica, J.C.C.; Imbachi, J.C.; Vega, J.F.B. Security in SDN: A comprehensive survey. *J. Netw. Comput. Appl.* **2020**, *159*, 102595. [CrossRef]
40. Ahmad, I.; Shahabuddin, S.; Kumar, T.; Okwuibe, J.; Gurtov, A.; Ylianttila, M. Security for 5G and Beyond. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3682–3722. [CrossRef]
41. Ranaweera, P.; Jurcut, A.D.; Liyanage, M. Survey on Multi-Access Edge Computing Security and Privacy. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1078–1124. [CrossRef]
42. Bascio, D.L.; Lombardi, F. On SRv6 Security. *Procedia Comput. Sci.* **2022**, *201*, 406–412. [CrossRef]
43. Filsfils, C.; Michielsen, K.; Camarillo, P.; Clad, F. SRv6Introduction. Cisco Tutorial. Available online: https://www.segment-routing.net/tutorials/2017-12-05-srv6-introduction/ (accessed on 18 March 2024).
44. ETSI GS MEC 021 V2.2.1. Multi-access Edge Computing (MEC); Application Mobility Service API. Feb. 2022. Available online: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/021/02.02.01_60/gs_mec021v020201p.pdf. (accessed on 12 April 2024).
45. Puliafito, C.; Vallati, C.; Mingozzi, E.; Merlino, G.; Longo, F.; Puliafito, A. Container Migration in the Fog: A Performance Evaluation. *Sensors* **2019**, *19*, 1488. [CrossRef] [PubMed]
46. ONF SDN Tutorial. Available online: https://github.com/opennetworkinglab/ngsdn-tutorial (accessed on 18 March 2024).
47. Mininet. Available online: http://mininet.org (accessed on 18 March 2024).
48. CRIU. Available online: https://criu.org/Main_Page (accessed on 18 March 2024).
49. RSYNC. Available online: https://rsync.samba.org (accessed on 18 March 2024).
50. Yu, Y.; Calagna, A.; Giaccone, P.; Chiasserini, C.F. TCP Connection Management for Stateful Container Migration at the Network Edge. In Proceedings of the 2023 21st MedComNet, Island of Ponza, Italy, 13–15 June 2023; pp. 151–157.

future internet

MDPI

Article

Wireless and Fiber-Based Post-Quantum-Cryptography-Secured IPsec Tunnel

Daniel Christian Lawo [1,2], Rana Abu Bakar [3,4], Abraham Cano Aguilera [1,2], Filippo Cugini [3], José Luis Imaña [5], Idelfonso Tafur Monroy [1,*] and Juan Jose Vegas Olmos [2]

[1] Department of Electrical Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands; d.c.lawo@tue.nl (D.C.L.); a.c.a.cano.aguilera@tue.nl (A.C.A.)
[2] Software Architecture, Nvidia Corporation, Yokneam Illit 2066730, Israel; juanj@nvidia.com
[3] Consorzio Nazioinale Interuniversitario per le Telecomunicazioni, 56124 Pisa, Italy; rana.abubakar@santannapisa.it (R.A.B.); filippo.cugini@cnit.it (F.C.)
[4] Istituto di Telecomunicazioni, Informatica e Fotonica, Scuola Superiore Sant'Anna, 56124 Pisa, Italy
[5] Department of Computer Architecture and Automation, Universidad Complutense de Madrid, 28040 Madrid, Spain; jluimana@ucm.es
* Correspondence: i.tafur.monroy@tue.nl

Abstract: In the near future, commercially accessible quantum computers are anticipated to revolutionize the world as we know it. These advanced machines are predicted to render traditional cryptographic security measures, deeply ingrained in contemporary communication, obsolete. While symmetric cryptography methods like AES can withstand quantum assaults if key sizes are doubled compared to current standards, asymmetric cryptographic techniques, such as RSA, are vulnerable to compromise. Consequently, there is a pressing need to transition towards post-quantum cryptography (PQC) principles in order to safeguard our privacy effectively. A challenge is to include PQC into existing protocols and thus into the existing communication structure. In this work, we report on the first experimental IPsec tunnel secured by the PQC algorithms Falcon, Dilithium, and Kyber. We deploy our IPsec tunnel in two scenarios. The first scenario represents a high-performance data center environment where many machines are interconnected via high-speed networks. We achieve an IPsec tunnel with an AES-256 GCM encrypted east–west throughput of 100 Gbit/s line rate. The second scenario shows an IPsec tunnel between a wireless NVIDIA Jetson and the cloud that achieves a 0.486 Gbit/s AES-256 GCM encrypted north–south throughput. This case represents a mobile device that communicates securely with applications running in the cloud.

Keywords: post-quantum cryptography; falcon; dilithium; kyber; data processing unit; data center; IPsec

Citation: Lawo, D.C.; Abu Bakar, R.; Cano Aguilera, A.; Cugini, F.; Imaña, J.L.; Tafur Monroy, I.; Vegas Olmos, J.J. Wireless and Fiber-Based Post-Quantum-Cryptography-Secured IPsec Tunnel. *Future Internet* **2024**, *16*, 300. https://doi.org/10.3390/fi16080300

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 26 July 2024
Revised: 15 August 2024
Accepted: 20 August 2024
Published: 21 August 2024

1. Introduction

For several years now, quantum computing has been a focal point of rigorous investigation ultimately resulting in the creation of a quantum processor [1]. The arrival of a powerful commercially available quantum computer is expected in the near future, with prototype systems [2], digital annealers [3], and quantum annealers [4] already being on the market. This presents a significant challenge to contemporary communication systems reliant on classical cryptographic infrastructure and methods. The vulnerability of asymmetric cryptography, such as RSA [5], to quantum processors poses a serious threat to our communication. Unlike asymmetric cryptography, symmetric cryptography such as the Advanced Encryption Standard (AES)/Rijndael cipher [6] is said to remain secure against quantum threats if its key size is doubled [7]. Hence, a shift from AES-128 to AES-256 is required. However, the urgency to replace current asymmetric cryptography algorithms with quantum-resistant alternatives, known as post-quantum cryptography (PQC), is paramount. It is imperative to account for the threat to our digital communications posed by the arrival of quantum computers. Therefore, in December 2016, the National Institute

of Standards and Technology (NIST) launched a competition for the standardization of new, quantum-resilient algorithms that are hard to crack not only by classical computers but also by quantum computers [8].

Recently, the NIST announced their decision to standardize three PQC signature algorithms and one Key Exchange Mechanism (KEM) [8]. Many different candidates were submitted to the NIST competition. The security of the candidates is mostly based on one of the following approaches [9]: multivariate cryptography, hash-based cryptography, code-based cryptography, isogeny-based cryptography, or lattice-based cryptography. Multivariate cryptographic systems base their security on solving multivariate equation systems. An example for a PQC candidate which is multivariate-based is the PQC signature scheme Rainbow [10]. Hash-based schemes base their security on the well-known and well-understood technique of hashing. The NIST candidate SPHINCS+ [11] is a hash-based signature scheme that was one of the four candidates in the NIST competition that was announced to be standardized. Code-based schemes are relying on algorithmic primitives [12]. To name an example, Classic McEliece [13] is a code-based KEM that was submitted to the NIST competition that was not elected to be standardized. However, three out of four candidates for PQC standardization, namely Falcon [14], Dilithium [15], and Kyber [16], are based on cryptographic lattices. The fourth candidate that is going to be standardized, SPHINCS+ [11], is hash-based. In the literature, extensive comparisons have been made regarding the performance of SPHINCS+, Falcon, and Dilithium. Notably, studies such as [17,18] identify certain drawbacks of SPHINCS+ that are particularly relevant to our work. Specifically, SPHINCS+ produces signatures that are substantially larger in size, as shown in Table 1. These significantly larger signature sizes pose a concern for our client-to-data-center application. Additionally, SPHINCS+ is the most computationally intensive algorithm among the three. Therefore, we have excluded SPHINCS+ from consideration and are focusing our PQC work on three out of the four algorithms that have been chosen for standardization: Falcon [14], Dilithium [15], and Kyber [16].

The current digital technologies confirm identities using digital certificates that create a so-called chain of trust. Figure 1 illustrates how a chain of trust is established using a sequence of certificates, which can include one or multiple intermediate certificates. A digital certificate contains the public key and the signature of a certificate authority (CA). The end-entity certificate includes the certificates within the certificate chain. Therefore, the sizes of the signatures and the public keys heavily influence the size of the resulting certificate. Naturally, that holds true for all certificates included in the certificate chain that need to be validated by the end entity. In Table 1, the signature and public key sizes of the three PQC signature algorithms chosen by the NIST are shown in bytes. Additionally, Kyber's public key and encapsulation sizes in bytes can be seen. For reference purposes, Table 2 shows the sizes in bytes for the most commonly used classical signature algorithms. The signature algorithms mentioned in Table 2 are not quantum-safe. Comparing Table 1 with Table 2 shows clearly that the sizes of the PQC algorithms are significantly greater compared to their classical counterparts. Hence, the size of a PQC certificate is considerably greater than the size of a certificate that is signed using classical cryptographic algorithms such as RSA. While PQC algorithms have yet to be standardized and rolled out into production systems, efforts are being taken to examine so-called hybrid certificates [19,20]. Hybrid certificates are certificates that support the use of multiple algorithms during the transitional phase where classical algorithms are still in use and PQC algorithms are being deployed.

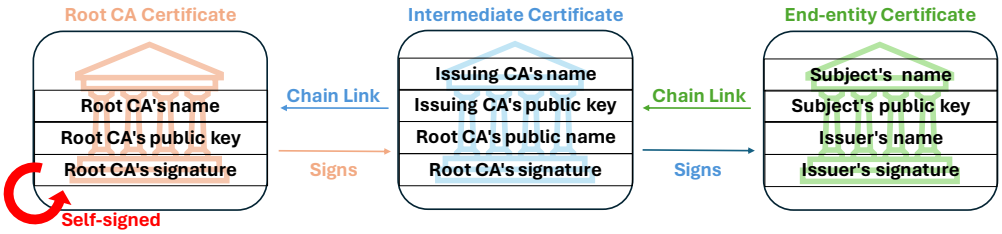


Figure 1. Chain of trust: The root CA signs the intermediate certificate. The end-entity certificate's authenticity is confirmed by one or more intermediate CAs. Hence, trust is established.

Table 1. Signature (Sig) and public key (Pub key) sizes of Falcon and Dilithium in bytes for different NIST security levels (I, II, III, and V). Public key (Pub key) and encapsulation (Encaps) size of Kyber in bytes. For SPHINCS+, the sizes in bytes are indicated for the use of SHA-256 128-bit (NIST I), SHA-256 192-bit (NIST III), and SHA-256 256-bit hashing (NIST V).

Algorithm		I	II	III	V
Kyber	Pub key	800		1184	1568
Kyber	Encaps	768		1088	1568
Dilithium	Pub key		1312	1952	2592
Dilithium	Sig		2420	3293	4595
Falcon	Pub key	897			1793
Falcon	Sig	666			1280
SPHINCS+	Pub key	32		48	64
SPHINCS+	Sig	17,088		35,664	49,856

Table 2. Public key (Pub key) and signature (Sig) sizes of classical signature algorithms in bytes [19]. Classical signature and public key sizes are considerably smaller compared to the sizes employed in PQC signature algorithms.

Algorithm	Pub Key	Sig
RSA 1024	128	128
RSA 2048	256	256
RSA 4096	512	512
SECP384r1	48	96
SECP521r1	65	132

For the transition towards the use of quantum-resilient algorithms, PQC must be integrated into existing protocols that are used nowadays such as Internet Protocol security (IPsec). Our work contributes to the goal of integrating PQC into the widely established IPsec protocol. In this work, we use the reference implementations of Falcon, Dilithium, and Kyber that were submitted to the NIST competition. We do not use an optimized, accelerated, or in any form modified version of the algorithms. We benchmark the PQC algorithms Falcon, Dilithium, and Kyber on our experimental setup. We execute the algorithms on different processors for reference purposes. The main challenge that we address in the work we present here is the experimental integration of PQC into the IPsec protocol for the purpose of quantum-resilient communication. We set up a PQC-secured IPsec tunnel and implement the tunnel for two scenarios: (1) an intra-data-center high-speed connection between devices in the data center and (2) a client-to-data-center connection. With this work, it is our goal to contribute to the state of the art by presenting a quantum-safe software stack for setting up a high-speed IPsec tunnel. We demonstrate the use of the IPsec tunnel in a high-performance environment and in an environment with a mobile, low-power client.

2. Related Works

Since the NIST started the competition for the standardization of PQC algorithms, a lot of work on the implementation of the presented PQC algorithms has been performed. Those efforts have not only been focused on Central Processing Unit (CPU) implementations but also on a variety of platforms such as FPGA [21,22], GPU [23], RISC-V [24,25], or a combination of different platforms [26] to improve the performance of the computationally challenging parts of the PQC algorithms. The Number Theoretic Transform (NTT) method, for example, is a mathematical operation of fundamental importance for Falcon [14], Dilithium [15], and Kyber [16]. NTT serves for the efficient multiplication of polynomials. The procedure can be parallelized, and hence, the performance benefits strongly from implementations on platforms that can provide heavy parallelization such as FPGAs [27] or GPUs [28]. Other subroutines or functions of PQC algorithms are recursive and therefore do not benefit from parallelization. As an example, Falcon uses floating point operations and recursive functions which would require a modification prior to being implemented in hardware. In [29], the authors state that their implementation of Falcon's key generation on a FPGA shows a high latency while having high hardware utilization compared to a software-based implementation on an Intel I7 CPU.

The IPsec protocol has existed for decades. Therefore, numerous publications with different implementations on a large variety of platforms have been reported. In [30], the authors compare IPsec solutions implemented in Data Plane Development Kit (DPDK), in the Linux userspace, in the Linux kernel, on the Network Interface Card (NIC), and on the host CPU. They claim a 3.54× improvement in throughput and a 2.54× improvement in latency with their implementation compared to the existing control plane design. They achieve a 4.795 Gbit/s throughput. However, they achieve this throughput using 128-bit AES Galois-counter mode (GCM). For being considered quantum-safe, AES-256 must be used [7].

In [31], the authors examine the reference implementations of Dilithium and Kyber on data processing unit (DPU) devices and how the algorithms can be accelerated using an optimized version for ARM core processors. In [32], the authors use a PQC software stack similar to the one that we present in this work. They investigate the performance impact of Falcon and Kyber in the stack and evaluate the advantages and disadvantages of choosing one over the other. In [33], an IPsec tunnel using Dilithium and Kyber with a different software stack and a different methodology is established. In the work that we present here, we apply our knowledge about PQC and focus on establishing a PQC-secured IPsec tunnel.

In the context of quantum-secure communication, it is necessary to mention Quantum Key Distribution (QKD). While PQC is based on mathematical challenges, the security of QKD relies on the physical properties of quantum mechanics to achieve secure communications. In [34], the authors report on a 10 Gbit/s IPsec tunnel between two JP Morgan Chase data centers. QKD is a highly important approach to securing future networks against attackers. However, unlike PQC, QKD requires extensive specialized hardware. Moreover, the rate at which quantum-safe keys are exchanged is low. Consequently, a key management system [35] is required, which increases the overhead and complexity of such systems. Given the high costs associated with QKD, we expect it to be used primarily in areas with very high security needs, such as military or government applications. We envision a co-existence of PQC and QKD systems in the form of hybrid PQC-QKD schemes [35].

Despite extensive research on PQC and IPsec individually, little has been reported on the integration of PQC with IPsec. In [36], the authors use PQC in a custom protocol and combine it with IPsec. However, they neither use Falcon [14] nor report on the throughput achieved with their IPsec implementation. The focus of their work is on integrating PQC into an encryption daemon used by IPsec. Like us, the authors use strongSwan as the encryption daemon. However, without the necessary firmware support, IPsec hardware offloading cannot be used, forcing the device's CPU to handle all cryptographic operations for the IPsec tunnel. The authors employ a version of strongSwan that does

not support hardware offloading. This lack of offloading would significantly reduce the tunnel's throughput due to the CPU's increased workload. In our work, we demonstrate a complete setup of a PQC-secured IPsec tunnel using Dilithium, Falcon, and Kyber, including hardware offloading of the tunnel's AES-256 GCM operations, and report on the performance.

3. IPsec Protocol

IPsec [37] is an OSI layer 3 (network layer) protocol that is part of the IPv4 suite. For comparison, MACsec is a layer 2 (data link layer) protocol, while TLS acts on layer 4 (transport layer), and SSH operates on layer 7 (application layer) of the OSI model. IPsec is used to end-to-end encrypt and decrypt data in transit [38]. Multiple different algorithms are supported. In this work, we use AES GCM, in particular AES-256 GCM, because the DPU devices allow for hardware offloading of AES GCM. IPsec works in either transport or tunnel mode. Using the transport mode, the payload of the ip packet is encrypted. The IP header is not modified nor encrypted. The tunnel mode encrypts the entire IP packet and authenticates it. Therefore, the original IP packet is encapsulated into a new IP packet including a new IP header. The tunnel mode is used to create virtual private networks for network-to-network communications.

Moreover, IPsec can be used in either the Authentication Header (AH) mode or in Encapsulation Security Payload (ESP) mode [39]. The AH mode serves as a means for authentication exclusively. It guarantees data integrity, data origin authentication, and optionally, a replay protection service. Data integrity is maintained through the utilization of a message digest created by algorithms like HMAC-MD5 or HMAC-SHA. Data origin authentication is established by employing a shared secret key to generate the message digest. Replay protection is implemented through a sequence number field within the AH header. The AH mode verifies the authenticity of IP headers and their payloads, except for specific header fields that may undergo legitimate alterations during transit, such as the Time To Live (TTL) field. The ESP protocol offers both data confidentiality through encryption and authentication, which includes ensuring data integrity, data origin authentication, and replay protection. ESP can operate solely for confidentiality, solely for authentication, or for both confidentiality and authentication simultaneously. When ESP incorporates authentication, it employs identical algorithms to those used in AH, although with different coverage. AH-style authentication verifies the complete IP packet, including the outer IP header, whereas the ESP authentication mechanism authenticates only the IP datagram segment of the IP packet. In this paper, we use IPsec exclusively in the ESP mode. We do not use the AH mode.

IPsec uses security policies and security associations. Every Security Association (SA) is identified by an Security Parameter Index (SPI) and a sequence number. Authentication can be carried out via two different options: with standard public key encryption or with the so-called pre-shared key method. IPsec supports a variety of public key schemes, such as RSA [40] or Diffie–Hellman [41]. If, however, the pre-shared key method is used for establishing the IPsec tunnel, the symmetric key that is used for encryption is already in possession of the devices. The devices send each other hashes of the pre-shared keys and hence proof that they are indeed in possession of the correct key. As IPsec does not support PQC public key encryption, we perform our own PQC-secured authentication and key exchange and then ultimately set an IPsec connection using the pre-shared key method. This is possible because both parties have the key that they exchanged before via PQC. An important parameter when setting up an IPsec connection is the re-keying parameter that indicates how often a new key is exchanged. As an encryption daemon, we use strongSwan (https://www.strongswan.org/, accessed on 25 January 2025) version 5.9.10. Since this version does not support PQC yet, we deactivate re-keying in this work as re-keying would inevitably employ non-quantum-safe key exchange algorithms. Instead of re-keying, we run the herein presented software stack again, thus setting up a new connection that uses a new key. The IPsec rules are installed into the Linux kernel using the ip-xfrm command

that is part of the iproute2 (https://github.com/iproute2/iproute2 accessed on 3 July 2024) Linux tool set.

IPsec Hardware Acceleration

IPsec is a widely used protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet in a data stream. BlueField-2 DPUs offer hardware acceleration for IPsec, enabling efficient packet processing and encryption/decryption without burdening the host CPU. The software framework that is used to program the DPU is called Data Center-on-a-Chip Architecture (DOCA). By design, DOCA is very similar to NVIDIA's Compute Unified Device Architecture (CUDA) that is used for the programming of a Graphic Processing Unit (GPU). Like in CUDA, using the Application Programming Interfaces (APIs) enables the user to access on-board hardware accelerators of the device. Figure 2 presents the workflow of DOCA IPsec, detailing the steps involved in setting up IPsec policies and processing network traffic within the BlueField environment. We configure the NIC similar to the procedure presented on the NVIDIA DOCA webpage (https://docs.nvidia.com/doca/sdk/nvidia+doca+east-west+overlay+encryption+application/index.html, accessed on 15 July 2024) regarding the acceleration of east–west traffic. Hence, the DPU offloads the complex cryptographic operations to the hardware accelerators that are accessible via DOCA.

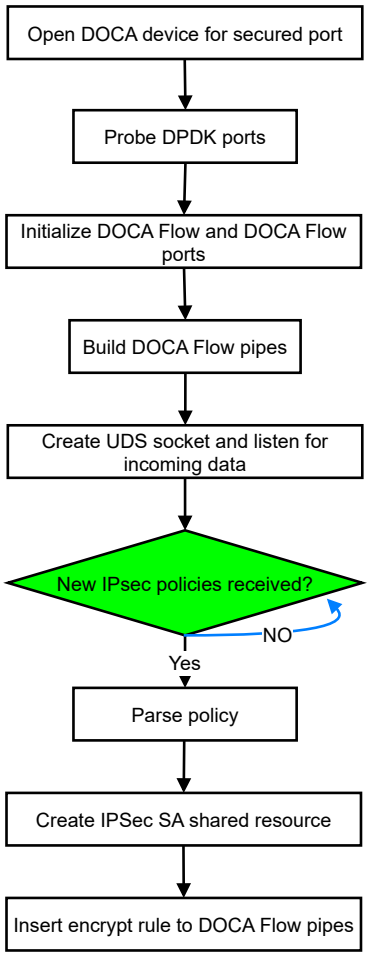


Figure 2. Overview of DOCA IPsec workflow. The sequential steps involved in setting up IPSec policies and processing network traffic within the BlueField environment are illustrated.

When the IPsec connection is configured, the process begins by initializing the DOCA device for the secure port. This step ensures that the network interface is ready to handle incoming and outgoing traffic securely. To prepare the infrastructure for packet processing

and flow management, DPDK ports are probed to identify available network interfaces. This is followed by initializing DOCA Flow (https://docs.nvidia.com/doca/archive/doca-v1.2/flow-programming-guide/index.html accessed on 10 January 2024) and DOCA Flow ports. DOCA Flow pipes are then constructed to define the flow of packets through various stages of processing. These pipes facilitate actions such as the filtering, forwarding, and manipulation of packet headers.

A Unix Domain Socket (UDS) is created to establish a communication channel for receiving incoming data. This socket serves as the interface for interacting with external systems and applications. The system periodically checks for new IPsec policies. Upon receiving a new policy, it is parsed to extract relevant information such as encryption and decryption rules. If the policy corresponds to an encryption rule, an IPsec SA shared resource is created. This resource manages encryption parameters and states for packets matching the specified criteria. The encryption rule is then inserted into the appropriate DOCA Flow pipes. This ensures that packets matching the encryption criteria undergo the specified encryption process before further processing or transmission. After processing the IPsec policy, the system resumes listening for incoming data on the UDS. This allows it to continue handling network traffic while enforcing the defined security policies.

As shown in Figure 3, the IPsec offload process on BlueField begins with the initialization and configuration of the DPU. This involves opening and initializing a DOCA device for the unsecured port and setting up the control pipe as the root for packet processing. Incoming packets are classified based on the protocol type (TCP or UDP) and IP version (IPv4 or IPv6). BlueField-2 DPUs use dedicated pipes to match packet headers against predefined criteria, such as 5-tuple (source/destination IP addresses, source/destination ports, and protocol). The 5-tuple includes the IP address of the device that sends the packet, as well as the IP address of the intended recipient device. Additionally, it includes the source port number used by the application to send data on the source device, as well as the destination port number used by the application to receive data on the destination device. Finally, it specifies the type of protocol used for communication, such as TCP or UDP. Once a packet is classified and matched, it undergoes encryption or decryption based on the established IPsec policies. BlueField-2 DPUs support hardware-accelerated encryption/decryption operations, leveraging dedicated cryptographic engines (https://docs.nvidia.com/doca/sdk/nvidia+doca+ipsec+security+gateway+application+guide accessed on 10 January 2024) for fast and secure data processing.

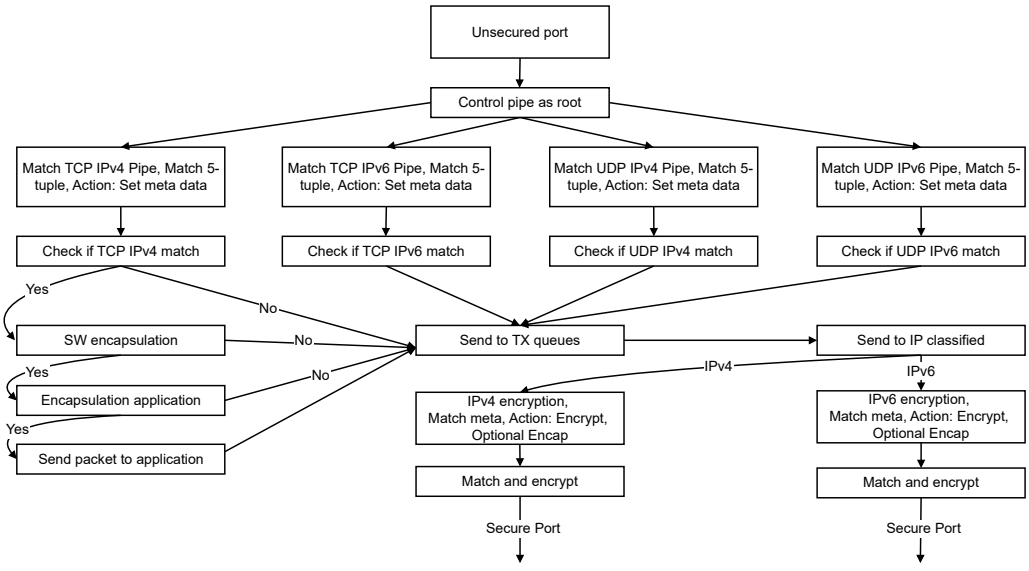


Figure 3. DOCA IPsec Flow diagram.

Encrypted packets are encapsulated with additional IPsec headers before being forwarded to the appropriate destination. BlueField-2 DPUs handle encapsulation efficiently, ensuring minimal overhead and latency in the data transmission process. In the secure egress domain, BlueField-2 DPUs perform additional processing to ensure proper packet routing and security enforcement. This may involve IP classification, encryption pipe selection (IPv4 or IPv6), and the application of IPsec policies based on metadata matching. Metadata refer to additional information attached to a packet that can provide context beyond the basic header information. For IPsec, metadata might include details about the specific SA used for encryption, enabling the DPU to select the appropriate algorithms and keys. BlueField-2 DPUs seamlessly integrate with the application layer, providing APIs and interfaces for application developers to define and enforce IPsec policies, monitor network traffic, and manage security configurations. The hardware acceleration capabilities of BlueField-2 DPUs significantly enhance the performance and scalability of IPsec implementations in data center and cloud environments. By offloading intensive cryptographic operations to dedicated hardware, BlueField-2 DPUs optimize resource utilization and improve the overall system efficiency.

4. Implementation

4.1. PQC-Algorithms

Falcon [14], Dilithium [15], and SPHINCS+ [11] are the three candidates in the NIST competition for PQC signature algorithms that are chosen to be standardized [8]. All three signature algorithms have in common that they execute the same procedural steps: key generation, verification, and sign. Key generation and signing are performed by the server machine. The client machine needs to verify the signature.

Unlike Dilithium and Kyber, Falcon offers only two NIST security levels: Falcon 512 (NIST level I), and Falcon 1024 (NIST level V). Falcon's signature and public key sizes in bytes for the NIST security level I and V can be seen in Table 1. For research purposes, Falcon's reference implementation also includes Falcon 256. However, Falcon 256 is not considered secure against quantum attacks [14] and is therefore not considered in this work. The security of the algorithm is based on the theoretical framework developed by Gentry, Peikert, and Vaikuntanathan for lattice-based signature schemes [42]. This framework is applied to NTRU lattices utilizing a trapdoor sampler that is called "fast Fourier sampling". The fundamental mathematical challenge to solve is the Short Integer Solution (SIS) problem over NTRU lattices [14,43]. In this work, we use the reference implementation of Falcon that has been submitted to the NIST competition. During the key generation process, this algorithm's implementation utilizes AES-generated pseudo-random numbers as initial seeds to set up SHAKE-256 for generating random polynomials following a Gaussian distribution. If the squared norm of these polynomials exceeds the bounds, or if the norms of orthogonalized vectors deviate, the algorithm rejects them and generates new polynomials. The Fast-Fourier Transform (FFT) is employed to calculate the norms of orthogonalized vectors. Leveraging these polynomials, the algorithm produces a public key polynomial. The key generation module resolves the NTRU equation to compute the key polynomials [14].

Dilithium offers three NIST security levels: Dilithium 2 (NIST level II), Dilithium 3 (NIST level III), and Dilithium 5 (NIST level V). Signature and public key sizes in bytes can be seen in Table 1. The implementation of Dilithium that is used in this work employs SHAKE for matrix expansion, vector masking, and sampling of the secret polynomials. A Dilithium version that uses AES in counter mode for these steps exists. However, this specific version requires Advanced Vector Extensions (AVX2) operations which are not supported by the DPU's ARM cores. Hence, we employ SHAKE instead of AES for the key generation of Dilithium. During the signature generation, NTT is used [15].

Initially, many different candidates for possible KEM algorithms were submitted to the NIST competition. To name an example, other KEM candidates were BIKE (code-based) [44] or SIKE (isogeny-based) [45]. Regardless, Kyber is the only KEM in the NIST

competition that was chosen for standardization. It thus will most likely become the main algorithm for the key exchange stage of PQC-based digital communication. As KEM, the three main procedural steps of Kyber are called key generation, key encapsulation, and key decapsulation. In the key generation process, a key pair comprising a public key and a private key is created. The key encapsulation aims to encrypt the key that is intended for obtaining a shared secret using the public key. Subsequently, the key decapsulation is employed to recover the key that was encrypted with the public key during the encapsulation phase. Like Dilithium, Kyber offers three different levels of security: Kyber 512 (NIST level 1), Kyber 768 (NIST level III), and Kyber 1024 (NIST level V) [16]. Similar to Dilithium, Kyber employs NTT to enhance its security. The algorithm conducts arithmetic operations on 256-bit polynomials within a polynomial ring. Despite variations in security levels, the size and modulus of the polynomials remain consistent. The increase in security level solely leads to a rise in the number of polynomials utilized [16].

4.2. Algorithmic Procedure

We use the procedure shown in Figure 4 to set up the IPsec tunnel that we present in this work. First, we establish an OpenSSL (https://www.openssl.org/ accessed on 10 January 2024) connection between the two devices. The cipher that we used for the OpenSSL session is TLS_AES_256_GCM_SHA384. The required certificates are self-signed. In a real-life scenario, the self-signed certificates would need to be replaced by certificates issued by a certificate authority. The second step is the exchange of PQC signatures. For this purpose, we use the reference implementation of Falcon [14] and the reference implementation of Dilithium [15]. After that, we exchange a PQC key using Kyber's reference implementation [16]. The reference implementations of the three PQC algorithms used in this work are available online (https://falcon-sign.info/, https://pq-crystals.org/dilithium/index.shtml, https://pq-crystals.org/kyber/index.shtml accessed on 10 January 2024). The fourth step is the combination of the OpenSSL key with the PQC key by performing an XOR operation with the OpenSSL key and the PQC key. This is called key mixing. The resulting key is secure for as long as at least one of the two mixed keys is not compromised [46]. Ultimately, we use the ephemeral key resulting from mixing the keys to set up the IPsec connection using the pre-shared key method. The IPsec connection is protected by AES-256 GCM encryption which is considered to be secure against quantum attacks [7]. Superuser privileges are required for the execution. The session key remains active for as long as the IPsec SA is active. In case a new key needs to be exchanged, the procedure shown in Figure 4 is repeated.

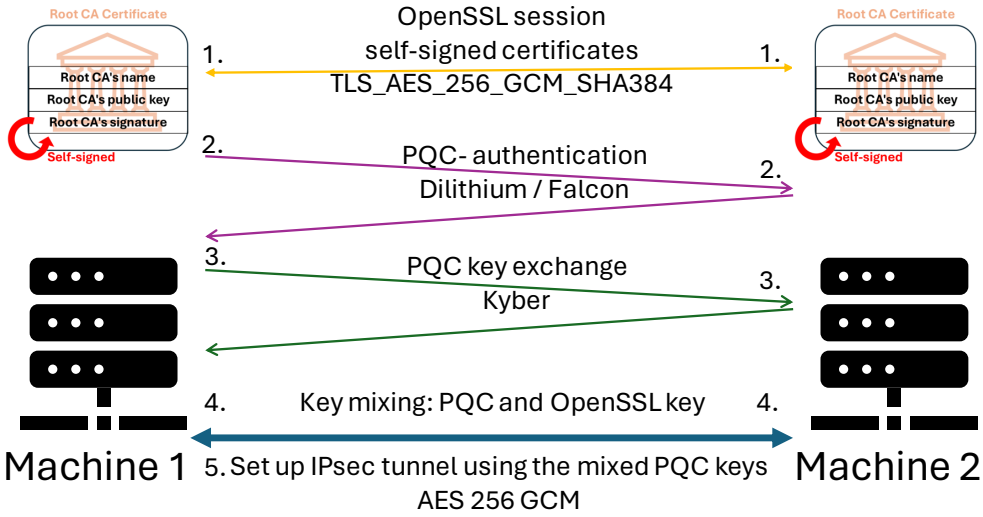


Figure 4. Methodology: First, using self-signed certificates, an OpenSSL session is established. The used cipher was TLS_AES_256_GCM_SHA384. Then, PQC signatures are exchanged. As a third step, Kyber is used to exchange a key. Fourthly, the key retrieved by Kyber and the key established by the OpenSSL session are mixed. Ultimately, an IPsec tunnel is set up using the mixed ephemeral key.

We execute the procedure shown in Figure 4 on the server machine's processor directly, on the DPU, and on the Jetson. If the cryptographic operations are executed on the server's CPU, the DPU is used as simple NIC. The server encrypts the traffic and sends the outgoing traffic already encrypted to the NIC via PCIe 4. While receiving traffic, the NIC forwards the encrypted traffic to the host, and the host machine decrypts the traffic on its own CPU. In this case, the NIC only manages outgoing and incoming connections. It is not used for processing of any kind. If the cryptographic are executed on the NIC, the server sends the outgoing and incoming traffic unencryptedly to the NIC. It thus saves valuable CPU cycles of its own CPU. In this case, the NIC executes all cryptographic operations. It receives the data as plaintext from its host and sends out the data encryptedly. The receiving DPU decrypts the incoming packets and forwards them to the host via PCIe 4.

5. Experimental Setup and Methodology

Figure 5 shows the schematical representation of a data center. In a data center, multiple servers are stashed in several racks. All servers are interconnected via a high-speed network within the data center. Different applications are running on the servers. The applications are potentially running a number of virtual machines (VMs) on various different machines that, hence, are required to communicate with each other. In a data center, this is called east–west traffic. External users and clients access the services provided by the applications that are running in the data center via the public internet. This kind of traffic is referred to as north–south traffic.

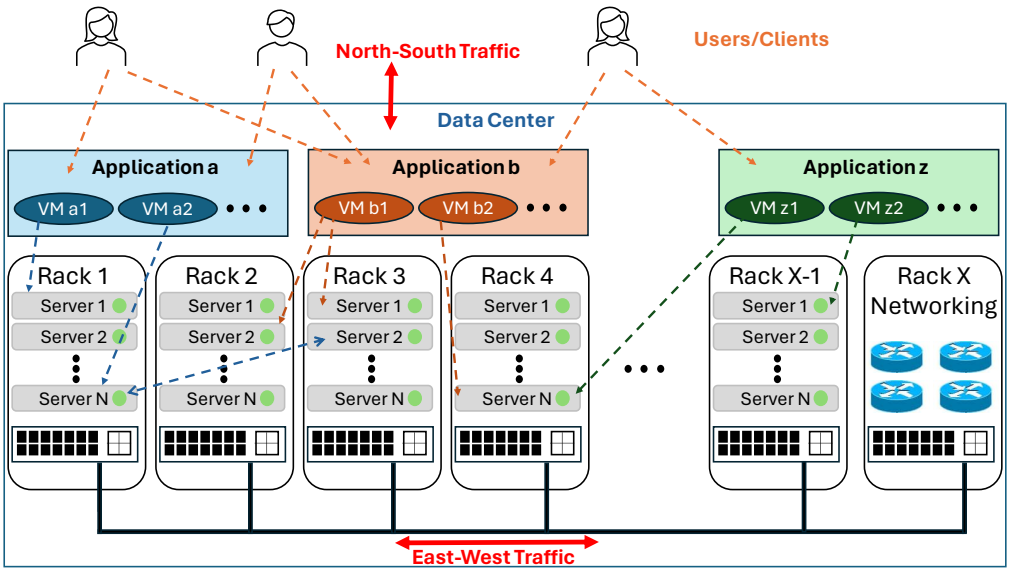


Figure 5. Schematic representation of a data center: Multiple racks host many servers, all interconnected via the local intra-data-center network. Different applications are hosted. Traffic within the data center is referred to as east–west traffic. Incoming/outgoing traffic is called north–south traffic. External users and clients access the services.

In this work, we present two scenarios for the application of the PQC-secured IPsec tunnel that we demonstrate in this work. The first scenario emulates an intra-data-center east–west communication at line rate. Our experimental setup for this, shown in Figure 6, represents this scenario: Two Dell PowerEdge 740xd (https://www.dell.com/en-us/shop/dell-poweredge-servers/poweredge-r740xd-rack-server/spd/poweredge-r740xd/pe_r740xd_tm_vi_vp_sb accessed on 10 January 2024) servers are each equipped with an NVIDIA BlueField 2 DPU (https://www.nvidia.com/en-us/networking/products/data-processing-unit/ accessed on 10 January 2024). Each DPU is equipped with eight on-board ARMv8 processors that are clocked with 2750 MHz. The DPUs are connected via optical fiber with an optical switch. The throughput between DPU and DPU is measured using the

VIAVI [47] traffic generator. The VIAVI traffic generator can achieve throughputs of up to 400 Gbit/s. However, the DPUs that we use are capable of 100 Gbit/s, and therefore, we do not make use of VIAVI's full capabilities. In order to test the throughput, the VIAVI traffic generator is placed in between the two DPUs.

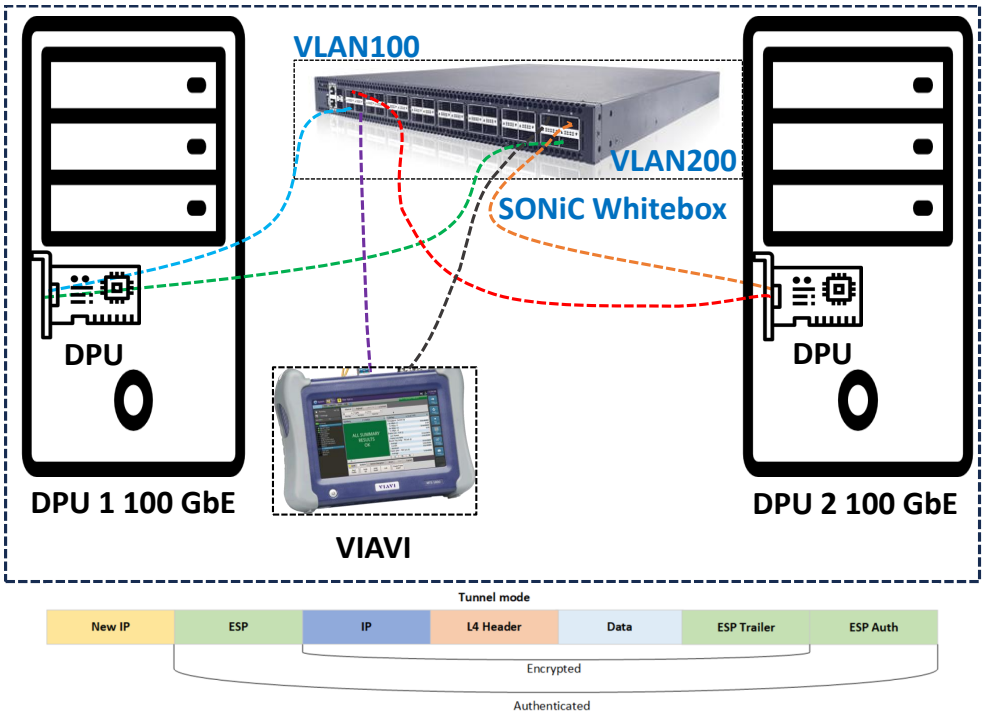


Figure 6. Two identical servers are each equipped with an Nvidia BlueField 2 100 G DPU. The DPUs are connected via optical fiber to an optical switch and IPsec connection established with following packet header fields. This emulates the east–west traffic in the intra-data-center scenario.

The second scenario that we present in our work is a quantum-safe way for clients to establish a north–south IPsec connection for accessing the services from outside the data center. We demonstrate our north–south IPsec tunnel using a wireless connection. Our setup for this can be seen in Figure 7: a mobile device that seeks to exchange encrypted information with the cloud. As a mobile device, we use an Nvidia Jetson Nano (https://developer.nvidia.com/embedded/jetson-nano accessed on 10 January 2024) that is equipped with a WiFi antenna extension. The Jetson connects to the cloud via a WiFi router. We measure the throughput between the Jetson and a 25 G DPU in the cloud using the iperf (https://iperf.fr/ accessed on 15 January 2024) traffic generator and achieve a 0.486 Gbit/s throughput. We connect the Jetson to a 25 G DPU instead of a 100 G DPU because the WiFi antenna's maximum throughput is equal to 1 Gbit/s, and therefore, a 25 G DPU is suitable to provide the necessary performance.

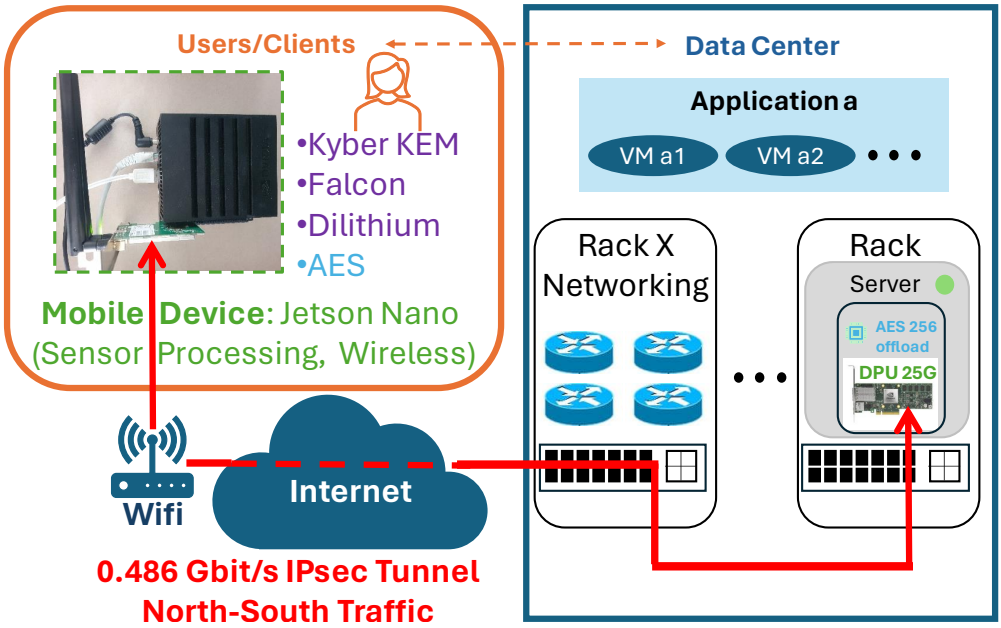


Figure 7. Wireless setup: An Nvidia Jetson Nano connects to a WiFi using a WiFi antenna extension. The Jetson establishes a PQC-secured IPsec tunnel and connects through the network to a server that is equipped with a 25G DPU.

6. Results

6.1. Signature Algorithms

Figure 4 shows the methodology that we use for setting up the quantum-secure IPsec tunnel that we present in this work. The server machine and the client machine have to perform different tasks. As Dilithium and Falcon are signature algorithms, they are part of step 2 in Figure 4. Hence, the latency introduced by the execution of the algorithms can be attributed to step 2. While executing a signature algorithm, the server machine needs to generate a key and create a signature using the sign process. Figure 8 shows the average required clock cycles while executing Dilithium on the server side on the server directly (blue background), on the Jetson (red background), and on the DPU (yellow background). While signing, a security downgrade from Dilithium 5 down to Dilithium 3 comes with a performance gain between 6 % (server) and 39 % (DPU) in terms of CPU cycles. Further trading off security for the velocity of execution by using Dilithium 2 instead of 5 yields a performance boost between 41 % (server) and 89 % (DPU). While generating a key, it is between 35 % (server) and 40 % (Jetson) faster to use Dilithium 3 instead of Dilithium 5. At the lowest security level, it consumes 89 % (server, Jetson, and DPU) less CPU cycles to use Dilithium 2 instead of Dilithium 5.

The cryptographic operations of Dilithium's client side are shown in Figure 9. The client needs only to verify the signature. The verification process of a Dilithium 5 signature takes between 25 % (server) and 43 % (Jetson) longer than the verification of a Dilithium 3 signature. Verifying a Dilithium 2 signature is between 65 % (server) and 83 % quicker than the equivalent process for a Dilithium 5 signature.

The cryptographic latency introduced by Falcon on the server side can be seen in Figure 10. Generating a key for Falcon 1024 takes between 89 % (Jetson) and 94 % (server) more clock cycles in comparison with Falcon 512. It is of note that the key generation of Falcon is particularly more challenging than Dilithium's key generation. Falcon's key generation requires about two orders of magnitude more clock cycles compared to Dilithium's key generation. Signing requires between 63 % (server) and 73 % (DPU) more clock cycles for Falcon 1024 while comparing it with Falcon 512.

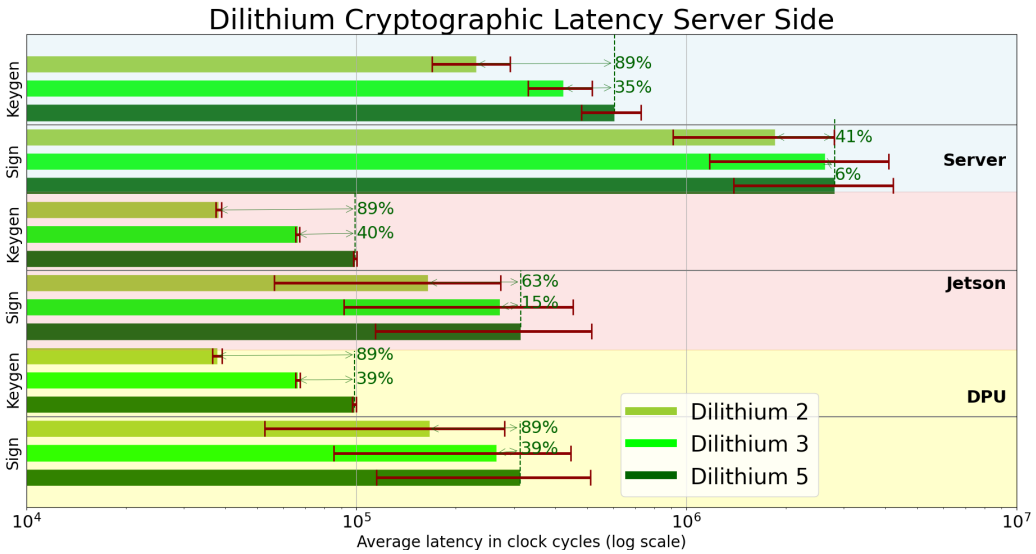


Figure 8. Cryptographic latency introduced by the execution of Dilithium's keygen and Dilithium's sign executed on the server side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

The cryptographic latency introduced by Falcon on the client side can be seen in Figure 11. The client requires between 54 % (Jetson) and 71 % (DPU) more clock cycles to verify a Falcon 1024 signature compared to a Falcon 512 signature. Falcon's and Dilithium's verification processes are similarly competitive in terms of performance. Dilithium is slightly faster while Falcon is within the same order of magnitude regarding the required CPU clock cycles for the execution.

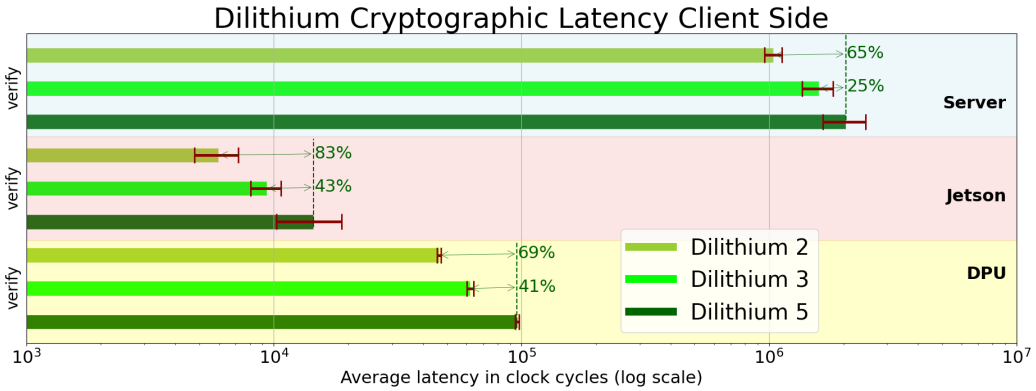


Figure 9. Cryptographic latency introduced by the execution of Dilithium's verification executed on the client side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

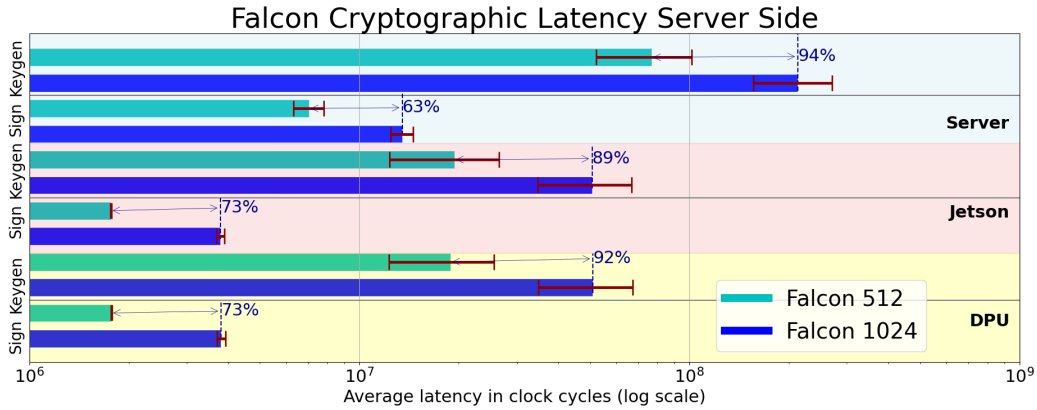


Figure 10. Cryptographic latency introduced by the execution of Falcon's key generation and Falcon's sign executed on the server side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

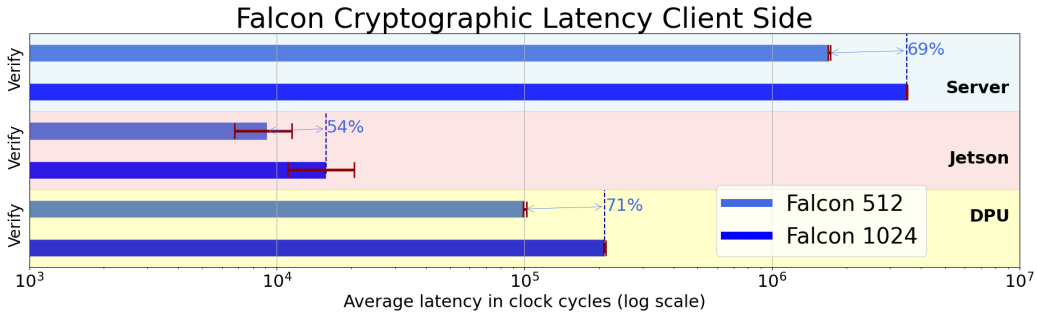


Figure 11. Cryptographic latency introduced by the execution of Falcon's verification executed on the client side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

6.2. Key Exchange Mechanism

Kyber as a key exchange mechanism represents step 3 in Figure 4. The latency added in this step equals the latency caused by the execution of Kyber. The server machine has to perform the key generation and the key decapsulation while the client machine has to execute the key encapsulation. The latency introduced for the server machine can be seen in Figure 12. Upgrading the security level from Kyber 512 to Kyber 768 comes with a penalty in terms of CPU cycles between 33 % (DPU) and 40 % (server) during the key decapsulation. Furthermore, increasing the security level from Kyber 768 to Kyber 1024 costs between 45 % (server) and 57 % more CPU (DPU) cycles while performing the key decapsulation. The key generation is generally a more expensive operation compared with key decapsulation. While generating a key for Kyber 512, an additional charge of 32 % (server) to 34 % (DPU, Jetson) applies. Upgrading the strength from Kyber 768 to Kyber 1024 costs between 28 % (server) and 66 % more CPU clock cycles while generating a key. The Jetson and the DPU perform very similarly. The server requires more CPU cycles by almost one order of magnitude regarding only the number of clock cycles that are required for the execution of the algorithm. This does not take into account the clock frequency the different processors are operating at.

