symmetry

MDPI

Article

High-Throughput and Low-Latency DrDoS Detection Using Quantized ONNX Models

Salih Eren [1], Alperen Gültekin [1], Ömer Özkan [1] and İlker Özçelik [2,*]

[1] Department of Computer Engineering, Faculty of Engineering and Architecture, Eskisehir Osmangazi University, Eskisehir 26040, Türkiye; saliheren266@gmail.com (S.E.); alperen06gultekin@gmail.com (A.G.); o.ozkan4393@gmail.com (Ö.Ö.)

[2] Department of Software Engineering, Faculty of Engineering and Architecture, Eskisehir Osmangazi University, Eskisehir 26040, Türkiye

* Correspondence: ilker.ozcelik@ogu.edu.tr

Abstract

Distributed Reflection Denial-of-Service (DrDoS) attacks, such as DNS Amplification, exploit inherent asymmetries in standard network protocols to generate devastating traffic volumes. To restore defense-side balance against these asymmetric threats, we present a comprehensive, systematic comparison of ONNX compilation and quantization configurations across Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Gated Recurrent Unit (GRU) architectures evaluated on a unified reference platform. To achieve this, our evaluation analyzes training durations, throughput scalability, and sample latency across varying batch sizes to align with operational network environments. We demonstrate that while small batch sizes suffer from data transfer overhead, increasing batch configurations significantly accelerates GPU throughput, particularly for statically quantized ONNX models. Additionally, training durations exhibit an inverse scaling relationship with batch size, yielding massive temporal savings as workloads expand. Furthermore, larger batch sizes effectively amortize fixed execution costs across thousands of samples, reducing average per-sample latency. Ultimately, this systematic evaluation provides a deployment blueprint for highly efficient intrusion detection engines capable of neutralizing asymmetric network threats at line-rate.

Keywords: DrDoS; DDoS; DNS amplification; ONNX; quantization; deep learning; intrusion detection; MLP; real-time detection; network security

Academic Editor: Shuang Xu

Received: 27 April 2026
Revised: 10 July 2026
Accepted: 12 July 2026
Published: 14 July 2026

1. Introduction

The Domain Name System (DNS), a cornerstone of contemporary digital infrastructure, enables users to access internet resources through memorable domain names instead of IP addresses. However, DNS was architected during the internet's formative stages, a period when security was not a primary design consideration. Consequently, the system remains inherently vulnerable to Distributed Denial-of-Service (DDoS) attacks [1]. These threats primarily manifest as DNS reflection and DNS resource exhaustion, which are fundamentally driven by structural and protocol asymmetries. Attackers exploit DNS servers as reflection vectors to generate high-volume traffic, leveraging minimal request payloads to elicit disproportionately amplified responses, a phenomenon known as traffic asymmetry. This escalating threat landscape is evidenced by 2023 data showing a DNS component in 60% of all DDoS incidents [2], followed by a staggering 80% surge in such attacks during the first quarter of 2024 [3].

As the complexity and dynamic nature of these threats outpace traditional security solutions, deep learning-based methods have emerged as a robust alternative for intrusion detection. By leveraging large-scale network traffic data, these approaches can effectively identify anomalous behavior that evades rule-based systems. Although these models demonstrate high detection accuracy, their practical deployment is hindered by a significant operational hurdle. This challenge primarily concerns the necessity for real-time operation within critical infrastructure, such as Internet Service Providers (ISPs) and Internet Exchange Points (IXPs).

In these high-demand environments, model size and inference speed are of paramount importance [4]. The substantial memory requirements and prolonged latency associated with conventional deep learning architectures often create severe performance bottlenecks [5]. Consequently, the operational success of anomaly-based detection now depends heavily on model optimization and compression techniques. Among these, quantization has become indispensable, offering a means to significantly reduce both model footprint and execution time by converting model weights into lower-precision numerical formats [6].

Although a comprehensive intrusion detection pipeline entails several consecutive operations, including packet capture, line-rate parsing, flow aggregation, and classification, this study specifically isolates and benchmarks the deep learning inference stage. In modern high-speed security architectures, upstream network telemetry and preprocessing tasks are increasingly offloaded to high-performance kernel spaces or hardware acceleration interfaces (e.g., eBPF/XDP or DPDK), leaving the deep learning model itself as the primary software bottleneck. Thus, isolating the inference stage allows us to systematically evaluate model optimization and compression frameworks without introducing confounding variables from volatile network-stack performance. Within this scope, the primary contributions of this study to the literature are summarized below:

- We establish a systematic, side-by-side comparative optimization framework evaluating the integration of ONNX conversion and quantization (both static and dynamic) across three structurally distinct neural network architectures (MLP, CNN, and GRU) developed for DrDoS detection. To the best of our knowledge, no existing study provides such a comprehensive, multi-architecture benchmark under a single experimental setup.
- We demonstrate that integrating the ONNX Runtime with hardware acceleration significantly enhances deep learning model efficiency. A 100-run evaluation demonstrates that CPU/GPU optimization reduces average inference latency, while dynamic quantization decreases the model memory footprint. These efficiency trends remain consistent across three structurally distinct baseline models evaluated under the identical deployment and optimization pipeline: a Multi-Layer Perceptron (MLP) for capturing global non-linear features, a 1D Convolutional Neural Network (1D-CNN) for spatial-feature extraction, and a Gated Recurrent Unit (GRU) network for temporal dependency capture.
- We provide extensive, multi-dimensional analysis, mapping training, and inference behavior across both CPU and GPU environments. By systematically varying batch sizes, we identify the trade-off thresholds where hardware acceleration offsets overhead, serving as a practical deployment guide for real-time systems.