Figure 13 shows the latency introduced on the client machine. The only operation that the client has to perform is the key encapsulation. Downgrading the security level from Kyber 1024 to Kyber 768 boosts the performance by between 37 % (server) and 75 % (Jetson). Decreasing the security level to Kyber 512 yields a performance gain of 75 % (server) to 113 % (DPU) with respect to Kyber 1024.

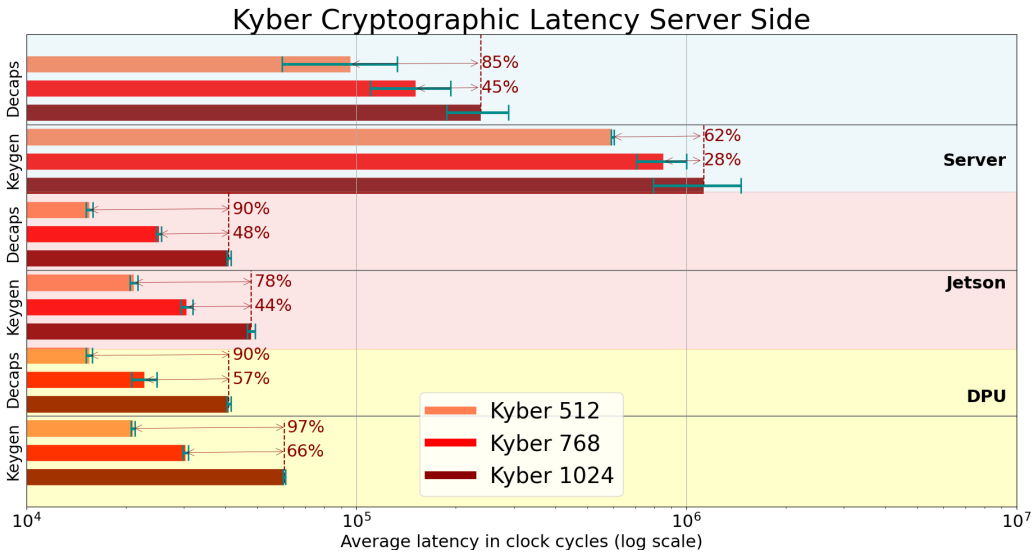


Figure 12. Cryptographic latency introduced by the execution of Kyber's key generation and Kyber's key decapsulation executed on the server side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

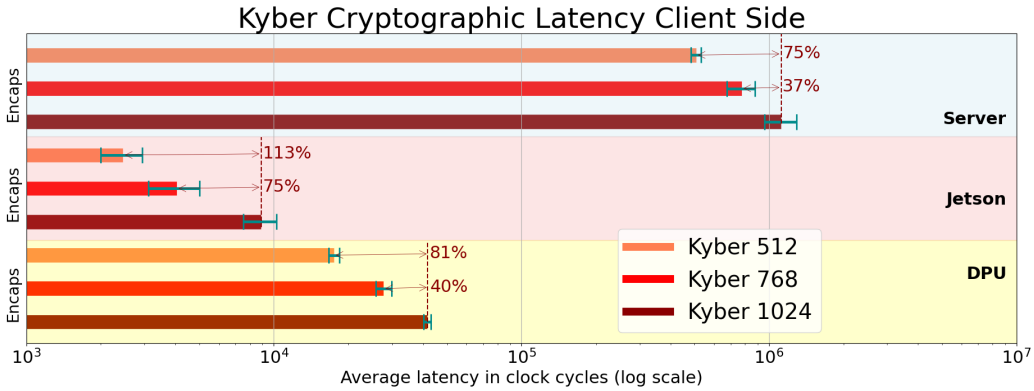


Figure 13. Cryptographic latency introduced by the execution of Kyber's key encapsulation executed on the client side. The server device was a Dell PowerEdge server, equipped with an Intel Xeon CPU, an NVIDIA Jetson (the wireless device in our setup), and an NVIDIA BlueField 2 DPU.

To examine the performance of the individual steps of the various algorithms, we execute on different processors the main three steps of Falcon and Dilithium (key generation, verify, and sign), as well the main three steps of Kyber (key generation, key encapsulation, and key decapsulation). The results can be seen in Figure 14 for an AMD Ryzen 7 3700X desktop processor, in Figure 15 for the ARMv8 processor on the DPU, and in Figure 16 for the Intel Xeon processor that our servers are equipped with. The metric is executions per second. It is of note here that the Intel Xeon, shown in Figure 16, performs slightly better than the ARMv8, shown in Figure 16, even though the Intel Xeon requires a significant amount of CPU cycles more for the execution of the PQC algorithms. This is due to the higher clock frequency that the Intel Xeon is operating at compared to the ARMv8 processor.

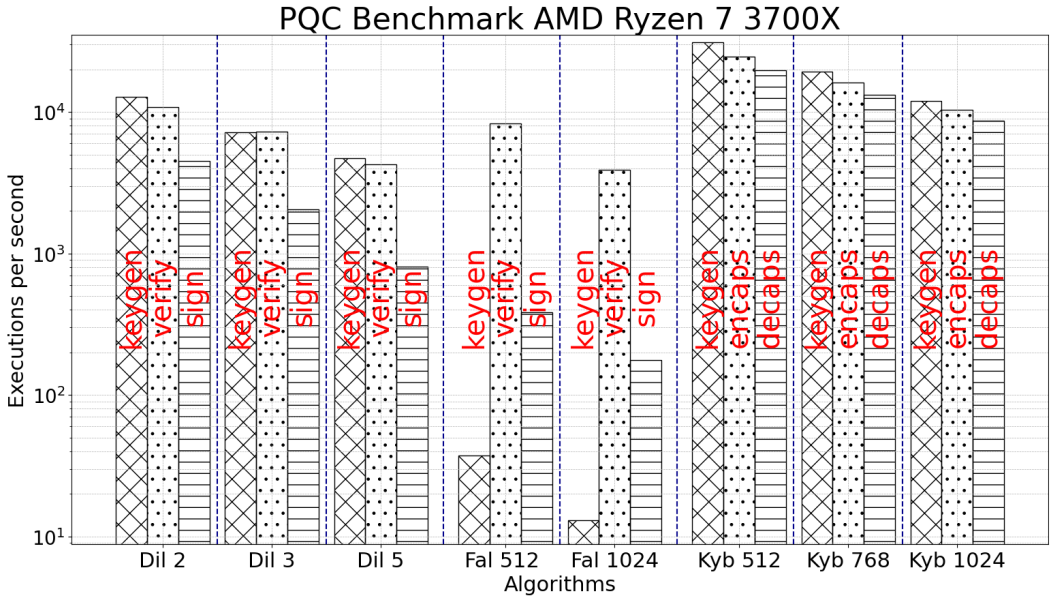


Figure 14. Falcon's and Dilithium's main steps (key generation, verification, and sign) and Kyber's main steps (key generation, key encapsulation, and key decapsulation) executed on an AMD Ryzen 7 processor.

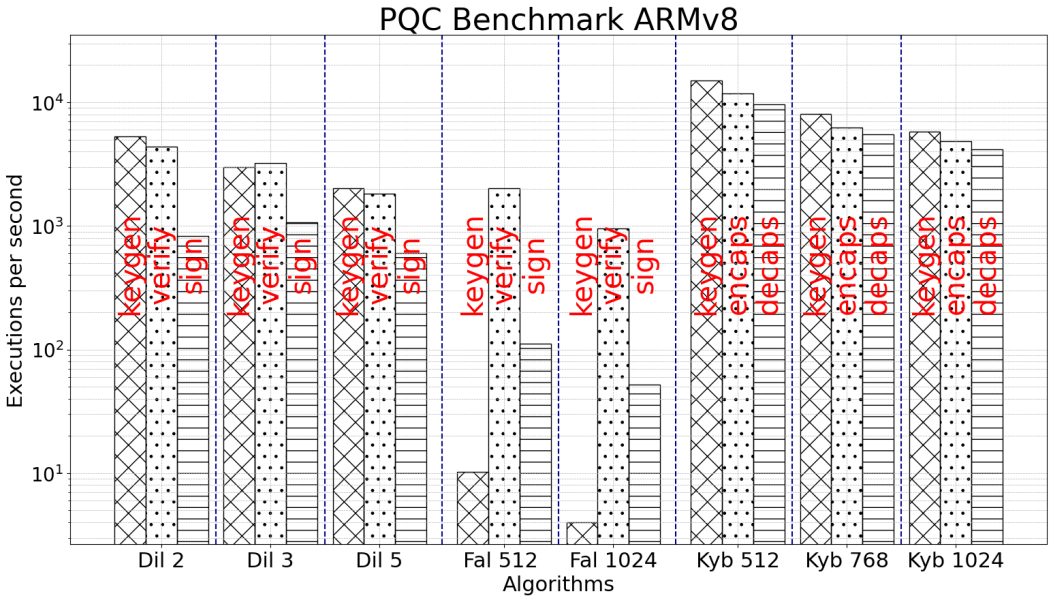


Figure 15. Falcon's and Dilithium's main steps (key generation, verification, and sign) and Kyber's main steps (key generation, key encapsulation, and key decapsulation) executed on an ARMv8 processor.

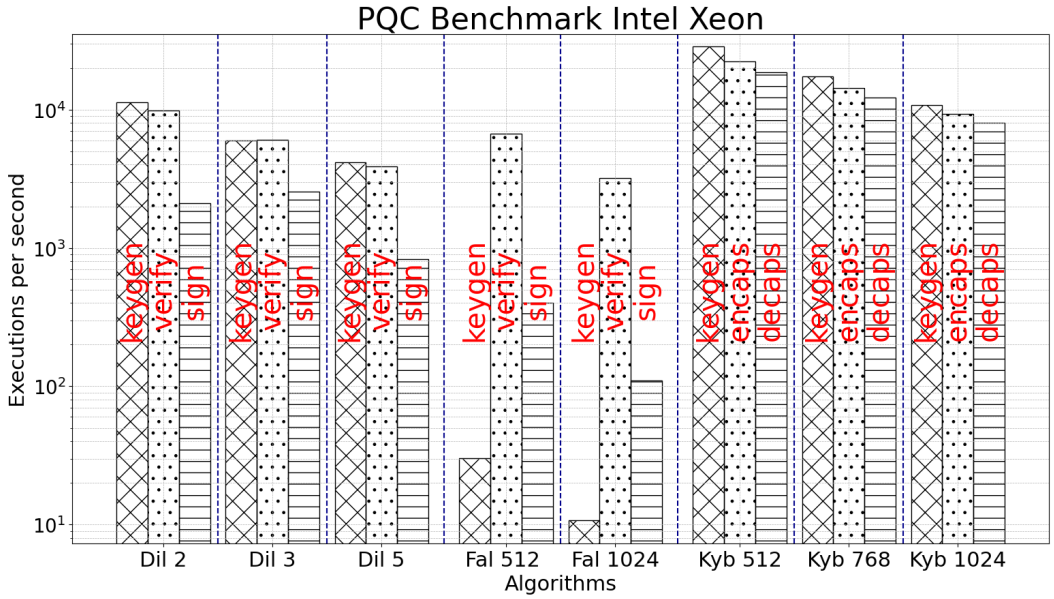


Figure 16. Falcon's and Dilithium's main steps (key generation, verification, and sign) and Kyber's main steps (key generation, key encapsulation, and key decapsulation) executed on an Intel Xeon processor.

After the keys have been exchanged successfully, during step 4 in Figure 4, we perform an XOR operation between the cryptographic key that we retrieved while setting up an OpenSSL session using classical cryptography and the keys that we have exchanged using Kyber. Ultimately, we perform step 5 in Figure 4 and set up the IPsec tunnel. We do that for both scenarios, east–west traffic and north–south traffic.

6.3. IPsec Tunnel

Using the iperf traffic generator, we analyze the performance of the north–south IPsec tunnel that we set up between the Jetson as a wireless device and the 25 G DPU in the cloud, shown in Figure 7. We achieve an AES-256 GCM encrypted wireless throughput of 0.486 Gbit/s. As this scenario emulates a mobile device communicating via the cloud, the signal travels through a chain of multiple hops. We therefore do not set the maximum transmission unit (MTU) because every device in the chain between the mobile device and the 25G DPU in the cloud can modify the MTU size.

In our intra-data-center east–west traffic scenario, however, we do have control over the MTU size. Thus, after we set up the east–west IPsec tunnel from DPU to DPU, we characterize the tunnel's throughput with different MTU sizes using the VIAVI traffic generator. The results can be seen in Figure 17. The original plots generated by VIAVI for each MTU can be seen in Figure 18. With small packets, 64 B MTU, we achieve an encrypted throughput of 34 Gbit/s. This can be seen in Figure 18a. Doubling the MTU to 128 B yields an encrypted throughput of 58 Gbit/s, as shown in Figure 18b. Setting the MTU to 256 B in Figure 18c results in an encrypted throughput of 73 Gbit/s. In Figure 18d, at 512 B MTU, we obtain an encrypted throughput of 95 Gbit/s. Ultimately, in Figure 18e,f, starting from 1024 B MTU, we observe an encrypted throughput of 100 Gbit/s line rate. For all MTU sizes that are bigger than, and including 1024 B, the encrypted throughput converges towards 100 Gbit/s. That holds true for jumbo-sized packets due to the fact that the on-board hardware accelerators of the DPU support only operation up to 100 Gbit/s.

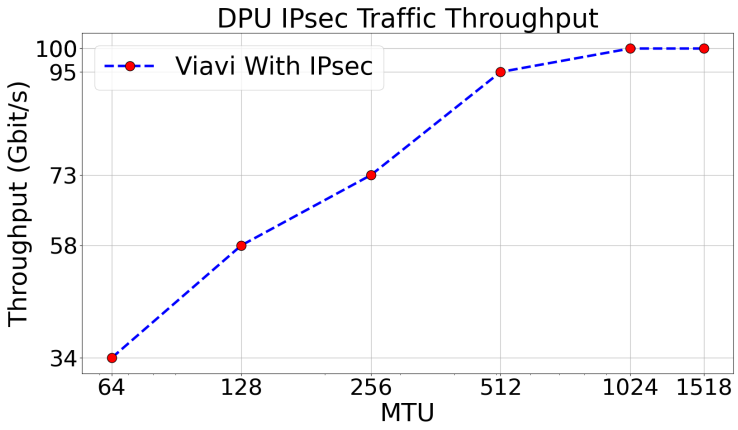


Figure 17. AES-256 GCM encrypted IPsec throughput between DPU and DPU depending on the set MTU. We achieve 100 Gbit/s from 1024 B MTU on.

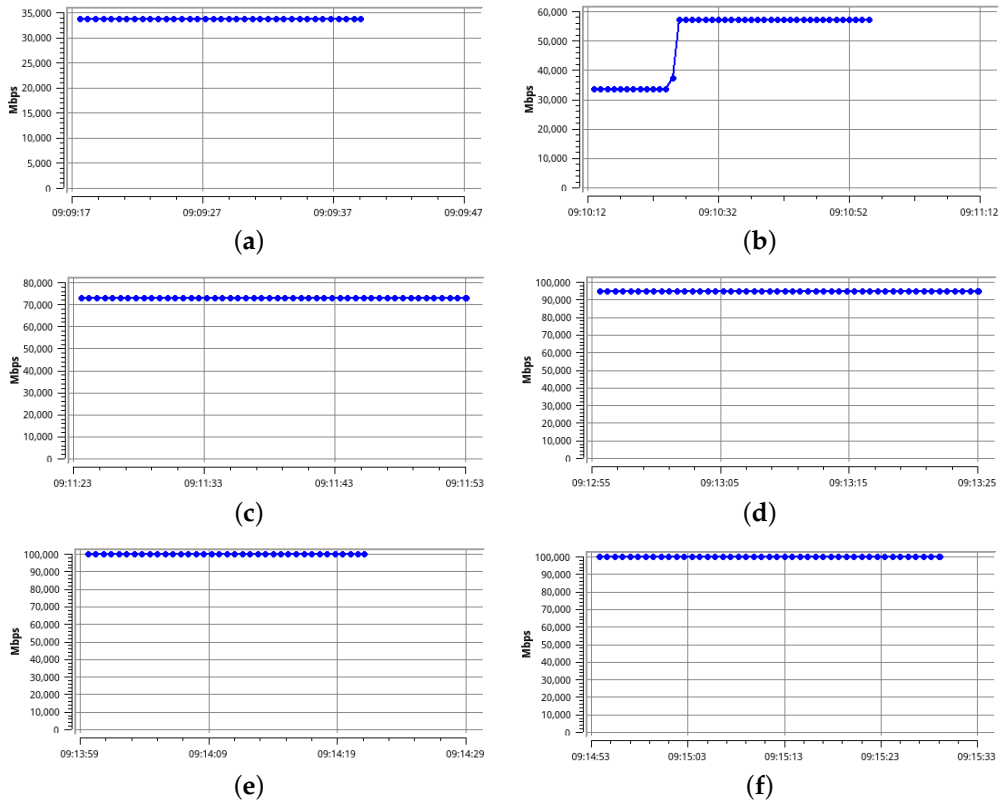


Figure 18. Throughput of the IPsec tunnel with different MTU sizes. The traffic is generated by VIAVI and passed through the IPsec tunnel that we present in this work. For all MTU sizes equal to or bigger than 1024 B, we achieve the maximum supported line rate of 100 Gbit/s. (**a**) 64 B MTU; (**b**) 128 B MTU; (**c**) 256 B MTU; (**d**) 512 B MTU; (**e**) 1024 B MTU; (**f**) 1518 B MTU.

7. Discussion

Kyber [16], being the only remaining KEM in the NIST competition chosen for standardization, is integral to every version of the encryption stack presented in this work. The choice of security level for Kyber (NIST levels I, III, or V) can vary based on numerous factors, including security requirements and the processing power of the devices involved. This decision is typically made by the application or, ultimately, by the software developer.

Dilithium [15], developed by the same group (https://pq-crystals.org/ accessed 19 March 2024) that submitted Kyber, offers excellent performance in terms of execution speed

when combined with Kyber [16]. However, Dilithium's signatures are larger compared to Falcon's, resulting in larger certificates and more data that need to be transmitted over networks. This might not be an issue in high-performance environments, such as data centers, where network bandwidth is abundant, making signature sizes irrelevant. However, in mobile applications with limited and potentially unstable network connections, the larger signature size could negatively impact performance.

Falcon's performance is inferior to Dilithium's in terms of computation. Falcon requires more CPU clock cycles per execution, and especially its key generation process is significantly more demanding. Despite this, in scenarios where keys are generated infrequently, the impact of this disadvantage is minimal. However, in high-performance environments with numerous sessions, each requiring its own key, this becomes a significant drawback of Falcon. The advantage Falcon offers is its smaller signature size, which may be more suitable for mobile, low-power, and low-performance environments.

8. Conclusions and Future Work

In this work, we present a software stack to establish a fully offloaded, quantum-safe IPsec tunnel. We first perform a quantum-safe authentication using the PQC signature algorithms Falcon and Dilithium, followed by the key exchange algorithm Kyber. Then, using the quantum-secure key, we set up an IPsec tunnel that is secured by AES-256 GCM. Moreover, we benchmark the performance of the PQC algorithms on multiple CPUs. Dilithium outperforms Falcon in terms of execution speed. However, Falcon's signature is smaller than Dilithium's which poses an advantage in environments with low network capacities.

In our experimental setup, we demonstrated an IPsec connection between a mobile device connected to WiFi and a 25 G DPU in the cloud. We did not modify the MTU, considering that every device and hop linking the connection between the Jetson and the DPU could modify the MTU. Using our setup, we achieved an end-to-end encrypted throughput of 0.486 Gbit/s between the Jetson and the 25 G DPU.

Once the IPsec tunnel was established, NVIDIA's NIC showed excellent performance by offloading cryptographic operations to the NIC's hardware accelerators, effectively liberating the host machine's CPU from cryptographic calculations. We confirmed the IPsec tunnel's performance relative to the MTU size using the VIAVI traffic generator. With small packets of 64 B MTU, we achieved a throughput of 34 Gbit/s. With an MTU of 1024 B, we achieved a full encrypted throughput of 100 Gbit/s line rate, which is the maximum performance attainable with this NIC model. We observed a 100 Gbit/s line rate for all MTU sizes larger than 1024 B, including jumbo packets.

To further accelerate the transition to quantum-resistant algorithms, we identify two major tasks that need to be addressed in the future. First, PQC algorithms must be fully integrated into existing software stacks, transitioning from research environments into production code used in real-life applications. Our work represents a first step towards this direction. Second, the processing power required for PQC algorithms is significantly higher than that of classical algorithms. We therefore anticipate the development of dedicated hardware accelerators specifically for PQC algorithms, similar to the hardware accelerators currently used for classical asymmetric cryptography. This would accelerate the execution speed and reduce the energy consumption while using PQC algorithms.

Author Contributions: Conceptualization, D.C.L., R.A.B. and F.C.; methodology, D.C.L., R.A.B., A.C.A. and F.C.; software, D.C.L., R.A.B. and A.C.A.; validation, D.C.L., R.A.B., A.C.A. and F.C.; writing—original draft preparation, D.C.L. and R.A.B.; writing—review and editing, F.C., J.L.I., I.T.M. and J.J.V.O.; visualization, D.C.L. and R.A.B.; supervision, F.C., J.L.I., I.T.M. and J.J.V.O.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly funded by the QUARC project by the European Union Horizon Europe research and innovation program within the framework of Marie Skłodowska-Curie Actions with grant number 101073355 and the CLEVER project by the Key Digital Technologies Joint Undertaking program with grant number 101097560.

Data Availability Statement: The data presented in this study are available in this article.

Conflicts of Interest: J.J.V.O. is employed by NVIDIA. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
AH	Authentication Header
API	Application Programming Interface
AVX2	Advanced Vector Extensions
CA	Certificate Authority
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DOCA	Data Center-on-a-Chip Architecture
DPDK	Data Plane Development Kit
DPU	Data Processing Unit
ESP	Encapsulation Security Payload
GCM	Galois-counter mode
GPU	Graphics Processing Unit
IPsec	Internet Protocol security
KEM	Key Exchange Mechanism
MTU	Maximum Transmission Unit
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
PQC	Post-Quantum Cryptography
QKD	Quantum Key Distribution
SA	Security Association
SIS	Short Integer Solution
SPI	Security Parameter Index

References

1. Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J.C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F.G.S.L.; Buell, D.A.; et al. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510. [CrossRef] [PubMed]
2. Crippa, L.; Tacchino, F.; Chizzini, M.; Aita, A.; Grossi, M.; Chiesa, A.; Santini, P.; Tavernelli, I.; Carretta, S. Simulating Static and Dynamic Properties of Magnetic Molecules with Prototype Quantum Computers. *Magnetochemistry* **2021**, *7*, 117. [CrossRef]
3. Codognet, P.; Diaz, D.; Abreu, S. Quantum and Digital Annealing for the Quadratic Assignment Problem. In Proceedings of the 2022 IEEE International Conference on Quantum Software (QSW), Barcelona, Spain, 10–16 July 2022; pp. 1–8. [CrossRef]
4. Hu, F.; Lamata, L.; Wang, C.; Chen, X.; Solano, E.; Sanz, M. Quantum Advantage in Cryptography with a Low-Connectivity Quantum Annealer. *Phys. Rev. Appl.* **2020**, *13*, 054062. [CrossRef]
5. Sharma, M.; Choudhary, V.; Bhatia, R.S.; Malik, S.; Raina, A.; Khandelwal, H. Leveraging the power of quantum computing for breaking RSA encryption. *Cyber-Phys. Syst.* **2021**, *7*, 73–92. [CrossRef]
6. Dworkin, M.J.; Barker, E.B.; Nechvatal, J.R.; Foti, J.; Bassham, L.E.; Roback, E.; Dray, J.F., Jr. Advanced Encryption Standard (AES). 2001. Available online: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (accessed on 17 January 2024).
7. Bonnetain, X.; Naya-Plasencia, M.; Schrottenloher, A. Quantum Security Analysis of AES. *IACR Trans. Symmetric Cryptol.* **2019**, *2019*, 55–93. [CrossRef]
8. Alagic, G.; Cooper, D.; Dang, Q.; Dang, T.; Kelsey, J.M.; Lichtinger, J.; Liu, Y.K.; Miller, C.A.; Moody, D.; Peralta, R.; et al. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. 2022. Available online: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf (accessed on 18 January 2024).
9. Bernstein, D.J.; Buchmann, J.; Dahmen, E. (Eds.) Introduction to post-quantum cryptography. In *Post-Quantum Cryptography*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1–14. [CrossRef]
10. Ding, J.; Schmidt, D. Rainbow, a New Multivariable Polynomial Signature Scheme. In *Applied Cryptography and Network Security*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 164–175.
11. Bernstein, D.J.; Hülsing, A.; Kölbl, S.; Niederhagen, R.; Rijneveld, J.; Schwabe, P. The SPHINCS+ Signature Framework. Cryptology ePrint Archive, Paper 2019/1086. 2019. Available online: https://eprint.iacr.org/2019/1086 (accessed on 20 December 2023).

12. Overbeck, R.; Sendrier, N. Code-based cryptography. In *Post-Quantum Cryptography*; Bernstein, D.J., Buchmann, J., Dahmen, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 95–145. [CrossRef]
13. Albrecht, M.R.; Bernstein, D.J.; Chou, T.; Cid, C.; Gilcher, J.; Lange, T.; Maram, V.; Von Maurich, I.; Misoczki, R.; Niederhagen, R.; et al. Classic McEliece: Conservative Code-Based Cryptography. 2022. Available online: https://inria.hal.science/hal-04288769/document (accessed on 13 January 2024).
14. Fouque, P.-A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Fast-Fourier Lattice-Based Compact Signatures over NTRU. 2019. Available online: https://falcon-sign.info/ (accessed on 15 January 2024).
15. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 238–268. [CrossRef]
16. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehle, D. CRYSTALS—Kyber: A CCA-Secure Module-Lattice-Based KEM. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018; pp. 353–367. [CrossRef]
17. Fitzgibbon, G.; Ottaviani, C. Constrained Device Performance Benchmarking with the Implementation of Post-Quantum Cryptography. *Cryptography* **2024**, *8*, 21. [CrossRef]
18. Vidaković, M.; Miličević, K. Performance and Applicability of Post-Quantum Digital Signature Algorithms in Resource-Constrained Environments. *Algorithms* **2023**, *16*, 518. [CrossRef]
19. Rubio García, C.; Rommel, S.; Takarabt, S.; Vegas Olmos, J.J.; Guilley, S.; Nguyen, P.; Tafur Monroy, I. Quantum-resistant Transport Layer Security. *Comput. Commun.* **2024**, *213*, 345–358. [CrossRef]
20. Paul, S.; Kuzovkova, Y.; Lahr, N.; Niederhagen, R. Mixed Certificate Chains for the Transition to Post-Quantum Authentication in TLS 1.3. In Proceedings of the ASIA CCS '22: 2022 ACM on Asia Conference on Computer and Communications Security, New York, NY, USA, 30 May–3 June 2022; pp. 727–740. [CrossRef]
21. Karabulut, E.; Aysu, A. A Hardware-Software Co-Design for the Discrete Gaussian Sampling of FALCON Digital Signature. *IACR Cryptol. ePrint Arch.* **2023**, *2023*, 908.
22. Howe, J.; Oder, T.; Krausz, M.; Güneysu, T. Standard Lattice-Based Key Encapsulation on Embedded Devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 372–393.
23. Gupta, N.; Jati, A.; Chauhan, A.K.; Chattopadhyay, A. PQC Acceleration Using GPUs: FrodoKEM, NewHope, and Kyber. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 575–586.
24. Gupta, N.; Jati, A.; Chattopadhyay, A.; Jha, G. Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium. Cryptology ePrint Archive, Paper 2022/496. 2022. Available online: https://eprint.iacr.org/2022/496 (accessed on 16 January 2024).
25. Karl, P.; Schupp, J.; Fritzmann, T.; Sigl, G. Post-Quantum Signatures on RISC-V with Hardware Acceleration. Cryptology ePrint Archive, Paper 2022/538. 2022. Available online: https://eprint.iacr.org/2022/538 (accessed on 20 January 2024).
26. Yaman, F.; Mert, A.C.; Öztürk, E.; Savaş, E. A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021; pp. 1020–1025.
27. Mert, A.C.; Öztürk, E.; Savaş, E. Design and Implementation of a Fast and Scalable NTT-Based Polynomial Multiplier Architecture. In Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 28–30 August 2019; pp. 253–260. [CrossRef]
28. Şah Özcan, A.; Savaş, E. Two Algorithms for Fast GPU Implementation of NTT. Cryptology ePrint Archive, Paper 2023/1410. 2023. Available online: https://eprint.iacr.org/2023/1410 (accessed on 20 January 2024).
29. Schmid, M.; Amiet, D.; Wendler, J.; Zbinden, P.; Wei, T. Falcon Takes Off—A Hardware Implementation of the Falcon Signature Scheme. Cryptology ePrint Archive, Paper 2023/1885. 2023. Available online: https://eprint.iacr.org/2023/1885 (accessed on 20th January 2024).
30. Ullah, S.; Choi, J.; Oh, H. IPsec for high speed network links: Performance analysis and enhancements. *Future Gener. Comput. Syst.* **2020**, *107*, 112–125. [CrossRef]
31. Aguilera, A.C.; Clemente, X.A.I.; Lawo, D.; Monroy, I.T.; Olmos, J.V. First end-to-end PQC protected DPU-to-DPU communications. *Electron. Lett.* **2023**, *59*, e12901. [CrossRef]
32. Lawo, D.C.; Frantz, R.; Aguilera, A.C.; Clemente, X.A.I.; Podleś, M.P.; Imaña, J.L.; Monroy, I.T.; Olmos, J.J.V. Falcon/Kyber and Dilithium/Kyber Network Stack on Nvidia's Data Processing Unit Platform. *IEEE Access* **2024**, *12*, 38048–38056. [CrossRef]
33. Aguilera, A.C.; Abu Bakar, R.; Alhamed, F.; Garcia, C.R.; Imaña, J.; Monroy, I.T.; Cugini, F.; Olmos, J.V. First Line-rate End-to-End Post-Quantum Encrypted Optical Fiber Link Using Data Processing Units (DPUs). In Proceedings of the 2024 Optical Fiber Communications Conference and Exhibition (OFC), San Diego, CA, USA, 26–28 March 2024; pp. 1–3.
34. Alia, O.; Huang, A.; Luo, H.; Amer, O.; Pistoia, M.; Lim, C. Quantum-safe 10 Gbps Site-to-Site IPsec VPN Tunnel over 46 km Deployed Fibre. In Proceedings of the Optical Fiber Communication Conference (OFC) 2024, San Diego, CA, USA, 24–28 March 2024; Optica Publishing Group: Washington, DC, USA, 2024; p. Th3B.5. [CrossRef]
35. Rencis, E.; Vīksna, J.; Kozlovičs, S.; Celms, E.; Lāriņš, D.J.; Petručeņa, K. Hybrid QKD-based framework for secure enterprise communication system. *Procedia Comput. Sci.* **2024**, *239*, 420–428. [CrossRef]

36. Bae, S.; Chang, Y.; Park, H.; Kim, M.; Shin, Y. A Performance Evaluation of IPsec with Post-Quantum Cryptography. In *Information Security and Cryptology—ICISC 2022*; Seo, S.H., Seo, H., Eds.; Springer: Cham, Switzerland, 2023; pp. 249–266.
37. Kumar, S.; Dalal, S.; Dixit, V. The osi model: Overview on the seven layers of computer networks. *Int. J. Comput. Sci. Inf. Technol. Res.* **2014**, *2*, 461–466.
38. Hamed, H.; Al-Shaer, E.; Marrero, W. Modeling and verification of IPSec and VPN security policies. In Proceedings of the 13TH IEEE International Conference on Network Protocols (ICNP'05), Boston, MA, USA, 6–9 November 2005; pp. 10–278. [CrossRef]
39. Dhall, H.; Dhall, D.; Batra, S.; Rani, P. Implementation of IPSec Protocol. In Proceedings of the 2012 Second International Conference on Advanced Computing & Communication Technologies, Rohtak, India, 7–8 January 2012; pp. 176–181. [CrossRef]
40. Sadikin, M.A.; Wardhani, R.W. Implementation of RSA 2048-bit and AES 256-bit with digital signature for secure electronic health record application. In Proceedings of the 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA), Lombok, Indonesia, 28–30 July 2016; pp. 387–392. [CrossRef]
41. Maurer, U.M.; Wolf, S. The Diffie–Hellman Protocol. *Des. Codes Cryptogr.* **2000**, *19*, 147–171. [CrossRef]
42. Gentry, C.; Peikert, C.; Vaikuntanathan, V. Trapdoors for Hard Lattices and New Cryptographic Constructions. Cryptology ePrint Archive, Paper 2007/432. 2007. Available online: https://eprint.iacr.org/2007/432 (accessed on 20 January 2024).
43. Soni, D.; Basu, K.; Nabeel, M.; Aaraj, N.; Manzano, M.; Karri, R. Hardware Architectures for Post-Quantum Digital Signature Schemes. In *Hardware Architectures for Post-Quantum Digital Signature Schemes*; Springer International Publishing: Cham, Switzerland, 2021. [CrossRef]
44. Aragon, N.; Barreto, P.; Bettaieb, S.; Bidoux, L.; Blazy, O.; Deneuville, J.C.; Gaborit, P.; Ghosh, S.; Gueron, S.; Güneysu, T.; et al. BIKE: Bit Flipping Key Encapsulation. 2022. Available online: https://bikesuite.org/ (accessed on 15 January 2024).
45. Jao, D.; Azarderakhsh, R.; Campagna, M.; Costello, C.; De Feo, L.; Hess, B.; Jalili, A.; Koziel, B.; LaMacchia, B.; Longa, P.; et al. SIKE: Supersingular Isogeny Key Encapsulation. 2017. Available online: https://static1.squarespace.com/static/5fdbb09f31d71c1227082339/t/5ff378bdac5ecf06b683b05b/1609791681245/2017-ECCinvitedtalk.pdf (accessed on 15 January 2024).
46. Meher, K.; MidhunChakkaravarthy, D. New Approach to Combine Secret Keys for Post-Quantum (PQ) Transition. *Indian J. Comput. Sci. Eng.* **2021**, *12*, 629–633.
47. Suzuki, T.; Kim, S.Y.; Kani, J.i.; Yoshida, T. Low-latency PON PHY implementation on GPUs for fully software-defined access networks. *IEEE Netw.* **2022**, *36*, 108–114.

future internet

MDPI

Article

An Intelligent System for Determining Driver Anxiety Level: A Comparison Study of Two Fuzzy-Based Models

Yi Liu [1,*] and Leonard Barolli [2,*]

[1] Department of Computer Science, National Institute of Technology, Oita College, 1666 Maki, Oita 870-0152, Japan
[2] Department of Information and Communication Engineering, Fukuoka Institute of Technology (FIT), 3-30-1 Wajiro-Higashi, Higashi-Ku, Fukuoka 811-0295, Japan
* Correspondence: ryuui1010@gmail.com (Y.L.); barolli@fit.ac.jp (L.B.)

Abstract: While driving, stress and frustration can affect safe driving and pose the risk of causing traffic accidents. Therefore, it is important to control the driver's anxiety level in order to improve the driving experience. In this paper, we propose and implement an intelligent system based on fuzzy logic (FL) for deciding the driver's anxiety level (DAL). In order to investigate the effects of the considered parameters and compare the evaluation results, we implement two models: DAL Model 1 (DALM1) and DAL Model 2 (DALM2). The input parameters of DALM1 include driving experience (DE), in-car environment conditions (IECs), and driver age (DA), while for DALM2, we add a new parameter called the accident anxiety state (AAS). For both models, the output parameter is DAL. We carried out many simulations and compared the results of DALM1 and DALM2. The evaluation results show that the DAL is very good for drivers' ages between 30 to 50 years old. However, when the driver's age is below 30 or above 50, DAL tends to decline. With an increase in DE and IECs, the DAL value is decreased. But when the AAS is increased, the DAL is increased. DALM2 is more complex because the rule base is larger than DALM1, but it makes a better decision of DAL value.

Keywords: VANETs; fuzzy logic; driver anxiety level

Citation: Liu, Y.; Barolli, L. An Intelligent System for Determining Driver Anxiety Level: A Comparison Study of Two Fuzzy-Based Models. *Future Internet* **2024**, *16*, 348. https://doi.org/10.3390/fi16100348

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 9 August 2024
Revised: 14 September 2024
Accepted: 17 September 2024
Published: 24 September 2024

1. Introduction

As the number of COVID-19 infections declines, economic activities are resuming, and the traffic volume will be increased in the future. Thus, drivers may face heightened stress due to traffic congestion, which could lead to irritation. Such emotions can impair the ability to drive safely and increase the risk of causing traffic accidents. Driving a car is a constant source of stress, as it demands that drivers remain vigilant of surrounding traffic conditions in order to make appropriate decisions and precise maneuvers. A survey on Japanese drivers' behaviors indicates that the primary emotions experienced while driving include anger/irritation, fear of accidents, and impatience, as shown in Figure 1 [1].

There are many factors that cause these emotions such as the behavior of other drivers with aggressive cutting in, dangerous actions of cyclists and pedestrians, and the surrounding traffic environment, such as traffic jams. Also, when other vehicles do not adhere to traffic rules or are driven aggressively, it often irritates many drivers. Controlling emotions is crucial for safe driving. Regardless of driving proficiency, the inability to manage emotions, especially under significant stress, can impair concentration. This often results in hazardous driving behaviors, potentially leading to severe accidents [2].

Substantial research is currently being conducted by industry, governmental bodies, and academic entities to develop effective systems and infrastructure for preventing car accidents. This has led to an increased emphasis on collaboration among researchers, culminating in the creation of intelligent transportation systems (ITSs).

ITSs are dedicated to the application of advanced transportation technologies, merging state-of-the-art information, communication, and control technologies to craft sustainable

networks that integrate people, vehicles, and roadways. Within ITSs, vehicular ad hoc networks (VANETs) provide critical roles, primarily focusing on enhancing safety, streamlining traffic flow, boosting efficiency, and facilitating the convenience of both drivers and passengers. In VANETs, vehicles, functioning as network nodes, are equipped with capabilities to relay vital data, including safety alerts and road or traffic updates, among themselves through vehicle-to-vehicle interactions and to roadside units via vehicle-to-infrastructure communication. Despite the practical deployment of VANETs incorporating various applications, existing architectures encounter a myriad of challenges.

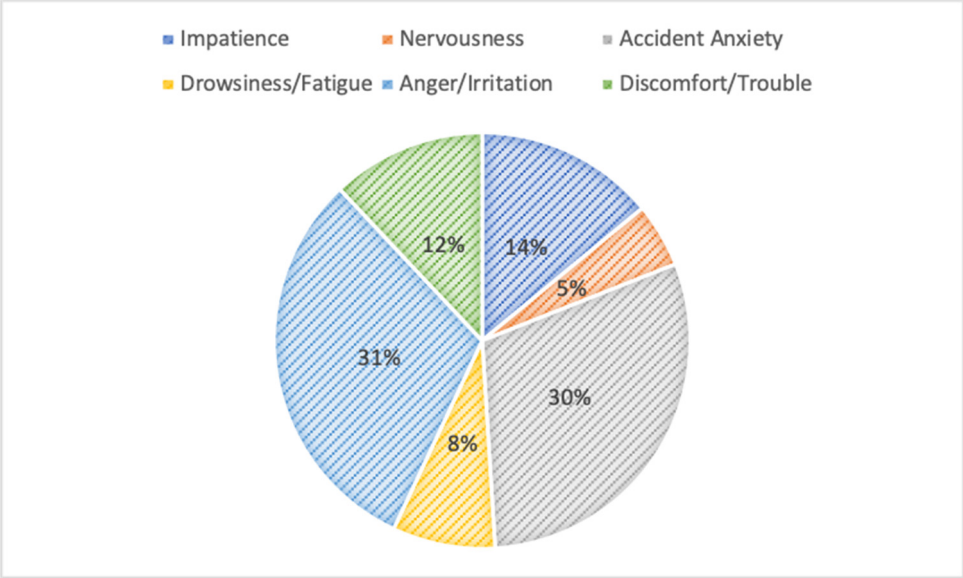


Figure 1. Classification of experienced emotions.

Driving anxiety is related to many factors such as transportation, clinical, cognitive psychological, social, and technical aspects. However, most of the studies about driving anxiety are based on questionnaires, and they investigate only the feelings and emotions associated with this problem, while the quantification of its detrimental effects while driving is rarely undertaken. Most of the behavioral measures are determined by accident reports and assessments of subjective experiences. They are mainly related to the extent of driving avoidance in different situations [3–8].

Fuzzy logic (FL) significance arises from the general nature of human reasoning, which is typically approximate, particularly in the realm of common sense. FL employs linguistic variables to articulate control parameters, allowing complex issues to be expressed and understood through straightforward linguistic terms. Additionally, the use of linguistic variables enables the description of vague parameters. Fuzzy sets, representing linguistic labels, deploy membership functions to rank preferences for the potential interpretations of these labels. There are many applications of FL in different fields of science and engineering [9,10].

This paper presents an intelligent FL-based system for deciding the driver's anxiety level (DAL). In order to investigate the effects of the considered parameters and compare the evaluation results, we implemented two models: DAL Model 1 (DALM1) and DAL Model 2 (DALM2). The input parameters of DALM1 were the driving experience (DE), in-car environment conditions (IECs), and the driver's age (DA), while for DALM2, we added a new parameter called the accident anxiety state (AAS). For both models, the output parameter was DAL. We carried out many simulations and compared the results of DALM1 and DALM2. The evaluation results show that DALM2 is more complex because the rule base is larger than DALM1, but it makes a better decision of DAL value.

By our study, we aimed to close the gap between psychological and emotional fields and the technical approaches. To the best of our knowledge, there are no tech-

nical papers that consider FL to determine the driver's anxiety level. Presently, in different fields, researchers have used artificial intelligence (AI) and machine learning (ML) algorithms [11,12]. However, they are used as black boxes. By using FL, we can build explainable intelligent systems.

The organization of this paper is outlined as follows: Section 2 introduces the concept of VANETs. A brief overview of FL and a FL controller (FLC) is provided in Section 3. The design of our FL-based system is presented in Section 4. In Section 5, we present the simulation results, while in Section 6, we expand on them through discussion. The paper concludes with a summary and suggestions for prospective research in Section 7.

2. VANETs

A VANET (vehicular ad hoc network) is a type of mobile ad hoc network (MANET), which can facilitate the exchange of information between vehicles as well as between vehicles and roadside infrastructure. These networks can support V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communication. They are expected to evolve with advancements in autonomous vehicles, smart cities and 5G technology. More sophisticated applications such as platooning, where vehicles travel in a group formation to reduce fuel consumption and improve traffic flow, will be integrated into VANETs. Thus, VANETs represent a crucial advancement in transportation technology, promising significant improvements in safety, efficiency, and convenience for future road networks.

Key components of VANETs are as follows.

- Vehicles: The vehicles are equipped with Onboard Units (OBUs) that facilitate communication with other vehicles and roadside units.
- Roadside units (RSUs): The RSUs are fixed units along the road, which provide connectivity to vehicles, serving as gateways to the internet and other networks.
- OBUs: The OBUs are communication devices installed in vehicles, which enable the transmission and reception of data.
- Application Units (AUs): The AUs are part of vehicle's onboard system, which utilize VANET data to operate various applications.

In VANETs, the V2V communication is the direct communication between vehicles, which is essential for collision avoidance, traffic information dissemination, and cooperative driving. While V2I refers to the communication between vehicles and infrastructure, such as traffic signals or toll booths. This facilitates traffic management, toll collection, and the provision of real-time traffic updates. While VANETs possess many advantages, they have extensive and constantly changing topologies as well as fluctuations in the capacity of wireless links. They are constrained by bandwidth and hard delay constraints. Also, the high mobility, speed, and low density of vehicles lead to brief contact durations. In addition, limited transmission ranges, physical barriers, and interference result in intermittent and disrupted connectivity.

For the implementation of VANET applications, it is essential to develop appropriate networking mechanisms capable of addressing challenges inherent in vehicular environments [13–19].

Mobility pattern recognition is a complex task in VANETs because the driving state of each vehicle is different. For traffic control and accident prevention, the human driver behavior should be analyzed to identify mobility patterns and predict the driver's state [20]. The diver's state is affected by different factors such as the external environment, the car safety device, their driving skill, driver emotions, driver age, transportation information, and so on, as shown in Figure 2.

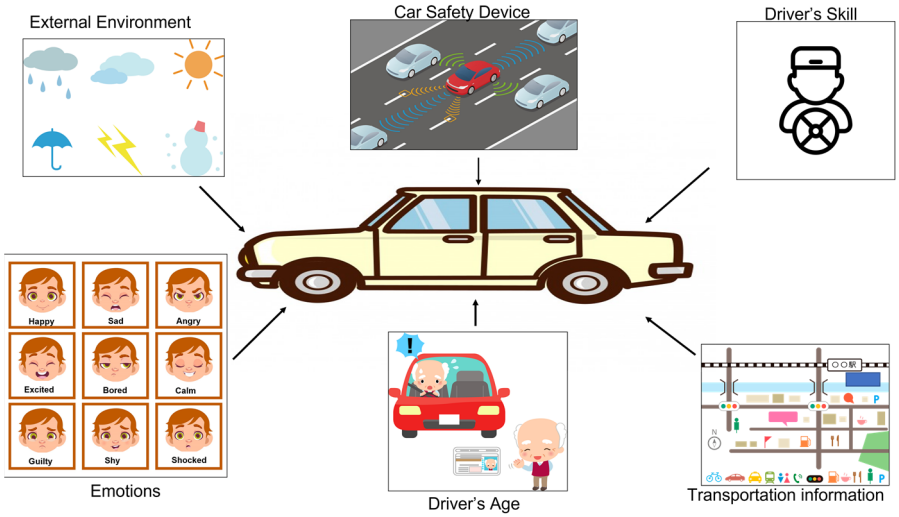


Figure 2. Factors affecting driver's state.

3. FC and FLC

The FL system considers values between true and false, rather than being confined to binary extremes, thus enhancing traditional logic frameworks. It can process both linguistic and numerical data by mapping from input vectors to a singular output value via an online algorithm. Unlike conventional binary logic, which operates strictly at 0 or 1, FL functions across a continuum from 0 to 1. This allows for more nuanced interpretations of data, aligning well with the mimicking complex, nonlinear relationships and managing situations with varying degrees of certainty. Consequently, FL offers a closer approximation to human reasoning and decision-making processes. Also, it proves beneficial in a variety of applications.

Fuzzy control (FC) emerges as the predominant field of application. The FLC process is initiated with fuzzification, in which crisp numerical input values are transformed into fuzzy sets based on predefined linguistic variables. This transformation enables the system to manage imprecision and ambiguity effectively. Such a step is crucial for translating real-world sensor data into formats that the FLC can interpret in terms of degrees of truth, rather than binary on/off states. The inference engine assesses the fuzzy inputs along with the rule base, which consists of a collection of if–then rules encoding expert knowledge and decision logic. These rules determine the output of the controller in fuzzy terms, taking into account all possible input combinations and their respective actions.

Finally, the defuzzification mechanism transforms the set of fuzzy outputs into a crisp value, which is subsequently utilized for control purposes. This output is a single value representing the aggregated result of the fuzzy inference process, optimized to achieve the desired control objective. Consequently, the FLC can make decisions that mimic human reasoning, facilitating sophisticated control strategies in complex and nonlinear systems [21–24].

4. Proposed Simulation System

In this paper, we consider FL to implement the proposed simulation system. The proposed system structure is shown in Figure 3. In the implemented system, sensors will detect and quantify the input parameters for the FL-based system.

We implement two models: DALM1 and DALM2. The input parameters of DALM1 include driving experience (DE), in-car environment conditions (IECs), and driver age (DA), while for DALM2, we add a new parameter called the accident anxiety state (AAS). For both models, the output parameter is the DAL. We explain each parameter in the following paragraphs.

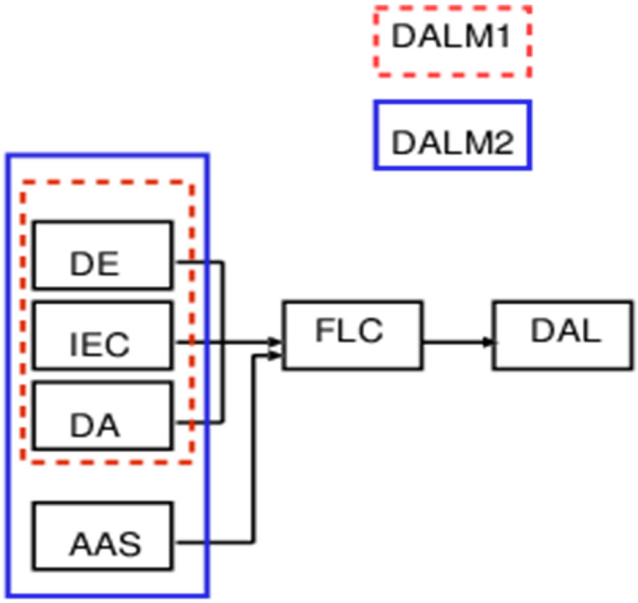


Figure 3. Structure of the proposed system.

- Driving Experience (DE)

In general, it is known that experienced drivers are able to make more appropriate responses than inexperienced drivers. Also, the driving experience can affect the traffic safety behavior. Specifically, young people have peculiar driving behavior such as driving faster, decelerating and accelerating more abruptly, being less likely to come to a full stop at stop signs and tailgating other cars, in comparison to middle-aged and older drivers [25].

- In-car Environment Conditions (IECs)

The IECs are related to the vehicle conditions such as the air quality aspects influencing environmental conditions in vehicles. Especially with the mobile age and associated lengthy periods of stay in enclosed cars, maintaining healthy air hygiene in the car interior is very important. There are different factors that influence the vehicle environment such as temperature, window-opening, the correct usage of automated air conditioning systems, and indoor air filters. These factors are useful for improving the environmental conditions of vehicles and optimizing the interior air hygiene. These are related also with the different endowments, ages and models of vehicles [26].

- Driver Age (DA)

Aging combined with injury or disease can affect the physical and mental capabilities required for the task of driving. There are different difficulties for older people such as maintaining a constant speed and keeping within the lane, turning the head and body during parking, the reduced field of view and unintentional speeding, identifying road signs and driving at night, reduced reactions and longer decision times, and difficulty with ingress/egress. Thus, the age of drivers has a great effect on DAL [27].

- Accident Anxiety State (AAS)

The AAS refers to a specific type of anxiety experienced by individuals who have been involved in, witnessed, or even heard about car accidents. This anxiety can significantly affect the person ability to drive or even be a passenger in a vehicle. The symptoms and impacts of the driver AAS include physical symptoms, emotional symptoms, behavioral symptoms, cognitive symptoms, and so on. The AAS can severely limit the person independence and freedom, making it challenging to carry out daily activities that involve driving. It can also affect one's ability to commute to work, attend social events, or perform assigned errands [28].

The terms set for input and output parameters are defined as shown in Table 1. For the implementation of the proposed system, we consider triangular and trapezoidal membership functions as shown in Figure 4 because they are good for real-time systems and applications [29]. The membership functions are shown in Figure 5. We set the DE, IEC, AAS and DAL values to range from 0 to 1 unit. The DE value range we set from 20 to 90 considering the driver ages in Japan [25]. The DAL is considered in 6 levels for DALM1 and 7 levels for DALM2. When the DAL value is DAL1, the amount of the driver's anxiety is good, while when it is DAL6 (DALM1) or DAL7 (DALM2), this is the worst case.

Table 1. Parameters and term set.

Parameters	Term Sets
Driving Experience (DE)	Not Good (NG), Good (G), Very Good (VG)
In-car Environment Conditions (IECs)	Bad (Ba), Normal (Nor), Good (Gd)
Driver Age (DA)	Young (Yo), Middle (Mi), Old (Ol)
Accident Anxiety State (AAS)	Low (Lo), Middle (Mid), High (Hi)
DALM1: Driver Anxiety Level (DAL)	Driver Anxiety Level1 (DAL1), DAL2, DAL3, DAL4, DAL5, DAL6
DALM2: DAL	DAL1, DAL2, DAL3, DAL4, DAL5, DAL6, DAL7

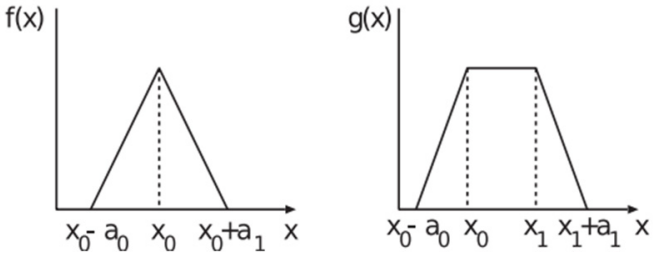


Figure 4. Triangular and trapezoidal membership functions.

The fuzzy rule base (FRB) for DALM1 is shown in Table 2 and has 27 rules, while the FRB for DALM2 is shown in Table 3 and has 81 rules. The control rules are constructed by "if . . . then . . ." expressions, such as the following:

Table 2. FRB for DALM1.

Rule	DE	IEC	DA	DAL	Rule	DE	IEC	DA	DAL	Rule	DE	IEC	DA	DAL
1	NG	Ba	Yo	DAL6	10	G	Ba	Yo	DAL5	19	VG	Ba	Yo	DAL4
2	NG	Ba	Mi	DAL4	11	G	Ba	Mi	DAL3	20	VG	Ba	Mi	DAL2
3	NG	Ba	Ol	DAL6	12	G	Ba	Ol	DAL6	21	VG	Ba	Ol	DAL5
4	NG	Nor	Yo	DAL5	13	G	Nor	Yo	DAL4	22	VG	Nor	Yo	DAL3
5	NG	Nor	Mi	DAL3	14	G	Nor	Mi	DAL2	23	VG	Nor	Mi	DAL1
6	NG	Nor	Ol	DAL6	15	G	Nor	Ol	DAL5	24	VG	Nor	Ol	DAL4
7	NG	Gd	Yo	DAL3	16	G	Gd	Yo	DAL2	25	VG	Gd	Yo	DAL2
8	NG	Gd	Mi	DAL2	17	G	Gd	Mi	DAL1	26	VG	Gd	Mi	DAL1
9	NG	Gd	Ol	DAL5	18	G	Gd	Ol	DAL4	27	VG	Gd	Ol	DAL3

If the DE is NG, the IECs are Ba and the DA is Yo, then the DAL is DAL6 (Rule 1 of Table 2), or If the AAS is Hi, the DE is VG, the IECs are Gd and the DA is Ol, then the DAL is DAL3 (Rule 81 of Table 3).

We explain the membership functions as follows. The triangular membership function is denoted by $f()$, while the trapezoidal membership function is denoted by $g()$. The maximum value for the triangular membership function is denoted by x_0, while the edges are denoted by $x_0 - a_0$ and $x_0 + a_1$, for the left and right sides, respectively. For the trapezoidal membership function, the maximum values are denoted by x_0 and x_1, while

the edges are denoted by $x_0 - a_0$ and $x_1 + a_1$, for the left and right sides, respectively. The FRB for both models is tuned based on our experience and the related work on a driver's anxiety [25–28]. When we want a strong effect of the parameter for a peak value, we select the triangular membership function, while in cases when we want the strong effect of the parameter in a region, we use the trapezoidal membership function.

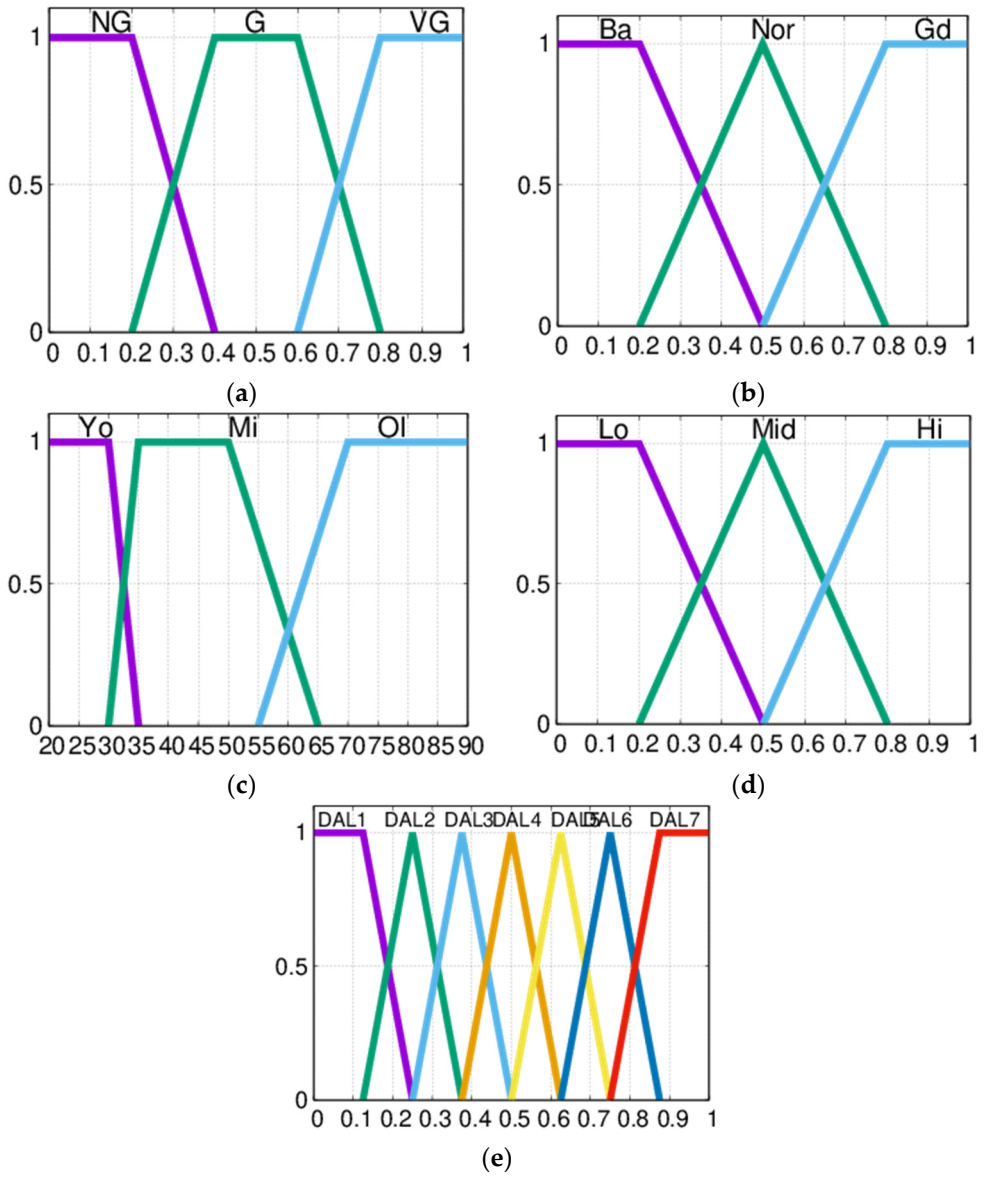


Figure 5. Membership functions for parameters. (**a**) Driving Experience (DE). (**b**) In-car Environment Conditions (IECs). (**c**) Driver Age (DA). (**d**) Accident Anxiety State (AAS). (**e**) Driver Anxiety Level (DAL).

$$\mu_{NG}\,(DE) = g\,(DE; NG_0, NG_1, NG_{w0}, NG_{w1})$$

$$\mu_{G}\,(DE) = g\,(DE; G_0, G_1, G_{w0}, G_{w1})$$

$$\mu_{VG}\,(DE) = g\,(DE; VG_0, VG_1, VG_{w0}, VG_{w1})$$

$$\mu_{Ba}\,(IEC) = g\,(IEC; Ba_0, Ba_1, Ba_{w0}, Ba_{w1})$$

$$\mu_{Nor}\ (IEC) = f\ (IEC; Nor_0, Nor_{w0}, Nor_{w1})$$

$$\mu_{Gd}\ (IEC) = g\ (IEC; Gd_0, Gd_1, Gd_{w0}, Gd_{w1})$$

$$\mu_{Yo}\ (DA) = g\ (DA; Yo0, Yo1, Yow0, Yow1)$$

$$\mu_{Mi}\ (DA) = g\ (DA; Mi0, Mi1, Miw0, Miw1)$$

$$\mu_{Ol}\ (DA) = g\ (DA; Ol0, Ol1, Olw0, Olw1)$$

$$\mu_{Lo}\ (AAS) = g\ (AAS; Lo0, Lo1, Low0, Low1)$$

$$\mu_{Mid}\ (AAS) = f\ (AAS; Mid0, Midw0, Midw1)$$

$$\mu_{Hi}\ (AAS) = g\ (AAS; Hi0, Hi1, Hiw0, Hiw1)$$

Table 3. FRB for DALM2.

Rule	AAS	DE	IEC	DA	DAL	Rule	AAS	DE	IEC	DA	DAL	Rule	AAS	DE	IEC	DA	DAL
1	Lo	NG	Ba	Yo	DAL6	28	Mid	NG	Ba	Yo	DAL7	55	Hi	NG	Ba	Yo	DAL7
2	Lo	NG	Ba	Mi	DAL4	29	Mid	NG	Ba	Mi	DAL6	56	Hi	NG	Ba	Mi	DAL7
3	Lo	NG	Ba	Ol	DAL7	30	Mid	NG	Ba	Ol	DAL7	57	Hi	NG	Ba	Ol	DAL7
4	Lo	NG	Nor	Yo	DAL5	31	Mid	NG	Nor	Yo	DAL6	58	Hi	NG	Nor	Yo	DAL7
5	Lo	NG	Nor	Mi	DAL3	32	Mid	NG	Nor	Mi	DAL5	59	Hi	NG	Nor	Mi	DAL6
6	Lo	NG	Nor	Ol	DAL6	33	Mid	NG	Nor	Ol	DAL7	60	Hi	NG	Nor	Ol	DAL7
7	Lo	NG	Gd	Yo	DAL4	34	Mid	NG	Gd	Yo	DAL6	61	Hi	NG	Gd	Yo	DAL7
8	Lo	NG	Gd	Mi	DAL2	35	Mid	NG	Gd	Mi	DAL4	62	Hi	NG	Gd	Mi	DAL5
9	Lo	NG	Gd	Ol	DAL5	36	Mid	NG	Gd	Ol	DAL6	63	Hi	NG	Gd	Ol	DAL7
10	Lo	G	Ba	Yo	DAL4	37	Mid	G	Ba	Yo	DAL5	64	Hi	G	Ba	Yo	DAL6
11	Lo	G	Ba	Mi	DAL2	38	Mid	G	Ba	Mi	DAL3	65	Hi	G	Ba	Mi	DAL5
12	Lo	G	Ba	Ol	DAL5	39	Mid	G	Ba	Ol	DAL6	66	Hi	G	Ba	Ol	DAL7
13	Lo	G	Nor	Yo	DAL3	40	Mid	G	Nor	Yo	DAL4	67	Hi	G	Nor	Yo	DAL6
14	Lo	G	Nor	Mi	DAL1	41	Mid	G	Nor	Mi	DAL2	68	Hi	G	Nor	Mi	DAL4
15	Lo	G	Nor	Ol	DAL4	42	Mid	G	Nor	Ol	DAL5	69	Hi	G	Nor	Ol	DAL6
16	Lo	G	Gd	Yo	DAL2	43	Mid	G	Gd	Yo	DAL3	70	Hi	G	Gd	Yo	DAL5
17	Lo	G	Gd	Mi	DAL1	44	Mid	G	Gd	Mi	DAL2	71	Hi	G	Gd	Mi	DAL3
18	Lo	G	Gd	Ol	DAL3	45	Mid	G	Gd	Ol	DAL4	72	Hi	G	Gd	Ol	DAL6
19	Lo	VG	Ba	Yo	DAL2	46	Mid	VG	Ba	Yo	DAL3	73	Hi	VG	Ba	Yo	DAL5
20	Lo	VG	Ba	Mi	DAL1	47	Mid	VG	Ba	Mi	DAL2	74	Hi	VG	Ba	Mi	DAL3
21	Lo	VG	Ba	Ol	DAL3	48	Mid	VG	Ba	Ol	DAL4	75	Hi	VG	Ba	Ol	DAL6
22	Lo	VG	Nor	Yo	DAL1	49	Mid	VG	Nor	Yo	DAL2	76	Hi	VG	Nor	Yo	DAL4
23	Lo	VG	Nor	Mi	DAL1	50	Mid	VG	Nor	Mi	DAL1	77	Hi	VG	Nor	Mi	DAL2
24	Lo	VG	Nor	Ol	DAL2	51	Mid	VG	Nor	Ol	DAL3	78	Hi	VG	Nor	Ol	DAL5
25	Lo	VG	Gd	Yo	DAL1	52	Mid	VG	Gd	Yo	DAL2	79	Hi	VG	Gd	Yo	DAL3
26	Lo	VG	Gd	Mi	DAL1	53	Mid	VG	Gd	Mi	DAL1	80	Hi	VG	Gd	Mi	DAL1
27	Lo	VG	Gd	Ol	DAL1	54	Mid	VG	Gd	Ol	DAL2	81	Hi	VG	Gd	Ol	DAL4

The membership functions of DAL are defined as follows.

$$\mu_{DAL1}\ (DAL) = g\ (DAL; DAL10, DAL11, DAL1w0, DAL1w1)$$

$$\mu_{DAL2}\ (DAL) = f\ (DAL; DAL20, DAL2w0, DAL2w1)$$

$$\mu_{DAL3}\ (DAL) = f\ (DAL;\ DAL30,\ DAL3w0,\ DAL3w1)$$

$$\mu_{DAL4}\ (DAL) = f\ (DAL;\ DAL40,\ DAL4w0,\ DAL4w1)$$

$$\mu_{DAL5}\ (DAL) = f\ (DAL;\ DAL50,\ DAL5w0,\ DAL5w1)$$

$$\mu_{DAL6}\ (DAL) = f\ (DAL;\ DAL60,\ DAL6w0,\ DAL6w1)$$

$$\mu_{DAL7}\ (DAL) = g\ (DAL;\ DAL70,\ DAL71,\ DAL7w0,\ DAL7w1)$$

5. Simulation Results

In this section are presented the simulation results, illustrating the correlations between input and output parameters as shown in Figures 6–9. We discuss the relations among various parameters in DALM1 and DALM2. For implementation of our system, we use FuzzyC software developed in our Lab [2,29].

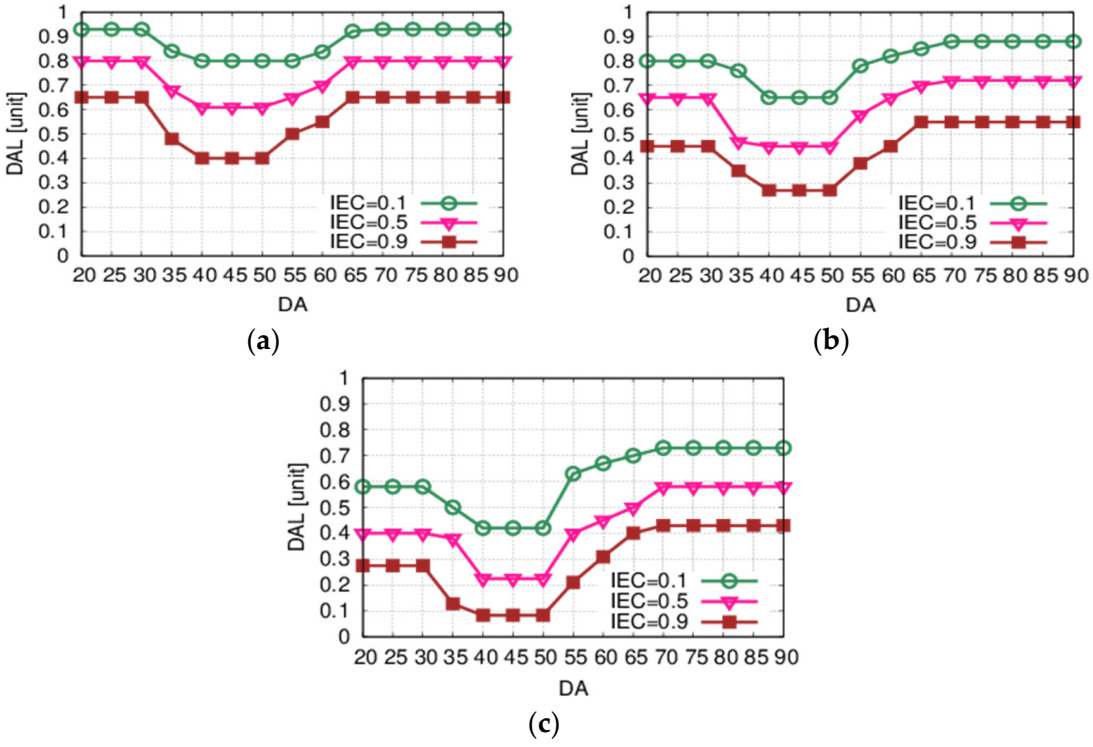


Figure 6. Relationship between DAL and DA for different values of IECs and DE. (**a**) DE = 0.1. (**b**) DE = 0.5. (**c**) DE = 0.9.

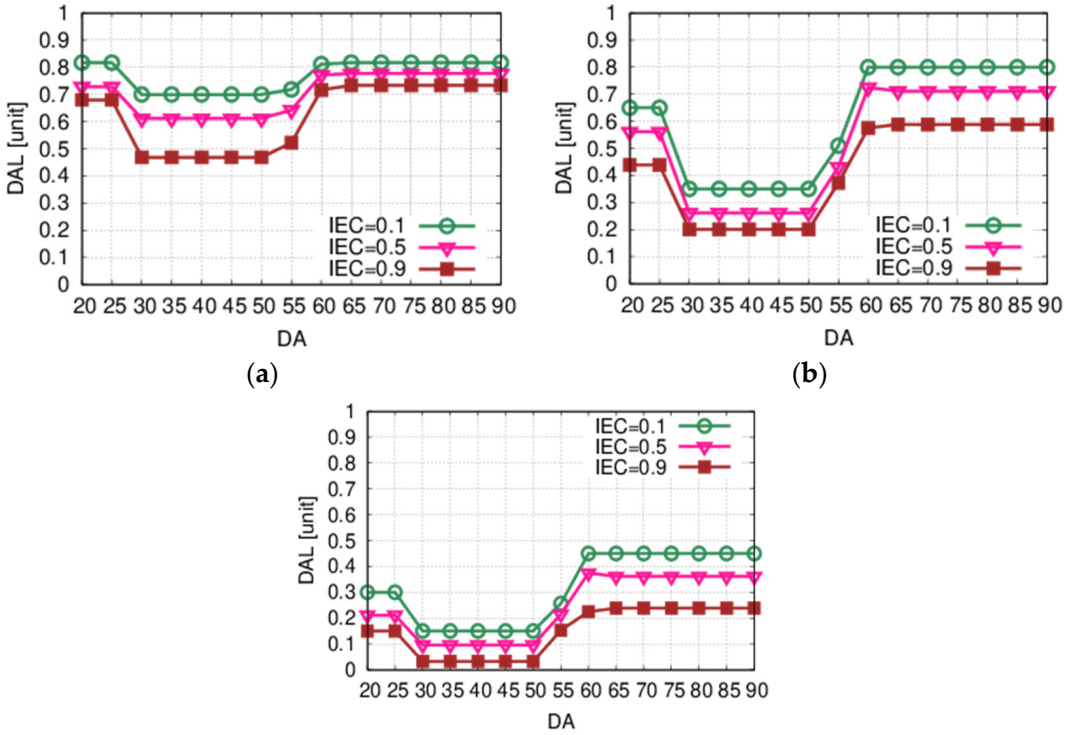


Figure 7. Relationship between DAL and DA for different values of IECs and DE when AAS is 0.1. (**a**) AAS = 0.1, DE = 0.1. (**b**) AAS = 0.1, DE = 0.5. (**c**) AAS = 0.1, DE = 0.9.

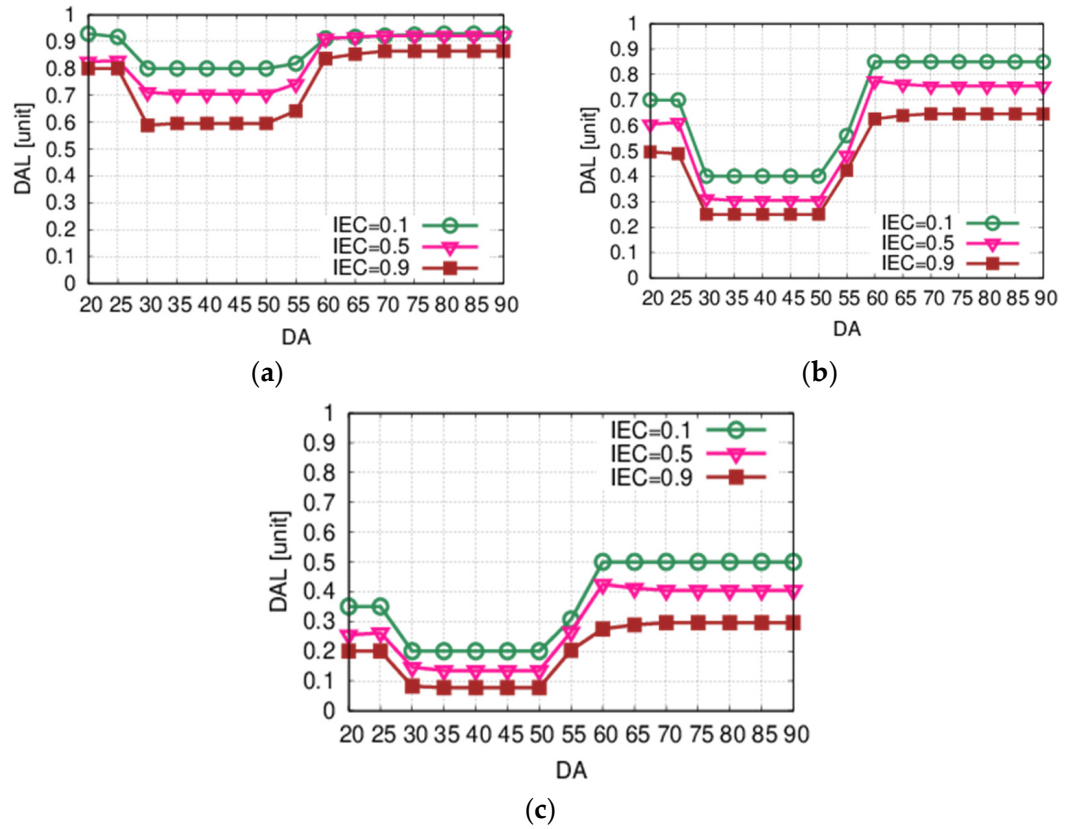


Figure 8. Relationship between DAL and DA for different values of IECs and DE when AAS is 0.5. (**a**) AAS = 0.5, DE = 0.1. (**b**) AAS = 0.5, DE = 0.5. (**c**) AAS = 0.5, DE = 0.9.

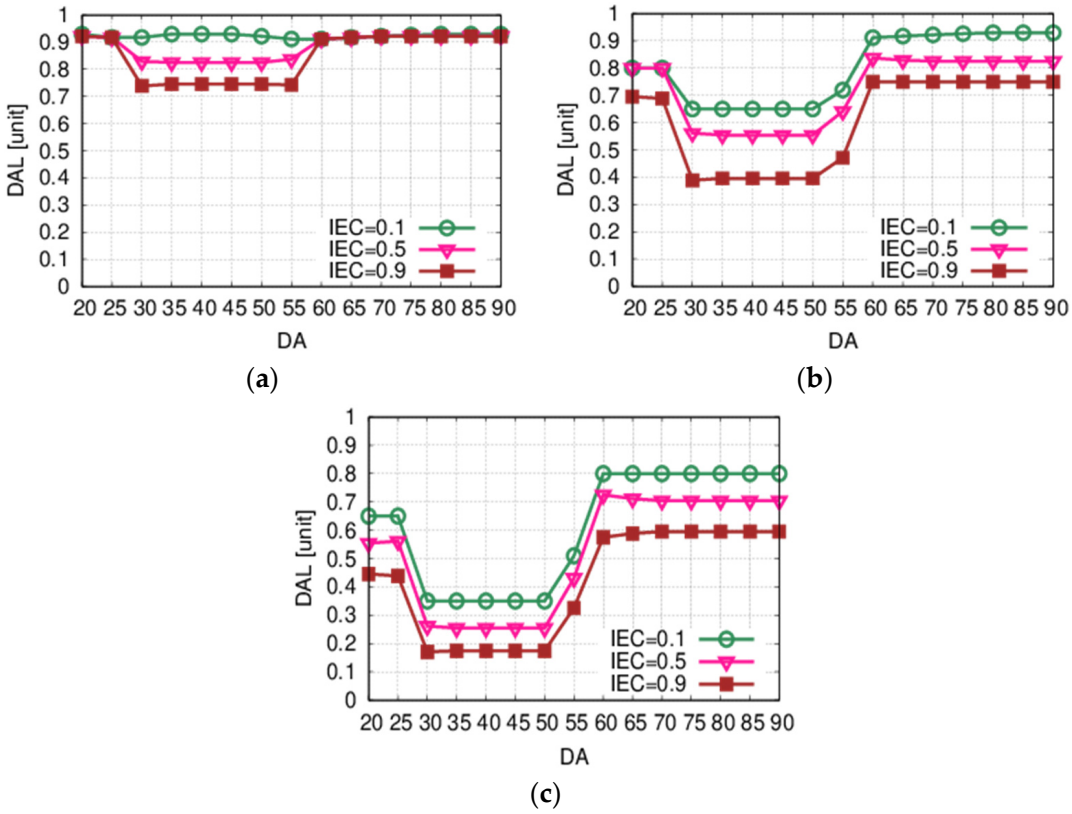


Figure 9. Relationship between DAL and DA for different values of IECs and DE when AAS is 0.9. (**a**) AAS = 0.9, DE = 0.1. (**b**) AAS = 0.9, DE = 0.5. (**c**) AAS = 0.9, DE = 0.9.

5.1. Simulation Results of DALM1

The simulation results of DALM1 are shown in Figure 6. We show the relationship between DAL and DA, DE and IECs input parameters. The DE is considered a constant parameter. We increased the values of the DE and IECs from 0.1 to 0.9 and the DA from 20 to 90.

In Figure 6a, when IEC is 0.1 and DA is 40, the DAL is 0.8. When the IEC value is increased to 0.5 and 0.9, the DAL is 0.62 and 0.4. So, the DAL value is decreased about 18% and 40%, respectively. In Figure 6b,c, we changed the DE to 0.5 and 0.9, respectively. We see that with increase of DE parameter, the DAL is decreased. For DE 0.5 and 0.9, comparing with Figure 6a, when the DA is 40 and the IEC is 0.9, the DAL is decreased by 12% and 28%, respectively.

5.2. Simulation Results of DALM2

The simulation results of DALM2 are shown in Figures 7–9. We show the relationship between DAL and DA for different values of IECs, AAS and DE. We investigate the effect of AAS on DAL, which is considered as a new parameter. We consider the AAS and DE as constant parameters and we change the IECs value from 0.1 to 0.9, and the DA value from 20 to 90.

We can see in Figure 7a that when the DA value is 40, IEC is 0.1, AAS is 0.1 and DE is 0.1, the DAL is 0.7. Comparing the same point with Figure 6, the DAL is decreased 10%. In Figure 7b, we changed the DE from 0.1 to 0.5. It can be seen that the DAL is decreased compared with Figure 7a. Also, in Figure 7c, the values of DAL have decreased much more.

In Figures 8 and 9, we changed the AAS values from 0.1 to 0.5 and 0.9. We see that for the same point when the DA value is 40, IECs is 0.1 and DE is 0.1, the DAL is 0.8 and 0.92, respectively. Comparing with the same point of Figure 7a, the DAL is increased by 10% and 22%, respectively.

Thus, when the AAS value is increased, the DAL in DALM2 increases much more compared with DALM1. This shows that the AAS has a great effect on the driver's anxiety level.

6. Discussion

In this section, we discuss the simulation results. In the case of DALM1, when dealing with inexperienced drivers (DE = 0.1), the results show the highest values of DAL. The low experience coupled with bad in-car environment conditions increase the driver's anxiety. For a moderate value of driver experience (DE = 0.5), the anxiety level is decreased, and the best behavior is that for middle-aged drivers. In the case of experienced drivers (DE = 0.9), we see that the anxiety level is very low, especially for middle-aged drivers and good in-car environment conditions.

For DALM2, we have scenarios: a low-level accident anxiety state (AAS = 0.1: Scenario 1), a moderate-level accident anxiety state (AAS = 0.5: Scenario 2) and a high-level accident anxiety state (AAS = 0.9: Scenario 3).

In the case of Scenario 1, the drivers have a low level of accident anxiety, so the driver experience coupled with in-car environment conditions and driver age have a great effect on the driver's anxiety level. The low level of DAL occurs when the driver is middle-aged, the in-car environment conditions are good and they have good experience. In the case of Scenario 2, the accident anxiety state is moderate, so the values of DAL are increased compared with Scenario 1. The values of DAL are increased more in Scenario 3 because the level of the accident anxiety state is high. The worst case of Scenario 3 occurs when the driving experience is low, in-car environment conditions are bad, with young and old-aged drivers.

7. Conclusions and Future Work

In this paper, we presented a FL-based system for assessing driver anxiety levels by considering various input parameters. We examined two models: DALM1 and DALM2. We evaluated the proposed models through simulations. From the evaluation results, we conclude that the DAL is very good for drivers within the ages of 30 to 50 years old. However, when the driver's age is below 30 or above 50, the DAL tends to decline. With increasing values of DE and IECs, the DAL value is decreased. But when the AAS is increased, the DAL is increased. This shows that the AAS has a great effect on the driver's anxiety level. Comparing complexity, DALM2 is more complex than DALM1. However, DALM2 also considers the AAS, which makes the system more reliable.

In future research, we would like to make extensive simulations to evaluate the proposed system. Also, we will implement a testbed and compare the simulation results with experimental results.

Author Contributions: Conceptualization, Y.L. and L.B.; methodology, Y.L.; software, L.B.; validation, Y.L; formal analysis, Y.L.; investigation, Y.L. and L.B.; resources, Y.L. and L.B.; data curation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, L.B.; visualization, Y.L.; supervision, L.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Driver Emotional Characteristics and Their Impact on Driving Behavior: Aiming to Develop an Educational Program for Emotion Control, International Association of Traffic and Safety Sciences (IATSS), Research Study, in Japanese. 2008. Available online: https://www.iatss.or.jp/research/entry_img/h076.pdf (accessed on 10 June 2024).
2. Bylykbashi, K.; Qafzezi, E.; Ampririt, P.; Ikeda, M.; Matsuo, K.; Barolli, L. Performance evaluation of an integrated fuzzy-based driving-support system for real-time risk management in VANETs. *Sensors* **2020**, *20*, 6537. [CrossRef]

3. Stephens, A.; Collette, B.; Hidalgo-Munoz, A.; Fort, A.; Evennou, M.; Jallais, C. The impacts of anxiety over driving on self-reported driving avoidance, work performance and quality of life. *J. Transp. Heal.* **2020**, *19*, 100929. [CrossRef]
4. Stephens, A.; Collette, B.; Hidalgo-Munoz, A.; Fort, A.; Evennou, M.; Jallais, C. Help-seeking for driving anxiety: Who seeks help and how beneficial is this perceived to be? *Transp. Res. Part F Traffic Psychol. Behav.* **2024**, *105*, 182–195. [CrossRef]
5. Pawar, N.M.; Yadav, A.K.; Velaga, N.R. A comparative assessment of subjective experience in simulator and on-road driving under normal and time pressure driving conditions. *Int. J. Inj. Control. Saf. Promot.* **2022**, *30*, 116–131. [CrossRef]
6. Brzezinska, D.; Bryant, P. Performance-based analysis in evaluation of safety in car parks under electric vehicle fire conditions. *Energies* **2022**, *15*, 649. [CrossRef]
7. Ertan, D.; Hubert-Jacquot, C.; Maillard, L.; Sanchez, S.; Jansen, C.; Fracomme, L.; Schwan, R.; Hopes, L.; Javelot, H.; Tyvaert, L.; et al. Anticipatory anxiety of epileptic seizures: An overlooked dimension linked to trauma history. *Seizure* **2021**, *85*, 64–69. [CrossRef]
8. Balzarotti, S.; Sullman, M.; Abati, D.; Biassoni, F. The expression of driving anger in a sample of Italian drivers. *Transp. Res. Part F Traffic Psychol. Behav.* **2023**, *97*, 383–395. [CrossRef]
9. Zimmermann, H.J. *Fuzzy Set Theory and Its Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 1996. [CrossRef]
10. Balaji, T.S.; Srinivasan, S.; Bharathi, S.P.; Ramesh, B. Fuzzy-Based Secure Clustering with Routing Technique for VANETs. *Comput. Syst. Sci. Eng.* **2022**, *43*, 291–304. [CrossRef]
11. Tselentis, D.I.; Papadimitriou, E.; van Gelder, P. The usefulness of artificial intelligence for safety assessment of different transport modes. *Accid. Anal. Prev.* **2023**, *186*, 107034. [CrossRef]
12. Chen, J.; Teo, T.H.; Kok, C.L.; Koh, Y.Y. A Novel Single-Word Speech Recognition on Embedded Systems Using a Convolution Neuron Network with Improved Out-of-Distribution Detection. *Electronics* **2024**, *13*, 530. [CrossRef]
13. Qu, F.; Wu, Z.; Wang, F.-Y.; Cho, W. A security and privacy review of VANETs. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 2985–2996. [CrossRef]
14. Zeadally, S.; Hunt, R.; Chen, Y.-S.; Irwin, A.; Hassan, A. Vehicular ad hoc networks (VANETS): Status, results, and challenges. *Telecommun. Syst.* **2012**, *50*, 217–241. [CrossRef]
15. Manvi, S.S.; Tangade, S. A survey on authentication schemes in VANETs for secured communication. *Veh. Commun.* **2017**, *9*, 19–30. [CrossRef]
16. Lee, B.-G.; Chung, W.-Y. Wearable glove-type driver stress detection using a motion sensor. *IEEE Trans. Intell. Transp. Syst.* **2016**, *18*, 1835–1844. [CrossRef]
17. Aljabry, I.A.; Al-Suhail, G.A. A survey on network simulators for vehicular ad-hoc networks (VANETS). *Int. J. Comput. Appl.* **2021**, *174*, 1–9. [CrossRef]
18. Babu, S.; Raj Kumar, P.A. A comprehensive survey on simulators, emulators, and testbeds for VANETs. *Int. J. Commun. Syst.* **2022**, *35*, e5123. [CrossRef]
19. Malik, S.; Sahu, P.K. A comparative study on routing protocols for VANETs. *Heliyon* **2019**, *5*, e02340. [CrossRef]
20. Swamynathan, C.; Ravi, V.; Ranganayakulu, D.; Kandasamy, R. Driver behaviour prediction and enhanced ad hoc on-demand distance vector routing protocol in VANET. *Int. J. Commun. Syst.* **2023**, *37*, e5650. [CrossRef]
21. Mendel, J.M.; John, R.I.; Liu, F. Interval type-2 fuzzy logic systems made simple. *IEEE Trans. Fuzzy Syst.* **2006**, *14*, 808–821. [CrossRef]
22. Ampririt, P.; Higashi, S.; Qafzezi, E.; Ikeda, M.; Matsuo, K.; Barolli, L. An intelligent fuzzy-based system for handover decision in 5G-IoT networks considering network slicing and SDN technologies. *Internet Things* **2023**, *23*, 100870. [CrossRef]
23. Mittal, K.; Jain, A.; Vaisla, K.S.; Castillo, O.; Kacprzyk, J. A comprehensive review on type 2 fuzzy logic applications: Past, present and future. *Eng. Appl. Artif. Intell.* **2020**, *95*, 103916. [CrossRef]
24. Ramesh, G.; Aravindarajan, V.; Logeshwaran, J.; Kiruthiga, T. Estimation analysis of paralysis effects for human nervous system by using Neuro fuzzy logic controller. *NeuroQuantology* **2022**, *20*, 3195–3206.
25. Begum, S. Intelligent driver monitoring systems based on physiological sensor signals: A review. In Proceedings of the 2013 16th International IEEE Conference on Intelligent Transportation Systems—(ITSC 2013), Hague, The Netherlands, 6–9 October 2013; pp. 282–289. [CrossRef]
26. Buera, L.; Lleida, E.; Miguel, A.; Ortega, A. Multi-environment models based linear normalization for speech recognition in car conditions. In Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Montreal, QC, Canada, 17–21 May 2004. [CrossRef]
27. Shea, A.; Evans, L. Traffic Safety and the Driver. *J. Oper. Res. Soc.* **1992**, *43*, 728. [CrossRef]
28. Măirean, C. Fear and avoidance of driving among drivers involved in a road traffic crash. The role of traumatic fear and driving cognitions. *Transp. Res. Part F Traffic Psychol. Behav.* **2020**, *74*, 322–329. [CrossRef]
29. Bylykbashi, K.; Elmazi, D.; Matsuo, K.; Ikeda, M.; Barolli, L. Effect of security and trustworthiness for a fuzzy cluster management system in VANETs. *Cogn. Syst. Res.* **2019**, *55*, 153–163. [CrossRef]

future internet

MDPI

Article

Empowering Healthcare: TinyML for Precise Lung Disease Classification

Youssef Abadade [1], Nabil Benamar [2,3], Miloud Bagaa [4,*] and Habiba Chaoui [1]

[1] System Engineering Laboratory, National School of Applied Siences, Ibn Tofail University of Kenitra, Kenitra B.P 242, Morocco; youssef.abadade@uit.ac.ma (Y.A.); habiba.chaoui@uit.ac.ma (H.C.)
[2] School of Technology, Moulay Ismail University of Meknes, Meknes 50050, Morocco; n.benamar@umi.ac.ma
[3] School of Science and Engineering, Al Akhawayn University in Ifrane, P.O. Box 104, Hassan II Avenue, Ifrane 53000, Morocco
[4] Department of Electrical and Computer Engineering, University of Quebec at Trois-Rivieres, Trois-Rivieres, QC G8Z 4M3, Canada
* Correspondence: miloud.bagaa@uqtr.ca

Abstract: Respiratory diseases such as asthma pose significant global health challenges, necessitating efficient and accessible diagnostic methods. The traditional stethoscope is widely used as a non-invasive and patient-friendly tool for diagnosing respiratory conditions through lung auscultation. However, it has limitations, such as a lack of recording functionality, dependence on the expertise and judgment of physicians, and the absence of noise-filtering capabilities. To overcome these limitations, digital stethoscopes have been developed to digitize and record lung sounds. Recently, there has been growing interest in the automated analysis of lung sounds using Deep Learning (DL). Nevertheless, the execution of large DL models in the cloud often leads to latency, dependency on internet connectivity, and potential privacy issues due to the transmission of sensitive health data. To address these challenges, we developed Tiny Machine Learning (TinyML) models for the real-time detection of respiratory conditions by using lung sound recordings, deployable on low-power, cost-effective devices like digital stethoscopes. We trained three machine learning models—a custom CNN, an Edge Impulse CNN, and a custom LSTM—on a publicly available lung sound dataset. Our data preprocessing included bandpass filtering and feature extraction through Mel-Frequency Cepstral Coefficients (MFCCs). We applied quantization techniques to ensure model efficiency. The custom CNN model achieved the highest performance, with 96% accuracy and 97% precision, recall, and F1-scores, while maintaining moderate resource usage. These findings highlight the potential of TinyML to provide accessible, reliable, and real-time diagnostic tools, particularly in remote and underserved areas, demonstrating the transformative impact of integrating advanced AI algorithms into portable medical devices. This advancement facilitates the prospect of automated respiratory health screening using lung sounds.

Keywords: TinyML; lung disease classification; early detection

Citation: Abadade, Y.; Benamar, N.; Bagaa, M.; Chaoui, H. Empowering Healthcare: TinyML for Precise Lung Disease Classification. *Future Internet* **2024**, *16*, 391. https://doi.org/10.3390/fi16110391

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 24 August 2024
Revised: 17 October 2024
Accepted: 21 October 2024
Published: 25 October 2024

1. Introduction

According to the World Health Organization (WHO), lung diseases are among the leading causes of mortality worldwide, resulting in the deaths of millions of people each year [1]. Respiratory diseases are often detected late, making treatment less effective [2].

Various clinical approaches have been developed to diagnose and assess lung health issues, including computed tomographic scans, chest X-rays, and pulmonary function tests. However, these techniques are restricted to specialized medical facilities due to their complexity, high cost, and time-consuming nature [3]. Additionally, medical professionals in hospitals are often overworked, which increases the likelihood of errors and patient waiting times [3]. Therefore, it becomes apparent that a different approach is needed to better assist practitioners in making an initial diagnosis.

In contrast, the stethoscope is used as a non-invasive and patient-friendly tool for diagnosing respiratory conditions through lung auscultation [4]. This procedure involves listening to the sounds produced by air moving in and out of the lungs.

During lung auscultation, experts are able to identify various abnormal respiratory sounds, like wheezing and crackling [4]. These sounds serve as indicators of possible respiratory conditions for the patient. However, traditional stethoscopes come with several associated challenges. Firstly, their effectiveness relies heavily on the physician's expertise and judgment, introducing potential for diagnostic errors [5]. Secondly, they lack a recording feature, preventing other medical personnel from analyzing the sounds heard during consultations [6]. Thirdly, they are not equipped with noise-canceling capabilities, making it difficult to hear lung sounds in noisy environments such as emergency rooms or busy clinics [7].

The digital stethoscope has introduced a new approach to auscultation, benefiting research, education, and clinical practice [8]. It digitizes lung sounds, allowing for recording and playback, which reduces reliance on a single physician's judgment and enables collaboration with other medical professionals [8]. It incorporates digital filters to eliminate noise and isolate the relevant acoustic signals within specific frequency bands [8]. This enhances diagnostic accuracy and improves clinical decision making. It also allows for the visualization and retrospective analysis of lung sounds. The integration of wireless transmission capabilities, such as Bluetooth or WiFi, with the digital stethoscope will facilitate remote diagnosis, greatly enhancing convenience and application in a variety of medical contexts [8].

In recent years, there has been growing interest in the automated analysis of lung sounds. By using machine learning, particularly Deep Learning (DL) techniques, the experience, quality of diagnosis, and care for both patients and healthcare professionals have significantly improved [3,8]. The utilization of DL algorithms to examine lung sound patterns captured by digital stethoscopes represents a promising approach for the early and precise detection of disease [9]. Moreover, these technologies aim not only to reduce dependency on specialist facilities but also to overcome the limitations of traditional stethoscopes, making diagnosis more accurate by removing human error [3].

However, coupling digital stethoscopes with DL presents certain limitations. DL algorithms require significant computational resources, posing challenges in resource-constrained environments [10,11]. Latency in cloud-based solutions can impact real-time analysis, particularly in areas with insufficient internet bandwidth for large data transmission [10–12], and privacy concerns arise when transmitting sensitive health data over the internet [10,11]. To address these limitations, Tiny Machine Learning (TinyML) [13] offers a compelling solution by enabling efficient ML codes to run on small, energy-efficient devices.

TinyML is a fast-growing field of ML including hardware, algorithms, and software that aims to facilitate running ML models on ultra-low-power devices having very limited power (under 1 mW), less memory, and limited processor capabilities [14]. TinyML offers tiny IoT devices the ability to analyze data collected by various sensors and act based on the decisions made by the embedded ML model without the need for the cloud. TinyML finds applications in diverse fields [11], including agriculture [11,15], healthcare [10,11,16], and environmental monitoring [11,17].

The hardware limitations of tiny IoT devices require the minimizing of the ML model in order to deploy it in extremely resource-limited devices. The minimization of the model can be performed by the following techniques: pruning and quantization [14]. These techniques aim to reduce the size of the ML model while trying not to impact its accuracy. The pruning technique is the process of removing unused weights in the model to increase speed inference and minimize its size, while quantization reduces the precision of the model parameters from floating-point (e.g., 32-bit) to lower (e.g., 8-bit) precision [11]; this decreases the model's memory footprint as well as the amount of processing required.

TinyML holds immense potential in the healthcare sector [10,11,16]. TinyML's ability to run directly on devices at the edge offers numerous advantages. One of the most

significant benefits is the ability to perform real-time data analysis without the need for continuous data transmission to centralized cloud systems [11,16,18]. This reduces latency, enhances data privacy by minimizing the transfer of sensitive patient data, and lowers dependency on reliable internet connections [10,16], which is particularly beneficial for remote health monitoring in remote and undeserved areas [12,16].

Many studies have demonstrated the practical applications of TinyML in healthcare, showcasing its potential to optimize real-time health monitoring and diagnostic tools. The authors of [19] optimized a Convolutional Neural Network (CNN) model through pruning and quantization, making it deployable on low-cost microcontrollers like the Raspberry Pi Pico and ESP32 for real-time blood pressure estimation using photoplethysmogram (PPG) signals. This approach enables efficient, low-power solutions for continuous blood pressure monitoring. The authors of [20] proposed a TinyML-based solution for predicting and detecting falls among elderly individuals, utilizing a wearable device placed on the leg to capture movement data. The system employs a nonlinear support vector machine classifier for real-time fall detection and prediction. Similarly, the authors of [21] used CNN models combined with audio data to detect falls. In [22], researchers developed a system that predicts blood glucose levels in individuals with type 1 diabetes by deploying DL models on edge devices. This system, which relies on recurrent neural networks (RNNs), processes continuous glucose monitoring (CGM) data on low-power, resource-constrained devices, enabling real-time monitoring without the need for cloud infrastructure. Additionally, the authors of [23] introduced a lightweight solution based on Temporal Convolutional Networks (TCNs) for heart rate estimation in wearable devices. By leveraging optimized TCN models, the system achieved accurate heart rate monitoring while maintaining low latency and energy consumption, making it suitable for use in resource-constrained environments like wearable health devices.

In this paper, we present a new approach focused on creating TinyML models to distinguish between asthma and non-asthma conditions by using lung sound recordings. We developed and compared various ML models based on different metrics. To ensure these models are suitable for cost-effective platforms such as the Arduino Nano 33 BLE, we employed quantization techniques.

The remainder of the paper is organized as follows: Section 2 reviews existing studies that have addressed similar challenges. Section 3 details the materials and methodologies used in our experiments. Section 4 provides a comprehensive analysis of the empirical results and their implications. Finally, Section 5 presents our conclusions and proposes directions for future research.

2. Related Works

Numerous research papers have examined the application of DL to identify patterns and distinguish among various lung conditions by using raw respiratory sound data.

The authors of [24] developed a framework for classifying various respiratory diseases by using lung sound recordings. They employed a CNN with Mel-Frequency Cepstral Coefficients (MFCC) for feature extraction. The proposed model achieved a classification accuracy of 95.7%, effectively distinguishing among different respiratory diseases, such as asthma, COPD, URTI, LRTI, and bronchiectasis, and a class representing healthy people.

In [25], the authors developed a framework for classifying lung sounds, addressing the challenge of noise interference from the heart and lung sounds. By utilizing two public datasets with 280 lung sounds of varying durations and sampling rates, the study preprocessed signals for uniformity and employed Mel-Frequency Cepstral Coefficients (MFCCs) [26] and Short-Time Fourier Transform (STFT) [27] for feature extraction. It tested several models, achieving the highest accuracy with an STFT+MFCC-ANN combination, demonstrating promising results for automatic respiratory diagnosis with high precision and recall rates.

In the paper [5], the authors evaluated the efficacy of various deep learning models for diagnosing respiratory pathologies by using lung auscultation sounds. The study

compared three deep learning models across both non-augmented and augmented datasets, revealing that the CNN–LSTM model outperformed others with high accuracy rates in all scenarios. Augmentation significantly enhances model performance, with the CNN–LSTM hybrid showing particular strength by combining the CNN's feature extraction capabilities with LSTM's [28] classification efficiency.

In [29], the authors explored the effectiveness of Mel Frequency Cepstral Coefficients (MFCCs) in classifying cough sounds for diagnosing five respiratory diseases, such as asthma and chronic obstructive pulmonary diseases (COPDs). The study employed a unique ensemble of recurrent neural network models with LSTM cells and tested various meta-classifiers, achieving over 87% accuracy. This approach underscores MFCCs' potential as standalone features for cough signal classification and suggests future directions, including further disease characteristic diagnosis and COVID-19 cough classification.

In [30], the authors tackled the challenge of detecting respiratory pathologies from sounds, using the ICBHI Benchmark dataset. Given the dataset's imbalance, the study employed a Variational Convolutional Autoencoder for data augmentation, alongside traditional oversampling techniques. A CNN was employed for classification into healthy, chronic, and non-chronic categories, achieving an F-score of 0.993 in the three-label classification. For the six-class classification, which included RTI, COPD, Bronchiectasis, Pneumonia, and Bronchiolitis, the CNN achieved an F-score of 0.99.

In [31], the authors developed a non-invasive method for classifying respiratory sounds by using an electronic stethoscope and audio recording software. By using MFCC with SVM and spectrogram images with CNN, they benchmarked the CNN's performance against the SVM method across various sound classifications. The CNN and SVM both reached 86% in distinguishing healthy versus pathological sounds; for rale, rhonchus, and normal sounds, the CNN achieved 76% and SVM 75%; in singular-sound-type classification, both achieved 80%. These results underline the effectiveness of CNNs and SVM in respiratory sound analysis.

In [32], the authors developed a system for diagnosing asthma using deep learning by analyzing respiratory sounds from asthmatic and non-asthmatic individuals. They developed a web interface and a mobile app for real-time prediction, aiding doctors in performing accurate diagnoses. Utilizing features such as chroma, RMS, Spectral centroid, Rolloff, and MFCCs, the ConvNet model demonstrated impressive performance metrics, including 99.8% accuracy, 100% sensitivity, 100% specificity, and a 99% F-score.

In [33], the authors proposed RDsLINet, a novel lightweight inception network for classifying a broad spectrum of respiratory diseases through lung sound signals. The framework involves preprocessing, melspectrogram image conversion, and classification via RDsLINet. The proposed RDsLINet achieved impressive accuracy rates: 96% for seven-class, 99.5% for six-class, and 94% for healthy vs. asthma classifications.

In [34], the authors proposed a novel approach to respiratory disease detection through a wearable auscultation device. They developed a Respiratory Sound Diagnosis Processor Unit (RSDPU) utilizing LSTM networks to analyze respiratory sounds in real time. The study highlights the implementation of Dynamic Normalization Mapping (DNM) to optimize quantization and reduce overfitting, crucial to maintaining model accuracy with limited computational resources. The hardware implementation of the RSDPU includes a noise corrector to enhance diagnostic reliability. The results show that the RSDPU achieved an 81.4% accuracy in abnormality diagnosis, with a minimal power consumption of 381.8 μW. The study demonstrates the potential of combining advanced machine learning techniques with efficient hardware design to create effective and practical healthcare solutions for continuous respiratory monitoring

A startup called Respira Labs [35] has introduced an innovative, cost-effective wearable sensor that leverages TinyML to analyze cough sounds for signs of respiratory diseases like pneumonia. This compact device integrates a microphone and a microcontroller executing a neural network to discern specific cough characteristics such as wheezing and

crackling. Designed for ease of use, it features a simple strap mechanism, operates without batteries, and communicates results via LED indicators and sound signals.

Table 1 summarizes existing research on lung disease detection and classification using audio data, covering diseases such as asthma, COPD, lung fibrosis, bronchitis, and pneumonia and various pathological lung sounds. Datasets vary from publicly available ICBHI 2017 to self-collected data, indicating diversity in data sources. The extracted audio features include STFT, MFCC, and spectrograms, with MFCC being the most common. The models used range from ANN, CNN, and LSTM, to hybrid CNN-LSTM, achieving high accuracy rates and mostly deploying solutions on cloud platforms. While some studies [33,34] have explored computation directly on edge devices, our work leverages TinyML to push the boundaries of what can be achieved on resource-limited hardware. This approach allows for efficient real-time analysis and model deployment on compact, low-power devices, making advanced diagnostics more accessible in a wide range of settings.

Table 1. Related work summary.

Work	Lung Diseases	Dataset	Feature(s)	Model	Results	Deployment
[24]	Asthma, COPD, URTI, LRTI, and bronchiectasis and normal	Respiratory Sound Database	MFCC	CNN	95.7%	Cloud
[25]	Normal and abnormal	ICBHI 2017 and KAUH	STFT and MFCC	ANN, SVM, KNN, DT, and RF	ANN: 98.61%; SVM: 93%; KNN: 93%; DT: 88%; RF: 95%	Cloud
[5]	Asthma, BRON, COPD, heart failure, lung fibrosis, normal, and pleural effusion pneumonia	ICBHI 2017 and KAUH	-	CNN and LSTM and CNN-LSTM	CNN: 99.81%; LSTM: 99.81%; CNN-LSTM: 100%	Cloud
[29]	Asthma, COPD, ILD, bronchitis, and pneumonia	Self-collected data	MFCC	LSTM	88.5%	Cloud
[30]	Healthy, chronic, and non-chronic	ICBHI 2017	MFCC	CNN	Accuracy: 99%	Cloud
[31]	Healthy and pathological	Self-collected data	MFCC and spectrogram	SVM-MFCC, CNN-Spectrogram	MFCC-SVM: 86%; Spectrogram-CNN: 86%	Cloud
[31]	Rale, rhonchus and normal	Self-collected data	MFCC and spectrogram	SVM-MFCC, CNN-Spectrogram	MFCC-SVM: 76%; Spectrogram-CNN: 76%	Cloud
[32]	Asthma and non-asthma	Self-collected data	Chroma, RMS, and MFCC	ConvNet	Accuracy: 99.8%	Mobile
[33]	Healthy and asthma	KAUH	Mel-spectrogram	RDsLINet	Accuracy: 94%	Edge
[33]	Asthma, BRON, COPD, pneumonia, heart failure, and pleural effusion	KAUH	Mel-spectrogram	RDsLINet	Accuracy: 94%	Edge
[34]	Normal and unnormal	ICBHI 2017	MFCC	LSTM	Accuracy: 81.4%	Edge

3. Materials and Methods

In this section, we describe the various methods employed to acquire, preprocess, and build models for distinguishing between normal and asthma conditions. The development process was conducted by using Edge Impulse, a platform specifically designed for training models, adjusting hyperparameters, and optimizing them for operation on various edge devices. Figure 1 depicts our approach, which involves the standard stages found in traditional machine learning projects, with added steps for real-time processing on the microcontroller:

Step 1: Data collection and preparation: Lung sound data were acquired from a publicly accessible dataset, divided into 80% for training and 20% for testing. These data were then processed by using bandpass filtering to reduce external noise.

Step 2: The lung sound data were uploaded to the Edge Impulse platform [36] and segmented into 5-second windows, with MFCC features extracted from each window.

Step 3: A custom CNN, a CNN proposed by Edge Impulse, and a custom LSTM were created, trained, and optimized for deployment on a microcontroller.

Step 4: The three models were tested by using the test data, and performance metrics were computed.

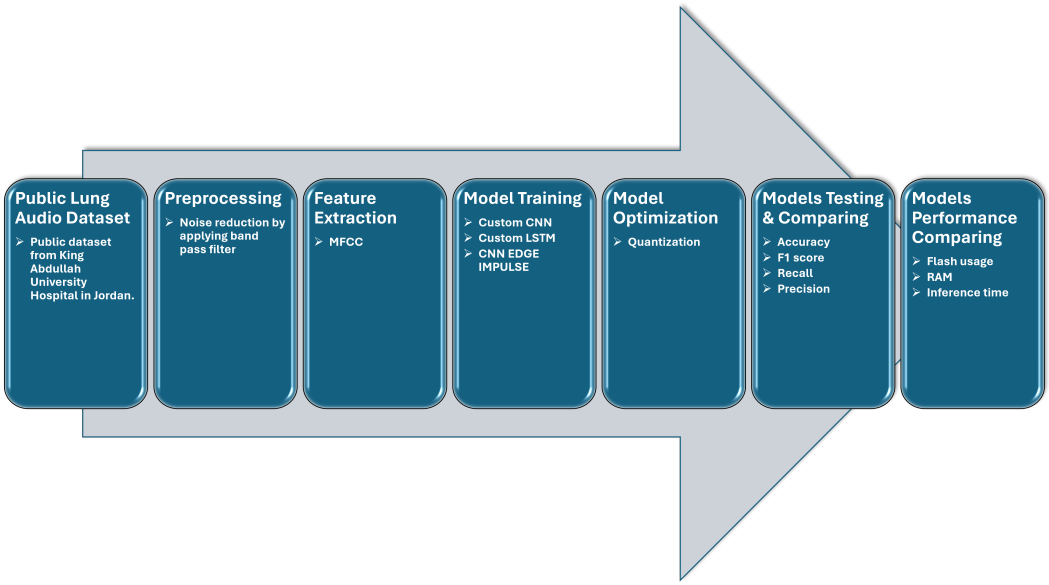


Figure 1. Workflow of lung sound analysis and model training process.

3.1. Dataset

In this study, we used a publicly available dataset from King Abdullah University Hospital in Jordan [37]. The dataset comprises 310 records from 105 patients with various respiratory conditions, including normal, asthma, pneumonia, heart failure, bronchiectasis, and chronic obstructive pulmonary disease. The duration of each recording ranges from 5 to 30 s, for a total of 88 min of data. The patients' ages range from 12 to 90 years, with a mean ± SD age of 48 ± 18 years. For each patient, three types of recordings were obtained, each using a different filtering mode—bell mode filtration, diaphragm mode filtration, and extended mode filtration—to minimize interference from heartbeats and external noise.

The audio recordings were captured by using a single channel at a sampling rate of 4 kHz. The audio files, in WAV format, were captured by using a single-channel stethoscope-based acquisition system (electronic stethoscope 3200; 3M Littmann) positioned at various locations on the chest wall. Table 2 presents the quantity of patients in each disease classification along with the associated count of recordings utilized from this dataset. The dataset demonstrates an unequal distribution, containing more data for the normal and asthma categories compared with the others. Given this situation, we decided to distinguish between two distinct kinds of lung sounds: normal and asthma. Before uploading data to

Edge Impulse, each audio file was processed with a 5th-order Butterworth bandpass filter, with upper and lower cut-off frequencies set to 100–1800 Hz, to reduce external noise [4,34].

Table 2. Demographic summary and recording details by disease category.

Category	Subjects	Age (Mean ± SD)	Number of Records	Duration
Normal	35 (11F, 24M)	43 ± 19	105	31 m 10 s
Asthma	32 (17F, 15M)	45 ± 15	94	28 m 06 s
Heart failure	21 (9F, 12M)	58 ± 18	56	13 m 59 s
Pneumonia	5 (2F, 3M)	55 ± 13	17	4 m 53 s
Bronchiectasis	3 (1F, 2M)	37 ± 26	9	2 m
COPD	9 (1F, 8M)	57 ± 9	29	07 m 54 s

Upon uploading our dataset, we proceeded to create an ML pipeline called an "impulse". This impulse comprises three primary building blocks: the input block, the processing block, and the learning block. The input block identifies the data type employed during the training of the model. It can either be an image or a time series. In our specific case, we utilized time series as the input data. The window size was set to 5000 ms, and the window increase was set to 5000 ms [34]. This duration is sufficient to cover at least one respiratory cycle, given the average resting respiration rates for adults (12–20 breaths per minute) [37].

3.2. Feature Extractor

Feature extraction is an essential phase in lung sound analysis, where raw audio signals are converted into useful representations for classification. This process includes several categories: time-domain, frequency-domain, and time–frequency-domain features. Time-domain features capture the temporal characteristics of lung sounds, including metrics such as the zero-crossing rate, root mean square, and signal envelope. Frequency-domain features provide insights into the energy distribution across different frequency bands and include measures like MFCCs. Time–frequency-domain features, like wavelet transform and spectrograms, present a combined view of both time and frequency characteristics [6].

MFCCs are commonly used as features [6,25,29] in lung analysis derived from the Fourier transform, which can capture the distribution of energy in different frequency bands. The MFCCs consist of a series of coefficients that capture the characteristics of the signal's spectrum. These coefficients are derived by taking the logarithm of the discrete cosine transform applied to the signal's spectrum. The particular parameters utilized to produce MFCCs are detailed in Table 3.

Table 3. Relevant parameters used to generate MFCCs.

Parameter	Value
Number of coefficients	13
Frame length	0.256
Frame stride	0.064
Filter number	20
FFT length	256

In order to obtain the MFCC parameters presented in Table 3, we were inspired by the methodology used in [34], where the authors tested multiple configurations of MFCC parameters to optimize model performance and resource consumption. Following a similar procedure, we conducted multiple iterative tests to explore different parameter settings and find the optimal balance between performance and computational efficiency. While we initially tested the parameters used in their study, we found that these configurations did

not yield the desired results in our specific case. As a result, we experimented with other configurations to achieve the best balance between model accuracy and resource efficiency.

Given the constraints of TinyML, where devices are resource-limited and power-sensitive, it was essential to choose parameters that minimized memory usage and computational time. The final selected parameters consume 38 KB of RAM and have a processing time of 763 ms, based on estimates provided by the Edge Impulse platform for an Arduino Nano 33 BLE [38].

3.3. Model Proposition

In this work, we developed three classifiers, i.e., a custom CNN model, the CNN model proposed by Edge Impulse, and a custom LSTM model, that can diagnose two distinct respiratory conditions, that is, asthma and normal. We drew inspiration from the existing literature [31,34], which demonstrated promising results in lung disease detection. Tables 4–6 illustrate the architectures of these three models.

Table 4. Our CNN model.

Custom CNN Model
Input(shape=(975,))
Reshape((975 / 13, 13))
Conv1D(32, kernel_size=3, activation='relu')
MaxPooling1D(pool_size=2)
BatchNormalization()
Dropout(20%)
Conv1D(64, kernel_size=3, activation='relu')
MaxPooling1D(pool_size=2)
BatchNormalization()
Dropout(20%)
Conv1D(128, kernel_size=3, activation='relu')
MaxPooling1D(pool_size=2)
BatchNormalization()
Dropout(20%)
Flatten()

Table 5. Custom LSTM model.

Our LSTM Model
Input(shape=(975,))
Reshape((975 / 13, 13))
LSTM(128)
Dropout(20%)
LSTM(64)
Dropout(20%)
LSTM(32)
Dropout(20%)
Dense(512, activation='relu')
Dropout(40)

Table 5. *Cont.*

Our LSTM Model
Dense(8, activation='relu')
Dropout(40%)
Dense(classes, activation='softmax')

Table 6. CNN model proposed by Edge Impule.

CNN Model Proposed by Edge Impule
Input(shape=(975,))
Reshape((975 / 13, 13))
Conv1D(8, kernel_size=3, activation='relu')
MaxPooling1D(pool_size=2)
Dropout(25%)
Conv1D(16, kernel_size=3, activation='relu')
MaxPooling1D(pool_size=2)
Dropout(25%)
Dense(classes, activation='softmax')

To tune the hyperparameters, a total of 396 combinations of epochs, learning rates, and mini-batch sizes were explored across all three models. Specifically, we tested epochs ranging from 10 to 300, with learning rates of 10^{-3}, 10^{-4}, 6×10^{-3}, and 6×10^{-4}, and mini-batch sizes of 32, 16, and 8. The Adam optimizer was consistently applied across all models to facilitate the optimization process. Table 7 summarizes the optimal hyperparameters, presenting the configurations that show the highest accuracy.

Table 7. Optimal hyperparameters showing the highest accuracy.

Model	**Learning Rate**	**Epochs**	**Batch Size**
Custom CNN	6×10^{-3}	300	32
CNN Edge Impulse	6×10^{-3}	100	16
Custom LSTM	6×10^{-3}	100	32

3.4. Target Device

The target device for deployment is an Arduino Nano 33 BLE Sense [38], featuring a Cortex-M4F 64 MHz processor, chosen for its compact size and versatility across various applications. With 256 kB of RAM and 1024 kB of ROM, the device has limited computational resources. To optimize model deployment, we used post-training quantization provided by the Edge Impulse platform, utilizing implementations from the TensorFlow Lite Micro library [39]. This technique reduces the precision of a model's internal representations by converting 32-bit floating-point parameters into lower-precision int8, significantly reducing the model's memory (ROM) requirements. This optimization makes deployment on a constrained device more feasible and accelerates computation, enabling quicker predictions and responses. Post-training quantization is thus a crucial step, preserving core functionality and accuracy while adapting the models to the device's limited resources.

Given the deployment on the Arduino Nano 33 BLE Sense, additional metrics, such as inference time, memory usage, and storage footprint, were considered. These factors are essential to ensuring the models' efficiency and smooth performance on the device. By evaluating classification metrics, computational efficiency, and suitability for the tiny

device, we made informed decisions regarding model selection. Edge Impulse offers a functionality that estimates the performance of the model on the target device before deployment, facilitating a more informed deployment process.

4. Results and Discussion

In this section, we compare the performance of the three models by using metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC), as well as model size, inference time, and peak RAM usage, which are critical for deployment on TinyML devices. The dataset was split into 72% training, 10% validation, and 18% testing. Each model was trained and quantized by Edge Impulse for deployment suitability.

Table 8 shows the results of our experiments. the custom CNN model achieved the highest performance, with an accuracy of 96% on the test set and an AUC of 0.96. In contrast, the CNN Edge Impulse model, while faster and less resource-intensive, demonstrated lower accuracy, 85%, and an AUC of 0.85. This difference can be attributed to the simpler architecture of the Edge Impulse model. The custom LSTM model, however, achieved lower accuracy, 90%, compared with the CNN.

Table 8. Model evaluation on validation and testing sets.

Model	Accuracy	Precision	Recall	F1-Score	Loss Value	Area Under ROC Curve
Custom CNN						
Validation	100%	100%	100%	100%	0.03	1.00
Testing	96%	97%	97%	97%	-	0.96
CNN Edge Impulse						
Validation	92%	92%	92%	92%	0.24	0.91
Testing	85%	86%	86%	86%	-	0.85
Custom LSTM						
Validation	93%	93%	93%	93%	0.28	0.93
Testing	90%	93%	92%	92%	-	0.90

The confusion matrix in Table 9 provides a detailed evaluation of the classification performance of the CNN, LSTM, and CNN-EDGE-IMPULSE models on the test set data. The custom CNN outperformed the other models, correctly classifying 94.1% of "Asthma" cases and 98.3% of "Normal" cases, while the LSTM achieved 82.4% and 98.3%, respectively. CNN Edge Impulse, while more resource-efficient, had the lowest performance, with accuracy of 76.5% for "Asthma" and 88.1% for "Normal".

Table 9. Confusion matrix of test set data. Green: correct classifications, Red: misclassifications.

	Asthma (CNN)	Normal (CNN)	Asthma (LSTM)	Normal (LSTM)	Asthma (CNN-EDGE-IMPULSE)	Normal (CNN-EDGE-IMPULSE)	Uncertain
Asthma (CNN)	94.1%	5.9%	0%	0%	0%	0%	0%
Normal (CNN)	1.7%	98.3%	0%	0%	0%	0%	0%
Asthma (LSTM)	0%	0%	82.4%	17.6%	0%	0%	0%
Normal (LSTM)	0%	0%	1.7%	98.3%	0%	0%	0%
Asthma (CNN-EDGE-IMPULSE)	0%	0%	0%	0%	76.5%	20.6%	2.9%
Normal (CNN-EDGE-IMPULSE)	0%	0%	0%	0%	10.2%	88.1%	1.7%
F1-score	0.96	0.97	0.89	0.94	0.79	0.88	

Table 10 compares the models based on resource consumption, as estimated by the Edge Impulse cloud platform on the Arduino Nano 33. The CNN Edge Impulse model demonstrates its clear advantage for low-power, resource-constrained environments, requiring only 4.5 KB of RAM. However, this comes at the cost of lower classification performance. The custom CNN, which uses more RAM (12 KB) and has a longer inference time, strikes a balance between resource usage and accuracy, making it a more suitable option when both high classification performance and moderate resource usage are required.

Table 10. Performance comparison of different models.

Model	Inferencing Time	Peak RAM USAGE	Flash Usage
Our CNN	127 ms	12.0 K	249.6 K
CNN Edge Impulse	6 ms	4.5 K	31.7 K
Our LSTM	324 ms	23.2 K	190 K

On the other hand, the LSTM model, while providing good accuracy (90%) and the ability to process sequential data, has the highest resource demands, consuming 23.2 KB of RAM. This makes it less practical for highly resource-constrained devices.

Table 11 presents a comparative analysis between our proposed CNN model and similar works that utilize Edge ML models. Our TinyML model demonstrates superior performance, achieving higher accuracy, 96%, while also excelling in resource efficiency, making it more suitable for deployment on low-power devices.

Table 11. Qualitative performance comparison of the proposed CNN with existing works.

Work	[33]	[34]	**Our Work**
Dataset	KAUH	ICBHI 2017	**KAUH**
Feature	Mel-spectrogram	MFCC	**MFCC**
Preprocessing	Discrete Fourier transform (DFT)-based filtering and segmentation into 5 s windows	Zero padding, segmentation into 3 s windows, Butterworth bandpass filter, and Z-score normalization	**Zero padding, Butterworth bandpass filter, and segmentation into 5 s windows**
Lung diseases	Asthma and healthy	Normal and subnormal	**Asthma and healthy**
Target device	-	Alinx AX7A200 FPGA	**Arduino Nano 33 BLE**
Model	RDsLINet	LSTM	**CNN**
Accuracy	91%	81.4%	**96%**
Total execution time	5.439 s	-	**890 ms**
Peak RAM USAGE	-	32 KB	**12.0 KB**
Flash usage	498 KB	-	**249.6 KB**

One of the key reasons our model outperforms that of [33], which used the same dataset, lies in the application of pruning and quantization techniques. These methods allowed us to significantly reduce both model size and inference time, optimizing the model for resource-constrained environments. Pruning effectively removes less critical weights from the network, thus speeding up computation and reducing memory usage, while quantization lowers the precision of the model's parameters without substantially affecting its accuracy, leading to more efficient deployment on embedded devices.

In contrast, in [33], the authors applied depthwise separable convolutions and global average pooling (GAP) layers instead of fully connected layers to reduce the model size and execution time. This likely contributes to the differences in execution time and accuracy observed in our model compared with [33].

5. Conclusions and Future Work

In this work, we have exploited TinyML models for detecting respiratory diseases, particularly asthma, using lung sound recordings. Our custom CNN model achieved an accuracy of 96% while maintaining efficient resource usage. This demonstrates the feasibility of deploying real-time, accurate diagnostic tools on resource-constrained devices, making them suitable for portable medical applications.

The potential impact of this approach on healthcare is significant. By offering a low-cost, portable solution for respiratory disease detection, our models can enhance access to reliable diagnostics in remote and underserved areas, reducing the reliance on traditional medical facilities and expensive equipment. This advancement is crucial to improving early disease detection and patient outcomes.

In future work, we will address the challenges encountered with dataset imbalance, which limited the diversity of the training data. To overcome this challenge, we will explore merging multiple publicly available lung sound datasets and applying data augmentation techniques, such as variational autoencoders [30]. Additionally, real-world clinical validation and the inclusion of more respiratory conditions will be key steps toward refining and extending the applicability of our models.

Author Contributions: Conceptualization, Y.A.; Methodology, Y.A., N.B. and H.C.; Validation, Y.A.; Formal analysis, Y.A.; Investigation, Y.A.; Data curation, Y.A.; Writing—original draft, Y.A.; Writing—review & editing, N.B. and M.B.; Supervision, N.B. and H.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are publicly available at https://data.mendeley.com/datasets/jwyy9np4gv/3 (accessed on 21 August 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. World Health Organization. The Top 10 Causes of Death. 2021. Available online: https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death (accessed on 31 March 2024).
2. Hashoul, D.; Haick, H. Sensors for detecting pulmonary diseases from exhaled breath. *Eur. Respir. Rev.* **2019**, *28*. [CrossRef] [PubMed]
3. Sfayyih, A.H.; Sulaiman, N.; Sabry, A.H. A review on lung disease recognition by acoustic signal analysis with deep learning networks. *J. Big Data* **2023**, *10*, 101. [CrossRef] [PubMed]
4. Andrès, E.; Gass, R.; Charloux, A.; Brandt, C.; Hentzler, A. Respiratory sound analysis in the era of evidence-based medicine and the world of medicine 2.0. *J. Med. Life* **2018**, *11*, 89. [PubMed]
5. Alqudah, A.M.; Qazan, S.; Obeidat, Y.M. Deep learning models for detecting respiratory pathologies from raw lung auscultation sounds. *Soft Comput.* **2022**, *26*, 13405–13429. [CrossRef] [PubMed]
6. Huang, D.M.; Huang, J.; Qiao, K.; Zhong, N.S.; Lu, H.Z.; Wang, W.J. Deep learning-based lung sound analysis for intelligent stethoscope. *Mil. Med. Res.* **2023**, *10*, 44. [CrossRef]
7. McLane, I.; Emmanouilidou, D.; West, J.E.; Elhilali, M. Design and Comparative Performance of a Robust Lung Auscultation System for Noisy Clinical Settings. *IEEE J. Biomed. Health Inform.* **2021**, *25*, 2583–2594. [CrossRef]
8. Seah, J.J.; Zhao, J.; Wang, D.Y.; Lee, H.P. Review on the advancements of stethoscope types in chest auscultation. *Diagnostics* **2023**, *13*, 1545. [CrossRef]
9. Lella, K.K.; Jagadeesh, M.; Alphonse, P. Artificial intelligence-based framework to identify the abnormalities in the COVID-19 disease and other common respiratory diseases from digital stethoscope data using deep CNN. *Health Inf. Sci. Syst.* **2024**, *12*, 22. [CrossRef]
10. Tsoukas, V.; Boumpa, E.; Giannakas, G.; Kakarountas, A. A review of machine learning and tinyml in healthcare. In Proceedings of the 25th Pan-Hellenic Conference on Informatics, Volos, Greece, 26–28 November 2021; pp. 69–73. [CrossRef]

11. Abadade, Y.; Temouden, A.; Bamoumen, H.; Benamar, N.; Chtouki, Y.; Hafid, A.S. A Comprehensive Survey on TinyML. *IEEE Access* **2023**, *11*, 96892–96922. [CrossRef]
12. Ooko, S.O.; Muyonga Ogore, M.; Nsenga, J.; Zennaro, M. TinyML in Africa: Opportunities and Challenges. In Proceedings of the 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 7–11 December 2021; pp. 1–6. [CrossRef]
13. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [CrossRef]
14. Dutta, D.L.; Bharali, S. TinyML Meets IoT: A Comprehensive Survey. *Internet Things* **2021**, *16*, 100461. [CrossRef]
15. Nicolas, C.; Naila, B.; Amar, R.C. TinyML Smart Sensor for Energy Saving in Internet of Things Precision Agriculture platform. In Proceedings of the 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN), Barcelona, Spain, 5–8 July 2022; pp. 256–259. [CrossRef]
16. Bhamare, M.; Kulkarni, P.V.; Rane, R.; Bobde, S.; Patankar, R. Chapter 14—TinyML applications and use cases for healthcare. In *TinyML for Edge Intelligence in IoT and LPWAN Networks*; Academic Press: Cambridge, MA, USA, 2024; pp. 331–353. [CrossRef]
17. Bamoumen, H.; Temouden, A.; Benamar, N.; Chtouki, Y. How TinyML Can be Leveraged to Solve Environmental Problems: A Survey. In Proceedings of the 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sakheer, Bahrain, 20–21 November 2022; pp. 338–343. [CrossRef]
18. Diab, M.S.; Rodriguez-Villegas, E. Embedded Machine Learning Using Microcontrollers in Wearable and Ambulatory Systems for Health and Care Applications: A Review. *IEEE Access* **2022**, *10*, 98450–98474. [CrossRef]
19. Sun, B.; Bayes, S.; Abotaleb, A.M.; Hassan, M. The Case for tinyML in Healthcare: CNNs for Real-Time On-Edge Blood Pressure Estimation. In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, 27–31 March 2023; pp. 629–638. [CrossRef]
20. Saadeh, W.; Butt, S.A.; Altaf, M.A.B. A Patient-Specific Single Sensor IoT-Based Wearable Fall Prediction and Detection System. *IEEE Trans. Neural Syst. Rehabil. Eng.* **2019**, *27*, 995–1003. [CrossRef]
21. Fang, K.; Xu, Z.; Li, Y.; Pan, J. A Fall Detection using Sound Technology Based on TinyML. In Proceedings of the 2021 11th International Conference on Information Technology in Medicine and Education (ITME), Wuyishan, China, 19–21 November 2021; pp. 222–225. [CrossRef]
22. Zhu, T.; Kuang, L.; Li, K.; Zeng, J.; Herrero, P.; Georgiou, P. Blood Glucose Prediction in Type 1 Diabetes Using Deep Learning on the Edge. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021. [CrossRef]
23. Risso, M.; Burrello, A.; Pagliari, D.J.; Benatti, S.; Macii, E.; Benini, L.; Pontino, M. Robust and Energy-Efficient PPG-Based Heart-Rate Monitoring. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Republic of Korea, 22–28 May 2021. [CrossRef]
24. Alghamdi, N.S.; Zakariah, M.; Karamti, H. A deep CNN-based acoustic model for the identification of lung diseases utilizing extracted MFCC features from respiratory sounds. In *Multimedia Tools and Applications*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 1–33. [CrossRef]
25. Ullah, A.; Khan, M.S.; Khan, M.U.; Mujahid, F. Automatic Classification of Lung Sounds Using Machine Learning Algorithms. In Proceedings of the 2021 International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, 13–14 December 2021; pp. 131–136. [CrossRef]
26. Abdul, Z.K.; Al-Talabani, A.K. Mel Frequency Cepstral Coefficient and its Applications: A Review. *IEEE Access* **2022**, *10*, 122136–122158. [CrossRef]
27. Owens, F.; Murphy, M. A short-time Fourier transform. *Signal Process.* **1988**, *14*, 3–10. [CrossRef]
28. Yu, Y.; Si, X.; Hu, C.; Zhang, J. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Comput.* **2019**, *31*, 1235–1270. [CrossRef]
29. Sreeram, A.; Ravishankar, U.; Sripada, N.R.; Mamidgi, B. Investigating the potential of MFCC features in classifying respiratory diseases. In Proceedings of the 2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Paris, France, 14–16 December 2020; pp. 1–7. [CrossRef]
30. García-Ordás, M.T.; Benítez-Andrades, J.A.; García-Rodríguez, I.; Benavides, C.; Alaiz-Moretón, H. Detecting Respiratory Pathologies Using Convolutional Neural Networks and Variational Autoencoders for Unbalancing Data. *Sensors* **2020**, *20*, 1214. [CrossRef]
31. Aykanat, M.; Kılıç, Ö.; Kurt, B.; Saryal, S. Classification of lung sounds using convolutional neural networks. *EURASIP J. Image Video Process.* **2017**, *2017*, 65. [CrossRef]
32. Tawfik, M.; Al-Zidi, N.M.; Fathail, I.; Nimbhore, S. Asthma Detection System: Machine and Deep Learning-Based Techniques. In *Artificial Intelligence and Sustainable Computing;* Pandit, M., Gaur, M.K., Rana, P.S., Tiwari, A., Eds.; Springer: Singapore, 2022; pp. 207–218. [CrossRef]
33. Roy, A.; Satija, U. RDLINet: A Novel Lightweight Inception Network for Respiratory Disease Classification Using Lung Sounds. *IEEE Trans. Instrum. Meas.* **2023**, *72*, 4008813. [CrossRef]
34. Zhou, W.; Yu, L.; Zhang, M.; Xiao, W. A low power respiratory sound diagnosis processing unit based on LSTM for wearable health monitoring. *Biomed. Eng. Tech.* **2023**, *68*, 469–480. [CrossRef]
35. Harvard. AI for Good-Healthcare. 2024. Available online: https://harvard-edge.github.io/cs249r_book/contents/ai_for_good/ai_for_good.html#healthcare (accessed on 31 March 2024).

36. Hymel, S.; Banbury, C.; Situnayake, D.; Elium, A.; Ward, C.; Kelcey, M.; Baaijens, M.; Majchrzycki, M.; Plunkett, J.; Tischler, D.; et al. Edge Impulse: An MLOps Platform for Tiny Machine Learning. *arXiv* **2023**, arXiv:2212.03332.
37. Fraiwan, M.; Fraiwan, L.; Khassawneh, B.; Ibnian, A. A dataset of lung sounds recorded from the chest wall using an electronic stethoscope. *Data Brief* **2021**, *35*, 106913. [CrossRef] [PubMed]
38. Arduino. Nano 33 BLE Sense. Available online: https://docs.arduino.cc/hardware/nano-33-ble-sense/ (accessed on 13 October 2024).
39. David, R.; Duke, J.; Jain, A.; Janapa Reddi, V.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. *Proc. Mach. Learn. Syst.* **2021**, *3*, 800–811.

future internet

MDPI

Article

EdgeGuard: Decentralized Medical Resource Orchestration via Blockchain-Secured Federated Learning in IoMT Networks

Sakshi Patni and Joohyung Lee *

School of Computing, Gachon University, Seongnam 13120, Republic of Korea; sakshichhabra555@gmail.com or drsakshi5@gachon.ac.kr
* Correspondence: j17.lee@gachon.ac.kr

Abstract: The development of medical data and resources has become essential for enhancing patient outcomes and operational efficiency in an age when digital innovation in healthcare is becoming more important. The rapid growth of the Internet of Medical Things (IoMT) is changing healthcare data management, but it also brings serious issues like data privacy, malicious attacks, and service quality. In this study, we present EdgeGuard, a novel decentralized architecture that combines blockchain technology, federated learning, and edge computing to address those challenges and coordinate medical resources across IoMT networks. EdgeGuard uses a privacy-preserving federated learning approach to keep sensitive medical data local and to promote collaborative model training, solving essential issues. To prevent data modification and unauthorized access, it uses a blockchain-based access control and integrity verification system. EdgeGuard uses edge computing to improve system scalability and efficiency by offloading computational tasks from IoMT devices with limited resources. We have made several technological advances, including a lightweight blockchain consensus mechanism designed for IoMT networks, an adaptive edge resource allocation method based on reinforcement learning, and a federated learning algorithm optimized for medical data with differential privacy. We also create an access control system based on smart contracts and a secure multi-party computing protocol for model updates. EdgeGuard outperforms existing solutions in terms of computational performance, data value, and privacy protection across a wide range of real-world medical datasets. This work enhances safe, effective, and privacy-preserving medical data management in IoMT ecosystems while maintaining outstanding standards for data security and resource efficiency, enabling large-scale collaborative learning in healthcare.

Keywords: federated learning; resource orchestration; Internet of Medical Things; blockchain; scalability; resource efficiency

Citation: Patni, S.; Lee, J. EdgeGuard: Decentralized Medical Resource Orchestration via Blockchain-Secured Federated Learning in IoMT Networks. *Future Internet* **2025**, *17*, 2. https://doi.org/10.3390/fi17010002

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 19 October 2024
Revised: 29 November 2024
Accepted: 16 December 2024
Published: 25 December 2024

Citation: Patni, S.; Lee, J. EdgeGuard: Decentralized Medical Resource Orchestration via Blockchain-Secured Federated Learning in IoMT Networks. *Future Internet* **2025**, *17*, 2. https://doi.org/10.3390/fi17010002

1. Introduction

In the era of digital healthcare transformation, the Internet of Medical Things (IoMT) has emerged as a form of critical infrastructure, enabling healthcare providers to collect, analyze, and utilize vast amounts of patient data efficiently [1]. The proliferation of IoMT devices has facilitated unprecedented levels of patient monitoring and care, enabling rapid innovations in personalized medicine and remote healthcare delivery [2]. The goal of collaborative healthcare management in geographically distributed IoMT networks is to efficiently allocate and manage resources across various edge devices to meet healthcare demands effectively. However, as IoMT adoption grows, so do the challenges associated with ensuring security, privacy, and compliance, particularly in distributed healthcare

environments [3]. A recent survey highlights the landscape of IoMT adoption, revealing that wearable devices are the leading IoMT category, utilized by 69.6% of surveyed healthcare organizations [2]. Remote patient monitoring systems follow with a 62.3% adoption rate, and smart medical equipment is used by 60.2% of organizations. This diverse usage emphasizes the necessity for robust strategies to manage multi-device IoMT environments effectively [4,5].

The geographically distributed IoMT networks, where medical devices and edge computing resources are deployed across multiple locations, offer key advantages such as reduced latency in critical care scenarios, improved resilience, and enhanced compliance with regional health data regulations. These architectures are vital for meeting the demands of global healthcare delivery and ensuring uninterrupted service during localized disruptions or emergencies [6]. However, they offer significant safety concerns. The need to protect sensitive health information across a variety of locations, defend against a broader spectrum of online cyber threats, and ensure compliance under distinct healthcare regulatory standards increases layers of complexities as cited in [7]. Although several advanced security measures have been designed for IoMT, significant barriers still remain in meeting the security requirements of geographically dispersed healthcare networks. These problems include critical communication overhead that causes latency in time-sensitive medical applications, challenges with data integrity across various medical devices, and scalability issues arising from the incapacity of current security mechanisms to manage increasing amounts of health data [8]. Additionally, it is common for medical device combinations to be non-standardized, which leads to hazards and discrepancies. Inadequate visibility in auditing and monitoring further contributes to the inability to identify and address medical security issues in healthcare settings [9]. Furthermore, providing comprehensive security will require a lot of processing and storage power for a medical device with limited resources, which will affect its overall effectiveness and cost. Due to these difficulties, an effective solution for IoMT contexts is urgently needed. The suggested architecture, EdgeGuard, combines federated learning's privacy with blockchain's decentralized security [9,10]. It is specifically designed for the IOMT. This is the whole framework that, on the one hand, enhances health data security and optimizes resource management in collaborative healthcare environments and, on the other, permits safe and effective cooperation between dispersed medical devices and edge nodes.

EdgeGuard is an innovative framework that seeks to redefine the contours of secure and efficient data management in IoMT networks. It leverages blockchain technology, adaptive federated learning, and edge computing capabilities to address significant limitations of current approaches in healthcare data security and privacy [3,11]. EdgeGuard implements a novel decentralized architecture that optimizes resource utilization across diverse IoMT devices while ensuring robust data privacy and integrity [12]. Our framework introduces several key innovations: a lightweight blockchain consensus mechanism specifically designed for IoMT networks, an adaptive aggregation function for privacy-preserving federated learning, and intelligent resource allocation through reinforcement learning [13]. As illustrated in Figure 1, EdgeGuard introduces a secure blockchain layer that enables safe collaborative learning while preventing information leakage of patient data. The main contributions of this work are as follows:

- **Privacy-preserving federated learning architecture:** A novel adaptive aggregation mechanism that enables secure model training across distributed healthcare institutions [14]. This architecture incorporates differential privacy techniques and secure aggregation protocols, ensuring patient privacy while optimizing model performance through quality-aware aggregation.

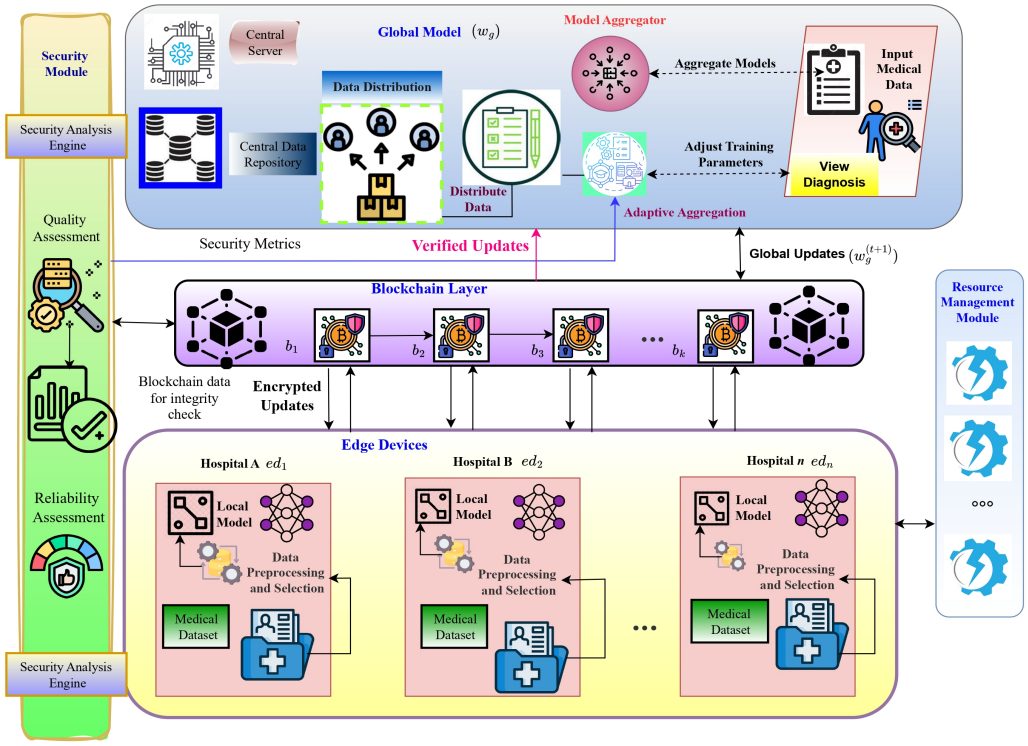


Figure 1. EdgeGuard: a secure federated learning framework for IoMT.

- **IoMT-optimized blockchain consensus:** A lightweight consensus mechanism specifically engineered for resource-constrained medical devices, providing robust security guarantees while maintaining efficiency. This mechanism ensures data integrity and creates immutable audit trails for regulatory compliance.
- **Intelligent resource management:** Advanced optimization techniques for heterogeneous IoMT environments include the following:
  - A dynamic model complexity adaptation based on available computational resources.
  - Adaptive learning rate scheduling, considering resource constraints.
  - Quality-aware device selection for optimal federated learning rounds.
  - Efficient model update compression for bandwidth-constrained scenarios.
- **Comprehensive performance framework:** A multi-dimensional evaluation framework that assesses not only diagnostic accuracy but also computational efficiency, communication effectiveness, energy consumption, and fairness across diverse IoMT devices, ensuring practical deployability in real-world healthcare settings.

This paper is structured as follows: Section 2 discusses the related work and Section 3 presents EdgeGuard's system formulation, including the system model, assumptions, threat model, problem statement, and design goals. Section 3 describes the proposed EdgeGuard framework in detail. Section 4 explains the operational design and algorithm analysis. Section 5 discusses the experimental setup, results, and analysis, and Section 6 concludes the paper.

2. Related Work

The integration of blockchain technology with federated learning in IoMT networks has emerged as a significant research area in recent years, driven by concerns over data privacy and security vulnerabilities in healthcare systems. In this section, we will analyze some of the most recent advances in this area, focusing on three main aspects: blockchain mechanisms, federated learning in healthcare, and integrated decentralized blockchain-FL

architectures. In the context of IoMT networks, ref. [8] presented a security framework that provides security when medical data are being transferred based on a combination of encryption techniques, pattern recognition modules, and adaptive learning mechanisms. Although this approach has made advances in both the detection of anomalies as well as attack resistance, it neither considers the quality of data in a distributed setting nor does it take into account the computational capabilities of IoMT products. In addition, the framework lacks mechanisms for enabling decentralized yet secure collaborative machine learning across healthcare institutions—a must-have component to enhance diagnostic models while guaranteeing the privacy of records. This shortcoming is something EdgeGuard addresses with its blockchain-secured federated learning architecture.

Mohammed et al. [15] proposed a federated learning paradigm toward collaborative usage of health information while keeping it private based on both secure multi-party computations and differential privacy. Even though it has provided significant privacy preservation, there has been no proper consideration given to securing the model updates, the participating devices, or the data quality. A recent work proposed by Biken Singh et al. [16] introduces a blockchain-supported federated learning system for WBANs, focusing on energy efficiency and privacy through QNNs, differential privacy, and homomorphic encryption. However, their approach primarily focuses on energy optimization and basic privacy preservation, with no consideration of data quality assessment and device reliability in medical contexts. The authors failed to provide explicit security requirements for IoMTs in healthcare environments. EdgeGuard bridges this gap through its adaptive aggregation and lightweight consensus mechanisms, which are specifically tailored for IoMT.

Prior work by Yu et al. [17] proposed an I-UDEC framework combining blockchain, AI, and federated learning to optimize computation offloading and resource allocation in ultra-dense edge computing. The work obtained great improvements in task execution time but was mainly focused on general IoT scenarios and did not consider specific medical data sensitivity and healthcare regulatory compliance. Furthermore, their blockchain implementation was not designed for resource-constrained medical devices—something EdgeGuard addresses in its healthcare-specific design and lightweight consensus mechanism. The paper by Ali Kashif et al. [18] explored the integration of federated learning in healthcare Metaverse applications, highlighting potential benefits and challenges in medical diagnosis, patient monitoring, and drug discovery. While comprehensive in scope, the paper primarily focused on theoretical aspects and future possibilities, lacking practical implementation details or specific solutions for current IoMT security and privacy challenges that EdgeGuard addresses through its concrete blockchain-secured federated learning architecture. Previous research [19] proposed combining DFT with differential privacy in federated learning for healthcare, achieving better accuracy and reduced communication costs, but their approach lacked security mechanisms for model updates and did not address device reliability or data quality validation in medical networks—gaps that EdgeGuard specifically addresses.

Although the existing works are tremendous in terms of healthcare security through federated learning and integrating blockchain, they merely focus on individual aspects like privacy preservation or energy efficiency and cannot achieve a holistic solution in IoMT environments. Most approaches usually miss critical points related to data quality assessment and reliability of devices and, most importantly, come up with lightweight security mechanisms tailored for resource-constrained medical devices.

EdgeGuard has addressed this by designing an integrated framework that contains blockchain-secured federated learning and IoMT-specific optimizations. Our solution uniquely integrates adaptive aggregation based on data quality and device reliability, a lightweight consensus mechanism designed for medical devices, and comprehensive security measures that maintain HIPAA compliance while enabling efficient collaborative

learning. The comparative analysis in Table 1 demonstrates the current work addresses particular aspects of privacy and security for health networks but not in totality. As such, EdgeGuard clearly distinguishes itself by combining privacy preservation, lightweight blockchain security, resource optimization, data quality assessment, and device reliability monitoring within the proposed healthcare-specific framework.

Table 1. Comparison of related works in blockchain-secured federated learning for healthcare.

Work	Privacy Preservation	Security Mechanism	Resource Optimization	Data Quality Assessment	Device Reliability	Healthcare Specific
[8] Pattern Recognition	DP	Encryption	No	No	No	Yes
[15] Privacy-preserving FL	DP + MPC	No	No	No	No	Yes
[16] WBAN-based FL	DP + HE	Blockchain	Energy-aware	No	No	Yes
[17] I-UDEC Framework	FL	Blockchain	2Ts-DRL	No	No	No
[18] DFT-based FL	DP + DFT	No	Communication	No	No	Yes
EdgeGuard (Ours)	DP + MPC	Lightweight Blockchain	Resource-aware	Yes	Yes	Yes

DP: differential privacy, MPC: multi-party computation, HE: homomorphic encryption, DFT: discrete Fourier transform, 2Ts-DRL: two-timescale deep reinforcement learning, FL: Federated Learning.

3. Problem Formulation

3.1. System Model

More than a thousand devices form a vast network of IoMT healthcare data resources. The adaptive federated learning model, with its dynamic aggregation, is recognized as a well-known system in the field of IoMT. This novel idea has changed the concept and analysis of medical data. It ensures privacy and security in these increasingly developing digital health systems. Edge devices, local datasets, central servers, edge servers, local and global models, adaptive aggregation functions, blockchain security layers, and more, are the main components used in this proposed model. The following are the main parts of our system:

1. **Edge devices** ($\mathcal{ED} = \{ed_1, ed_2, \ldots, ed_N\}$): The different numbers of IOMT N edge devices range in terms of their computing and sensing capabilities. These edge devices are important components of our distributed learning setup from simple fitness trackers to cutting-edge medical equipment.
2. **Local datasets** ($\mathcal{DS} = \{DS_1, DS_2, \ldots, DS_N\}$): Every edge device ed_i tracks a local dataset, thus reflecting a different portion of global health data. These datasets are characterized by their size $|DS_i|$ and a quality metric $q_i \in [0, 1]$.
3. **Edge servers** ($\mathcal{ES} = \{es_1, es_2, \ldots, es_M\}$): A collection of M edge servers arranged specifically to enable intermediate aggregation and computational offloading. These servers improve system scalability and responsiveness by acting as a link between the central server and resource-limited peripheral devices. This makes our whole system work better and faster, especially when dealing with lots of devices that might not be very powerful on their own.
4. **Central server** ($\mathcal{CS}$): A high-performance central server that orchestrates the federated learning process, aggregates model updates, and maintains the global model. It is responsible for initiating learning rounds and disseminating the updated global model to edge devices.
5. **Global and local models** (w_g, w_l): A shared neural network with d parameters; the global model $w_g \in \mathbb{R}^d$ represents the collective knowledge extracted from various medical data sources across the network. Collection $\mathcal{W}$ consists of local model instances, where the model parameters trained on the local dataset DS_i of edge device

ed_i are represented by each $w_l \in \mathbb{R}^d$. These local models feed into the global model and are updated regularly, enabling a distributed learning process that protects data privacy and makes use of the IoMT network's collective insights.

6. **Communication links** ($\mathcal{L} = \{l_{ij}\}$): The group of communication lines that link the central server, edge servers, and edge devices. The bandwidth b_{ij} and latency λ_{ij} of each connection l_{ij} are its key characteristics.
7. **Adaptive aggregation function** ($f : \mathbb{R}^d \times [0,1] \times [0,1] \to \mathbb{R}^d$): A completely novel function that dynamically weighs each edge device's contribution according to measures for data quality and reliability. It is defined as follows:

$$w_g = f(\{w_i\}_{i=1}^N, \{q_i\}_{i=1}^N, \{r_i\}_{i=1}^N) \tag{1}$$

where r_i denotes the device e_i's reliability and q_i denotes the data quality measure.

8. **Blockchain Security Layer (BSL)** $\mathcal{B}$ To ensure the security, integrity, and traceability of the federated learning process [20], our system incorporates a BSL. This decentralized ledger system, denoted as $\mathcal{B} = (B, T, \sigma, V)$, consists of a chain of blocks $B = \{b_1, b_2, \ldots, b_K\}$, each containing a set of verified transactions. The function $T : \mathcal{E} \cup \mathcal{S} \cup \{\mathcal{C}\} \to B$ maps entities to transactions recorded in blocks, while the cryptographic hash function $\sigma : B \to \{0,1\}^*$ ensures the immutability of the blockchain. A validation function $V : B \times \mathcal{E} \cup \mathcal{S} \cup \{\mathcal{C}\} \to \{0,1\}$ verifies the legitimacy of transactions and blocks. Each model update and aggregation operation is recorded as a transaction in the blockchain, ensuring the integrity and traceability of the learning process:

$$b_{k+1} = \sigma(b_k \parallel T(w_g^{t+1}) \parallel T(\{w_i^{t+1}\}_{i \in \mathcal{E}_t})) \tag{2}$$

where the symbol *parallel* denotes concatenation. This blockchain layer gives our system a higher level of security by providing an editable and tamper-proof record of all learning activities inside the IoMT network.

9. **Quality assessment module** ($QA : \mathcal{DS} \to [0,1]$): A new module that grades the quality of a local dataset based on various criteria, such as data distribution, label accuracy, and task relevance, around the world. The quality score that is derived for every local dataset is $q_i = Q(D_i)$.
10. **Reliability evaluation function** ($REL : \mathcal{ED} \times \mathbb{T} \to [0,1]$): The feature that scores the reliability of the edge devices over time, accounting for the needs of hardware, uptime, and consistency of contributions. For each device at time t it returns a reliability score $Rel_i = REL(ed_i, t)$.

In the complex ecosystem, the adaptive federated learning scheme requires a series of communication rounds. For each round t, we choose a subset of edge devices $\mathcal{ED}_t \subseteq \mathcal{ED}$. Based on their own datasets, this subset of edge devices computes local updates:

$$w_i^{t+1} = w_i^t - \eta \nabla LF(w_i^t, DS_i) \tag{3}$$

where LF denotes the loss function and η represents the learning rate. Subsequently, using our adaptive aggregation function, the central server aggregates these updates:

$$w_g^{t+1} = f(\{w_i^{t+1}\}_{i \in \mathcal{ED}_t}, \{q_i^t\}_{i \in \mathcal{ED}_t}, \{r_i^t\}_{i \in \mathcal{ED}_t}) \tag{4}$$

In this dynamic aggregation technique by IoMT, worldwide learning depends not just on the volume of data but on the goodness of data along with the credibility of origin. This, in turn, opens up the gates for an even more adaptive and robust learning process.

3.2. Assumptions and Threat Model

The main assumptions and threats are considered, which define our system architecture and security measures while designing EdgeGuard for safe and effective federated learning in IoMT healthcare networks.

3.2.1. Main Assumptions

1. **Data privacy and locality:** The local dataset $D_i \in \mathcal{DS}$ for every edge device $ed_i \in \mathcal{ED}$ remains on the device. According to healthcare data standards and patient privacy, only model updates $w_i \in \mathcal{W}$ are shared. Formally:

$$\forall ed_i \in \mathcal{ED}, \text{ share } w_i \text{ but not } DS_i \tag{5}$$

2. **Device heterogeneity and intermittent connectivity:** The edge devices, $\mathcal{ED} = \{ed_1, ed_2, \ldots, ed_N\}$, are heterogeneous in terms of processing capability and network reliability. We represent this heterogeneity by defining a time-varying subset of devices that are active, $\mathcal{ED}_t \subseteq \mathcal{ED}$ in each round t:

$$\mathcal{ED}_t = \{ed_i \in \mathcal{ED} \mid \text{device } ed_i \text{ is active at time } t\} \tag{6}$$

3. **Semi-honest participants:** While implementing the steps, participants have the opportunity to learn from the data they have received. We assume the reliability assessment function $REL : \mathcal{ED} \times \mathbb{T} \to [0, 1]$ represents this behavior over time:

$$\forall ed_i \in \mathcal{ED}, t \in \mathbb{T} : 0 < REL(e_i, t) \leq 1 \tag{7}$$

3.2.2. Primary Threats μ

1. **Data poisoning attacks** (μ_d)**:** The malicious entities may manipulate the local dataset or introduce fake data into the global model w_g updates. We represent this threat as a perturbation δ of local updates:

$$\tilde{w}_i^{t+1} = w_i^{t+1} + \delta, \quad \delta \sim \mathcal{D}_{\text{adversary}} \tag{8}$$

where $\mathcal{D}_{\text{adversary}}$ is an adversarial distribution.

2. **Privacy breaches** (μ_p)**:** The adversaries could try to recreate private information from model updates. In such cases, the differential privacy anticipates and assists the users from the privacy risks:

$$\Pr[f(DS) \in S] \leq e^{\epsilon} \cdot \Pr[f(DS') \in S] + \delta \tag{9}$$

where datasets D and DS' differ by one record; here, f is our learning algorithm, ϵ is the privacy budget, and δ is the failure probability.

3. **Integrity and authentication attacks** (μ_i)**:** The attackers might fabricate the devices or interfere with model updates. In order to solve this, our blockchain security layer $\mathcal{B} = (B, T, \sigma, V)$ ensures the following:

$$V(b_k, e_i) = 1 \iff T(w_i^{t+1}) \text{ is a valid transaction in } b_k \tag{10}$$

where $b_k \in B$ is a block in the blockchain, and V is the validation function.

With its blockchain-based integrity verification $\mathcal{B}$, adaptive aggregation function f, and privacy-preserving methods, EdgeGuard confronts these fundamental assumptions and threats. The reliability evaluation function R and quality assessment module $Q : \mathcal{D} \to [0, 1]$ are designed to counteract data poisoning attacks and the blockchain layer offers an

unchangeable audit trail to guarantee the authenticity and integrity of model modifications. By keeping raw data localized and introducing noise to model updates, the federated learning method, when paired with differential privacy measures, naturally contributes to privacy protection against breaches.

3.3. Problem Statement and Design Goals

In this study, a federated learning system, called EdgeGuard, is developed using blockchain technology to enable safe and effective cooperation in the Internet of Medical Things (IoMT) networks with N-distributed edge devices. The challenge involves effectively carrying out federated learning tasks across many IoMT devices while reducing threats (μ_d, μ_p, and μ_i). This is conducted to enhance the learning process's overall efficiency and provide strong security in a cooperative healthcare setting. During the federated learning process over periods $\{t_1, t_2, \ldots, T\}$, the objective is to maximize data utility (DU) and model accuracy (MA) while minimizing communication overhead (CO), potential threats (μ), and model divergence (MD) in the presence of unknown malicious clients, such that we have the following:

$$\{\omega \mid \omega : \prod_{i=1}^{N} \prod_{j=1}^{M} \prod_{k=1}^{K} \omega_{ijk}\} \tag{11}$$

where $\omega_{ijk} = 1$ if edge device e_i participates in learning round j at local edge devices. Equation (12) formulates the threat model by incorporating the considered potential threats (μ_d: data poisoning, μ_p: privacy breaches, and μ_i: integrity attacks), aiming to minimize these risks and enhance the overall security of the collaborative environment:

$$\mu_k = \min(\mu_d, \mu_p, \mu_i) \tag{12}$$

Equation (13) is formulated in a way that guarantees the effective distribution of edge devices to learning rounds across geographically dispersed data centers:

$$\omega : \mathcal{E} \times \mathbb{T} \times \mathcal{S} \times \mu_k \rightarrow \{0, 1\} \tag{13}$$

The critical constraints that must be satisfied during the federated learning process within the IoMT network are stated in Equations (15)–(18):

$$C1 : \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} \omega_{ijk} = 1 \tag{14}$$

$$C2 : \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} E^i_{CPU} \times \omega_{ijk} \leq S^k_{CPU} \tag{15}$$

$$C3 : \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} E^i_{RAM} \times \omega_{ijk} \leq S^k_{RAM} \tag{16}$$

$$C4 : \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} E^i_{BW} \times \omega_{ijk} \leq S^k_{BW} \tag{17}$$

$$C5 : \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} \omega_{ijk} \leq AvailED \tag{18}$$

The constraint $C1$ specifies that each edge device e_i must participate in exactly one learning round; the constraints $\{C2–C4\}$ state that the available resource capacity (CPU, RAM, bandwidth) of the edge network must be greater than or equal to the total requesting resource capacity of participating edge devices; constraint $C5$ specifies the geographical

constraints, indicating the availability of edge devices within the IoMT network. The design goals of EdgeGuard are, thus, formulated as a multi-objective optimization problem:

$$\min_{w_g,\omega}\{CO, \mu, MD\} \quad \text{and} \quad \max_{w_g,\omega}\{DU, MA\} \tag{19}$$

subject to constraints $C1$–$C5$, where w_g denotes the global model parameters. To achieve these goals, EdgeGuard employs the following:

- An adaptive aggregation function f to balance data utility and privacy:

$$w_g^{t+1} = f(\{w_i^{t+1}\}_{i\in\mathcal{E}_t}, \{q_i^t\}_{i\in\mathcal{E}_t}, \{r_i^t\}_{i\in\mathcal{E}_t}) \tag{20}$$

- A blockchain security layer $\mathcal{B} = (B, T, \sigma, V)$ to ensure integrity and traceability.
- A quality assessment module $Q : \mathcal{D} \to [0, 1]$ to evaluate data quality.
- A reliability evaluation function $R : \mathcal{E} \times \mathbb{T} \to [0, 1]$ to assess device trustworthiness.

These components work in concert to create a secure, efficient, and privacy-preserving federated learning system for IoMT healthcare networks.

4. Proposed Framework

The EdgeGuard framework in Figure 1 enables secure federated learning across distributed IoMTs by allowing the edge devices to perform local model training over the sensitive health data, and the encrypted model updates are transmitted over the blockchain layer safeguarding the data integrity and privacy. With the direction of continuous security analysis and resource management, the central server then aggregates these changes into a global model via adaptive aggregation. This very sophisticated technique enables collaborative learning with raw patient data remaining localized, therefore balancing usefulness and privacy against system efficiency in challenging medical situations. EdgeGuard will, therefore, address particular challenges when processing remote medical data and aid in providing more insight into healthcare by combining various modules without compromising individual patient privacy or system security.

4.1. EdgeGuard Framework

The EdgeGuard architecture consists of six main steps: local model training, local model upload, cross-verification, block generation and propagation, adaptive aggregation, and global model update. These procedures provide safe and efficient federated learning in IoMT healthcare networks.

4.1.1. Local Model Training

The local training process is independently conducted using locally stored data on each IoMT edge device. We consider N edge devices as $\mathcal{E} = \{ed_1, ed_2, \ldots, ed_N\}$. Let there be M medical sensors at each edge device denoted as $\mathcal{S} = \{s_1, s_2, \ldots, s_M\}$ sending their health data to the IoMT environment for processing and analysis. Each sensor generates a set of health measurements, $mathcalH = h_1, h_2, \ldots, h_Z$, along with particular parameters like the timestamp, sensor type, and measurement value $(h_timei, h_typei, h_valuei)$. The edge device consists of computational resources characterized by CPU, memory, and bandwidth capacities $(E^i_{CPU}, E^i_{Mem}, E^i_{BW})$. A convolutional neural network (CNN) is utilized at the edge devices to process and analyze the health data. The neural network comprises p-q-r number of neurons at the input, hidden, and output layers. These layers are interconnected

through NN weights ($\{w_{I_1}, w_{I_2}, \ldots, w_{I_p}, w_{H_1}, w_{H_2}, \ldots, w_{H_q}, w_{O_1}, \ldots, w_{O_r}\}$), with the size of the NN as S, such that we have the following:

$$S = (p+1) \times q + (q \times r) = q(p+r+1) \Rightarrow q(p+2) \text{ as } r = 1 \tag{21}$$

The NN weights and biases (b) are initialized randomly in the range of $[0, 1]$. The CNN collects the historical health data and normalizes the data to create and provide p input values, such as $\{DS_1, DS_2, \ldots, DS_p\}$, into the input layer. The prediction process consists of three main steps: training, testing, and prediction. Data validation is conducted to improve the model's performance, using the mean absolute percentage error (MAPE) as the error function to assess the model's accuracy. The pre-processing of data extracts health measurements from different sensors and aggregates them over a fixed time interval. To normalize the input data within the range of $[0, 1]$, the data aggregation process is applied, as shown in Equation (22):

$$\hat{D}_i = \frac{D_i - D_{min}}{D_{max} - D_{min}} \tag{22}$$

In the dataset, DS_{max} represents the highest value obtained, while DS_{min} corresponds to the lowest value. The normalized data, denoted as $\hat{DS}$, comprise a collection of all normalized data values, represented as $\hat{DS}_1, \hat{DS}_2, \ldots, \hat{DS}_n$. These normalized one-dimensional values are utilized as input to the input layer of the CNN. This model analyzes previous p health data values to predict the health status (Y_{out}) at the $p+1th$ time instance. The *ReLU* activation function, as depicted in Equation (23), is used in the hidden layers:

$$ReLU(x) = \max(0, x) \tag{23}$$

The evaluation of the accuracy and performance of the LM training process is carried out by utilizing the *MAPE* score, as given in Equation (24):

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|Y_{predicted} - Y_{actual}|}{Y_{actual}} \tag{24}$$

where n represents the total number of data samples, while Y_{actual} and $Y_{predicted}$ correspond to the actual and predicted health status, respectively. Stochastic gradient descent with momentum (SGD-M) is employed to achieve dynamic and adaptive optimization of the network weights. In this context, velocity (v) represents the gradient change needed to reach the global minimum, as expressed in Equation (25):

$$w_{t+1} = w_t - v_t \tag{25}$$

The updated weight vector is represented as w_{t+1}, while the current weight vector is denoted as w_t. The calculation of v_t can be performed using Equation (26):

$$v_t = \beta \cdot v_{t-1} + \eta \nabla w_t \tag{26}$$

Here, the momentum is represented by the term $\beta \cdot v_{t-1}$. The constant β has a value between 0 and 1, the learning rate is denoted as η, and ∇w_t corresponds to the gradient of the loss function for the weight. v_{t-1} represents the velocity at the previous step. Then, the local model w_{t+1} is uploaded to the blockchain to complete federated learning aggregation.

4.1.2. Local Model Upload

In EdgeGuard, we collect model updates from our medical devices, represented as $\{\Delta w_1, \Delta w_2, \ldots, \Delta w_N\}$. These updates are added to our blockchain, forming a series of

connected blocks $\{B_1, B_2, \ldots, B_N\}$. Each block has two parts: a body and a header. The body, shown in Equation (27), contains the model updates and calculation times, as follows:

$$\text{Body}_i = \{(\Delta w_k^l, T_{\text{local}}^i) | \forall k, l, i\} \tag{27}$$

Here, Δw_k^l denotes the update from device k for training round l, and T_{local}^i denotes how long device i takes to compute. The header, given by Equation (28), is like an information tag, as follows:

$$\text{Header}_i = \{P_{\text{prev}}, \lambda, \text{PoW}\} \tag{28}$$

P_{prev} links to the previous block, λ denotes how fast we make blocks, and PoW is proof that the block is legit. We also track the block size using Equation (29):

$$B_{\text{size}} = h + \delta m \cdot N_D \tag{29}$$

This depends on the header size h, the size of each update δm, and the number of devices N_D. This setup helps us keep our medical AI updates organized and secure, balancing technical precision with practical application in our IoMT network [21].

4.1.3. Cross-Verification

Miners broadcast and verify model updates, accumulate verified updates in a candidate block B, and finalize the block if it follows Equation (30):

$$B_{size} \geq h + \delta m \cdot N_E \text{ or } t \geq T_{wait} \tag{30}$$

4.1.4. Block Generation and Propagation

EdgeGuard employs a proof of work (PoW) mechanism for secure block generation and propagation. This process unfolds in three key steps: hash generation, block generation rate determination, and block propagation with ledger update. In the hash generation step, a miner m in the network computes a hash value H by iteratively modifying a nonce value N. The goal is to find a hash that satisfies the condition expressed in Equation (31):

$$H(N) < T \tag{31}$$

Here, H represents the hash function, N is the nonce, and T denotes the target value that defines the PoW difficulty. The block generation rate, denoted as λ, is inversely proportional to the PoW difficulty, which is reflected in the target hash value T. This relationship is captured in Equation (32):

$$\lambda \propto \frac{1}{T} \tag{32}$$

This inverse relationship implies that a more stringent target value T results in a lower λ, thereby reducing the frequency of block generation. As stated in Equation (33), upon generation of a valid hash by a miner, the new block B is verified, approved, and disseminated to all miners. At this point, the miners cease their proof of work calculations and append B to their local ledgers.

$$\text{If } H(T(\tilde{w}_i^{t+1}) \parallel n) < \text{Target, then } B \text{ is added to all local ledgers} \tag{33}$$

The proof-of-work method, thus, ensures integrity and security within the EdgeGuard architecture and provides a good base for decentralized storage and validation of the model changes within the IoMT network.

4.1.5. Adaptive Aggregation and Global Model Update

Another innovation is the adaptive aggregation function that weighs contributions from each edge device within the function of data quality and reliability of devices. Such a feature is extremely critical for ensuring that the federated learning process remains robust and effective in the eventuality of a possible problem that results from data quality and security threats. Let $\{w_1, w_2, \ldots, w_N\}$ represent the gradient updates acquired from the blockchain layer. Finally, the central server takes the adaptive aggregation function f to obtain the new global gradient at the time step $t+1$. Mathematically, we have the following:

$$w_{t+1} = f(\{w_i^{t+1}\}_{i\in\mathcal{E}_t}, \{q_i^t\}_{i\in\mathcal{E}_t}, \{r_i^t\}_{i\in\mathcal{E}_t}) \tag{34}$$

where $\mathcal{E}_t$ is the set of participating devices in round t, q_i^t is the quality score of device i's data, and r_i^t is the reliability score of device i. The adaptive aggregation function f is defined as follows:

$$f(\{w_i\}, \{q_i\}, \{r_i\}) = \frac{\sum_{i\in\mathcal{E}_t} \alpha_i w_i}{\sum_{i\in\mathcal{E}_t} \alpha_i} \tag{35}$$

where α_i is the weight assigned to device i's update and is calculated as follows:

$$\alpha_i = q_i \cdot r_i \cdot \exp(-\beta \|w_i - \bar{w}\|^2) \tag{36}$$

Here, $\bar{w}$ is the average of all updates, and β is a hyperparameter controlling the influence of update similarity. This formulation ensures the following:

1. Higher quality data (higher q_i) have more influence on the global model.
2. More reliable devices (higher r_i) contribute more significantly.
3. Updates that are closer to the average (potentially more trustworthy) are given higher weight.

The quality score q_i is determined by the quality assessment module (QAM):

$$q_i = Q(D_i) = \frac{1}{1 + \exp(-(\gamma_1 C_i + \gamma_2 V_i - \gamma_3 O_i))} \tag{37}$$

where C_i denotes the completeness of the data, V_i denotes the validity, O_i denotes the outlier ratio, and $\gamma_1, \gamma_2, \gamma_3$ denote learnable parameters. The reliability score r_i is calculated by the reliability evaluation function (REF), as follows:

$$r_i = R(e_i, t) = \delta r_{i,t-1} + (1-\delta)\left(\frac{1}{1 + \exp(-(\lambda_1 U_i + \lambda_2 A_i - \lambda_3 E_i))}\right) \tag{38}$$

where $r_{i,t-1}$ is the previous reliability score, U_i is the uptime ratio, A_i is the contribution accuracy, E_i is the error rate, δ is a smoothing factor, and $\lambda_1, \lambda_2, \lambda_3$ are learnable parameters. This adaptive aggregation mechanism allows EdgeGuard to conduct the following:

1. Mitigate the impact of low-quality or malicious updates.
2. Adapt to changing device behaviors and data characteristics.
3. Improve the overall robustness and accuracy of the global model.
4. Provide an implicit defense against various attacks, including data poisoning and free-riding.

The aggregated gradient w_{t+1} is then used to update the global model GM. The GM has the same architecture as the local models. The update process is expressed in Equation (39):

$$GM_{(t+1)} = GM_{(t)} + w_{t+1} \tag{39}$$

4.1.6. Local Model Update

The global gradient w_{t+1} is broadcast to all local clients using the blockchain layer, ensuring that the privacy of each client is maintained. Then each local model LM_i for the ith edge device is updated using the broadcasted global gradient w_{t+1} as expressed in Equation (40):

$$LM_i^{(t+1)} = LM_i^{(t)} + w_{t+1} \tag{40}$$

where $LM_i^{(t)}$ represents the local model parameters at epoch t, and $LM_i^{(t+1)}$ represents the updated local model parameters for the next epoch. Local training for the next epoch begins with the updated parameters of the global model, ensuring consistency across all local models while maintaining the privacy and security of individual health data. This iterative process of local training, secure aggregation, and model distribution allows EdgeGuard to facilitate collaborative learning across IoMT devices, enabling improved healthcare insights while maintaining stringent privacy and security standards essential to medical applications.

4.2. Smart Contract Implementation for Access Control and Model Updates

The IoMT network, running on the EdgeGuard framework, promises to have extremely complicated architectures for safe and verifiable smart contracts. Our implementation includes three leading types: device access control, model update verification, and secure aggregation protocols. The layer of smart contracts is important as it facilitates the transition between the federated learning process and the incorporation of blockchain security.

The proposed smart contract manages device registration, validates model updates, and ensures secure aggregation according to the privacy and security requirements of healthcare data. Specifically, this study targets three main aspects:

- **Access control**: Ensures only authorized IoMT devices participate in the federated learning process.
- **Model update verification**: Validates and records model updates in an immutable manner.
- **Secure aggregation**: Implements privacy-preserving model aggregation using multi-party computation.

Algorithm 1 presents the comprehensive smart contract protocol that governs these interactions within EdgeGuard.

This protocol further completes our description of the blockchain security layer B in Section 3.1, such that the federated learning process can remain robust and free of attacks only when legitimate devices are involved in the training model. It makes use of the adaptive aggregation function in Equations (34)–(36), in which quality scores q_i and reliability scores r_i are used for weighing each edge device.

The implementation guarantees the following three important properties:

1. **Security**: Through robust access control and validation mechanisms.
2. **Privacy**: Via secure multi-party computation during aggregation.
3. **Verifiability**: Through immutable blockchain records of all operations.

This kind of architecture of the smart contract provides a needed base in the IoMT environment for secure and verifiable federated learning, yet it certainly aligns with EdgeGuard's decentralized approach to medical resource management.

Algorithm 1 EdgeGuard smart contract protocol.

Require: Set of edge devices $\mathcal{ED} = \{ed_1, ed_2, \ldots, ed_N\}$, local models $\mathcal{W} = \{w_1, w_2, \ldots, w_N\}$, quality scores $\mathcal{Q} = \{q_1, q_2, \ldots, q_N\}$, reliability scores $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$
Ensure: Verified global model update w_g

Initialize DeviceInfo, ModelUpdate structures
for each registered device $ed_i \in \mathcal{ED}$ **do**
  $DeviceInfo[ed_i] \leftarrow \{active : \text{true}, reliability : r_i, timestamp : t\}$
  $ModelUpdate[ed_i] \leftarrow \{model : w_i, quality : q_i, verified : \text{false}\}$
end for
for each update round t **do**
  for each active device $ed_i \in \mathcal{ED}$ **do**
    if $ValidateDevice(ed_i)$ **then**
      $commitment \leftarrow GenerateCommitment(w_i)$
      $Commitments[ed_i] \leftarrow commitment$
      $shares[ed_i] \leftarrow SecureShare(w_i)$
    end if
  end for
  $maskedUpdates \leftarrow \emptyset$
  for each committed device ed_i **do**
    if $ValidateCommitment(ed_i)$ **then**
      $\alpha_i \leftarrow q_i \cdot r_i \cdot \exp(-\beta \|w_i - \bar{w}\|^2)$ ▷ From Equation (36)
      $maskedUpdates \leftarrow maskedUpdates \cup \{DecryptShare(shares[ed_i]) \times \alpha_i\}$
    end if
  end for
  $w^{t+1} \leftarrow \sum(maskedUpdates) / \sum(\alpha_i)$ ▷ Adaptive aggregation
  if $H(T(w^{t+1} \parallel n)) < Target$ **then**
    $RecordUpdate(w^{t+1})$
    for each $ed_i \in \mathcal{ED}$ **do**
      $ModelUpdate[ed_i].model \leftarrow w^{t+1}$
      $ModelUpdate[ed_i].verified \leftarrow \text{true}$
    end for
    if convergence criteria are met **then**
      break
    end if
  end if
end for
return $w_g \leftarrow w^{t+1}$

4.3. Operational Design and Complexity Analysis

Algorithm 2 presents the functional design of the suggested EdgeGuard framework. The four main parts of the EdgeGuard algorithm—model update distribution, adaptive aggregation, blockchain operations, and edge device computations—determine how long it takes to run. Edge device computations including local model training and data preparation have temporal complexity $O(I \times n)$, where n is the local dataset's data point count and I is the number of training iterations. Using a temporal complexity of $O(M \times 2^d)$, where M is the number of blocks and d is the PoW difficulty level, the blockchain layer generates and verifies secure blocks using the proof of work (PoW) consensus technique. With ed denoting the number of edge devices and N denoting the overall number of model parameters, the adaptive aggregation processes of the central server introduce a complexity of $O(N \times ed)$.

This process aggregates updates from every edge device involved in participating. Moreover, the methods of quality assessment and dependability evaluation add a complexity of $O(ed \times K)$, where K is the count of evaluation criteria. As such, the entire time complexity of the EdgeGuard approach could be estimated as follows:

$$O(I \times n) + O(M \times 2^d) + O(N \times ed) + O(E \times K) \tag{41}$$

There is a balance between safe blockchain operations, local computations, and flexible global aggregation in the IoMT setting. This complexity analysis shows how EdgeGuard is spread out.

Algorithm 2 EdgeGuard: secure federated learning for IoMT.

Require: Set of edge devices $\mathcal{E} = \{ed_1, ed_2, \ldots, ed_N\}$, local datasets $\mathcal{D} = \{D_1, D_2, \ldots, D_N\}$, global model w_g, blockchain $\mathcal{B} = (B, T, \sigma, V)$, central server $\mathcal{C}$
Ensure: Updated global model w_g, secure and private federated learning

Initialize w_g, $\mathcal{B}$
for each communication round t **do**
  $\mathcal{E}_t \leftarrow$ Select participating devices
  for each $e_i \in \mathcal{E}_t$ in parallel **do**
    Preprocess local data D_i
    $w_i^t \leftarrow$ Train local model on D_i
    $q_i^t \leftarrow Q(D_i)$ ▷ Quality Assessment
    $r_i^t \leftarrow R(e_i, t)$ ▷ Reliability Evaluation
    $\tilde{w}_i^t \leftarrow w_i^t + \mathcal{N}(0, \sigma^2)$ ▷ Apply differential privacy
    $b_k \leftarrow \sigma(b_{k-1} \parallel T(\tilde{w}_i^t, q_i^t, r_i^t))$ ▷ Create blockchain block
    while $H(b_k) \geq$ Target **do**
      Adjust nonce in b_k
    end while
    Broadcast b_k to other miners
  end for
  $\mathcal{C}$ retrieves $\{\tilde{w}_i^t, q_i^t, r_i^t\}_{i \in \mathcal{E}_t}$ from $\mathcal{B}$
  $w_g^{t+1} \leftarrow f(\{\tilde{w}_i^t\}_{i \in \mathcal{E}_t}, \{q_i^t\}_{i \in \mathcal{E}_t}, \{r_i^t\}_{i \in \mathcal{E}_t})$ ▷ Adaptive aggregation
  $b_{k+1} \leftarrow \sigma(b_k \parallel T(w_g^{t+1}))$ ▷ Record global update in blockchain
  for each $e_i \in \mathcal{E}$ **do**
    e_i retrieves w_g^{t+1} from $\mathcal{B}$
    $w_i^{t+1} \leftarrow w_g^{t+1}$ ▷ Update local model
  end for
  if convergence criteria met **then**
    break
  end if
end for
return w_g

5. Performance Analysis

5.1. Experimental Setup

The EdgeGuard framework was evaluated using a comprehensive simulation environment built with PyTorch v1.9.0 for federated learning implementation, integrated with Ethereum Ganache v2.5.4 for blockchain simulation. The experiments were conducted on a server equipped with 2 Intel® Xeon® Silver 4114 CPUs (Santa Clara, CA, USA) (40 cores, 2.20 GHz), 128 GB RAM, running Ubuntu 20.04 LTS, and NVIDIA Tesla V100 GPU with 32 GB memory. The federated learning environment was implemented using PyTorch DistributedDataParallel, incorporating our custom adaptive aggregation mechanism with differential privacy support through PyTorch DP. The blockchain component utilized a private Ethereum network with smart contracts written in Solidity v0.8.0, specifically modified for IoMT requirements. Web3.py v5.28.0 facilitated smart contract interactions, while NumPy v1.21.0 and Pandas v1.3.0 were used for efficient data manipulation and numerical computations. To simulate the IoMT environment, we configured three types of edge devices with varying computational capabilities, as shown in Table 2. The network environment was configured to reflect real-world conditions, with bandwidth variations

from 1–10 Mbps, latency ranges of 10–100 ms, and packet loss rates of 0.1–1% in a star topology with a central aggregator.

Table 2. Edge device configurations.

Device	CPU Cores	MIPS	RAM (GB)	Storage (GB)	Power (W)
E1	2	2660	4	32	5
E2	4	3067	8	64	8
E3	8	3467	16	128	12

Dataset: We utilized the MIMIC-III dataset, performing comprehensive preprocessing including temporal alignment of vital signs, missing value imputation using forward fill, feature normalization, and time series segmentation into 24-h windows. The dataset was split into training (80%), validation (10%), and test (10%) sets. Performance monitoring was conducted using Linux perf-tools v5.15.0 for resource utilization, iperf3 v3.12 for network statistics, and Intel RAPL (Running Average Power Limit) through powercap-utils v0.6.0 for energy consumption measurements.

5.2. Baseline Implementation

For comparative analysis, we implemented FedAvg as our baseline following McMahan et al.'s seminal work [22]. The FedAvg implementation performs standard federated averaging without the security enhancements of EdgeGuard. In each communication round, the server selects a fraction of available clients ($C = 0.8$) and broadcasts the current global model. Each selected client trains the model on their local data for E epochs and returns the model updates. The global model is then updated using the following:

$$w_g^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^t \tag{42}$$

where n_k represents the size of the local dataset at client k, n denotes the total dataset size, and w_k^t represents the local model parameters at round t. This vanilla implementation differs from EdgeGuard in several key aspects:

- No quality-based weighting of client updates.
- No reliability assessment of participating devices.
- No blockchain-based security mechanisms.
- No differential privacy protections.

Both EdgeGuard and the FedAvg baseline were implemented using the same optimization framework to ensure a fair comparison. The optimization configuration incorporates SGD with momentum as the base optimizer and includes additional enhancements such as cosine annealing for learning rate scheduling and gradient clipping to improve training stability. Table 3 provides a comprehensive list of all parameters used in our experiments, including the detailed optimization configuration that was previously omitted.

Table 3. Simulation parameters.

Parameter	Value
System Configuration	
Number of Edge Devices	50–500
Number of IoMT Sensors	100–2000
CPU Cores per Server	40
RAM per Server	128 GB
GPU Memory	32 GB
Federated Learning Parameters	
Train/Test Split	80:20
Local Epochs	10–50
Batch Size	64
Communication Rounds	1–300
Client Selection Rate	0.8
Malicious Devices	{10%, 20%, . . . , 50%}
Optimization Parameters	
Base Learning Rate	0.01
Optimizer	SGD with Momentum ($\beta = 0.9$)
Learning Rate Scheduler	Cosine Annealing
Weight Decay	1×10^{-4}
Gradient Clipping	1.0
Early Stopping Patience	10 epochs
Momentum	0.9
Blockchain Parameters	
Block Generation Rate (λ)	{0.1, 0.3, 0.5, 0.7}
Consensus Algorithm	Proof of Work
Gas Limit	6,721,975
Block Time	15 s
Smart Contract Version	Solidity v0.8.0
Network Parameters	
Bandwidth Range	1–10 Mbps
Latency Range	10–100 ms
Packet Loss Rate	0.1–1%
Network Topology	Star
Dataset Configuration	
Training Set	80%
Validation Set	10%
Test Set	10%
Time Window	24 h
Sampling Rate	5 min
Security Parameters	
Differential Privacy ε	0.1–1.0
Privacy Budget δ	10^{-5}
Encryption Method	AES-256
Key Length	2048 bits

5.3. Evaluation and Simulation Results

We tested EdgeGuard on the MIMIC-III dataset in a simulated IoMT environment. Our evaluation focused on model accuracy, communication efficiency, security robustness, and resource utilization in four key aspects.

The experimentation environment maintains consistent conditions for both implementations, utilizing the MIMIC-III dataset with identical preprocessing steps and evaluation metrics. This setup ensures a fair comparison while highlighting EdgeGuard's enhanced security and efficiency features.

5.3.1. Model Accuracy

Figure 2 presents how the accuracy of the model converges to rounds of communication in EdgeGuard compared to standard federated learning (FedAvg) and a centralized approach.

This resulted in an average test error of 94.3%, which was higher than that of FedAvg at 91.7% and closer to the 95.5% error rate achieved by a centralized approach. The adaptive aggregation mechanism contributed to faster convergence toward this higher final accuracy in the federated setting.

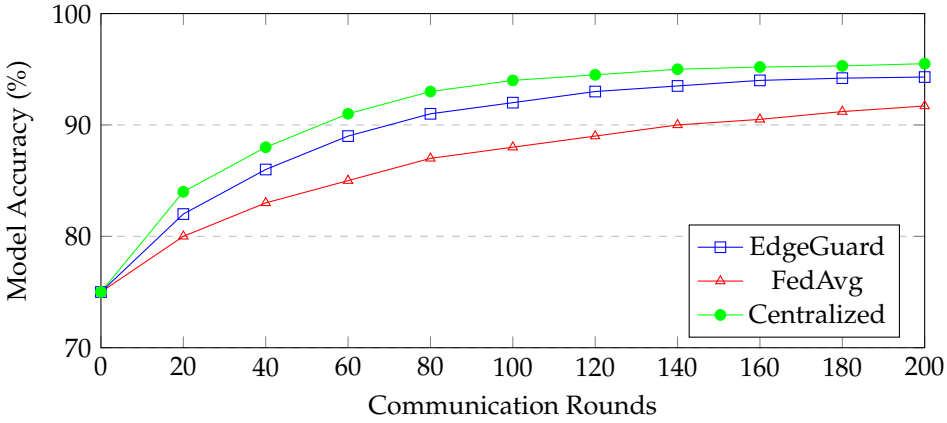


Figure 2. Model accuracy convergence.

5.3.2. Communication Efficiency

Figure 3 shows the total amount of data transferred during the training process for different numbers of edge devices.

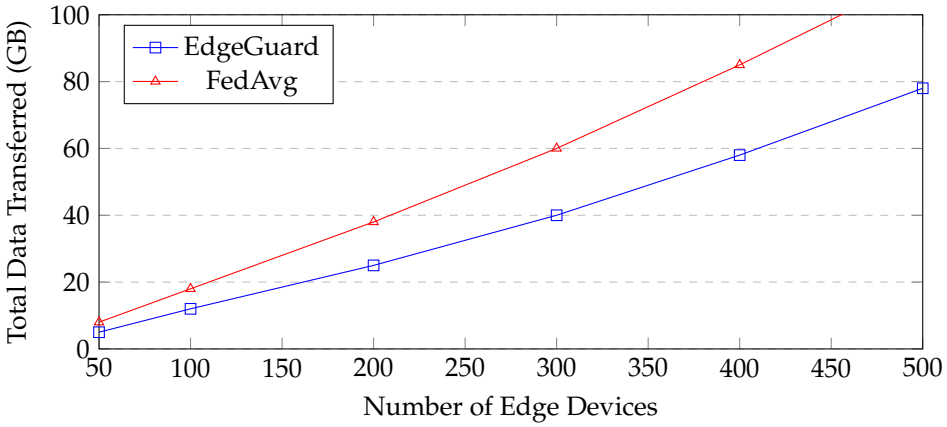


Figure 3. Communication efficiency.

EdgeGuard outperformed FedAvg in terms of communication efficiency, cutting the overall amount of data sent by as much as 30%, especially as the number of edge devices rose.

5.3.3. Security Robustness

Figure 4 illustrates EdgeGuard's performance under varying percentages of malicious nodes for different types of attacks.

EdgeGuard maintained an accuracy of over 90% even with up to 40% of the nodes being malicious, demonstrating strong resilience against data poisoning, model poisoning, and Sybil attacks.

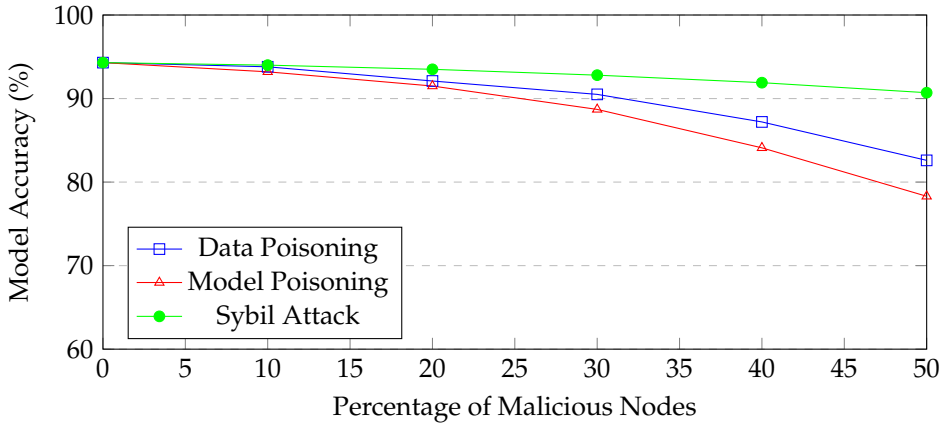


Figure 4. Security Robustness Under Different Attacks.

5.3.4. Resource Utilization

Table 4 presents the average resource utilization per edge device type during training.

EdgeGuard showed effective resource management across a variety of device types. Compared to standard federated learning, the integration of blockchain activities resulted in an average 15% increase in energy usage. However, a decrease in communication overhead made up for this. To summarize, EdgeGuard outperformed conventional federated learning in terms of model correctness, communication effectiveness, and security robustness while exhibiting tolerable resource use increases. The adaptive methods of the system demonstrated efficacy in preserving performance in adversarial IoMT scenarios while optimizing resource utilization among heterogeneous edge devices.

Table 4. Resource utilization.

Device	CPU (%)	RAM (GB)	Network (MB/s)	Energy (Wh)
E1	78.5	3.2	0.8	12.6
E2	65.3	6.1	1.2	20.1
E3	52.1	11.8	1.5	28.5

6. Conclusions

In this paper, we present EdgeGuard, a unique architecture that improves federated learning in IoMT contexts in terms of security, efficiency, and speed. Through EdgeGuard's integration of blockchain technology and adaptive federated learning, data privacy and integrity are guaranteed across distributed edge devices. Our analysis reveals that EdgeGuard resists up to 40.05% of malicious nodes while achieving a model accuracy of 94.34%, surpassing typical techniques by 2.68%. It also optimizes resource utilization in IoMT scenarios by reducing communication overhead by 30.67%. Coupled with proof of work consensus and differential privacy approaches, the framework's adaptive aggregation mechanism provides a robust defense against various threats and ensures the protection of patient data. Thus, EdgeGuard offers a complete solution for machine learning in decentralized healthcare ecosystems that is safe, effective, and privacy-preserving, greatly advancing the development of reliable AI-driven healthcare applications.

Author Contributions: S.P.: Conceptualization, methodology, formal analysis, validation, writing original draft, Conceptualization; J.L.: Formal analysis, validation, visualization, review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to privacy and ethical restrictions.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Al-Turjman, F.; Nawaz, M.H.; Ulusar, U.D. Intelligence in the Internet of Medical Things era: A systematic review of current and future trends. *Comput. Commun.* **2020**, *150*, 644–660. [CrossRef]
2. Huang, C.; Wang, J.; Wang, S.; Zhang, Y. Internet of medical things: A systematic review. *Neurocomputing* **2023**, *557*, 126719. [CrossRef]
3. Chang, Y.; Fang, C.; Sun, W. A Blockchain-Based Federated Learning Method for Smart Healthcare. *Comput. Intell. Neurosci.* **2021**, *2021*, 4376418. [CrossRef] [PubMed]
4. Raul, S.; Das, S.; Murty, C.S.; Devi, K. A Review on Intelligent Health Care System Using Learning Methods. In *Recent Developments in Electronics and Communication Systems*; IOS Press: Amsterdam, The Netherlands, 2023; pp. 154–159.
5. Mohammed, M.A.; Lakhan, A.; Abdulkareem, K.H.; Zebari, D.A.; Nedoma, J.; Martinek, R.; Kadry, S.; Garcia-Zapirain, B. Energy-efficient distributed federated learning offloading and scheduling healthcare system in blockchain based networks. *Internet Things* **2023**, *22*, 100815. [CrossRef]
6. Duan, Q.; Huang, J.; Hu, S.; Deng, R.; Lu, Z.; Yu, S. Combining Federated Learning and Edge Computing Toward Ubiquitous Intelligence in 6G Network: Challenges, Recent Advances, and Future Directions. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 2892–2950. [CrossRef]
7. Myrzashova, R.; Alsamhi, S.H.; Shvetsov, A.V.; Hawbani, A.; Wei, X. Blockchain meets federated learning in healthcare: A systematic review with challenges and opportunities. *IEEE Internet Things J.* **2023**, *10*, 14418–14437. [CrossRef]
8. Khan, M.F.; AbaOud, M. Blockchain-Integrated Security for real-time patient monitoring in the Internet of Medical Things using Federated Learning. *IEEE Access* **2023**, *11*, 117826–117850. [CrossRef]
9. Sai, S.; Chamola, V.; Choo, K.K.R.; Sikdar, B.; Rodrigues, J.J.P.C. Confluence of Blockchain and Artificial Intelligence Technologies for Secure and Scalable Healthcare Solutions: A Review. *IEEE Internet Things J.* **2023**, *10*, 5873–5897. [CrossRef]
10. Khubrani, M.M. Artificial Rabbits Optimizer with Deep Learning Model for Blockchain-Assisted Secure Smart Healthcare System. *Int. J. Adv. Comput. Sci. Appl.* **2023**, *14*. [CrossRef]
11. Manzoor, H.U.; Shabbir, A.; Chen, A.; Flynn, D.; Zoha, A. A Survey of Security Strategies in Federated Learning: Defending Models, Data, and Privacy. *Future Internet* **2024**, *16*, 374. [CrossRef]
12. Patni, S.; Lee, J. Explainable AI Empowered Resource Management for Enhanced Communication Efficiency in Hierarchical Federated Learning. *Comput. Electr. Eng.* **2024**, *117*, 109260. [CrossRef]
13. Otoum, Y.; Hu, C.; Said, E.H.; Nayak, A. Enhancing Heart Disease Prediction with Federated Learning and Blockchain Integration. *Future Internet* **2024**, *16*, 372. [CrossRef]
14. Solat, F.; Patni, S.; Lim, S.; Lee, J. Heterogeneous Privacy Level-Based Client Selection for Hybrid Federated and Centralized Learning in Mobile Edge Computing. *IEEE Access* **2024**, *12*, 108556–108572. [CrossRef]
15. Abaoud, M.; Almuqrin, M.A.; Khan, M.F. Advancing Federated Learning Through Novel Mechanism for Privacy Preservation in Healthcare Applications. *IEEE Access* **2023**, *11*, 83562–83579. [CrossRef]
16. Singh, M.B.; Singh, H.; Pratap, A. Energy-Efficient and Privacy-Preserving Blockchain Based Federated Learning for Smart Healthcare System. *IEEE Trans. Serv. Comput.* **2024**, *17*, 2392–2403. [CrossRef]
17. Yu, S.; Chen, X.; Zhou, Z.; Gong, X.; Wu, D. When Deep Reinforcement Learning Meets Federated Learning: Intelligent Multitimescale Resource Management for Multiaccess Edge Computing in 5G Ultradense Network. *IEEE Internet Things J.* **2021**, *8*, 2238–2251. [CrossRef]
18. Bashir, A.K.; Victor, N.; Bhattacharya, S.; Huynh-The, T.; Chengoden, R.; Yenduri, G.; Maddikunta, P.K.R.; Pham, Q.V.; Gadekallu, T.R.; Liyanage, M. Federated Learning for the Healthcare Metaverse: Concepts, Applications, Challenges, and Future Directions. *IEEE Internet Things J.* **2023**, *10*, 21873–21891. [CrossRef]
19. Hidayat, M.A.; Nakamura, Y.; Arakawa, Y. Enhancing Efficiency in Privacy-Preserving Federated Learning for Healthcare: Adaptive Gaussian Clipping with DFT Aggregator. *IEEE Access* **2024**, *12*, 88445–88457. [CrossRef]
20. Rajagopal, S.M.; Supriya, M.; Buyya, R. Blockchain Integrated Federated Learning in Edge/Fog/Cloud Systems for IoT-Based Healthcare Applications: A Survey. In *Federated Learning*; Taylor & Francis Group: Oxfordshire, UK, 2024; pp. 237–269. [CrossRef]

21. Rahman, M.A.; Hossain, M.S.; Islam, M.S.; Alrajeh, N.A.; Muhammad, G. Secure and provenance enhanced internet of health things framework: A blockchain managed federated learning approach. *IEEE Access* **2020**, *8*, 205071–205087. [CrossRef] [PubMed]
22. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, PMLR, Ft. Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.

future internet

MDPI

Article

On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture

Niroshinie Fernando *, Samir Shrestha, Seng W. Loke and Kevin Lee

School of Information Technology, Deakin University, Geelong, VIC 3220, Australia; shresthasam@deakin.edu.au (S.S.); seng.loke@deakin.edu.au (S.W.L.); kevin.lee@deakin.edu.au (K.L.)
* Correspondence: niroshinie.fernando@deakin.edu.au

Abstract: Edge, fog, and cloud computing provide complementary capabilities to enable distributed processing of IoT data. This requires offloading mechanisms, decision-making mechanisms, support for the dynamic availability of resources, and the cooperation of available nodes. This paper proposes a novel 3-tier architecture that integrates edge, fog, and cloud computing to harness their collective strengths, facilitating optimised data processing across these tiers. Our approach optimises performance, reducing energy consumption, and lowers costs. We evaluate our architecture through a series of experiments conducted on a purpose-built testbed. The results demonstrate significant improvements, with speedups of up to 7.5 times and energy savings reaching 80%, underlining the effectiveness and practical benefits of our cooperative edge-fog-cloud model in supporting the dynamic computational needs of IoT ecosystems. We argue that a multi-tier (e.g., edge-fog-cloud) dynamic task offloading and management of heterogeneous devices will be key to flexible edge computing, and that the advantage of task relocation and offloading is not straightforward but depends on the configuration of devices and relative device capabilities.

Keywords: edge computing; fog computing; cloud computing; device-enhanced edge

Citation: Fernando, N.; Shrestha, S.; Loke, S.W.; Lee, K. On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet* **2025**, *17*, 22. https://doi.org/10.3390/fi17010022

Academic Editors: Yuezhi Zhou and Xu Chen

Received: 12 November 2024
Revised: 20 December 2024
Accepted: 1 January 2025
Published: 7 January 2025

Citation: Fernando, N.; Shrestha, S.; Loke, S.W.; Lee, K. On Edge-Fog-Cloud Collaboration and Reaping Its Benefits: A Heterogeneous Multi-Tier Edge Computing Architecture. *Future Internet* **2025**, *17*, 22. https://doi.org/10.3390/fi17010022

1. Introduction

Cloud computing, despite its widespread adoption, struggles to satisfy the latency and bandwidth demands of IoT applications, necessitating the integration of edge and fog computing to complement cloud capabilities [1,2]. These paradigms, by situating computation closer to data at the network's edge, address latency and bandwidth issues and foster a cooperative dynamic among edge, fog, and cloud layers [3,4]. In this synergy, the device-enhanced edge model enables utilising idle computational resources of IoT devices themselves [5]. This approach not only alleviates the pressure on traditional edge servers but also promotes flexibility and resource efficiency. But the deployment of a cooperative device-enhanced edge-fog-cloud architecture presents significant challenges, including task allocation across heterogeneous resources, dynamic node availability, and maintaining Quality of Service (QoS).

Building upon the Honeybee model [6], we propose an integrated architecture for coordinating resources across edge, fog, and cloud tiers, and empirically evaluate its efficacy. This paper aims to investigate the following research questions within this architecture:

- RQ1: How does node collaboration across edge, fog, and cloud layers impact overall task performance?

- RQ2: How does adjusting task-sharing parameters affect the system's behaviour?
- RQ3: How can dynamic node availability, where heterogeneous nodes may leave or join without prior warning, be handled in a cooperative edge, fog, cloud setting?

Our contributions include a conceptual architecture for dynamic edge-fog-cloud collaboration, the development of a real-world prototype, and an empirical evaluation demonstrating the approach's effectiveness in a physical testbed.

The paper is organised as follows: Section 2 outlines the background and motivation for our architectural approach. Section 3 reviews related work in computation offloading. Section 4 details the Honeybee architecture and its enhancements. Section 5 presents our experimental evaluation, followed by conclusions and future work directions in Section 7.

2. Background and Motivation

The edge-fog-cloud architecture is commonly envisioned as a 3-layered model [7–9]. In this model the edge layer is at the bottom and directly connected to end-user IoT devices, the middle tier is composed of fog nodes, and the cloud layer is at the top, as shown in Figure 1. There are other interpretations of how the interactions between edge, fog, and cloud may occur [10], and this paper follows the model in Figure 1. In this paper, we envision a computing architecture where the IoT devices themselves are utilised as edge resource providers. The idea of a collection of end-user IoT devices collaboratively working as a collective computing resource has been explored in the literature under various terms, such as 'mobile device clouds' [11], 'mobile edge-clouds' [12], 'human-driven edge computing [13], 'collaborative multi-device computing' [14], 'mobile crowd computing' [6] and 'device-enhanced multi-access edge computing' [5] (device-enhanced MEC). In this paper, we use the term 'device-enhanced MEC' to refer to the bottom tier of Figure 1, which provides edge computing services to end-user IoT devices.

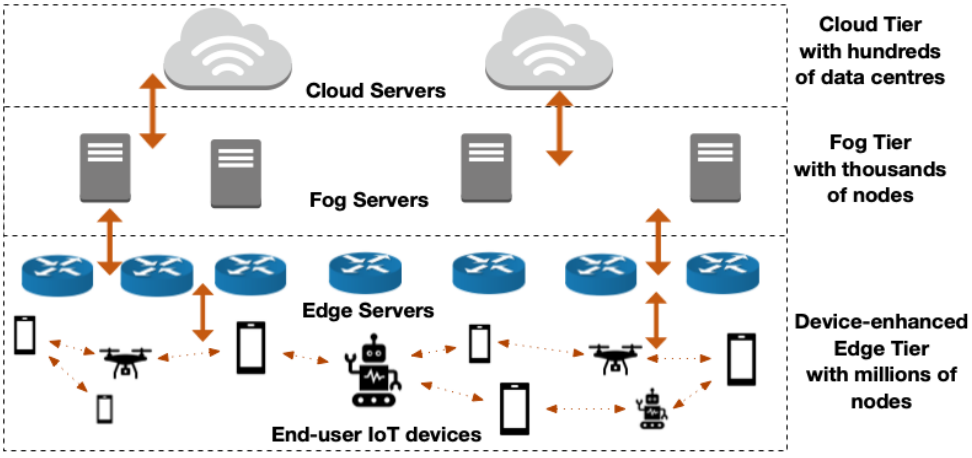


Figure 1. A 3-tier architecture for device-enhanced edge, fog, and cloud computing. End-user IoT devices such as smartphones, drones, and robots are integrated as edge resources, forming a local collective resource, and work collaboratively with conventional edge, fog, and cloud servers.

As shown in the aforementioned research, integrating end-user IoT devices to the edge tier as resource providers can provide performance gain, energy savings, and increased resource utilisation and availability. However, when considering real-life constraints, the co-cooperation amongst device-enhanced edge, fog, and cloud layers becomes essential. We illustrate this in the motivating scenario illustrated in Figure 2.

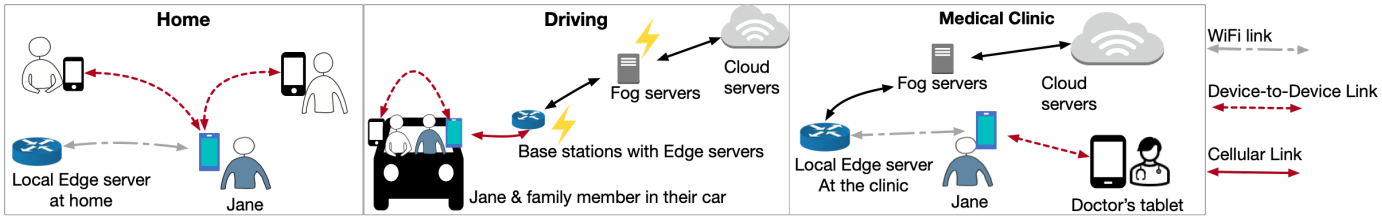


Figure 2. Scenario showing how different contexts require collaboration amongst resource nodes at edge, fog, and cloud tiers.

Here, Jane, has a medical condition which requires her to use a wearable device for constant monitoring of health data. The application QoS requirements in this case include near-real-time response time, a particular level of accuracy, high availability, and secure handling of Jane's sensitive medical data. She is also concerned about the battery drain of her mobile device and data access fees.

The wearable device sends the sensor data to her smartphone for analysis which is not powerful enough to run the AI algorithms required to process this high-velocity stream of data continuously. Hence, this application requires the support of external resources to provide accurate and timely results. Jane's smartphone has a collaborative resource-sharing middleware installed, which runs in the background of the smartphone and the resource nodes, and acts as an intermediary between applications and external resources. Applications that need external resources connect to available resources through the middleware.

In Figure 2, when Jane is at home, Jane's smartphone connects her to a local collective resource, made up of the collective resources of her family members' smartphones and her home edge server to carry out the sensor data processing tasks. In this situation, all of the participating resource nodes can be considered to be at the edge. Since all of the nodes are trusted, and connectivity is robust, Jane does not have the need to request the support of remote servers.

When Jane is traveling in the car with another family member, the local collective resource is made of just two smartphones-Jane's and her family member. Since this particular resource collection does not provide sufficient computing resources to satisfy the QoS requirements of Jane's app, she further shares the app workload with a conventional edge server located at a base station, as well as with fog and cloud servers. In this situation, the entire workload is not offloaded to the servers, but shared amongst the two smartphones (which form the local collective resource in this case) as well.

The app workload involves a large amount of data; hence, reducing data transfers via cellular links is beneficial, due to latency, data access fees, and energy usage. During the drive, Jane's smartphone suffers intermittent data connectivity, and so a considerable amount of work is being supported by the local collective resource, instead of the other servers. How much work is performed by the two smartphones (the local collective resource) and the conventional edge, fog, and cloud servers can depend on their availability and latency, as well as the task scheduling algorithm in Jane's collaborative resource sharing middleware. When the cellular data connectivity drops, this can increase the latency of conventional edge, fog, and cloud servers, impacting their performance. In this case, the two smartphones should be able to pick up most of the workload.

When Jane is at the medical clinic (rightmost situation in Figure 2), her smartphone forms a local collective resource with her doctors tablet via D2D and with the clinic edge server via WiFi. She further shares her work with fog and cloud servers because the doctor's diagnostic process requires the app to provide faster performance. As Jane uses the clinic's high-speed WiFi to connect to the remote servers, there is no concern of intermittent data

connectivity or data access fees. In this situation, the fog and cloud servers should be able to pick up most of the workload due to the availability of high-speed connectivity.

In the three aforementioned contexts described in the scenarios in Figure 2, Jane shared her work with resource nodes at the device level, edge, fog, and cloud levels in various degrees. The three situations had different constraints in terms of connectivity, trust and security, and amount of available resources, yet the application QoS requirements remained the same. Hence, to continue to meet the QoS requirements, the collaboration across edge, fog, and cloud needs to adapt dynamically depending on the context.

3. Related Work

Existing surveys on edge, fog, and cloud computing have comprehensively discussed the synergy between IoT and edge, fog, and cloud computing paradigms, explaining how edge and fog computing can bridge the gap between IoT and cloud computing by moving the computation closer to the end-user IoT devices, thereby addressing issues with energy, latency, and context awareness [4,5,15–18]. Numerous scholarly papers have extensively addressed diverse facets of edge, fog, and cloud tiers, taken as separate tiers. Specifically, for device-enhanced MEC, additional complexities need to be considered due to the dynamic nature of device-based resource providers and their intrinsic characteristics, such as distributed ownership, mobility, finite energy, resource constraints, increased proximity to other device-based resource providers, and Device-to-Device (D2D) communication capacity [5,15,19–22]. Various aspects of the technical feasibility of the lowest tier of device-enhanced MEC, where devices such as smartphones function as edge resource providers, have been demonstrated in frameworks such as MMPI [23], Hyrax [24], MClouds [25], Aura [26], and Honeybee [6]. However, these existing works have not investigated how the device-based resource providers can collaborate and share work with nodes at fog and cloud layers, considering various overheads, impacts on performance, and battery and various offloading parameters.

As highlighted in a recent work on the convergence of edge, fog, and cloud, only a few papers have yet investigated the interactions between these three paradigms in a 3-tier edge-fog-cloud architecture [27]. Below, we discuss related work that has explored this under-researched area. Here, we only focus on work that has investigated the interactions and collaborations between at least two of the edge, fog, and cloud tiers, and do not consider work that only focuses on one tier.

One of the first papers to explore **edge-fog-cloud** collaboration is Flores et al. [20], who proposed the HyMobi framework, which allows a mobile application to interoperate between device-enhanced MEC, fog, and cloud tiers. HyMobi has an incentive mechanism based on credit and reputation, and allows users to lease the resources of their devices as an open commodity in the edge tier, exploiting the social characteristics of the devices to form offloading communities. The effectiveness of HyMobi is demonstrated via a proof-of-concept implementation and a testbed, evaluating the performance and energy consumption of mobile applications and infrastructure awareness in a social-aware environment. However, the performance results are not explored thoroughly with different edge-fog-cloud combinations. Other papers that discuss all three tiers of edge-fog-cloud include [21,28,29]. In [21], a task can be offloaded to either edge, fog or cloud, or executed on the device itself; however, no collaboration amongst the tiers is mentioned. Although the algorithm in [28] can be applied to all three layers of edge-fog-cloud to reduce latency and power consumption, the paper only considers the fog layer. In [29], a Min-Min algorithm, considering cost, makespan, energy, and load balancing, is used to schedule tasks amongst the edge-fog-cloud tiers.

The coordination of work offloading amongst the **edge and cloud** tiers is discussed in [22,30,31]. In [22], the authors propose the HyFog framework, which considers interactions between device-enhanced MEC and cloud, and allows devices to choose the execution mode among local mode, D2D mode, and cloud mode. Simulations show that HyFog's three-layer graph-matching algorithm-based solution is able to minimise the power consumption while ensuring latency requirements. However, there is no discussion about adapting to dynamic conditions, such as device mobility. In comparison, refs. [19,32] only consider collaborated work offloading inside the device-enhanced MEC layer itself, between the devices as resource providers and the conventional edge server/s.

Offloading in hybrid **fog/cloud** systems is discussed in [33–35]. In [33], the authors present a scheme for the joint optimisation of transmit power control, computation and radio bandwidth allocation when offloading in hybrid fog/cloud systems, while guaranteeing user fairness and maximum tolerable delay. In [34], Zahoor et al. present a cloud-fog-based architecture in the context of a smart grid. The authors discuss simulation results of using round robin, throttled, and particle swarm optimisation algorithms to schedule requests from devices such as smart meters on the VMs of a fog-cloud architecture. However, it is unclear if there is any collaboration amongst the fog-cloud tiers. Kumar and Karri [35] present the EEOA (electric earthworm optimisation algorithm) for efficient resource assignment and work scheduling amongst fog-cloud tiers, considering the makespan, cost, and energy usage. Simulation results show that in many cases, the proposed EEOA performs better than alternative methods.

Overall, few papers have considered collaboration amongst all three tiers of edge-fog-cloud [20,21,28,29]. However, amongst all the papers considered in this section, only [20] has used an actual test bed without solely depending on simulations. Although important, simulations lack real-world variability, and do not always capture emergent behaviours arising from the interactions of various components in the edge, fog, and cloud tiers. Only work reported in [19,20,22,29–32] has provided clear evidence of supporting inter-tier interactions and collaborations, and a majority do not discuss support for dynamic conditions. In this paper, we address the gaps highlighted above.

4. An Architecture for Edge-Fog-Cloud Offloading

This section introduces our architecture for supporting inter-tier collaboration amongst edge, fog, and cloud tiers in a device-enhanced MEC context. We have chosen to extend and build on the Honeybee framework since it is open-source, supports proactive worker-centric offloading, and has automatic load-balancing and fault-tolerant mechanisms. Honeybee has also been extended in other work to work with drones [36], robots [37] and dependency-based task scheduling [38], showing its extensibility.

The Honeybee framework (https://github.com/niroshini/honeybee, accessed on 18 December 2023) is an Android implementation of the Honeybee mode [6,39], and enables the formation of mobile edge clouds, via peer-to-peer (P2P) connections using Wi-Fi Direct. In the Honeybee model, the device with the task to be completed is called the 'delegator'. The original task is first decomposed to a pool of jobs, and then is offloaded to 'worker' devices in the vicinity, while also carrying out a portion of the jobs by itself. Honeybee's task-scheduling algorithm is based on work stealing [40] to automatically load-balance the jobs amongst the delegator and the workers. The workers must proactively request to 'steal' jobs, and each time the delegator receives a steal request, it will respond to the requesting worker with a chunk of jobs. In this way, faster workers are able to steal more and more jobs, thus avoiding performance bottlenecks with faster nodes having to wait for slower nodes. The Honeybee model is also able to handle random disconnections as well as exploit random resource node encounters. To address the critical aspect of

incentivising participation of workers, we have previously explored the socio-technical requirements and engineering challenges of this paradigm, identifying effective incentive mechanisms [41]. Our findings provided insights into aligning user motivations with application goals through both intrinsic-social and extrinsic-personal incentives, ensuring sustained engagement while mitigating resource depletion concerns. However, in this work, we do not focus on the design of incentive structures, instead assuming that appropriate mechanisms are already in place across the edge, fog, and cloud layers to support inter-tier collaboration. In this paper, we build on the Honeybee model to extend the cooperative work-stealing mechanism beyond the original Honeybee's local device cloud. We have extended the Honeybee delegator to be able to simultaneously maintain connections with multiple P2P edge servers via Wi-Fi Direct, as well as fog and cloud servers via the LAN and the internet, respectively. The Honeybee worker component has been extended to include support for Java implementations on fog and cloud servers. While our previous work [38] extended the Honeybee framework to support sequential dependency tasks, this study does not utilise the dependency-enabled extension, as it lies beyond the scope of the present investigation. Figure 3 shows a high-level view of the architecture of the extended framework for Edge-Fog-Cloud collaboration.

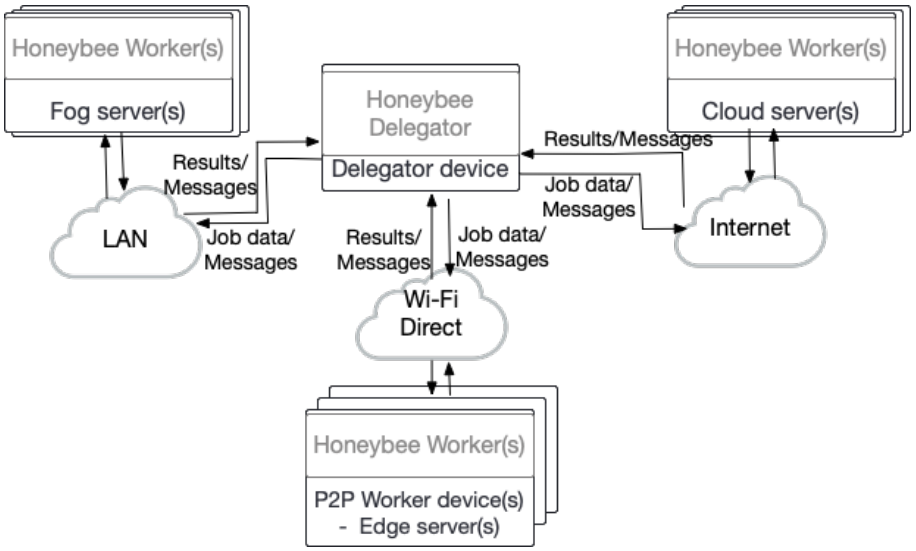


Figure 3. The edge-fog-cloud collaborative architecture.

Algorithms 1–4 provide an overview on how resource nodes at the edge, fog, and cloud collaboratively work on a distributed set of jobs. As shown in Algorithm 1, the delegator first initialises the job pool and starts consuming the jobs (for processing), while concurrently, it also starts the resource discovery threads for edge, fog, and cloud workers. Algorithm 3 describes the generalised periodic resource discovery thread. The delegator spawns three instances of this thread for workers at the three tiers of edge ($\mathcal{W}_e$), fog ($\mathcal{W}_f$), and cloud ($\mathcal{W}_c$). Each tier uses communication protocols applicable for that tier. Each instance of the three resource discovery threads operates independently, discovering resources specific to the worker type. Whenever a worker node is discovered, the system attempts to establish connections with them, via WiFi-Direct (for P2P edge), LAN (for fog), and internet (for cloud). Upon connecting, each worker will attempt to 'steal' from the delegator's job pool (see Algorithms 4 and 5). Whenever a worker's share of jobs is completed, it will send the results to the delegator, and without waiting to be assigned jobs, will **proactively** attempt to 'steal' jobs from the delegator.

Algorithm 1 Delegator's main thread

Input: Job pool $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ where $n > 0$, delegator d, edge workers $\mathcal{W}_e$, fog workers $\mathcal{W}_f$, cloud workers $\mathcal{W}_c$
Output: Completed jobs $\mathcal{J}c$, initially $\mathcal{J}c = \emptyset$
Initialise $\mathcal{J}$ to ensure $\mathcal{J} \neq \emptyset$
Start delegator's job consumer thread T_{d_con}
Start $T_{d\_comEdge}$ for edge workers
Start T_{d_comFog} for fog workers
Start $T_{d\_comCloud}$ for cloud workers
while $\mathcal{J} \setminus \mathcal{J}c \neq \emptyset$ **do**
    T_{d_con} consume $\mathcal{J}$
    Update $\mathcal{J}c$ with results from T_{d_con}
end while

Algorithm 2 Delegator's job consumer thread

Input: $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$, $n > 0$, d, $\mathcal{W}_e$, $\mathcal{W}_f$, $\mathcal{W}_c$
Output: $\mathcal{J}c$
Let $\mathcal{W} = \mathcal{W}_e \cup \mathcal{W}_f \cup \mathcal{W}_c$
while $\mathcal{J} \setminus \mathcal{J}c \neq \emptyset$ **do**
    if $\mathcal{J} \neq \emptyset$ **then**
        Consume $\mathcal{J}$
        Update $\mathcal{J}c$
    else Steal from a worker in $\mathcal{W}$ ▷ Traverse workers in connection order
    end if
end while

Algorithm 3 Delegator's periodic resource discovery thread for worker type W

Input: Worker type $W \in \{\mathcal{W}_e, \mathcal{W}_f, \mathcal{W}_c\}$; Communication protocol *Comm*
Output: Established connections stored in a ConcurrentHashMap $\mathcal{M}$
Initialise $\mathcal{M}$ as an empty ConcurrentHashMap
while resources are periodically discovered for W **do**
    for each worker $w \in W$ **do**
        Open socket connection using protocol *Comm*
        Create a *ClientSocketThread* instance for w
        $\mathcal{M}[w] \leftarrow$ ClientSocketThread instance
        Start a communication thread for w ▷ Algorithm 5
    end for
end while

Algorithm 4 Worker's main thread

Input: Worker job pool $\mathcal{J}_w$, initially $\mathcal{J}_w = \emptyset$, Stolen jobs $\mathcal{J}s$, d
Output: Worker's completed jobs $\mathcal{J}c_w$, initially $\mathcal{J}c_w = \emptyset$
Initialise: work_ongoing $\leftarrow$ true
while work_ongoing **do**
    if $\mathcal{J}s \neq \emptyset$ **then**
        $\mathcal{J}_w \leftarrow \mathcal{J}_w \cup \mathcal{J}s$
        while $\mathcal{J}_w \neq \emptyset$ **do**
            Consume $\mathcal{J}_w$
            Update $\mathcal{J}c_w$ and send to d
        end while
    else Steal from d
    end if
end while

Algorithm 5 Delegator's communication thread for each worker

Input: Worker w, job pool $\mathcal{J}$, connection map $\mathcal{M}$
Output: Updated $\mathcal{J}$, completed results $\mathcal{J}c$
Retrieve w's connection details from $\mathcal{M}$
Send *init_signal* to w
while *read* $\neq \emptyset$ **do**
    $m \leftarrow$ *read* ▷ m: received message
    if $m =$ steal_request **then**
        Send *no_jobs_left* if $\mathcal{J} = \emptyset$, else start victim thread T_{d_vic}
    else if $m =$ results **then**
        $\mathcal{J}c \leftarrow \mathcal{J}c \cup$ new results ▷ Store results
        if $\mathcal{J} \subseteq \mathcal{J}c$ **then**
            Execute *on_task_completed*
        end if
    else if $m =$ stolen_jobs **then**
        $\mathcal{J} \leftarrow \mathcal{J} \cup$ stolen jobs; start job consumer thread T_{d_con}
    else if $m =$ no_jobs **then**
        Mark w as idle
        if expired jobs $\neq \emptyset$ **then**
            $\mathcal{J} \leftarrow \mathcal{J} \cup$ expired jobs
        end if
    else if $m =$ worker_heartbeat **then**
        Update w's status in $\mathcal{M}$
    end if
end while

Algorithm 5 manages all communication between the worker node and the delegator node. This includes handling job-stealing requests, processing completed results, delegator functioning as a victim when a worker steals jobs from the delegator (described in Algorithm 6), expiring the oldest jobs (detailed in Algorithm 7), and sending termination signals upon task completion (detailed in Algorithm 8).

Algorithm 6 Delegator's victim thread

Input: Job pool $\mathcal{J}$, steal limit L_s, chunk size C_s
Output: Stolen job list $\mathcal{J}s$
Initialise $\mathcal{J}s \leftarrow \emptyset$
if $\mathcal{J} \neq \emptyset$ **then**
    if $|\mathcal{J}| > L_s$ **then**
        job $\leftarrow$ get first job in $\mathcal{J}$
    end if
    while *job* $\neq$ null **do**
        $\mathcal{J}s \leftarrow \mathcal{J}s \cup \{job\}$ ▷ Add job to stolen list
        if $|\mathcal{J}| \leq L_s$ **then**
            break
        end if
        if $|\mathcal{J}s| \geq C_s$ **then**
            break ▷ Steal chunk size is constant across all workers
        end if
        job $\leftarrow$ get next job in $\mathcal{J}$
    end while
end if
Transmit $\mathcal{J}s$ to worker

Algorithm 7 Delegator's job expiration method

1: **Input:** Pending results list $\mathcal{R}$
2: **Output:** Expired jobs $\mathcal{J}_{\text{expired}}$
3: Initialise $\mathcal{J}_{\text{expired}} \leftarrow \emptyset$
4: **if** $\mathcal{R} \neq \emptyset$ **then**
5: Select worker w_{ex} with the oldest pending jobs from $\mathcal{R}$
6: $\mathcal{J}_{\text{expired}} \leftarrow$ pending jobs of w_{ex}
7: **end if**
8: **return** $\mathcal{J}_{\text{expired}}$

Algorithm 8 Delegator's on_task_completed method

1: **Input:** Connection map of all connected workers $\mathcal{M}$
2: **Output:** Task completion signal sent, all connections closed
3: **for all** connected workers $w \in \mathcal{M}$ **do**
4: Send *termination_signal* to w
5: Close connection with w
6: **end for**

Over the course of program execution, any participating device can assume both roles of *thief* and *victim*. Hence, the worker can 'steal' from the delegator and the delegator can also 'steal' from workers if its own job pool is empty, but the task is not yet completed (see Algorithm 2). When a device receives a steal request, depending on its available job queue, it can decide to become a victim (see Algorithm 6). The victim then removes a C_s number of jobs (i.e., a chunk) from its own job list, and transmits them to the thief. The chunk size, C_s, defines the number of jobs a node can steal at a time. It is set as a small percentage of the total job count to minimise loss and enable easy reassignment if the worker disconnects.

- Stealing decisions: When the delegator steals from workers, workers are traversed sequentially in the order they connected to the delegator. The delegator attempts to steal jobs from each worker until successful or until all workers have been tried. Workers can only steal from the delegator due to an implementation constraint in peer-to-peer (P2P) connection protocols. In technologies such as Bluetooth and Wi-Fi Direct, only a star topology is supported, not a mesh. As a result, P2P connections between workers are not feasible.
- Conflict avoidance: The delegator maintains a synchronised lock on the job pool $\mathcal{J}$ during job allocation to prevent race conditions. This ensures that jobs are not simultaneously assigned to multiple workers.
- Complexity and overhead: The sequential worker selection ensures that worker traversals are $O(n)$, where n is the number of connected workers. Locking and job allocation operations remain $O(1)$, ensuring low overhead for each steal request.
- Fault tolerance: Worker heartbeats and job expiry are two main fault tolerance mechanisms in the framework. Each worker periodically sends a 'heartbeat' signal to the delegator to indicate its availability (see Algorithm 5). Other communications, such as job results, are also considered as heartbeats. If no heartbeat is received from a worker for a pre-determined consecutive interval, the delegator deems the worker as disconnected, and their assigned jobs are returned to the job pool for reassignment. If a job remains uncompleted beyond a predefined expiry time, it is marked as 'expired' and returned to the job pool, allowing other nodes, including the delegator, to process it (see Algorithms 5 and 7). The delegator invokes the job expiry mechanism only after its own job queue is exhausted and a steal attempt fails, ensuring efficient resource utilisation while preventing indefinite delays.

This proactive design lends itself for more opportunism as the availability and resourcefulness of each worker is unknown a priori, and subject to change any time [6]. For example, a worker's availability can be impacted if/when a worker node receives a call while it is participating in task execution, or if its location changes due to the owner moving away. This process continues until the pool is exhausted, or until a worker disconnects. The resource discovery threads are periodic; hence, potential workers can join at any point in time.

4.1. Upper and Lower Bounds for Speedup

In this section, extending previous work in Honeybee [6], we formulate theoretical upper and lower bounds to analyze the best- and worst-case speedup scenarios within a device-enhanced edge-fog-cloud computing architecture.

We assume that a given edge-fog-cloud computing environment consists of x P2P edge nodes, including the delegator, y fog workers, and z cloud workers, where x, y, z are non-negative integers.

Let us denote each node in the device-enhanced edge layer as n_{e_i}, where $1 \leq i \leq x$. The delegator is denoted as n_{e_1}. Each fog worker is denoted as n_{f_j}, where $1 \leq j \leq y$, and each cloud worker is denoted as n_{c_l}, where $1 \leq l \leq z$.

The time taken to complete m jobs on the delegator n_{e_1} is denoted as t_{e_1}. The time taken for an edge worker n_{e_i} to receive, complete, and return results for m jobs is denoted as t_{e_i}, where $i > 1$. Hence, the relative computational power of an edge worker n_{e_i} compared to the delegator n_{e_1} can be given as:

$$\frac{t_{e_i}}{t_{e_1}} = k_{e_i} \tag{1}$$

Similarly, the time taken for a fog server n_{f_j} to complete and return results for m jobs is denoted as t_{f_j}, and for a cloud server n_{c_l} as t_{c_l}. The relative computational powers of fog and cloud workers compared to the delegator n_{e_1} can be represented by constants k_{f_j} and k_{c_l}, respectively:

$$\frac{t_{f_j}}{t_{e_1}} = k_{f_j} \tag{2}$$

$$\frac{t_{c_l}}{t_{e_1}} = k_{c_l} \tag{3}$$

The parameters k_{e_i}, k_{f_j}, and k_{c_l}, introduced in Equations (1), (2), and (3), respectively, serve as indicators of the relative efficiency of task execution by worker nodes in comparison to the delegator. Specifically, a value of $k < 1$ signifies that the worker node, be it within the edge, fog, or cloud tier, can process and return the results of the assigned tasks more expediently than the delegator performing the same tasks in a monolithic manner. This computational advantage highlights the potential benefit of offloading tasks to worker nodes with $k < 1$. Conversely, a value of $k \geq 1$ suggests that task offloading may not yield substantial speedups, as the worker node does not demonstrate a computational speed advantage over the delegator.

Even when $k > 1$, indicating a worker's lower computational power compared to the delegator, offloading can still enhance overall performance. This improvement arises because the delegator and workers operate concurrently, increasing total system throughput. Essentially, the collective output of all computing nodes, despite individual limitations, can contribute to achieving speedup. Thus, the decision to offload tasks should consider the system-wide contribution, highlighting the importance of a holistic approach in optimising the edge-fog-cloud architecture's efficiency.

It is important to note that the relative computational power indicators k_{e_i}, k_{f_j}, and k_{c_l} are assumed to be constant for the purposes of this theoretical analysis. This simplification is intended to provide a tractable baseline for deriving upper and lower bounds on speedup. In practice, these indicators may vary dynamically due to fluctuations in network traffic (e.g., WLAN, LAN, and internet) and the computational load on worker devices, especially edge devices with constrained resources. These variations can influence task completion times and, consequently, the speedup achieved in real-world scenarios.

4.1.1. Best-Case Scenario

Deriving from [6], only considering the P2P edge workers, assuming jobs of equal computational complexity and ignoring overheads, the upper bound for speedup can be given as:

$$S_{\text{upper}} = 1 + \sum_{i=2}^{x} \left(\frac{1}{k_{e_i}} \right) \tag{4}$$

This formula illustrates the ideal scenario where the combined effort of all devices maximises the task processing speed. Realistically, it should be noted that given the presence of overheads and the variability in job sizes, the realised speedups are likely to fall below this theoretical maximum.

Incorporating the fog and cloud workers, the updated formula for the upper bound for speedup becomes:

$$S_{\text{upper}} = 1 + \sum_{i=2}^{x} \left(\frac{1}{k_{e_i}} \right) + \sum_{j=1}^{y} \left(\frac{1}{k_{f_j}} \right) + \sum_{l=1}^{z} \left(\frac{1}{k_{c_l}} \right) \tag{5}$$

4.1.2. Worst-Case Scenario

The lower bound for speedup is formulated, extending from [6], for edge, fog, and cloud workers as follows.

We consider the worst-case scenario where the collective capability of the worker devices across the edge, fog, and cloud tiers is significantly less than that of the delegator. This scenario might occur due to high network latency, worker unavailability, or extremely weak workers.

In such situations, the delegator undertakes the parallelised task execution without the assistance of any worker nodes, incurring overheads, including costs related to task parallelisation and ongoing worker search efforts, as well as communication costs across the edge, fog, and cloud layers.

The utmost job completion time, t_{worst}, is represented as:

$$t_{\text{worst}} = t_M + c_{\text{edge}} + c_{\text{fog}} + c_{\text{cloud}} + e_{\text{edge}} + e_{\text{fog}} + e_{\text{cloud}}$$

Under these assumptions, the lower bound for speedup can be given as:

$$S_{\text{lower}} > \frac{t_M}{t_M + c_{\text{edge}} + c_{\text{fog}} + c_{\text{cloud}} + e_{\text{edge}} + e_{\text{fog}} + e_{\text{cloud}}} \tag{6}$$

This formulation reflects the minimal speedup achievable under adverse conditions.

5. Experimental Evaluation

An image processing task (face detection) was chosen as the computational task to be shared amongst the nodes. The job pool on the delegator consists of 1000 unique PNG images at 1024 × 1024 resolution, obtained from the FFHQ dataset (https://github.com/NVlabs/ffhq-dataset, accessed on 4 March 2024). We selected images starting from

filenames 66000.png to 66999.png. The mean image size was 1.3 MB, with a median of 1.31 MB and a standard deviation of 0.161. The images were stored in the delegator device at the start of each experiment. The nodes used for the experiment are given in Table 1. These devices represent a range of computational capacities across the edge, fog, and cloud layers. The Moto G5S Plus (D1) represents a constrained edge device typical in IoT setups, while the OnePlus 6 (D2) demonstrates a more powerful edge environment. The Dell Inspiron 5502 (F1) serves as a fog server, representing resource-rich devices for intermediate computation and the AWS EC2 t3.xlarge instance (C1) exemplifies a high-performance cloud resource for centralised processing. These choices demonstrate the applicability of our approach to devices across a wide spectrum of computational capacities, from constrained edge devices to powerful cloud resources.

All results were obtained from experiments conducted on the specified physical devices, ensuring a realistic evaluation of the system's performance. No simulations were used. The aims of each experiment and related RQs are given in Table 2. Each experiment was conducted five times, and the results were averaged to ensure statistical reliability. The network capacity and latency details for the three different edge-fog-cloud tier networks in the experiments are given in Table 3.

Table 1. Specifications of the delegator and worker nodes.

Node	CPU	RAM	OS
Moto G5S Plus (D1)	Qualcomm MSM 8953 Snapdragon 625	4 GB	Android 8.1
Oneplus6 (D2)	Qualcomm SDM845 Snapdragon 845	8 GB	Android 11
Dell Inspiron 5502 (F1)	11th Gen Intel(R) Core(TM) i7-1165G7 @2.80GHz 1.69GHz	16 GB	Microsoft Windows 10 Pro (x64)
AWS EC2 instance t3.xlarge (Cx where $x \in [1..12]$)	4vCPU upto 3.1 GHz Intel Xeon Platinum Processor	16 GB	Ubuntu Server 20.04 LTS

Table 2. Experiment overview.

Experiment	Aim	RQ
1,2,3,4,5,6,9	Compare the performance of the system when there are multiple configurations of edge, fog, cloud nodes, with heterogeneous platforms, capacities, connection protocols, working together	RQ1
7	Investigate the impact of task sharing configuration parameters in an edge, fog, cloud collaboration setup	RQ2
8, 10	Investigate the impact of dynamically adding/removing or slowing down worker nodes	RQ3

Table 3. Network capacity and latency details for different network types.

Network Type	Latency Range (ms)	Bandwidth (Mbps)
P2P Edge (Wi-Fi Direct-based)	5–10	50–100
Fog Server (LAN-based)	5–30	100–150
Cloud Server (Internet-based)	200–400	30–50

5.1. Experiment 1: Baselines

This experiment examines the computational capacities of each worker node in comparison to the delegator. This sets the baseline values to understand the performance gains and battery usage for the later experiments. The delegator first executes the entire task monolithically (the 'monolithic version' refers to the task without any of the parallelising components). Next, the delegator offloads the entire task to each worker, and each node processes the entire task sequentially. Two sets of baseline experiments are carried out:

(1) D1 as delegator, with D2, F1, and C1 as workers, and (2) D2 as delegator, with D1, F1, and C1 as workers. The speedup metric, denoted as S, provides the relative performance gain achieved through offloading compared to monolithic execution. Table 4 shows the results of this study. Note that Avg. Tr. time denotes the average transmission time per job in the table. This includes all associated communication delays, such as the propagation time, any network routing or switching delays, and job preparation and acknowledgment processing at both ends. Battery % indicates the battery usage of the delegator during the execution of the experiment. This was measured by recording the battery percentage of each device just before the experiment started and immediately after it ended.

Table 4. Experiment 1: Baseline experimental results for each worker with D1 and D2 as delegator.

Node	D1 as Delegator				D2 as Delegator			
	S	**Battery (%)**	**Avg Tr. Time (ms)**	**Avg Total Time (ms)**	**S**	**Battery (%)**	**Avg Tr. Time (ms)**	**Avg Total Time (ms)**
Delegator	N/A	17.60	N/A	3,900,012	N/A	8.99	N/A	1,042,548
Fog	2.70	4.00	797	1,446,771	0.85	5.60	627	1,043,236
Cloud	1.43	8.39	1948	2,725,512	0.56	7.39	1286	1,674,836
Edge	2.57	5.20	184	1,515,956	0.17	23.60	2002	6,089,051

With D1 as delegator, the results indicate that D2, F1, and C1 exhibit speedup factors of approximately 2.57, 2.70, and 1.43 over D1, respectively, Note that this comparative performance takes transmission delays into account. This is why even though the fog (F1) and cloud (C1) nodes may be computationally more resource rich than the P2P edge node (D2), the overall performance in D1's perspective is less than D2. As can be seen from the battery usage on D1, offloading significantly reduces D1's battery consumption, even with the additional communication costs.

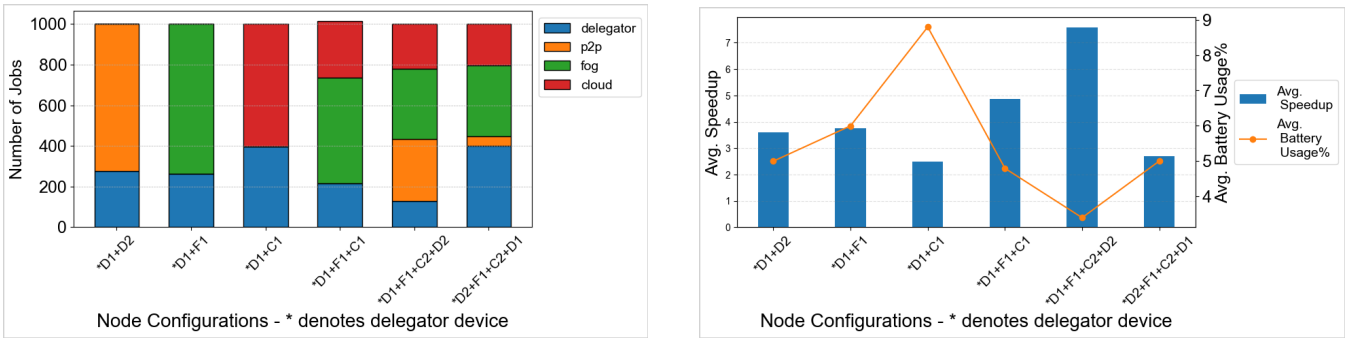
With D2 as delegator, F1's speedup (0.85) is only slightly lower than D2's monolithic execution, but C1 demonstrates a marked decrease in speedup with S = 0.56. Despite C1 having significant computational power, due to latency issues that are well known with cloud computing, jobs offloaded to C1 take more time to return results. Finally, the P2P edge node (D1) is the slowest, with S = 0.17. This is due to D1 being significantly less powerful than D2 (see Table 1).

5.2. Experiment 2: Delegator and P2P (Edge)

In this experiment, the delegator D1 collaborates with a peer-to-peer edge device D2. Comparative analysis indicates that D2 operates at a rate 2.57 times faster than D1, as detailed in Section 5.1. D2's superior computational capacity compared to D1 is further demonstrated by D2 executing approximately 2.6 times as many jobs as D1. This performance discrepancy is clearly depicted in Figure 4a. Hence, this configuration achieves a speedup of 3.59, as documented in Figure 4b. The average task completion time for this experiment was 1,084,998 ms.

5.3. Experiment 3: Delegator and Fog

In this experiment, D1 cooperates with F1 to complete the jobs together. This results in a speedup of 3.76, as seen in the Figure 4b. This is also due to the fact that F1 is more powerful than D1, as evidenced by the number of jobs completed by each node, as seen in Figure 4a. The average task completion time for this experiment was 1,038,146 ms.



(**a**) (**b**)

Figure 4. Results for Experiments 2–6 and 9. (**a**) Jobs completed by each node for Experiments 2–6,9. (**b**) Speedup gains and battery usage for varying node configurations.

5.4. Experiment 4: Delegator and Cloud

Here, D1 shares the workload with C1, resulting in a speedup of 2.48, as seen in Figure 4b. Node C1 is more powerful than the delegator D1, as evidenced by the number of jobs completed by each node in Figure 4a. The average task completion time for this experiment was 1,571,844 ms.

5.5. Experiment 5: Delegator, Fog, and Cloud

In this experiment, the delegator (D1) is collaborating with both F1 and C1. As can be seen from from Figure 4b, this results in a speedup of 4.85. The average task completion time for this experiment was 803,566 ms.

5.6. Experiment 6: Delegator, P2P, Fog, and Cloud

In this experiment, a comprehensive 3-tier configuration, comprising a P2P edge device (D2), a fog server (F1), a cloud server (C1), and the delegator (D1), collaboratively processes the tasks. This integrated architecture achieves a significant speedup of approximately 7.5, as documented in Figure 4b. The average task completion time for this experiment was 514,485 ms. D2 and F1, being the fastest workers, are able to complete almost two-thirds of the total jobs (Figure 4a). Figure 5 illustrates the cumulative number of jobs completed (y-axis) over time in milliseconds (x-axis) for each node (D1, D2, F1, C1). The lines represent the total number of jobs completed by each node at any given point in time. The gap between the lines provides an indication of how many jobs each device has completed relative to the others, with a larger gap signifying a higher contribution.

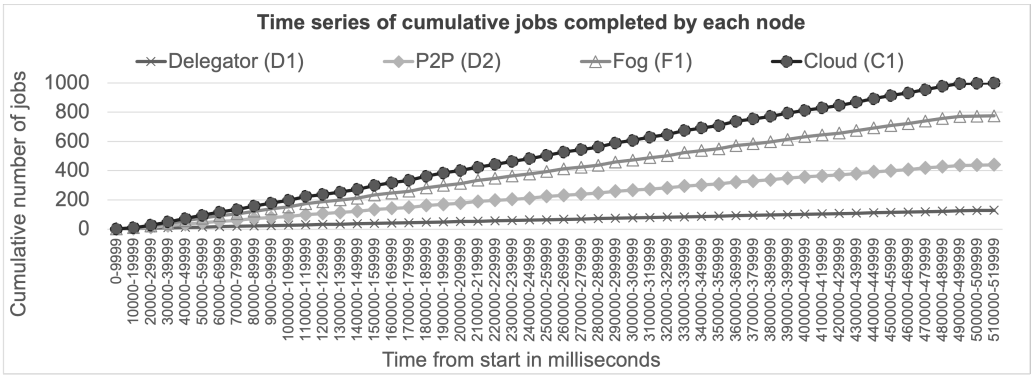
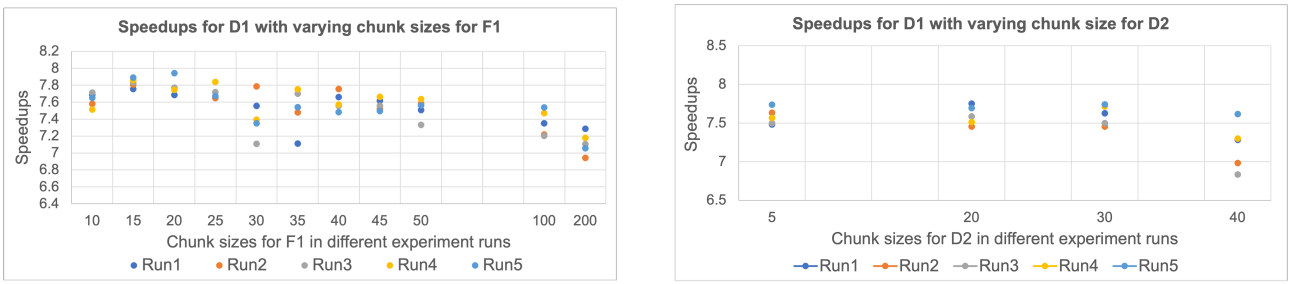


Figure 5. Time series of number of jobs completed by each node. Experiment 6: Results for D1 working with D2, F1, and C1: Time series of number of jobs completed by each node.

5.7. Experiment 7: Varying the Chunk Size

Chunk size, defined as the number of jobs dispatched per steal request to a worker, is a critical parameter within the Honeybee framework for optimising performance. An excessively large chunk size results in workers incurring substantial wait times for job reception rather than execution, while an overly small chunk size leads to too many steal requests, thereby increasing overheads.

The effect of chunk size is notably pronounced when job data are sizeable. In this paper, the job data are images, with an average size of 1.3 MB. The aim of this experiment is to calibrate chunk size to enhance performance. Initially, the experiment maintains a fixed chunk size for D2 and C1 at 5, iterating over various chunk sizes for F1 (see Figure 6b), which was identified as the most efficient worker in preceding experiments. Subsequently, the experiment is replicated with D2 (see Figure 6a) given its comparable efficacy to F1 in job completion.



(**a**) (**b**)

Figure 6. Experiment 7: Varying the chunk size. (**a**) Speedup gains for the D1 with varying chunk size for F1 and constant chunk size for nodes D2 and C1. (**b**) Speedup gain for the D1 with varying chunk size for D2 and constant chunk size for nodes F1 and C1.

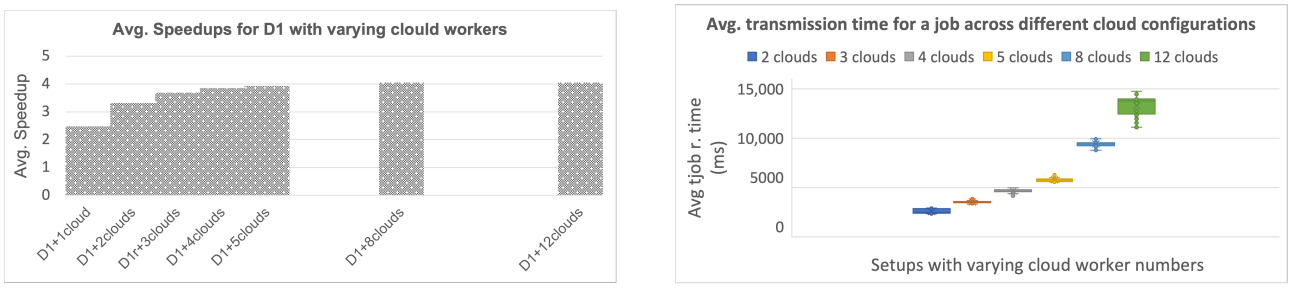
Increasing the chunk size beyond Honeybee's default of 5 yields performance gains for F1, with chunk sizes of 15 and 20 emerging as optimal for maximising speedups (Figure 6a). Nonetheless, augmenting the chunk size beyond these values appears to diminish returns. Conversely, for worker D2, the default chunk size gives the best speedup (Figure 6b). This phenomenon can be explained by examining the trade-off between communication time and job processing. An increase in chunk size results in proportionately longer job transmission times. Thus, an equilibrium must be struck between the latency of job arrival—a function of both the data volume per job and the chunk size—and the frequency of steal requests initiated by the worker. A diminutive chunk size prompts a higher rate of steal requests, while an excessive chunk size can lead to disproportionate waiting periods for job arrival relative to the job execution time.

This experiment highlights the need to configure chunk sizes for the different requirements for different tiers, as in this case, for fog and P2P edge. Fog servers, with greater processing power and bandwidth, can handle larger chunks effectively, while edge devices perform better with smaller chunks, compensating for their limited speed and capacity.

5.8. Experiment 8: Scalability

The results from previous experiments in this section show that adding more workers leads to increased speedups. However, it can be expected that speedups may eventually stabilise even if more and more nodes are added, due to the overheads of parallelisation and transmission costs. In this experiment, we investigate the scalability of adding more and more workers using instances of similar cloud workers. Starting with one cloud worker, the number of cloud workers is gradually increased till 12 to test the op-

timal number of workers for the highest speedups. Figure 7 illustrates the results of this experiment.



(**a**) (**b**)

Figure 7. Experiment 8: Results of scaling up cloud workers. (**a**) Speedups for delegator D1 with varying number of cloud workers (1 to 12). (**b**) Avg. job transmission time (ms) from delegator D1 to different setups of varying number of cloud workers.

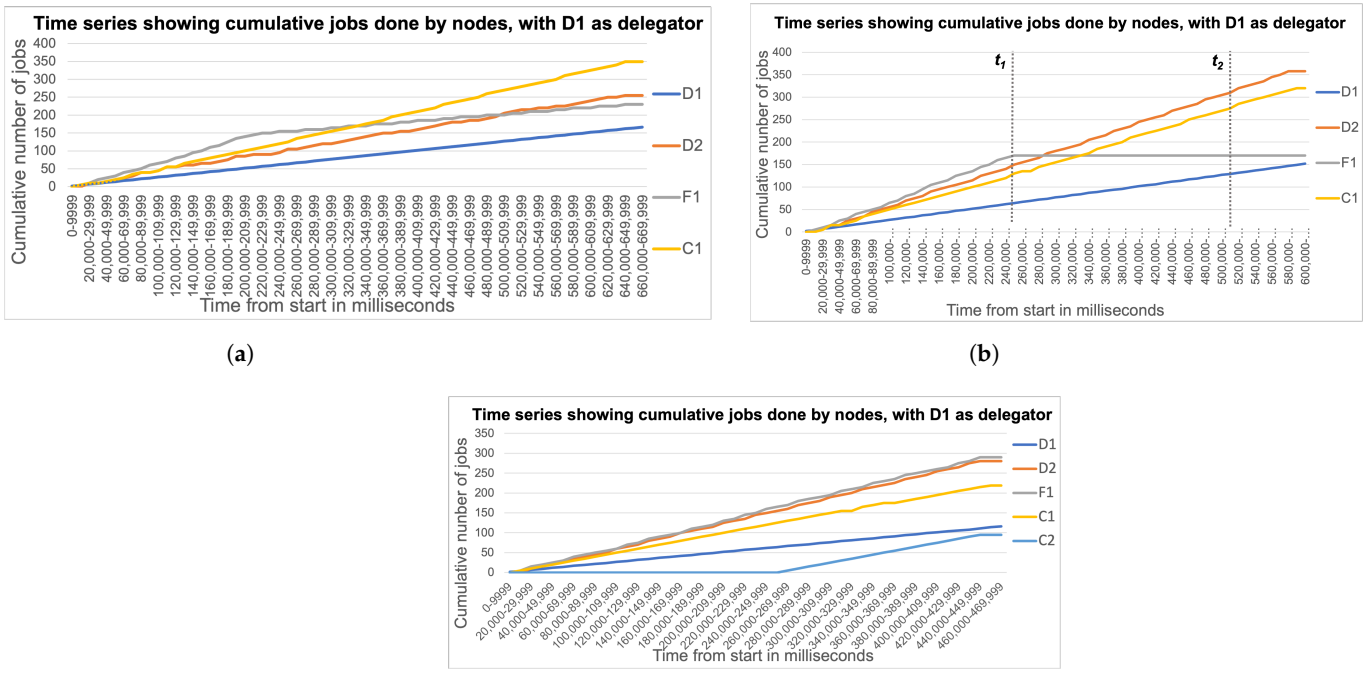
From Figure 7a, it can be seen that the system performance increases with the addition of cloud workers. However, the returns plateau after five cloud servers. From that point onwards, the speedups remain stable even though more cloud workers are added. This can be explained in that having more workers increases the job distribution/transmission overheads, as illustrated in Figure 7b. With an increasing number of cloud workers, the average transmission time for a job increases. As long as this transmission overhead is offset by the benefit of increased computing resources, the system could continue to have increasing speedups. However, it is evident from Figure 7a that the point of diminishing returns for this particular setup occurs at five cloud servers. This result with cloud workers is also consistent with the experimental results for P2P mobile workers investigated in [6], which highlights that regardless of the type of worker (whether P2P edge/fog/cloud), there will be an optimal number of workers for a given job setup.

5.9. Experiment 9: Relative Node Capability

In this experiment, the role of the devices D1 and D2 have been switched, so that D2 is the delegator and D1 is a P2P edge worker. When D2 acts as the delegator, and works with other workers F1, C1, and D1, the jobs can be completed with an average speedup of 2.72, as shown in the node configuration *D2*+F1+C1+D1* in Figure 4b. The average task completion time for this experiment was 382,891 ms. On average, out of 1000 jobs, D2 completed around 400 jobs and is the dominant node, as shown in Figure 4a. It is followed by F1, which completed around 349 jobs. C1 completed 203 jobs and D1 is the weakest of all, completing only 48 jobs.

5.10. Experiment 10: Dynamic Changing of the Node Availability

In real-world scenarios, it can be expected that the node availability would change dynamically. In this set of experiments, the node availability was manually changed to investigate how well this would be handled by the system by examining the rate of job completion (how many jobs were completed by each node at a given time). Figure 8 shows the results of this set of experiments. Note that the jobs are not completely identical (although generally similar), and therefore, the rate of job completion is not a 100% indicator of how much 'work' a node would have done, as the processing of some jobs may be more or less complex than others.



(**a**) (**b**)

(**c**)

Figure 8. Experiment 10: Results for D1 working with D2, F1, and C1 under dynamic conditions. (**a**) Time series of cumulative jobs completed by the nodes: Slowing down F1. (**b**) Time series of cumulative jobs completed by the nodes: Disconnect F1. (**c**) Time series of cumulative jobs completed by the nodes: Add new cloud worker.

5.10.1. Degradation of Network (Pausing Node to Emulate Reduced Availability)

We first investigated the impact of a node's availability being decreased with no prior warning, by programmatically slowing down F1. This was achieved by pausing F1 for 20 s before working on the stolen jobs once it has completed 150 jobs. As can be seen from Figure 8a, the rate of job completion of F1 starts to decrease around 220,000 ms. This is evident in the change of the slope of F1 in Figure 8a. The other nodes' (D1, D2, C1) rate of job completion does not appear to have been significantly increased to compensate for F1's slow down in this instance, although closer inspection reveals a slight uptick of C1's rate of job completion. This may have been because although F1 was slowed down, it was still stealing jobs from the pool. Hence, it may not have given sufficient opportunity for the other nodes to steal more jobs at a higher rate. This experiment was performed five times, and the average speedup obtained was 5.873. This can be compared with the average speedup gained when all the nodes D1, D2, F2, C1 were working with no degradation in Experiment 6, as given in Section 5.6, where the average speedup was 7.582. Although the impact of the F1's decreased performance is evident from the reduced speedup, Honeybee's job scheduling and load balancing methods were still able to handle the unexpected reduction in F1's availability without needing to reconfigure the system parameters.

5.10.2. Degradation of Network (Random Loss of Node)

As the second experiment in this set, we tested Honeybee's fault tolerance mechanisms by programmatically disconnecting F1 halfway through the task. As can be seen from Figure 8b, F1's slope is flat from 250,000 ms onwards. The slopes of D1, D2, F1, and C1's curves at the point of disconnection (at t_1 in Figure 8b) are $2.498, 6.142, 7.03$, and 5.195, respectively. After another 250,000 ms has elapsed, at 510,000 ms (at t_2 in Figure 8b), their slopes are $2.496, 6.333, 0$, and 5.795, where both D2 and C1 have noticeable upticks in their rates of job completion. This can be explained by D2 and C1 stepping up to steal more jobs in the absence of F1. Unlike in the previous experiment shown in Figure 8a, here, the total

absence of F1 provided an opportunity for the other nodes to engage in increased stealing. Although the disconnection here was random, Honeybee's load-balancing mechanisms were still able to (1) ensure any jobs assigned to F1 were added back to the pool so that job execution can complete, and (2) seamlessly load-balance the jobs amongst the remaining nodes automatically without any interventions. This experiment was performed five times, and the average speedup obtained was 6.421, compared to the average speedup of 7.582, when all nodes were performing without any disconnections, as given in Section 5.6. Interestingly, the average speedup that was obtained here is actually higher than that of the 5.873 recorded in the previous experiment in Figure 8a. This suggests that it is more optimal to have a lower number of nodes with high availability than to have more nodes with decreased availability.

5.10.3. Adding a New Node Opportunistically

In the third and last experiment in this set, we investigated Honeybee's ability to incorporate more resource nodes opportunistically halfway through task execution, and whether this can provide benefits to overall performance. As can be seen in Figure 8c, the new cloud worker C2 was added at 260,000 ms. This experiment was performed five times, and the average speedup obtained was 8.32, compared to the average speedup of 7.582, when nodes D1, D2, F1, and C1 were performing without any disconnections, as given in Section 5.6. As can be seen from the job completion curves in Figure 8c, the addition of the new node C2 has not caused any disruption to the performance of the other nodes. From the increased speedup obtained here, it is evident that Honeybee can handle the ad hoc addition of new nodes, and thereby exploit opportunistic node encounters without requiring any reconfiguration or interventions.

6. Summary and Discussion

The speedups and battery savings (for delegator) are summarised in Table 5. As shown in Table 5, the proposed collaborative and proactive method achieves the highest speedups and battery savings when the nodes across edge, fog, and cloud are all working together.

RQ1: The impact of node collaboration across edge, fog, and cloud layers on overall task performance

Experiments 1 to 6, and 9 investigated the impact on performance and battery life of delegating computations to different combinations of P2P edge, fog, and cloud nodes. Experiment 1 demonstrates that transmission (and latency) times must be considered when offloading computations to other hosts. We also noted that with today's mobile devices, fog nodes or edge nodes can sometimes perform better even than cloud nodes (of course, depending on the configuration of the cloud nodes). Experiment 2 showed that two peer devices (D1 and D2) can complete jobs as much as 3.6 times faster, in particular, when a device offloads some computations to a more powerful device. Experiment 3 showed that a device (i.e., D1) offloading to a powerful fog node (F1) can complete jobs as much as 3.76 times faster, in particular, when a device offloads some computations to a more powerful device. Also, though D1 using F1 is faster than D1 using D2, it is noted that a higher battery usage is required when D1 uses F1 than if it uses F1 (due to the transmission power required to a fog node as opposed to a peer node). Experiment 4 showed that a device (i.e., D1) offloading to a cloud node (C1) can complete jobs as much as 2.48 times faster, in particular, when a device offloads some computations to a more powerful device. It is noted that a higher battery usage is required when D1 uses C1 than if it uses F1, as we would expect. Experiment 5 showed that a device (i.e., D1) offloading to a fog node (F1) and cloud node (C1) can achieve a speed up of 4.85. However, because less work is performed by D1, even though it is transmitting jobs to the other devices, its battery usage

decreased. Experiment 6 showed that a device (i.e., D1) offloading to a a peer node (D2), a fog node (F1), and a cloud node (C1) can achieve a speed up of 7.5. However, because even less work is performed by D1, even though it is transmitting jobs to the other devices, its battery usage decreased. Experiment 9 shows that the more powerful (relative to workers) a device is, the less advantage there is in delegating, e.g., it shows that since D2 is more powerful, the speedup from delegation is less than would be gained if D1 was the delegator.

Table 5. Experiment summary.

Exp.	Configuration: D1 as Delegator	Avg Battery Saving %	Avg Speedup
1	D1 monolithic	-	1.00
1	offload to D2	77.27	2.57
1	offload to F1	52.33	2.70
1	offload to C1	70.45	1.43
2	D1 + D2	71.59	3.59
3	D1 + F1	65.91	3.76
4	D1 + C1	50.00	2.48
5	D1 + F1 + C1	72.78	4.85
6	D1 + D2 + F1 + C1	80.68	7.58
7	Varying chunks for D2		
7	Chunk size = (D2 = 5, F1 = 5, C1 = 5)	80.68	7.58
7	Chunk size = (D2 = 20, F1 = 5, C1 = 5)	80.68	7.57
7	Chunk size = (D2 = 30, F1 = 5, C1 = 5)	79.55	7.57
7	Chunk size = (D2 = 40, F1 = 5, C1 = 5)	79.55	7.10
7	Varying chunks for F1		
7	Chunk size = (D2 = 5, F1 = 10, C1 = 5)	77.27	7.62
7	Chunk size = (D2 = 5, F1 = 15, C1 = 5)	75.06	7.81
7	Chunk size = (D2 = 5, F1 = 20, C1 = 5)	75.00	7.74
7	Chunk size = (D2 = 5, F1 = 25, C1 = 5)	79.55	7.50
7	Chunk size = (D2 = 5, F1 = 30, C1 = 5)	76.19	7.46
7	Chunk size = (D2 = 5, F1 = 35, C1 = 5)	77.27	7.51
7	Chunk size = (D2 = 5, F1 = 40, C1 = 5)	80.68	7.64
7	Chunk size = (D2 = 5, F1 = 45, C1 = 5)	79.55	7.59
7	Chunk size = (D2 = 5, F1 = 50, C1 = 5)	78.47	7.51
8	Adding Cloud Workers		
8	D1 + 2 Clouds	58.01	3.33
8	D1 + 3 Clouds	57.44	3.69
8	D1 + 4 Clouds	59.15	3.87
8	D1 + 5 Clouds	67.05	3.93
8	D1 + 8 Clouds	60.23	4.01
8	D1 + 12 Clouds	60.23	4.06
Exp.	Configuration: D2 as delegator	Avg. Battery Saving%	Avg Speedup
9	D2 monolithic	-	1.00
9	offload to D1	−162.51	0.17
9	offload to F1	37.71	0.85
9	offload to C1	17.80	0.56
9	D2 + D1 + F1 + C1	44.38	2.72

RQ2: Impact of task sharing configuration parameters in an edge, fog, cloud collaboration setup

Experiment 7 shows that delegation parameters such as chunk size have a significant impact on performance, increasing speedup by as much as 3–5% (e.g., in a 10 h job, that would be 30 min of savings), and that parameter optimisation should be tailored to the specific requirements of each tier.

RQ3: Dynamically adding/removing or slowing down worker nodes

Experiment 8 shows that there are limits to delegation—employing more cloud workers does not always mean improvements to speedup due to increasing transmission times

and the delegator's limitations in managing multiple workers. The speedup saturates at around 4.1 even with eight cloud nodes or more. There are also diminishing returns so that the gain in speedup with each additional node is less. Experiment 10 demonstrates the ability of the Honeybee framework to compensate and adapt to nodes disconnecting or slowing down across diverse tiers of edge, fog, and cloud—other nodes automatically do more jobs and do so in a way where the additional jobs are distributed fairly uniformly among the remaining nodes so that no particular node suddenly increases in job uptake (the increases in cumulative jobs across different devices over time remain fairly linear). Conversely, the addition of a new worker reduces load or completed jobs on all nodes fairly uniformly.

Overall, for Honeybee-based work stealing style offloading computations, we note that, with the current networking technology, offloading (especially to nearby nodes, i.e., fog or peer nodes) becomes a way to reduce battery consumption on the delegator, i.e., with $D1$ as delegator, we have: $batt(D1, D2, F1, C1) < batt(D1, F1, C1) < batt(D1, D2) < batt(D1, F1) < batt(D1, C1)$, where $batt$ is the battery usage on the delegator. Also, speedups are to be gained with offloading, despite additional transmissions required, and in particular, offloading to more devices helps more, unless the delegator device is relatively much more powerful than the device(s) being offloaded to, and despite transmissions to fog or cloud nodes, the more powerful the device being offloaded to, the greater the speedups attainable: with D1 as delegator, we have: $su(D1)[1] < su(F1)[2.37] < su(D1, C1)[2.48] < su(D2)[2.53] < su(D1, D2)[3.6] < su(D1, F1)[3.76] < su(D1, F1, C1)[4.85] < su(D1, D2, F1, C1)[7.5]$, where su denotes the speed up with the given configuration (also given in square brackets), and with the more powerful D2 as delegator, the speedup attained is no more than 2.8. While speedups are to be gained via offloading, there are limits to how much gain can be achieved and the advantage of using more workers, and there are diminishing returns with more nodes being used, with clear implications. For estimations of costs (with paying for more cloud nodes) versus benefits (from improved speedups), a mechanism to determine the optimal number of nodes (minimising the cost/benefit ratio) is required. It might not be necessary to scale a system beyond some threshold number of devices (in our experiments, less than half a dozen other worker nodes are adequate). Since delegation or offloading requires resources of the delegator, a multilevel delegation/offloading architecture might be required, e.g., a hierarchical or graph-based delegation where workers themselves delegate to others, recursively, would be needed to obtain further speedups when a large number of devices are actually available.

There is a need to tune parameters to obtain good performance relative to the resources/costs employed, e.g., chunk size, number of workers, and there is a need to compensate for disconnections, slow downs or node failures, as well as addition of new nodes, which our Honeybee algorithm does successfully.

7. Conclusions and Future Work

In this work, we evaluated the proposed system of collaborative edge-fog-cloud architecture exclusively on a real testbed with actual devices, ensuring practical applicability and eliminating reliance on simulations. Our experiments suggest that dynamic cooperative computations involving edge, fog, and cloud nodes are advantageous and should be facilitated as a "normal" device function. Honeybee has provided a means to enable such cooperative computations in a seamless way that is dynamically responsive to the availability and changing capacities of devices at run-time. As computers in our pockets and surroundings become more powerful and increase in number, the problem is, and increasingly so, not the lack of computational capabilities, but an increase in untapped

idle resources (e.g., the case of idle resources on desktops [42] and cloud nodes [43] easily extends to smartphones and other devices).

This paper has experimented with a range of configurations involving edge, fog, and cloud nodes, but there are many more to explore. The optimal configurations for computations are not easily arrived at analytically, but best effort, heuristic-based, functional validation approaches are probably more practical, e.g., a rule of thumb might be "offload only when surrounded by more capable devices which are not too far away", where the capabilities of devices can only be detected by performing some jobs for a short monitored period of time (as opposed to requiring explicit sharing of device information due to privacy reasons). Moreover, our theoretical analysis of speedup bounds currently assumes constant relative computational power indicators k_{e_i}, k_{f_j}, and k_{c_l}. However, in real-world scenarios, these values vary due to fluctuating network conditions and computational loads. Extending our model with probabilistic or time series-based approaches could capture these dynamics, enabling adaptive task offloading and configuration optimisation, and enhancing the framework's robustness in variable environments. While our experiments used D1 as a representative of modestly resourced edge devices, future work will involve testing the system on smaller and more constrained IoT platforms. This will further validate the scalability and adaptability of our approach across diverse edge environments. Hence, much further work remains.

Author Contributions: Conceptualisation, N.F.; methodology, N.F., S.W.L. and K.L.; software, N.F. and S.S.; validation, N.F. and S.S.; investigation, N.F., S.S., S.W.L. and K.L.; data curation, N.F. and S.S.; writing—original draft preparation, N.F., S.S., S.W.L. and K.L.; writing—review and editing, N.F., S.S., S.W.L. and K.L.; visualisation, N.F. and S.S.; supervision, N.F., S.W.L. and K.L.; project administration, N.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wang, S.; Zhang, X.; Zhang, Y.; Wang, L.; Yang, J.; Wang, W. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* **2017**, *5*, 6757–6779. [CrossRef]
2. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
3. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
4. Cruz, P.; Achir, N.; Viana, A.C. On the Edge of the Deployment: A Survey on Multi-Access Edge Computing. *ACM Comput. Surv. (CSUR)* **2022**, *55*, 1–34. [CrossRef]
5. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
6. Fernando, N.; Loke, S.W.; Rahayu, W. Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds. *IEEE Trans. Cloud Comput.* **2016**, *7*, 329–343. [CrossRef]
7. Ortiz, G.; Zouai, M.; Kazar, O.; Garcia-de Prado, A.; Boubeta-Puig, J. Atmosphere: Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing. *Comput. Stand. Interfaces* **2022**, *79*, 103550. [CrossRef]
8. Stojmenovic, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In Proceedings of the 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC), Melbourne, Australia, 26–28 November 2014; pp. 117–122.
9. Alharbi, H.A.; Aldossary, M. Energy-efficient edge-fog-cloud architecture for IoT-based smart agriculture environment. *IEEE Access* **2021**, *9*, 110480–110492. [CrossRef]
10. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.

11. Fahim, A.; Mtibaa, A.; Harras, K.A. Making the case for computational offloading in mobile device clouds. In Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, Miami, FL, USA, 30 September–4 October 2013; pp. 203–205.
12. Drolia, U.; Martins, R.; Tan, J.; Chheda, A.; Sanghavi, M.; Gandhi, R.; Narasimhan, P. The case for mobile edge-clouds. In Proceedings of the 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing, Sorrento Peninsula, Italy, 18–21 December 2013; pp. 209–215.
13. Bellavista, P.; Chessa, S.; Foschini, L.; Gioia, L.; Girolami, M. Human-enabled edge computing: Exploiting the crowd as a dynamic extension of mobile edge computing. *IEEE Commun. Mag.* **2018**, *56*, 145–155. [CrossRef]
14. Zhang, W.; Flores, H.; Pan, H. Towards collaborative multi-device computing. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Athens, Greece, 19–23 March 2018; pp. 22–27.
15. Wang, B.; Wang, C.; Huang, W.; Song, Y.; Qin, X. A survey and taxonomy on task offloading for edge-cloud computing. *IEEE Access* **2020**, *8*, 186080–186101. [CrossRef]
16. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
17. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
18. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
19. He, Y.; Ren, J.; Yu, G.; Cai, Y. D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 1750–1763. [CrossRef]
20. Flores, H.; Sharma, R.; Ferreira, D.; Kostakos, V.; Manner, J.; Tarkoma, S.; Hui, P.; Li, Y. Social-aware hybrid mobile offloading. *Pervasive Mob. Comput.* **2017**, *36*, 25–43. [CrossRef]
21. Ranji, R.; Mansoor, A.M.; Sani, A.A. EEDOS: An energy-efficient and delay-aware offloading scheme based on device to device collaboration in mobile edge computing. *Telecommun. Syst.* **2020**, *73*, 171–182. [CrossRef]
22. Chen, X.; Zhang, J. When D2D meets cloud: Hybrid mobile task offloadings in fog computing. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
23. Doolan, D.C.; Tabirca, S.; Yang, L.T. MMPI a message passing interface for the mobile environment. In Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia, Linz, Austria, 24–26 November 2008; pp. 317–321.
24. Marinelli, E. Hyrax: Cloud Computing on Mobile Devices Using MapReduce. Master's Thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.
25. Miluzzo, E.; Cáceres, R.; Chen, Y.F. Vision: MClouds-computing on clouds of mobile devices. In Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services, Low Wood Bay, UK, 25 June 2012; pp. 9–14.
26. Hasan, R.; Hossain, M.; Khan, R. Aura: An incentive-driven ad-hoc IoT cloud framework for proximal mobile computation offloading. *Future Gener. Comput. Syst.* **2018**, *86*, 821–835. [CrossRef]
27. Firouzi, F.; Farahani, B.; Marinšek, A. The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT). *Inf. Syst.* **2022**, *107*, 101840. [CrossRef]
28. Abbasi, M.; Mohammadi-Pasand, E.; Khosravi, M.R. Intelligent workload allocation in IoT–Fog–cloud architecture towards mobile edge computing. *Comput. Commun.* **2021**, *169*, 71–80. [CrossRef]
29. Bisht, J.; Vampugani, V.S. Load and cost-aware min-min workflow scheduling algorithm for heterogeneous resources in fog, cloud, and edge scenarios. *Int. J. Cloud Appl. Comput. (IJCAC)* **2022**, *12*, 1–20. [CrossRef]
30. Gupta, S.; Lozano, A. Computation-bandwidth trading for mobile edge computing. In Proceedings of the 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 11–14 January 2019; pp. 1–6.
31. Huang, M.; Liu, W.; Wang, T.; Liu, A.; Zhang, S. A cloud–MEC collaborative task offloading scheme with service orchestration. *IEEE Internet Things J.* **2019**, *7*, 5792–5805. [CrossRef]
32. Diao, X.; Zheng, J.; Wu, Y.; Cai, Y. Joint computing resource, power, and channel allocations for D2D-assisted and NOMA-based mobile edge computing. *IEEE Access* **2019**, *7*, 9243–9257. [CrossRef]
33. Du, J.; Zhao, L.; Feng, J.; Chu, X. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Trans. Commun.* **2017**, *66*, 1594–1608. [CrossRef]
34. Zahoor, S.; Javaid, N.; Khan, A.; Ruqia, B.; Muhammad, F.J.; Zahid, M. A cloud-fog-based smart grid model for efficient resource utilization. In Proceedings of the 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 1154–1160.
35. Kumar, M.S.; Karri, G.R. Eeoa: Cost and energy efficient task scheduling in a cloud-fog framework. *Sensors* **2023**, *23*, 2445. [CrossRef] [PubMed]

36. Alwateer, M.; Loke, S.W.; Fernando, N. Enabling drone services: Drone crowdsourcing and drone scripting. *IEEE Access* **2019**, *7*, 110035–110049. [CrossRef]
37. Loke, S.W.; Abkenar, A.B.; Fernando, N. Towards an Edge-Cloud Platform for Multirobot-Multihuman Cooperation in Urban IoT Ecosystems. In Proceedings of the WiP IEEE International Conference on Pervasive Computing and Communications (PerCom), Austin, TX, USA, 23–27 March 2020.
38. Nagesh, S.S.; Fernando, N.; Loke, S.W.; Neiat, A.; Pathirana, P.N. Opportunistic mobile crowd computing: Task-dependency based work-stealing. In Proceedings of the 28th Annual International Conference on Mobile Computing And Networking, Sydney, NSW, Australia, 17–21 October 2022; pp. 775–777.
39. Fernando, N.; Loke, S.W.; Rahayu, W. Mobile crowd computing with work stealing. In Proceedings of the 2012 15th International Conference on Network-Based Information Systems, Melbourne, Australia, 26–28 September 2012; pp. 660–665.
40. Blumofe, R.D.; Leiserson, C.E. Scheduling multithreaded computations by work stealing. *J. ACM (JACM)* **1999**, *46*, 720–748. [CrossRef]
41. Fernando, N.; Arora, C.; Loke, S.W.; Alam, L.; La Macchia, S.; Graesser, H. Towards human-centred crowd computing: Software for better use of computational resources. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), Melbourne, Australia, 14–20 May 2023; pp. 90–94.
42. Rosales, E.; Sotelo, G.; de la Vega, A.; Díaz, C.O.; Gómez, C.E.; Castro, H. Harvesting Idle CPU Resources for Desktop Grid Computing While Limiting the Slowdown Generated to End-Users. *Clust. Comput.* **2015**, *18*, 1331–1350. [CrossRef]
43. Zhang, B.; Dhuraibi, Y.A.; Rouvoy, R.; Paraiso, F.; Seinturier, L. CloudGC: Recycling Idle Virtual Machines in the Cloud. In Proceedings of the 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, Canada, 4–7 April 2017; pp. 105–115. [CrossRef]

future internet

MDPI

Article

Analysis and Evaluation of Intel Software Guard Extension-Based Trusted Execution Environment Usage in Edge Intelligence and Internet of Things Scenarios

Zhiyuan Wang [1] and Yuezhi Zhou [1,2]*

[1] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; wzycs24@mails.tsinghua.edu.cn
[2] Zhongguancun Laboratory, Beijing, China
* Correspondence: zhouyz@mail.tsinghua.edu.cn

Abstract: With the extensive deployment and application of the Internet of Things (IoT), 5G and 6G technologies and edge intelligence, the volume of data generated by IoT and the number of intelligence applications derived from these data are rapidly growing. However, the absence of effective mechanisms to safeguard the vast data generated by IoT, along with the security and privacy of edge intelligence applications, hinders their further development and adoption. In recent years, Trusted Execution Environment (TEE) has emerged as a promising technology for securing cloud data storage and cloud processing, demonstrating significant potential for ensuring data and application confidentiality in more scenarios. Nevertheless, applying TEE technology to enhance security in IoT and edge intelligence scenarios still presents several challenges. This paper investigates the technical challenges faced by current TEE solutions, such as performance overhead and I/O security issues, in the context of the resource constraints and data mobility that are inherent to IoT and edge intelligence applications. Using Intel Software Guard Extensions (SGX) technology as a case study, this paper validates these challenges through extensive experiments. The results provide critical assessments and analyses essential for advancing the development and usage of TEE in IoT and edge intelligence scenarios.

Keywords: Trusted Execution Environment (TEE); Software Guard Extensions (SGX); Internet of Things (IoT); edge intelligence; performance evaluation; security

Citation: Wang, Z.; Zhou, Y. Analysis and Evaluation of Intel Software Guard Extension-Based Trusted Execution Environment Usage in Edge Intelligence and Internet of Things Scenarios. *Future Internet* **2025**, *17*, 32. https://doi.org/10.3390/fi17010032

Academic Editor: Gianluigi Ferrari

Received: 11 November 2024
Revised: 25 December 2024
Accepted: 6 January 2025
Published: 13 January 2025

Citation: Wang, Z.; Zhou, Y. Analysis and Evaluation of Intel Software Guard Extension-Based Trusted Execution Environment Usage in Edge Intelligence and Internet of Things Scenarios. *Future Internet* **2025**, *17*, 32. https://doi.org/10.3390/fi17010032

1. Introduction

The Internet of Things (IoT) has experienced rapid development in recent years, transforming industries to Internet-based paradigms by enabling the connection of millions of devices [1]. Owing to advancements in 5G and the upcoming 6G wireless communication technologies, data transmission has become significantly faster and more reliable, laying a solid foundation for IoT and edge intelligence applications [2]. By 2030, it is projected that approximately 500 billion IoT devices will be in use globally [3]. IoT data generation is expected to grow significantly, with estimates suggesting a rise to 73.1 ZB by 2025, a 422% increase from the 17.3 ZB produced in 2019 [4]. IoT devices are increasingly used in smart homes [5], healthcare [6,7], manufacturing [8,9], and transportation [10,11], where they collect vast amounts of data. These data are essential for deriving insights and powering edge intelligent applications through advanced analytics and machine learning [12], driving the convergence of IoT, edge devices, and cloud into a seamless computing continuum [13], which serves as a cornerstone of modern digital ecosystems.

With the expansion of IoT and edge intelligence computing, ensuring the security and privacy of the massive amount of data generated and processed by these devices has become a significant challenge [14]. Trusted Execution Environment (TEE) provides a secure, isolated space within a processor where sensitive data and code can be processed, safeguarding them from unauthorized access or tampering [15]. TEE is crucial for protecting the growing volume of sensitive data generated by billions of interconnected devices in IoT. These devices often operate in untrusted environments and are vulnerable to security breaches, making TEE essential for ensuring data integrity and privacy [16]. Various TEE technologies enhance data security, including AMD Secure Encrypted Virtualization (SEV) [17] for virtual machine encryption, Arm TrustZone [18] for embedded devices, and Penglai Enclave [19] on RISC-V platforms. Intel SGX [20], in contrast, focuses on application-level security through enclaves, making it ideal for protecting sensitive data in cloud and edge intelligence applications [21] and is widely used in IoT [22,23].

However, as edge intelligence becomes an essential component of IoT architectures, the usage of TEE in these scenarios faces several challenges. The resource constraints of edge devices, such as limited processing power and memory, coupled with the high-performance demands of real-time applications, make it difficult for TEE technologies to operate efficiently [24]. Additionally, ensuring data integrity [25,26], managing I/O operations securely [27], and handling the performance overhead from frequent enclave transitions [28] present significant hurdles, limiting the widespread deployment of TEE-based solutions in edge intelligence.

In this paper, we analyze the performance and security challenges of using SGX-based TEE in IoT and edge intelligence scenarios. Through a series of experiments, we evaluate the impact of TEE on system performance, focusing on key aspects such as I/O operations and data handling in resource-constrained environments. We concentrate on the performance of different SGX implementations in real-world scenarios, conducting comprehensive and detailed evaluations on SGX-based TEE through various carefully designed test cases. Our results reveal significant performance degradation and security limitations when using TEE technologies in edge scenarios. This research provides critical technical evaluations and security analyses that offer valuable insights for advancing the development and application of TEE in IoT and edge intelligence scenarios. The contributions of this paper could be summarized as follows:

1. A comprehensive survey of current TEE technologies in IoT scenarios is conducted, identifying key challenges such as I/O security and performance overhead.
2. A detailed testing framework is designed to evaluate the performance of SGX-based TEE in edge environments, covering implementations based on SGX Software Development Kit (SDK) and Library Operating System (LibOS), as well as their usage in a virtual environment.
3. Based on the survey and experimental results, performance bottlenecks in SGX are identified, offering insights and optimization directions for the application of TEE technologies in edge intelligence scenarios.

The rest of the paper is organized as follows. In Section 2, we provide a brief introduction to the fundamental concepts of SGX-based TEE, focusing on two key features: isolated execution and attestation. Section 3 discusses the application of TEE in IoT and the challenges it faces in edge intelligence scenarios, such as I/O security and performance degradation. Section 4 presents a detailed performance evaluation of SGX in edge scenarios and proposes performance improvement directions based on the test results. Section 5 covers a review of related research and Section 6 concludes the paper.

2. SGX-Based TEE

This section briefly introduces the basic concepts of SGX, and focuses on the two features of SGX: isolated execution and attestation. In addition, two SGX implementations based on SGX SDK and LibOS will be introduced.

2.1. Basic Concept About SGX

SGX is one of the widely used TEE technologies introduced by Intel Corporation in 2013. Through the combination of software and hardware, a special memory space called an enclave was built to protect users' sensitive data and code [20]. Entering and exiting the enclave requires special hardware instructions and involves complex security checks. SGX focuses on runtime data security, and even privileged software such as the OS or hypervisor cannot access data inside the enclave when the program is running. Isolated execution and attestation [29] are two important characteristics of SGX. Isolated execution is the guarantee that secret data will not be stolen by other software or programs and attestation is a common method for devices to prove their identities to each other in cloud computing scenarios. These two characteristics are the core part of SGX and also the security guarantee of SGX in IoT environments.

2.1.1. Isolated Execution

The isolation of physical memory is the primary condition for isolated execution. A protected contiguous memory space called Enclave Page Cache (EPC) is allocated in the memory and is physically isolated from other parts of the system. The data stored in the EPC are automatically encrypted by the CPU when written to memory and decrypted when read back into the CPU cache. The size of the EPC is limited, at 93 MB in the SGX version 1 [30], which leads to some performance degradation issues [31]. Special hardware instructions, Ecall and Ocall, are required to enter and exit the enclave. Furthermore, SGX records the allocation of each EPC page in the Enclave Page Cache Map (EPCM) and manages the access and modification of these pages. When any operation attempts to access the enclave, the CPU will sequentially check the permissions of the external access process, the validity of the EPC address, and the EPCM to ensure address space isolation.

2.1.2. Attestation

In cloud-based IoT scenarios, users transmit code and data to the cloud for processing. Attestation can ensure that users' sensitive data are not tampered by malicious software in the cloud. There are two kinds of attestation technology in SGX, local attestation and remote attestation. Local attestation is used to confirm the identity between two enclaves on the same platform while remote attestation is used to protect the data integrity on remote servers. The code, data, and some metadata within the enclave are hashed to create a unique abstract, which serves as the enclave's identity for attestation [29].

2.2. SGX Implementation

There are currently two common ways to deploy SGX applications, namely SGX SDK-based and LibOS-based. The SGX SDK-based approach as shown in Figure 1a requires porting or re-building the program in C or C++ based on SGX SDK [32]. In addition, the legacy code needs to be partitioned into secure and insecure portions, which requires a lot of effort. SGX SDK-based implementation effectively reduces the trusted computing base (TCB) size and allows for selective placement of sensitive data in the EPC, thereby optimizing the use of EPC space. However, this approach also has some drawbacks. First, the cost of code porting is high, leading to poor usability. Second, since system calls are not supported within the enclave, frequent context switching is required, resulting

in significant performance overhead. Finally, there is insufficient support for high-level languages, making programming in languages such as Python more difficult.

To overcome these drawbacks, LibOS-based approaches are proposed and has attracted considerable attention. As shown in Figure 1b, by providing library operating system with support within the enclave, interactions between the enclave and the operating system are reduced, enabling the rapid deployment of unmodified applications in the enclave. Although the TCB size increases, the usability is significantly enhanced through a LibOS shim layer. Many solutions have been proposed based on LibOS. For example, Haven [33] pulls most of the application-supporting OS code into the enclave to reduce dependence on the underlying OS, andSCONE [34] integrates SGX with Docker containers, enabling applications to run securely within SGX enclaves without requiring modifications. Among these solutions, Gramine [35] (formerly known as Graphene) and Occlum [36] are widely used LibOS-based SGX implementations that perform well in terms of both performance and security. Gramine is an open-source LibOS that provides a flexible and lightweight environment for running unmodified applications securely within enclaves. Occlum is written in Rust at its core, providing more reliable security guarantees. This paper will focus on these two LibOS-based implementations, analyzing their performance and security.

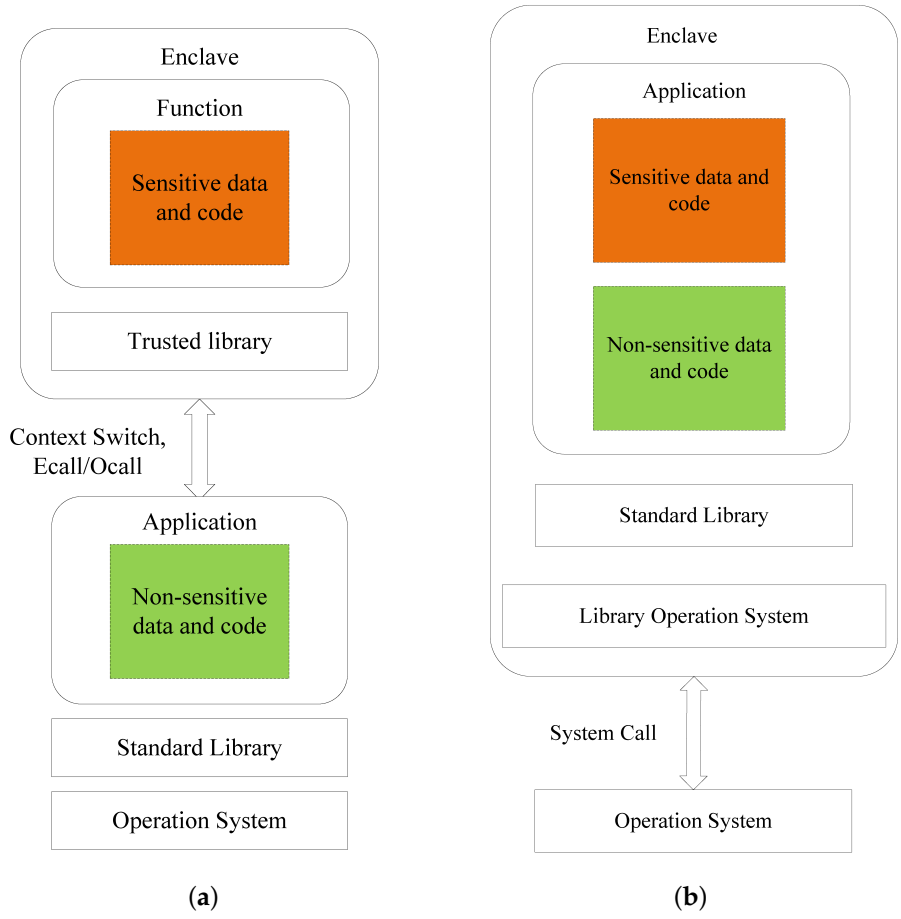


Figure 1. Different SGX implementations. (**a**) SGX SDK-based. (**b**) LibOS-based.

3. Analysis of TEE Usage in Edge Intelligence and IoT

IoT refers to a scenario where physical and digital devices are interconnected through specific protocols and communication methods, forming an extensive network [1]. Information exchange is a crucial component of IoT, involving the protection of privacy data and identity authentication between devices. These concerns align directly with the core

capabilities of TEE. In cloud computing, data are transmitted to the cloud for centralized processing, yet data owners do not fully trust the cloud infrastructure. As a result, TEE technologies such as ARM TrustZone [18], Confidential Compute Architecture (CCA) [37], Penglai Enclave [19] and SGX are crucial for ensuring data protection on remote servers. Next, we will analyze the applications of these different TEE technologies in the IoT and conduct a detailed investigation and analysis of SGX.

3.1. Arm TrustZone and CCA

In 2004, Arm TrustZone was proposed by incorporating hardware security extensions into Arm Cortex-A application processors. Arm TrustZone divides a system's hardware and software resources into two distinct worlds: the Secure World and the Normal World. The Secure World handles sensitive operations, such as cryptographic key management, authentication, and secure storage, while the Normal World runs the standard operating system and applications. The transition between the Secure World and the Normal World is managed through Monitor Mode. TrustZone extends memory security with optional components like the TrustZone Address Space Controller (TZASC) and TrustZone Memory Adapter (TZMA). TZASC manages secure and non-secure memory regions for DRAM, while TZMA handles off-chip ROM or SRAM, enabling secure world access to non-secure areas but not vice versa.

CCA, designed to enhance virtual machine protection, introduces the Realm Management Extension (RME) and adds two new worlds: Realm World and Root World. CCA introduces the Realm World to create secure, isolated environments for confidential VMs, completely separating them from other domains, including the host OS, hypervisor, TrustZone, and other realms. CCA uses a Granule Protection Table, an extension to the page table that tracks memory ownership with worlds to maintain separation. The Granule Protection Table is managed by the Monitor in the Root World, which ensures that the hypervisor or OS cannot alter it directly. This Monitor also controls the dynamic allocation of memory across worlds by updating the Granule Protection Table. Additionally, CCA features attestation mechanisms that validate the platform and ensure the integrity of the realms. The low power consumption of the ARM architecture makes it well suited for IoT scenarios. Owing to its wide deployment in mobile and low-end devices, ARM TrustZone and CCA are widely deployed in IoT and edge intelligence scenarios [38–40].

3.2. Penglai Enclave

Penglai Enclave is a software–hardware co-designed TEE technology aimed at enhancing the security and scalability of security-critical applications in cloud environments, specifically on RISC-V platforms. Penglai introduces two key hardware primitives: the Guarded Page Table and the Mountable Merkle Tree (MMT). The Guarded Page Table enables fine-grained, page-level memory isolation, ensuring that unauthorized software cannot access secure memory. MMT provides scalable memory encryption and integrity protection, supporting large secure memory regions. The secure monitor operates at the highest privilege level (e.g., machine mode in RISC-V), managing enclave creation, enforcing memory isolation and maintaining security guarantees.

Additionally, Penglai employs the Shadow Enclave and Shadow Fork mechanisms, which allow for the fast creation of multiple secure instances, significantly reducing the latency of memory initialization. The system can dynamically scale to thousands of concurrent secure instances, supporting up to 512 GB of encrypted memory with only approximately 5% overhead for memory-intensive applications. This framework effectively addresses the limitations of traditional TEE systems in terms of memory protection and

scalability, making it particularly suitable for large-scale, dynamic and serverless cloud computing scenarios.

3.3. Analysis of SGX in Edge Intelligence and IoT

SGX is widely used in IoT scenarios based on its security guarantees and remote attestation features. Table 1 summarizes some applications of SGX in IoT. The application scenarios can be broadly categorized into four types, with the first focusing on the use of SGX for IoT architecture. The work in [41] describes the entire process from data acquisition at the edge devices, to encrypted transmission, and finally, decryption and processing at the gateway, with SGX responsible for key management tasks. Based on device virtual cloning, SGX-Cloud [42] was constructed to allow users to choose which data to place into the cloud. The methods outlined in [43] utilized SGX's privacy-preserving mechanisms to achieve secure multi-party data sharing. Second, the remote attestation mechanism of SGX can be better utilized to achieve a security-enhanced identity authentication [44]. Third, blockchain is a key component in IoT scenarios, and its integration with SGX can achieve better privacy protection. The work in [45] sets up oracle nodes running in an SGX environment, enabling blockchain to securely obtain external data. Zhang et al. [46] leveraged SGX in industrial IoT to track data flow and prevent data abuse. According to the study in [4], SGX edge servers were set up to implement attribute-based fine-grained access control. Furthermore, SGX can also be applied in fields such as federated learning [47], smart grids [48] and health data protection [49].

Table 1. SGX application in IoT scenarios.

Category	Reference	IoT Application	SGX Use Case
Architecture	Ayoade et al., 2019 [41]	An end-to-end, SGX-based, IoT data protection framework	Key management, Remote attestation
	Showail et al., 2022 [42]	Building SGX-Cloud based on device virtual cloning	Data encryption, Data integrity
	Lei et al., 2021 [43]	Secure multi-party data interaction architecture	Privacy-preserving, Identity authentication
Attestation	Wang et al., 2018 [44]	Policy-based security-enhanced authentication mechanism between remote terminal and IoT devices	Remote attestation
Blockchain-based	Woo et al., 2020 [45]	Methods for securely obtaining external data on the blockchain	Trusted intermediary
	Zhang et al., 2022 [46]	Establishing a secure data trading environment to prevent data abuse	Identity authentication, Behavior monitoring
	Han et al., 2022 [4]	Attribute-based fine-grained access control	Edge intelligence, Identity authentication
Other Scenarios	Kalapaaking et al., 2023 [47]	Comprehensive federated learning framework from data collection to local model generation and secure model aggregation	Secure model aggregation
	Li et al., 2020 [48]	Efficient privacy-preserving architecture in smart grids	Confidential database
	Gao et al., 2021 [49]	Protection of sensitive medical data to build a secure Internet of Medical Things	Device registration

However, SGX usage in edge intelligence still has limitations. In edge intelligence scenarios, where device resources are limited and data mobility are complex, SGX, though capable of providing hardware-level data security, still faces significant challenges. Security and performance analyses are crucial in understanding the effectiveness and practicality

of SGX in these scenarios. On the one hand, data security is a significant concern in edge devices, as data exchanged with peripherals or the cloud are vulnerable to leakage. Additionally, SGX's inherent weakness in defending against side-channel attacks, combined with the closer proximity of attackers to edge devices, increases the likelihood of such attacks succeeding in edge scenarios. On the other hand, deploying SGX on edge devices often leads to performance degradation due to the added computational overhead and frequent context-switching, particularly in resource-constrained environments. Next, we will conduct an in-depth analysis of the three key challenges SGX faces in edge intelligence scenarios: I/O security issues, vulnerability to side-channel attacks, and performance degradation issues.

3.3.1. I/O Security Issues

Firstly, SGX does not include specific protection measures for I/O operations. While memory regions are protected by SGX, I/O operations still interact with external systems through unprotected channels. These channels, such as networks and storage devices, may become targets for attackers, potentially leading to the risk of I/O data being intercepted or alerted on edge devices. To establish a trusted path between the enclave and I/O devices, various solutions have been proposed, as shown in Table 2.

Table 2. Trusted I/O path construction in SGX environment.

Category	Reference	Methodology	Limitations
Hypervisor-based	Weiser et al., 2017 [50]	Based on trusted hypervisor	Large TCB and complex binding and attestation
Software-based	Liang et al., 2020 [51]	Leverages SMM and SMVisor to protect I/O operations	Occupies limited SMRAM
	Thalheim et al., 2021 [52]	Based on SPDK and DPDK for user–space I/O	Limited generality
External HW	Stancu et al., 2019 [53]	USB Dongle (HW) and DPE for attestation	Additional overhead for initialization
	Jang et al., 2024 [54]	Based on UDP (HW) for USB packet forwarding	Substantial overhead caused by remote attestation
Specific Scenarios	Peters et al., 2018 [55]	Modify the protocol stack and use the Bluetooth controller to protect Bluetooth device I/O	Limited generality
	Eskandarian et al., 2019 [56]	Two dongles for keyboard and monitor, enhancing security for browser	Limited screen protection area, Unprotected mouse

There are generally three approaches to construct a trusted I/O path: hypervisor-based, software-based and external hardware (HW)-based solutions. For hypervisor-based methods such as SGXIO [50], although it leverages a virtual machine to supervise the OS, it significantly increases the TCB. Additionally, because SGX does not trust the hypervisor, this approach involves a complex process of trusted domain binding and attestation. Software-based solutions like Aurora [51] protects I/O using the System Management Mode (SMM) but occupies valuable SMRAM resources and lacks distinction between secure and non-secure devices. Rocket-io [52] leverages the Storage Performance Development Kit (SPDK) and Data Plane Development Kit (DPDK) to accelerate communication between the enclave and peripherals. However, it is limited to the disk and network interface card, offering poor generality. The approaches based on external HW encounter challenges such as significant overhead when establishing trusted I/O paths. Ref. [53] designs an external HW called USB Dongle, and implements a Device Provisioning Enclave (DPE) for key exchange and information transmission, but during the initialization of the trusted path, a trusted OS

must first be booted to bind the DPE and the device, resulting in significant performance overhead. Furthermore, A USB proxy device (UPD) is designed in [54] for transmitting USB packets. However, the I/O path setup relies on remote attestation, which incurs significant overhead. Some I/O protection solutions for specific scenarios, such as Bluetooth [55] and browsers [56] are proposed but these also encounter issues such as limited generality and inadequate security protection. This demonstrates that SGX still lacks a universal solution for I/O security and I/O performance degradation issues. This is particularly concerning in edge intelligence scenarios, where attackers can more easily access devices and data sources, leading to increased security risks.

3.3.2. Vulnerability to Side-Channel Attacks

SGX provides hardware-level protection, preventing privileged software, such as the OS and hypervisor, from accessing sensitive data. However, the shared use of system resources between the enclave and untrusted applications significantly expands the attack surface for side-channel vulnerabilities. Currently, side-channel attacks exploiting page tables, branch prediction and cache present significant threats to the security of data within enclaves.

In the case of page tables, the enclave depends on the untrusted OS for management, while using additional data structures for validation. The OS can mark all pages as inaccessible, causing any access to trigger a page fault. By monitoring which pages the enclave accesses and analyzing the sequence of these events over time, the OS can potentially infer certain enclave states and compromise protected data. PigeonHole [57] demonstrates that page fault side-channel attacks can exploit unprotected SGX, leaking an average of 27% and up to 100% of secret bits. Additionally, Ref. [58] focuses on attacks targeting page table flags, such as modifying the "present" bit to track enclave page access, using updates to the "accessed" bit to detect pages accessed by the victim enclave and monitoring the "dirty" bit to infer the victim enclave's memory write operations. Additionally, attacks like the controlled-channel attacks [59] also pose significant challenges to the security of data within the enclave.

Branch prediction is a critical mechanism in modern processors that predicts the outcome of branch instructions before execution. By utilizing branch prediction, processors can preemptively fetch and decode instructions based on the predicted branch direction, enhancing performance. However, during context switching, the Branch Prediction Unit (BPU) is not flushed, potentially retaining sensitive information from isolated environments. This residual data can be exploited by attackers through software-based side-channel attacks, posing significant security risks. By analyzing the branch history information of the victim enclave, attacks such as Branch Shadowing [60] and BranchScope [61] can effectively extract sensitive data or cryptographic keys from within an enclave.

Cache-based side-channel attacks exploit the differing access times between the cache and main memory, stealing sensitive data by measuring the victim's execution time or data access patterns. For example, the Prime + Count cache side-channel attack [62] can establish covert channels across worlds within ARM TrustZone. For SGX, it is particularly susceptible to cache-timing attacks [25]. Experiments have shown that the Prime+Probe cache side-channel attack can extract the AES key from an SGX enclave in less than 10 s. To address the issue of side-channel attacks, several solutions have been proposed. For page table-based side-channel attacks, hardware isolation can provide enclaves with independent page tables [63]. Additionally, cache partitioning [64] prevents attackers and victims from sharing cache lines, while techniques like timing variation elimination [65] can help mitigate cache-based side-channel attacks. Although these methods can effectively defend against specific types of side-channel attacks, there is currently no universal solution to

protect SGX and other TEE technologies from side-channel vulnerabilities. Furthermore, many SGX projects, such as Gramine and Occlum, overlook side-channel attacks in their considerations, making SGX more susceptible to such attacks in practical deployments.

3.3.3. Performance Degradation Issues

In resource-constrained edge scenarios, the issue of SGX's performance degradation has drawn significant attention. SGX's performance degradation primarily arises from two sources, the overhead of data encryption and decryption, and the performance cost associated with entering and exiting the enclave during system calls or EPC page fault [66]. Encryption and decryption operations are performed by hardware, leaving limited room for optimization. In contrast, various approaches have been proposed to optimize the performance overhead caused by context switching. Hotcalls [28] based on a synchronization spin-lock mechanism significantly improve performance compared to Ecalls and Ocalls. The work in [67] achieves exitless system calls by delegating system calls to threads running outside the enclave, and performance overhead due to page faults is mitigated by implementing paging within the enclave. Another more general solution involves leveraging a LibOS, such as Gramine [35] and Occlum [36]. The LibOS approach can streamline interactions between the enclave and the application, potentially reducing the overhead associated with traditional system calls and improving overall performance. Next, we will provide a detailed discussion of the performance overheads of different SGX implementations in edge intelligence scenarios through a qualitative analysis and a quantitative assessment.

4. Performance Evaluation in Edge Scenarios

In this section, the performance evaluation of four different SGX implementations in edge scenarios are conducted, including the SGX SDK-based approach, two LibOS-based approaches (Gramine and Occlum), and the SGX implementation in a virtual environment. CPU-intensive instructions, I/O-intensive instructions, network programming, common system calls and performance under realistic workloads are the five main aspects of testing, covering most edge intelligence scenarios. Through qualitative analysis and quantitative evaluation, we will analyze the performance degradation of SGX in resource-constrained edge scenarios.

4.1. Experimental Setup

The hardware specifications, system parameters, and software versions used for the tests are shown in Table 3. The virtual environment utilizes the docker environment provided by Occlum, version 0.26.4. For both Gramine and Occlum, default values are used for the relevant parameters, and the tests are conducted on the SGX version 1. We disable CPU frequency boost features and fix the CPU frequency at 4.7 GHz to reduce performance data fluctuation. In the network tests, we disconnect external network connections and use local addresses to conduct the tests, minimizing external interference. To avoid the impact of other processes on the experimental results, we disable unnecessary background processes and services.

4.2. CPU-Intensive Workload

The Native-mode Benchmark (NBench) is a common test suite used to assess CPU performance, primarily focusing on the computational speed, floating-point operations and memory system efficiency of a computer. The NBench results are measured by the number of test iterations completed within 1 s. The more iterations are completed, the higher the efficiency. Taking the native Linux test results as the baseline, the performance of SGX implementations are illustrated in Figure 2.

Table 3. System configuration.

Hardware Environment	
CPU: Xeon(R) W-1250 CPU, 4.70 GHz	DRAM: 16 GB
CPUs: 1 Socket, 12 cores	
L1: 384 KB, L2: 1536 KB, L3: 12 MB	
System Settings	
Linux kernel: 6.6.8-060608-generic	GCC: 9.4.0
OS: Ubuntu 20.04	Architecture: x86_64
Intel SGX SDK	
Version: 2.22	PRM:128 MB
Gramine	
Version: 1.6	Enclave size: 256 MB
Max_threads: 4	
Occlum	
Version: 0.26.4	User_Space_Size: 300 MB
Max_threads: 32	

Overall, the implementation based on the SGX SDK has the worst performance, while Gramine performs the best. The performance of Occlum is similar to that of Gramine, and there is almost no difference between the virtual and native environments. The implementation based on the SGX SDK incurs significant context-switching overhead due to frequent enclave transitions, which leads to significant performance overhead. In the FOURIER test, it achieves only about 30% of the native performance. For the two LibOS-based approaches, Gramine performs well across all tests, with a minimum performance of about 80% of native performance. Occlum excels in numerical sorting, bitwise operations and LU decomposition, but in other tests, its performance is approximately 40% to 60% of the native performance. This is because Occlum's design is more oriented towards multi-process environments, allowing multiple processes to coexist within a single enclave, whereas Gramine excels in single-threaded processing capabilities.

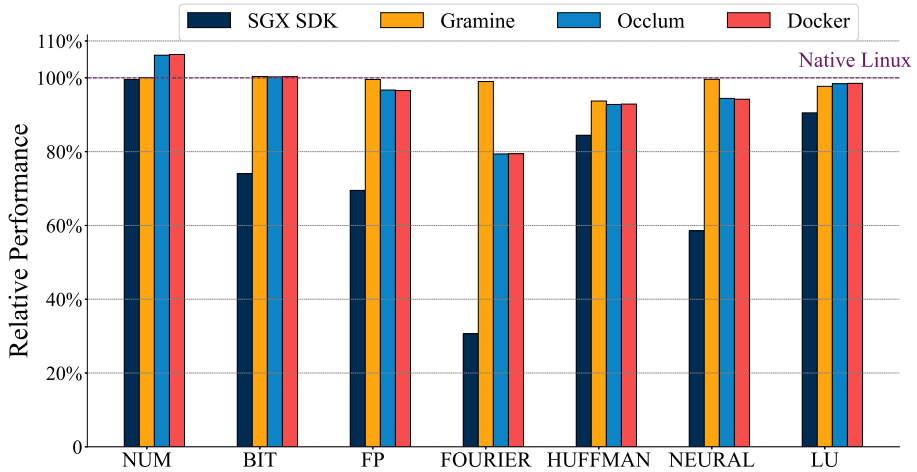


Figure 2. CPU-intensive workload.

Separately, in the numerical sorting task, all four SGX implementations perform well, with performance being close to that of native Linux. This is due to the use of heap

sort in the sorting task, and the small amount of numbers to be sorted in each group, which avoids the issue of insufficient enclave capacity. As a purely computational task, the performance of all SGX implementations is similar. In the bitwise operation test, LibOS-based approaches achieve performance close to native, while the SGX SDK shows a significant performance decline. In the floating-point computation tests, Occlum shows a performance decline relative to Gramine, indicating that Occlum is weaker in floating-point calculations. In purely CPU-intensive tasks, both Gramine and Occlum exhibit minimal performance degradation, while the SGX SDK experiences a notable performance decline. In edge intelligence scenarios, if using SGX SDK, it is essential to make efficient use of the limited memory space within the enclave and minimize the number of enclave transitions to reduce performance overhead.

4.3. I/O-Intensive Workload

In typical workflows, I/O is a common and crucial task. This is particularly relevant in edge intelligence applications, where devices are closer to the data source, making it important to focus on how to efficiently acquire and distribute data through I/O operations. Disk I/O is used for testing, with IOzone selected as the I/O-intensive workload. IOzone generates a series of file operations and measures their performance. IOzone is widely used and has been ported to various systems and platforms to assess file system performance. Since SGX does not support system calls natively, the SGX SDK-based solution requires using Ocalls to exit the enclave for I/O operations. This approach does not provide security guarantees, so we did not conduct I/O tests on the SGX SDK. Instead, we focused on testing LibOS-based solutions, as these solutions have their own file systems, allowing disk I/O to be performed within the enclave. We tested both Gramine and Occlum, which, despite both being based on LibOS, have significant differences in file systems. In Gramine, users need to customize a manifest file to specify trusted files, creating a virtual file system. Since Gramine only allows one process to exist within an enclave, processes cannot share encryption keys, which prevents them from sharing an encrypted file system. Gramine can only encrypt files marked as trusted, and it does not encrypt file metadata. In contrast, Occlum uses a fully encrypted file system divided into two layers: a read/write layer and a read-only layer. The read-only layer is encrypted from the "image" folder in the host's Occlum instance, while the read/write layer handles files generated during process execution.

To ensure the generality of the experimental results, each experiment was conducted a total of five times. Additionally, we ensured that only the current process performs file read/write and disk interaction operations. In the line charts, the short horizontal lines indicate the maximum and minimum values from the five tests, while the average of the five tests is connected to form the line chart. This paper mainly records the performance in four scenarios: sequential read/write and random read/write. Additionally, we focused on the system's read and write performance under different buffer sizes.

4.3.1. Sequential Read/Write Performance

Sequential read and write operations are both performed on files stored on disk, meaning relevant file information is not preloaded in memory. In the case of sequential write operations, new files are written, requiring not only data storage but also the overhead of tracking the data's location on the storage medium, known as metadata. The results are shown in Figure 3. Occlum's sequential read and write performance is significantly weaker than Gramine's, which is a result of the performance overhead introduced by the fully encrypted file system. When the buffer size was small, Gramine's sequential read and write speed could match or even surpass that of native Linux. However, once the size exceeded 1 MB, there was a noticeable decline in performance as the block size increased.

The reason for this is that Gramine reached its preset maximum read/write buffer size, and further increased its result in multiple read/write operations, which reduced efficiency. When the buffer size for sequential read reached 16 MB, Gramine's efficiency dropped to only about 20% of that of the native Linux, as shown in Figure 3a.

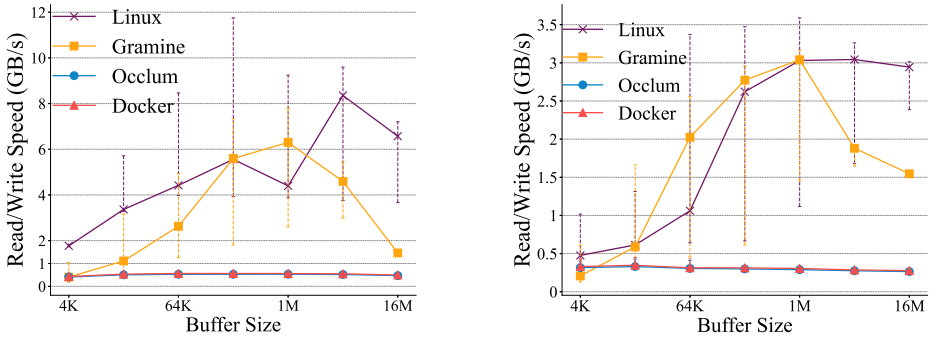


Figure 3. Sequential read/write performance. (**a**) Sequential read. (**b**) Sequential write.

4.3.2. Random Read/Write Performance

Random read and write operations were performed on cached files, which resulted in higher performance compared to sequential read and write operations. The results are shown in Figure 4. Occlum's read and write performance remained at a low level, while Gramine experienced a noticeable performance drop when the read/write buffer size exceeded 1 MB. Unlike sequential read and write operations, Gramine's performance with smaller buffer sizes did not surpass that of the native Linux. It merely approached native performance. This indicates that with caching, Gramine cannot achieve the same read and write performance as native Linux. The results in the virtual environment show almost no difference compared to those in the native environment.

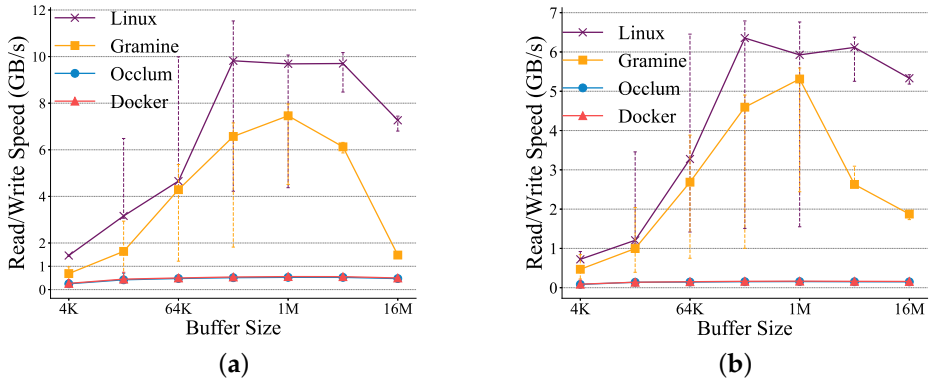


Figure 4. Random read/write performance. (**a**) Random read. (**b**) Random write.

Overall, SGX does not offer an I/O solution that balances performance and security in edge intelligence scenarios. The SGX SDK requires frequent enclave transitions for I/O operations, which introduces additional performance overhead without providing sufficient security. Occlum offers a secure file system, but its performance is too poor to be practically useful in real tasks. While Gramine achieves native performance with smaller buffer sizes in tests, it lacks comprehensive security guarantees and experiences significant performance degradation with larger buffer sizes.

4.4. Performance in Network Programming

In the context of IoT, the performance of network programming directly influences the speed of data exchange, thereby impacting application performance. This study simulated a scenario where a confidential server based on SGX was deployed remotely, and by adjusting the client load, server throughput and latency were observed to assess the impact of SGX on network programming performance. Lighttpd [68], a lightweight HTTP server, was enabled on the server side, while ApacheBench [69] was used on the client side to adjust the load. The simulation test was conducted by continuously downloading files from the server. The Lighttpd server version used was 1.4.40, and the ApacheBench version was 2.3. To simplify the experiment, both the server and the client ran on the same test machine, with the download address set to the local network 127.0.0.1 and port 8004. The test focused on two main aspects: first, testing multi-threaded performance by examining changes in system throughput as concurrency increases; and second, testing single-threaded performance by analyzing how latency changes as the size of the downloaded file increases. Since SGX SDK does not support system calls and cannot directly access ports, the experiment was conducted using SGX implementations based on LibOS, namely Gramine and Occlum.

4.4.1. Impact of Concurrency on Throughput

Since Occlum requires the use of the spawn method for process creation and does not support the fork system call, and fork is used in Lighttpd, the Lighttpd code needs to be modified to replace fork calls with spawn system calls. The code modification is planned for future implementation, and in this paper, testing is only performed on native Linux and Gramine. In the Gramine tests, the maximum number of threads for Lighttpd was set to 25, and the file downloaded by the client was a randomly generated file of 10 KB.

Figure 5 shows that as concurrency increases, both the native Linux and Gramine exhibit a trend where throughput initially increases and then stabilizes. When concurrency is low, Gramine's performance is significantly lower. For instance, when the concurrency is 1, Gramine achieves a throughput of only 7.6, while the native Linux exceeds 4000. The main reason for this is that with such a low load, the proportion of time spent handling download tasks decreases significantly, making the overhead of creating enclaves in Gramine particularly costly. However, as the load increases, both native Linux and Gramine show a rapid increase in throughput as concurrency rises from 1 to 50. When concurrency increases from 50 to 300, Gramine's throughput continues to rise, while the native Linux throughput stabilizes, and both eventually reach their maximum throughput. The test results indicate that Gramine's maximum throughput is about 75∼80% of that of native Linux.

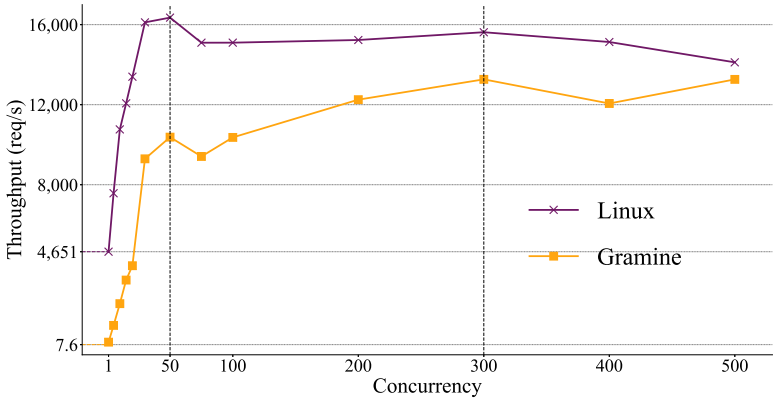


Figure 5. Impact of concurrency on throughput.

4.4.2. CPU Resource Usage

The CPU usage of the Gramine and Occlum frameworks is evaluated during network programming to address resource constraints in edge scenarios, where computational resources are limited. The evaluation focuses on the Lighttpd web server running within enclaves, with its CPU consumption monitored from an external terminal. To ensure the accuracy of the experimental results, the system cache is cleared prior to each test. The terminal used for testing is set to measure CPU usage every second. Results are shown in Figure 6. For Occlum, since all processes are initiated by a parent process, only the CPU usage of its child processes is measured. In contrast, for Gramine, the Lighttpd service is managed by the Platform Abstraction Layer (PAL) loader, making it the primary target for monitoring. The results show that both frameworks exhibit similar trends, with higher CPU usage during the initial runtime phase. However, as the server stabilizes, Occlum achieves slightly lower steady-state CPU overhead compared to Gramine. Overall, Occlum has a lower CPU overhead, but the initialization of the Lighttpd environment causes a significant number of context switches, resulting in the TEE's CPU overhead being much higher than that of a native Linux environment.

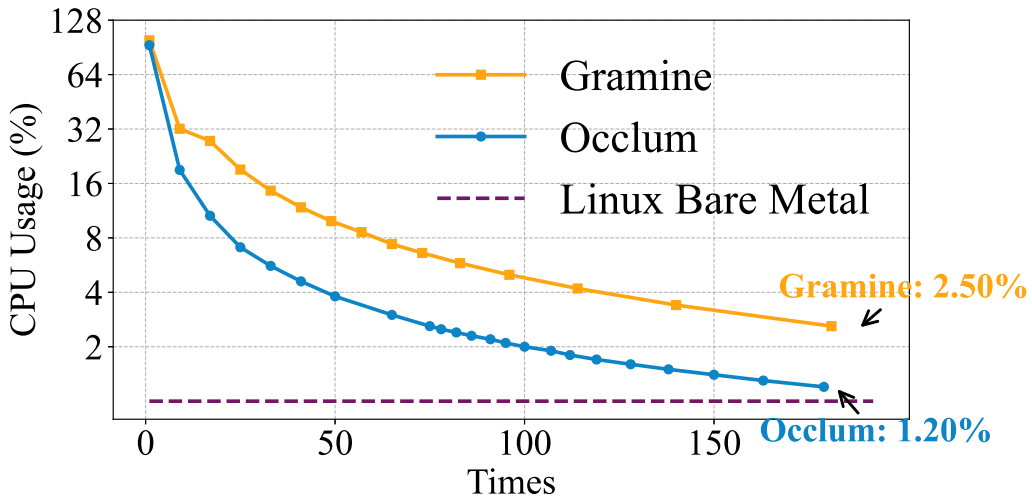


Figure 6. Comparison on CPU usage.

4.4.3. Impact of File Size on Latency

When measuring the impact of file size on download latency, a single-threaded mode was used, and native Linux, Gramine and Occlum were all tested. The client concurrency was set to 1, and the file size gradually increased from 1 KB to 100 KB to observe the changes in download latency. The results are shown in Figure 7. As the file size increases, there is no significant change in the latency of native Linux. For Occlum, latency increases both in virtual and native environments. Gramine is the most affected by file size, with latency rising significantly as the file size increases. Comparing server performance in single-threaded mode, Occlum demonstrates better performance in handling file downloads, particularly with larger files. Compared to native Linux, Occlum's latency is approximately 2 to 3 times higher, while Gramine, due to the impact of file size, can experience latency increases of over 10 times.

According to the results of the network programming performance tests, although Occlum performs better in file downloads in single-threaded mode, it still exhibits 2 to 3 times higher latency compared to native Linux, and is similarly affected by latency fluctuations due to file size. Additionally, Occlum's compatibility issues with the spawn-based pro-

cess creation mode can lead to usability problems. Gramine, on the other hand, has high download latency and is severely impacted by the size of the load.

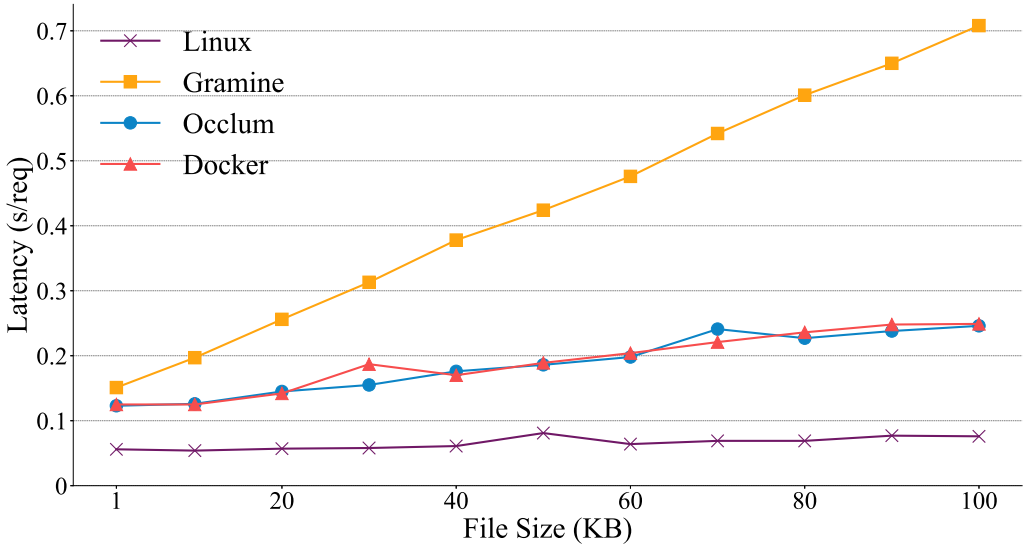


Figure 7. Impact of file size on latency.

4.5. System Call Overhead

A core challenge faced by SGX is how to manage its relationship with OS. In Section 2, two approaches to interacting OS are introduced. The SGX SDK fully distrusts OS, which means that system calls cannot be made within the enclave. While this approach enhances security, it presents challenges in terms of performance and code portability. On the other hand, when allowing for a moderate increase in TCB, LibOS-based solutions can perform most system calls within the enclave. Through security checks, these solutions can also avoid attacks from OS, such as Iago attacks [70]. We primarily tested the system call overhead of process creation and inter-process communication in Gramine and Occlum to observe SGX performance in multi-process tasks.

4.5.1. Process Creation Latency

Gramine and Occlum differ significantly in their process management models. Gramine only allows one process per enclave, while Occlum permits multiple processes within a single enclave. This leads to differences in their process creation methods: Occlum discards the fork system call and instead uses spawn to create new processes. Gramine, on the other hand, uses the traditional fork to create child processes, where the child and parent processes share the enclave properties defined by the manifest file. During testing, child processes are generated with varying memory sizes allocated via the malloc function. For parent processes, both native Linux and Gramine use fork system call, while Occlum uses spawn. The time required to create a new process with different memory sizes reflects the process creation latency.

In Figure 8, the process creation latency in Occlum is significantly lower than in Gramine. When the memory size of the new process is small (less than 100 MB), the performance can match that of native Linux. In Gramine, after a fork call, a new enclave is created, and complex interactions occur between the parent and child processes, such as identity authentication and key exchange, resulting in higher latency. Occlum, however, allows multiple processes to exist within a single enclave, where these processes share resources and are transparently managed by the operating system, greatly reducing process creation time. However, when the memory size of the new process exceeds the capacity of the EPC, it is constrained by the secure memory limit, requiring some of the memory to be

moved to regular memory. This triggers repeated encryption and decryption operations, leading to additional performance overhead and a sharp increase in latency.

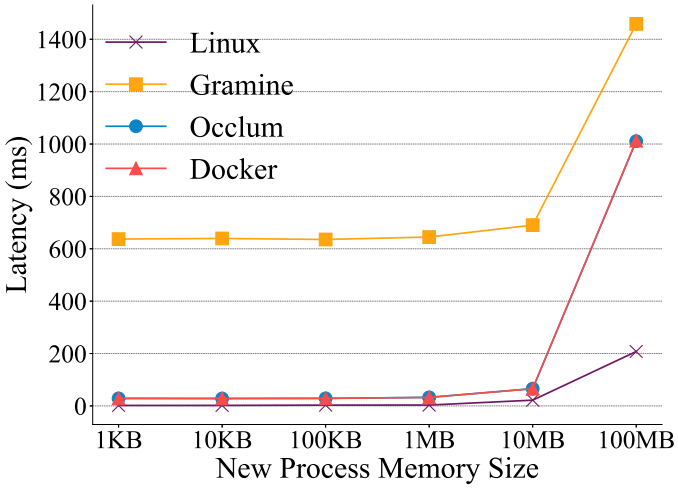


Figure 8. Process creation latency.

4.5.2. Inter-Process Communication Performance

Considering the different process execution models, in addition to process creation, inter-process communication is also a common and important scenario. First, we used the process creation methods mentioned earlier, whereas Gramine and native Linux used fork, and Occlum used spawn to create a child process. Communication between the parent and child processes was then established through the pipe system call. The parent process is responsible for writing data into the pipe, while the child process reads data from it. The throughput under different buffer sizes was used to reflect the efficiency of inter-process communication. In Figure 9, it can be observed that Occlum demonstrates high efficiency in inter-process communication, with almost no difference compared to native Linux. As the buffer size increases, the throughput of the pipe gradually increases. However, Gramine shows very low efficiency in inter-process communication. Through testing process-related system calls, although Gramine supports multi-process mode, its performance significantly declines. In contrast, Occlum achieves native level performance in both process creation and inter-process communication system calls.

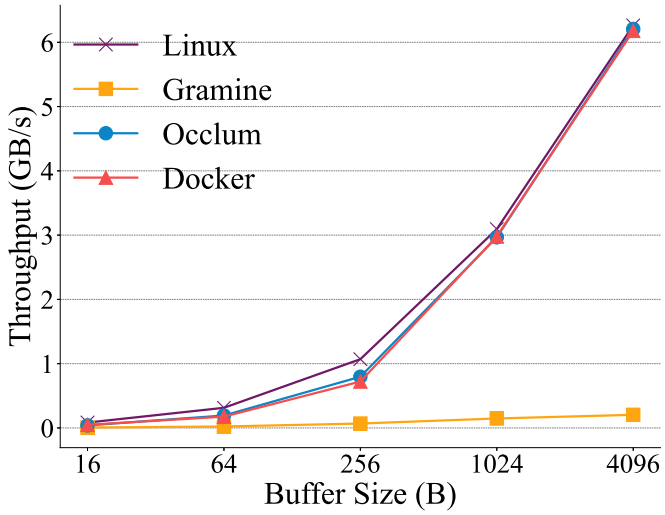


Figure 9. Inter-process communication.

4.6. Performance Under Realistic Workloads

In this section, we aim to evaluate the performance and applicability of SGX in the context of IoT workloads. IoT systems often involve extensive data caching and fast access requirements, which impose stringent demands on both performance and data security. SGX offers hardware-based TEE that ensures the confidentiality and integrity of sensitive data while maintaining system flexibility and scalability. To simulate a representative IoT workload, Memcached is selected as the target application for evaluation. Memcached, a lightweight and widely adopted in-memory caching system, is commonly utilized for efficient key-value storage and retrieval in distributed systems with high-speed data access requirements. Its workload characteristics closely align with the caching and access patterns frequently observed in IoT scenarios. By employing Memcached as the benchmark, this study replicates typical IoT workloads to systematically analyze the impact of SGX on both performance and security under realistic operating conditions. The results aim to provide meaningful insights into the feasibility and practicality of deploying SGX in real-world IoT environments.

The Memcached environment is configured for both Gramine and bare-metal Linux, with a focus on evaluating the relationship between throughput and latency. Memcached runs inside the SGX enclave, and load testing is performed using the memtier_benchmark [71] from a separate terminal. The default configuration uses four threads, and the load is increased by incrementally raising the number of concurrent clients per thread. The resulting performance metrics are presented in Figure 10. It is observed that the maximum throughput of Gramine is only about 30% of that achieved on bare-metal Linux, and its relative performance is even lower under light workloads. This indicates that in the Memcached scenario, the use of SGX still introduces significant performance overhead. Frequent encryption and decryption operations, along with transitions in and out of the enclave, severely limit the performance of edge devices.

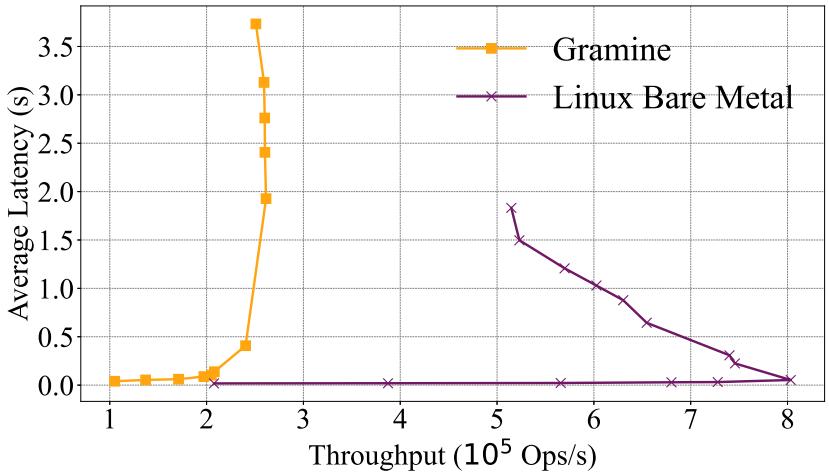


Figure 10. Throughput versus latency of Memcached.

4.7. Performance Improvement Strategies

Based on the performance tests, it is evident that SGX experiences significant performance degradation in edge intelligence scenarios. Although different SGX implementations excel in specific areas, such as Gramine in CPU-intensive tasks and Occlum in process management, the overall performance of SGX in real-world tasks like I/O and network programming remains poor. To improve SGX's performance in edge scenarios, three main approaches can be considered: optimizing enclave entry and exit, bypassing the kernel for I/O operations and adopting confidential virtual machine (CVM)-based TEE.

4.7.1. Optimizing Enclave Entry and Exit

Enclave switching poses a significant bottleneck in SGX-based systems, with the overhead of a single Ecall or Ocall exceeding 14,000 cycles [28]. In comparison, a typical system call incurs only around 150 cycles. Since nearly all system calls within an enclave require switching to the untrusted application to complete, frequent transitions result in substantial performance overhead. To address this issue, approaches like HotCalls [28] and Eleos [67] have been proposed. HotCalls reduces the latency of enclave transitions by leveraging a spin–lock mechanism and shared unencrypted memory, cutting the overhead of Ecalls and Ocalls to as low as 620 cycles. Eleos, on the other hand, employs an "exitless" architecture that batches system calls, caches results and utilizes shared memory buffers to minimize the need for enclave exits, achieving a 2–3x reduction in overhead for system call-intensive workloads. Both approaches effectively mitigate the severe performance degradation caused by frequent enclave transitions, making SGX more practical for real-world, high-performance scenarios. However, our experimental evaluation reveals that the LibOS-based approach offers a more balanced solution for performance, security and usability in SGX implementations. By integrating most system calls directly within the enclave, LibOS significantly reduces the frequency of enclave switches. With a moderate increase in the TCB size, it also eliminates the need for partitioning existing code, allowing applications to be deployed into enclaves efficiently and with minimal performance overhead. Consequently, we conclude that the LibOS-based approach is particularly well suited for IoT and edge intelligence scenarios, where both performance optimization and ease of deployment are critical.

4.7.2. Bypassing the Kernel for I/O Operations

SGX performs well in CPU-intensive tasks, but it shows poor performance in I/O-intensive tasks and real-world applications, necessitating specialized optimization for I/O operations. SGX inherently lacks support for trusted I/O and does not allow efficient methods such as DMA to access enclave memory. Traditional solutions involve multiple copying steps: from disk or network cards to the kernel space, then from the kernel space to untrusted application memory, and finally, into the enclave, resulting in significant performance degradation. To address these challenges and provide secure and reliable peripheral channels, Aurora [51] proposes a trusted I/O path based on the System Management Mode (SMM). Aurora uses System Management RAM (SMRAM) to isolate and protect I/O operations, creating a secure communication channel between the enclave and peripheral devices. Key features of Aurora include support for HID keyboards, USB storage, hardware clocks and serial printers, as well as a batch processing mechanism to optimize performance by reducing context switching. On the other hand, an efficient architecture called SMK [72] focuses on extending SGX capabilities to address specific issues such as secure networking and trusted timing in distributed environments. SMK provides a trusted network by running protocol stacks on trusted hardware, ensuring data integrity and authenticity. Additionally, SMK introduces a trusted clock system to ensure that applications relying on precise timestamps (e.g., blockchain and secure communications) remain secure and reliable.

Although these methods can achieve secure I/O, they do not provide significant performance improvements. We believe that since TEE does not inherently trust privileged software, bypassing the kernel is a more suitable solution for TEE systems. This approach aligns with the TEE's core security model, minimizing reliance on untrusted components and reducing the attack surface while ensuring both the integrity and confidentiality of data during I/O operations. Rocket-IO is a direct I/O stack for TEEs that bypasses the untrusted kernel using user–space libraries like DPDK and SPDK, enabling efficient and secure hard-

ware interaction. It eliminates multiple data copies and integrates encryption, significantly enhancing both I/O performance and security compared to existing frameworks. At the same time, we believe that new hardware and protocols can be leveraged to achieve the authentication of trusted peripherals, enabling direct data interaction. For instance, utilizing RDMA network cards, device authentication can be conducted through the SPDM protocol, extending the trust domain from the CPU to external devices. Subsequently, direct data exchange can be realized through methods such as DMA. Additionally, integrating advanced hardware like Persistent Memory can enhance both efficiency and security by enabling high-speed, non-volatile storage and direct data access. Combined with TEEs like Intel SGX, this ensures low-latency and secure data interaction across devices, while maintaining robust protection for sensitive operations [73].

4.7.3. Adopting CVM-Based TEE

Although the methods mentioned above can effectively improve the performance of SGX, its fundamental limitation lies in its design as a user–space TEE. While distrusting the OS, SGX still relies on OS-provided services, such as page table management, leading to inherent performance and security challenges. In contrast, VM-based TEEs, such as HyperEnclave [74], leverage a trusted hypervisor to build a flexible, general-purpose TEE architecture, offering multiple enclave types to suit diverse application needs. CVMs, in particular, are evolving rapidly, enabling fully functional virtual machines where unmodified applications can run seamlessly. Despite introducing a slightly larger TCB, CVMs significantly enhance performance for I/O-intensive tasks [75]. For example, Intel TDX [76] addresses SGX's I/O limitations by supporting trusted I/O, while solutions like Bifrost [77] optimize CVM-I/O performance through techniques such as zero-copy encryption and packet reassembly, achieving up to 21.50% performance gains over traditional VMs. Similarly, FOLIO [78], a DPDK-based software solution, improves network I/O performance for CVMs without relying on trusted I/O hardware, achieving performance within 6% of ideal TIO configurations while maintaining security and compatibility with DPDK applications. Given these advancements, CVM-based approaches are likely to represent the most practical and effective model for implementing TEEs in the future, provided there are no extreme constraints on TCB size.

5. Related Works

In this section, we primarily summarize the works related to the analysis and evaluation of TEE technologies.

For SGX-based TEE technologies, Hasan et al. [32] implemented LMbench using both the SGX SDK and Gramine, focusing on four scenarios: Forkless, SGX, NoSGX and Gramine. They compared the performance of the porting method and the shim-based method by analyzing system read/write performance and the overhead of certain system calls. They also tested Ecall and Ocall overhead, showing that the shim-based method is more optimization-friendly. After several iterations, LibOS-based technologies like Gramine and Occlum outperformed the port-based method in some scenarios. For SGXGauge, the authors of [66] examined SGX performance under different memory usage levels, using EPC size as a baseline and testing in low, medium and high memory usage conditions. They compared native Linux, SGX SDK-based and LibOS-based implementations, revealing a significant performance drop in the SGX SDK when memory exceeded the EPC, while LibOS-based implementations maintained stable performance. Weisse et al. [28] detailed Ecall and Ocall performance overheads in warm and cold cache environments. They also tested data exchange speeds between non-secure applications and enclaves across four buffer transfer modes: zero copy, copying in, copying out and copying in&out. The results

showed poor performance in the application–enclave interface, prompting the proposal of HotCalls to redesign the interface and improve performance.

For other TEE technologies, analysis and evaluation are essential components. For Arm TrustZone, a study [79] comparing it with other TEE technologies concluded that TrustZone is more hardware-efficient and avoids the risks of a highly privileged "black box" controlling the system. For ARM CCA, the authors of [37] conducted detailed performance tests under various workloads, including hypercalls, I/O instructions and application benchmarks like Apache and MySQL, identifying key performance bottlenecks. On the RISC-V platform, the authors of [19] evaluated the performance of Penglai Enclave using the SPECCPU benchmark and addressed security challenges such as controlled-channel and cache-based side-channel attacks, proposing mitigation strategies to enhance security. Additionally, some studies focus on TEE technologies that adopt VM-based protection, such as AMD SEV and Intel TDX. For instance, recent work [75] conducted a comparative evaluation of TDX, SEV, Gramine-SGX and Occlum-SGX, analyzing computational overhead and resource usage under various operational scenarios with legacy applications. This study uniquely evaluates TDX, providing valuable insights into the performance of CVM-based TEEs under realistic conditions. In contrast, the abovementioned works do not consider the integration of TEE technologies into IoT environments. In resource-constrained edge intelligence scenarios, the performance and applicability of TEEs face significant challenges, where factors such as limited computational power and energy efficiency can heavily impact their effectiveness.

6. Conclusions

This paper presented an in-depth analysis and evaluation of SGX-based TEE technologies in IoT and edge intelligence scenarios. Through comprehensive performance testing of various SGX implementations, including those based on SGX SDK and LibOS, we identified key challenges, such as performance degradation and I/O security issues, that arise when applying TEE in resource-constrained and latency-sensitive edge environments. Our experimental results highlight significant performance bottlenecks, particularly in areas like enclave transitions and secure I/O operations. These findings offer critical insights into the limitations of current SGX solutions and provide valuable benchmarks for improving the integration of TEE in IoT scenarios. Additionally, it proposes corresponding performance optimization strategies, offering practical approaches for deploying TEE in IoT and edge intelligence scenarios.

However, this study has some limitations. Primarily, it focuses on SGX-based TEE implementations and does not include evaluation with real workloads for other kinds of TEE technologies. Second, some security concerns related to TEE usage in edge environments, such as Man-in-the-Middle and Denial of Service attacks, are not covered in this work. Future work will focus on implementing the proposed performance optimizations and I/O security enhancements, while also broadening our experimental scope to cover other TEE technologies. This will provide a more comprehensive understanding of TEE utilization and find potential improvements in IoT–Edge Cloud Continuum.

Author Contributions: Conceptualization, investigation and funding acquisition Y.Z. ; methodology, software, data curation, visualization and draft preparation, Z.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key Research and Development Program of China under Grant No. 2023YFB4503000.

Data Availability Statement: The datasets used or analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
2. Al-Ansi, A.; Al-Ansi, A.M.; Muthanna, A.; Elgendy, I.A.; Koucheryavy, A. Survey on Intelligence Edge Computing in 6G: Characteristics, Challenges, Potential Use Cases, and Market Drivers. *Future Internet* **2021**, *13*, 118. [CrossRef]
3. Zikria, Y.B.; Ali, R.; Afzal, M.K.; Kim, S.W. Next-Generation Internet of Things (IoT): Opportunities, Challenges, and Solutions. *Sensors* **2021**, *21*, 1174. [CrossRef]
4. Han, J.; Zhang, Y.; Liu, J.; Li, Z.; Xian, M.; Wang, H.; Mao, F.; Chen, Y. A Blockchain-Based and SGX-Enabled Access Control Framework for IoT. *Electronics* **2022**, *11*, 2710. [CrossRef]
5. Samuel, S.S.I. A review of connectivity challenges in IoT-smart home. In Proceedings of the 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC), Muscat, Oman, 15–16 March 2016; pp. 1–4. [CrossRef]
6. Haghi Kashani, M.; Madanipour, M.; Nikravan, M.; Asghari, P.; Mahdipour, E. A systematic review of IoT in healthcare: Applications, techniques, and trends. *J. Netw. Comput. Appl.* **2021**, *192*, 103164. [CrossRef]
7. Selvaraj, S.; Sundaravaradhan, S. Challenges and opportunities in IoT healthcare systems: A systematic review. *SN Appl. Sci.* **2019**, *2*, 139. [CrossRef]
8. Kalsoom, T.; Ahmed, S.; Rafi-ul Shan, P.M.; Azmat, M.; Akhtar, P.; Pervez, Z.; Imran, M.A.; Ur-Rehman, M. Impact of IoT on Manufacturing Industry 4.0: A New Triangular Systematic Review. *Sustainability* **2021**, *13*, 12506. [CrossRef]
9. Trakadas, P.; Simoens, P.; Gkonis, P.; Sarakis, L.; Angelopoulos, A.; Ramallo-González, A.P.; Skarmeta, A.; Trochoutsos, C.; Calvo, D.; Pariente, T.; et al. An artificial intelligence-based collaboration approach in industrial iot manufacturing: Key concepts, architectural extensions and potential applications. *Sensors* **2020**, *20*, 5480. [CrossRef]
10. Zantalis, F.; Koulouras, G.; Karabetsos, S.; Kandris, D. A Review of Machine Learning and IoT in Smart Transportation. *Future Internet* **2019**, *11*, 94. [CrossRef]
11. Muthuramalingam, S.; Bharathi, A.; Rakesh kumar, S.; Gayathri, N.; Sathiyaraj, R.; Balamurugan, B. IoT Based Intelligent Transportation System (IoT-ITS) for Global Perspective: A Case Study. In *Internet of Things and Big Data Analytics for Smart Generation*; Balas, V.E., Solanki, V.K., Kumar, R., Khari, M., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 279–300. [CrossRef]
12. Hussain, F.; Hussain, R.; Hassan, S.A.; Hossain, E. Machine Learning in IoT Security: Current Solutions and Future Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1686–1721. [CrossRef]
13. Gkonis, P.; Giannopoulos, A.; Trakadas, P.; Masip-Bruin, X.; D'Andria, F. A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues. *Future Internet* **2023**, *15*, 383. [CrossRef]
14. Mohanty, J.; Mishra, S.; Patra, S.; Pati, B.; Panigrahi, C.R. IoT Security, Challenges, and Solutions: A Review. In *Progress in Advanced Computing and Intelligent Engineering*; Panigrahi, C.R., Pati, B., Mohapatra, P., Buyya, R., Li, K.C., Eds.; Springer: Singapore, 2021; pp. 493–504. [CrossRef]
15. Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted Execution Environment: What It is, and What It is Not. In Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 57–64. [CrossRef]
16. Valadares, D.C.G.; Will, N.C.; Spohn, M.A.; Santos, D.F.d.S.; Perkusich, A.; Gorgônio, K.C. Confidential computing in cloud/fog-based Internet of Things scenarios. *Internet Things* **2022**, *19*, 100543. [CrossRef]
17. Sev-Snp, A. Strengthening VM isolation with integrity protection and more. *White Pap. January* **2020**, *53*, 1450–1465.
18. Pinto, S.; Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* **2019**, *51*, 130:1–130:36. [CrossRef]
19. Feng, E.; Lu, X.; Du, D.; Yang, B.; Jiang, X.; Xia, Y.; Zang, B.; Chen, H. Scalable memory protection in the PENGLAI enclave. In Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), Online, 14–16 July 2021; pp. 275–294.
20. Hoekstra, M.; Lal, R.; Pappachan, P.; Phegade, V.; Del Cuvillo, J. Using innovative instructions to create trustworthy software solutions. In Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, 23–24 June 2013; p. 1. [CrossRef]
21. Will, N.C.; Gomes Valadares, D.C.; De Souza Santos, D.F.; Perkusich, A. Intel Software Guard Extensions in Internet of Things Scenarios: A Systematic Mapping Study. In Proceedings of the 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud), Rome, Italy, 23–25 August 2021; pp. 342–349. [CrossRef]
22. Chen, Y.; Sun, W.; Zhang, N.; Zheng, Q.; Lou, W.; Hou, Y.T. Towards Efficient Fine-Grained Access Control and Trustworthy Data Processing for Remote Monitoring Services in IoT. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 1830–1842. [CrossRef]
23. Ayoade, G.; Karande, V.; Khan, L.; Hamlen, K. Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment. In Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT, USA, 6–9 July 2018; pp. 15–22. [CrossRef]

24. Nilsson, A.; Bideh, P.N.; Brorsson, J. A Survey of Published Attacks on Intel SGX. *arXiv* **2020**, arXiv:2006.13598. [CrossRef]
25. Götzfried, J.; Eckert, M.; Schinzel, S.; Müller, T. Cache Attacks on Intel SGX. In Proceedings of the 10th European Workshop on Systems Security, New York, NY, USA, 23–26 April 2017; EuroSec'17; pp. 1–6. [CrossRef]
26. Van Bulck, J.; Piessens, F.; Strackx, R. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In Proceedings of the 2nd Workshop on System Software for Trusted Execution, Shanghai, China, 28 October 2017; pp. 1–6. [CrossRef]
27. Dhar, A.; Ulqinaku, E.; Kostiainen, K.; Capkun, S. ProtectIOn: Root-of-Trust for IO in Compromised Platforms. Cryptology ePrint Archive. 2019. Available online: https://eprint.iacr.org/2019/869 (accessed on 10 September 2024).
28. Weisse, O.; Bertacco, V.; Austin, T. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. *SIGARCH Comput. Archit. News* **2017**, *45*, 81–93. [CrossRef]
29. Scarlata, V.; Johnson, S.; Beaney, J.; Zmijewski, P. Supporting third party attestation for Intel SGX with Intel data center attestation primitives. *White Pap.* **2018**, *12*, 1–8.
30. Dinh Ngoc, T.; Bui, B.; Bitchebe, S.; Tchana, A.; Schiavoni, V.; Felber, P.; Hagimont, D. Everything You Should Know About Intel SGX Performance on Virtualized Systems. *Proc. ACM Meas. Anal. Comput. Syst.* **2019**, *3*, 5:1–5:21. [CrossRef]
31. Zhao, C.; Saifuding, D.; Tian, H.; Zhang, Y.; Xing, C. On the Performance of Intel SGX. In Proceedings of the 2016 13th Web Information Systems and Applications Conference (WISA), Wuhan, China, 23–25 September 2016; pp. 184–187. [CrossRef]
32. Hasan, A.; Riley, R.; Ponomarev, D. Port or Shim? Stress Testing Application Performance on Intel SGX. In Proceedings of the 2020 IEEE International Symposium on Workload Characterization (IISWC), Beijing, China, 27–30 October 2020; pp. 123–133. [CrossRef]
33. Baumann, A.; Peinado, M.; Hunt, G. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.* **2015**, *33*, 8:1–8:26. [CrossRef]
34. Arnautov, S.; Trach, B.; Gregor, F.; Knauth, T.; Martin, A.; Priebe, C.; Lind, J.; Muthukumaran, D.; O'Keeffe, D.; Stillwell, M.L.; et al. {SCONE}: Secure Linux Containers with Intel {SGX}. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), Savannah, GA, USA, 2–4 November 2016; pp. 689–703.
35. Tsai, C.C.; Porter, D.E.; Vij, M. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA, USA, 12–14 July 2017; pp. 645–658.
36. Shen, Y.; Tian, H.; Chen, Y.; Chen, K.; Wang, R.; Xu, Y.; Xia, Y.; Yan, S. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 16–20 March 2020; pp. 955–970. [CrossRef]
37. Li, X.; Li, X.; Dall, C.; Gu, R.; Nieh, J.; Sait, Y.; Stockwell, G. Design and verification of the arm confidential compute architecture. In Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), Carlsbad, CA, USA, 11–13 July 2022; pp. 465–484.
38. Lesjak, C.; Hein, D.; Winter, J. Hardware-security technologies for industrial IoT: TrustZone and security controller. In Proceedings of the IECON 2015—41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, 9–12 November 2015; pp. 002589–002595. [CrossRef]
39. Fitzek, A.; Achleitner, F.; Winter, J.; Hein, D. The ANDIX research OS—ARM TrustZone meets industrial control systems security. In Proceedings of the 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), Cambridge, UK, 22–24 July 2015; pp. 88–93.
40. Sang, F.; Lee, J.; Zhang, X.; Kim, T. PORTAL: Fast and Secure Device Access with Arm CCA for Modern Arm Mobile System-on-Chips (SoCs). In Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 12–14 May 2025; p. 13.
41. Ayoade, G.; El-Ghamry, A.; Karande, V.; Khan, L.; Alrahmawy, M.; Rashad, M.Z. Secure data processing for IoT middleware systems. *J. Supercomput.* **2019**, *75*, 4684–4709. [CrossRef]
42. Showail, A.; Tahir, R.; Zaffar, M.F.; Noor, M.H.; Al-Khatib, M. An internet of secure and private things: A service-oriented architecture. *Comput. Secur.* **2022**, *120*, 102776. [CrossRef]
43. Lei, H.; Yan, Y.; Bao, Z.; Wang, Q.; Zhang, Y.; Shi, W. SDSBT: A Secure Multi-party Data Sharing Platform Based on Blockchain and TEE. In *Cyberspace Safety and Security*; Cheng, J., Tang, X., Liu, X., Eds.; Springer: Cham, Switzerland, 2021; pp. 184–196. [CrossRef]
44. Wang, J.; Hong, Z.; Zhang, Y.; Jin, Y. Enabling Security-Enhanced Attestation With Intel SGX for Remote Terminal and IoT. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 88–96. [CrossRef]
45. Woo, S.; Song, J.; Park, S. A Distributed Oracle Using Intel SGX for Blockchain-Based IoT Applications. *Sensors* **2020**, *20*, 2725. [CrossRef]
46. Zhang, X.; Li, X.; Miao, Y.; Luo, X.; Wang, Y.; Ma, S.; Weng, J. A Data Trading Scheme with Efficient Data Usage Control for Industrial IoT. *IEEE Trans. Ind. Inform.* **2022**, *18*, 4456–4465. [CrossRef]

47. Kalapaaking, A.P.; Khalil, I.; Rahman, M.S.; Atiquzzaman, M.; Yi, X.; Almashor, M. Blockchain-Based Federated Learning With Secure Aggregation in Trusted Execution Environment for Internet-of-Things. *IEEE Trans. Ind. Inform.* **2023**, *19*, 1703–1714. [CrossRef]
48. Li, S.; Xue, K.; Wei, D.S.L.; Yue, H.; Yu, N.; Hong, P. SecGrid: A Secure and Efficient SGX-Enabled Smart Grid System with Rich Functionalities. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 1318–1330. [CrossRef]
49. Gao, Y.; Lin, H.; Chen, Y.; Liu, Y. Blockchain and SGX-Enabled Edge-Computing-Empowered Secure IoMT Data Analysis. *IEEE Internet Things J.* **2021**, *8*, 15785–15795. [CrossRef]
50. Weiser, S.; Werner, M. SGXIO: Generic Trusted I/O Path for Intel SGX. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 261–268. [CrossRef]
51. Liang, H.; Li, M.; Chen, Y.; Jiang, L.; Xie, Z.; Yang, T. Establishing Trusted I/O Paths for SGX Client Systems with Aurora. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 1589–1600. [CrossRef]
52. Thalheim, J.; Unnibhavi, H.; Priebe, C.; Bhatotia, P.; Pietzuch, P. rkt-io: A direct I/O stack for shielded execution. In Proceedings of the Sixteenth European Conference on Computer Systems, Online Event, UK, 26–28 April 2021; pp. 490–506. [CrossRef]
53. Stancu, F.A.; Trancă, D.C.; Chiroiu, M. TIO—Secure Input/Output for Intel SGX Enclaves. In Proceedings of the 2019 International Workshop on Secure Internet of Things (SIOT), Luxembourg, 26 September 2019; pp. 1–9, ISSN 2690-8557. [CrossRef]
54. Jang, Y.; Keem, S. SGX-USB: Secure USB I/O Path for Secure Enclaves. In Proceedings of the 57th Hawaii International Conference on System Sciences, HICSS 2024, Hilton Hawaiian Village Waikiki Beach Resort, Honolulu , HI, USA, 3–6 January 2024; pp. 7437–7446.
55. Peters, T.; Lal, R.; Varadarajan, S.; Pappachan, P.; Kotz, D. BASTION-SGX: Bluetooth and Architectural Support for Trusted I/O on SGX. In Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, Los Angeles, CA, USA, 2 June 2018; pp. 1–9. [CrossRef]
56. Eskandarian, S.; Cogan, J.; Birnbaum, S.; Brandon, P.C.W.; Franke, D.; Fraser, F.; Garcia, G.; Gong, E.; Nguyen, H.T.; Sethi, T.K.; et al. Fidelius: Protecting User Secrets from Compromised Browsers. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 264–280, ISSN 2375-1207. [CrossRef]
57. Shinde, S.; Chua, Z.L.; Narayanan, V.; Saxena, P. Preventing page faults from telling your secrets. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, Xi'an, China, 30 May–3 June 2016; pp. 317–328.
58. Wang, W.; Chen, G.; Pan, X.; Zhang, Y.; Wang, X.; Bindschaedler, V.; Tang, H.; Gunter, C.A. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October –3 November 2017; pp. 2421–2434.
59. Xu, Y.; Cui, W.; Peinado, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 640–656.
60. Lee, S.; Shih, M.W.; Gera, P.; Kim, T.; Kim, H.; Peinado, M. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 557–574.
61. Evtyushkin, D.; Riley, R.; Abu-Ghazaleh, N.; Ponomarev, D. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Not.* **2018**, *53*, 693–707. [CrossRef]
62. Cho, H.; Zhang, P.; Kim, D.; Park, J.; Lee, C.H.; Zhao, Z.; Doupé, A.; Ahn, G.J. Prime+ count: Novel cross-world covert channels on arm trustzone. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 441–452.
63. Costan, V.; Lebedev, I.; Devadas, S. Sanctum: Minimal hardware extensions for strong software isolation. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 857–874.
64. Kim, T.; Peinado, M.; Mainar-Ruiz, G. {STEALTHMEM}:{System-Level} protection against {Cache-Based} side channel attacks in the cloud. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 8–10 August 2012; pp. 189–204.
65. Werner, M.; Unterluggauer, T.; Giner, L.; Schwarz, M.; Gruss, D.; Mangard, S. {ScatterCache}: Thwarting cache attacks via cache set randomization. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 675–692.
66. Kumar, S.; Panda, A.; Sarangi, S.R. A Comprehensive Benchmark Suite for Intel SGX. *arXiv* **2022**, arXiv:2205.06415. [CrossRef]
67. Orenbach, M.; Lifshits, P.; Minkin, M.; Silberstein, M. Eleos: ExitLess OS Services for SGX Enclaves. In Proceedings of the Twelfth European Conference on Computer Systems, Belgrade, Serbia, 23–26 April 2017; pp. 238–253. [CrossRef]
68. Bogus, A. *Lighttpd*; Packt Publishing Ltd.: Birmingham, UK, 2008.
69. Apache HTTP Benchmarking Tool. ApacheBench Documentation. Available online: http://httpd.apache.org/docs/2.4/programs/ab.html (accessed on 10 September 2024).
70. Checkoway, S.; Shacham, H. Iago attacks: Why the system call API is a bad untrusted RPC interface. *ACM SIGARCH Comput. Archit. News* **2013**, *41*, 253–264. [CrossRef]

71. RedisLabs. *Memtier_BENCHMARK*. Available online: https://github.com/RedisLabs/memtier_benchmark (accessed on 20 December 2024).
72. Liang, H.; Li, M.; Chen, Y.; Yang, T.; Xie, Z.; Jiang, L. Architectural protection of trusted system services for SGX enclaves in cloud computing. *IEEE Trans. Cloud Comput.* **2019**, *9*, 910–922. [CrossRef]
73. Stavrakakis, D.; Giantsidi, D.; Bailleu, M.; Sändig, P.; Issa, S.; Bhatotia, P. Anchor: A Library for Building Secure Persistent Memory Systems. *Proc. ACM Manag. Data* **2023**, *1*, 1–31. [CrossRef]
74. Jia, Y.; Liu, S.; Wang, W.; Chen, Y.; Zhai, Z.; Yan, S.; He, Z. HyperEnclave: An open and cross-platform trusted execution environment. In Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22), Carlsbad, CA, USA, 11–13 July 2022; pp. 437–454.
75. Coppolino, L.; D'Antonio, S.; Iasio, D.; Mazzeo, G.; Romano, L. An Experimental Evaluation of TEE technology Evolution: Benchmarking Transparent Approaches based on SGX, SEV, and TDX. *arXiv* **2024**, arXiv:2408.00443.
76. Cheng, P.C.; Ozga, W.; Valdez, E.; Ahmed, S.; Gu, Z.; Jamjoom, H.; Franke, H.; Bottomley, J. Intel tdx demystified: A top-down approach. *ACM Comput. Surv.* **2024**, *56*, 1–33. [CrossRef]
77. Li, D.; Mi, Z.; Ji, C.; Tan, Y.; Zang, B.; Guan, H.; Chen, H. Bifrost: Analysis and Optimization of Network {I/O} Tax in Confidential Virtual Machines. In Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 23), Boston, MA, USA, 10–12 July 2023; pp. 1–15.
78. Li, M.; Srivastava, S.; Yan, M. Bridge the Future: High-Performance Networks in Confidential VMs without Trusted I/O devices. *arXiv* **2024**, arXiv:2403.03360.
79. Ngabonziza, B.; Martin, D.; Bailey, A.; Cho, H.; Martin, S. Trustzone explained: Architectural features and use cases. In Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; pp. 445–451.

MDPI AG
Grosspeteranlage 5
4052 Basel
Switzerland
Tel.: +41 61 683 77 34

Future Internet Editorial Office
E-mail: futureinternet@mdpi.com
www.mdpi.com/journal/futureinternet

Academic Open
Access Publishing

mdpi.com

ISBN 978-3-7258-3949-0