The remainder of this paper is organized as follows: Section 2 reviews related studies and methodologies in the literature. Section 3 provides a theoretical classification of DrDoS attack types and their characteristics. Section 4 details the experimental setup, dataset preprocessing, and feature selection procedures. The performance metrics utilized in the study are presented in Section 5. Section 6 introduces the tested deep learning models and their architectures, while Section 7 explains the ONNX conversion and quantization

optimization processes. Section 8 presents experimental results. Finally, Section 9 concludes with a discussion and concluding remarks.

2. Literature Review

While existing studies in the literature provide significant contributions to DrDoS attack detection, they often overlook the real-time performance requirements essential for high-volume traffic environments. This section examines studies that address this gap by evaluating model size and inference speed alongside anomaly detection performance.

In a study conducted by R. Doriguzzi-Corin et al. (2020) [7], a lightweight and practical deep learning model titled LUCID (Lightweight Usable CNN in DDoS Detection) was proposed. This model utilizes a Convolutional Neural Network (CNN) architecture to classify traffic flows as either malicious or benign by processing them within time windows. Evaluated on the UNB201X dataset, LUCID achieved an F1 score of 0.9946. The authors reported that the model operates approximately forty times faster than comparable models providing similar accuracy, with a processing capacity of 55,000 samples per second. When evaluated on low-power edge devices such as the NVIDIA Jetson TX2, the system demonstrated high efficiency and low resource consumption, proving its suitability for real-time intrusion detection.

Ibrahim et al. (2025) [8] analyzed machine learning approaches based on feature optimization to enhance DDoS detection processes using the CICDDoS2019 dataset. By reducing the feature set from 24 to 10 variables, the researchers achieved an accuracy of 99.85% with both Random Forest and Decision Tree models. The study reported inference times of four milliseconds (ms) for the Decision Tree and 240 ms for the Random Forest model.

Hernandez et al. (2025) [9] analyzed GPU-accelerated deep learning models using multivariate time series data for real-time DDoS detection in high-speed networks. The study found that the TCN architecture maintained consistent performance regardless of window size, achieving an F1 score of 99.86% with a low average inference time ranging between 20 and 55 microseconds.

Sever and Öğüt (2021) [10] conducted a comparative analysis of the inference latency of deep learning models converted to the ONNX format across different execution environments. In their research, a two-layer neural network developed in PyTorch was converted to Torch Script and ONNX formats, then executed on CPU and GPU via a C++ interface. Results indicated that the ONNX + TensorRT (GPU) combination provided the lowest latency at 390 microseconds. The study concludes that the combination of ONNX and TensorRT offers an effective infrastructure for high-speed AI inference on edge devices.

Alemayehu et al. (2026) [11] proposed a Deep Reinforcement Learning-based PPO agent for real-time DDoS detection in Industrial IoT (IIoT) environments. The policies were converted to the ONNX format, facilitating a reduction in model size to 1–2 MB and an inference latency of less than 23 microseconds per sample, with detection accuracy of up to 99.3%.

Sharmila and Nagapadma (2023) [12] introduced a Quantized Autoencoder (QAE) model for network anomaly detection on resource-constrained IoT edge devices. The QAE-u8 variant achieved a 70.01% reduction in average memory usage, a 92.23% reduction in model size, and a 27.94% reduction in CPU utilization on the RT-IoT2022 dataset.

In a study by Khantouchi et al. (2024) [13], the Eye-Net model was proposed for DDoS detection in IoT environments. Using Quantization-Aware Training (QAT) with INT8 precision, the model achieved 99.99% accuracy and an F1 score of 99.99% on the CICDDoS2019 dataset.

Nkoom et al. (2026) [14] proposed a federated learning framework for real-time DDoS detection on Internet of Robotic Things (IoRT) devices. Using QAT with differential privacy-based clustering, the model size was compressed from 0.52 MB to 0.13 MB. Inference speed was increased 433-fold, reducing latency to below 150 microseconds.

Most studies focusing on DrDoS and DDoS detection in the literature aim to reduce inference times to the millisecond range through lightweight architectures (LUCID [7]) or feature optimization [8]. In contrast, the ONNX-based MLP model presented in this study achieves an inference time of 89.5 nanoseconds on the GPU, significantly outperforming the performance metrics of Nkoom et al. [14].

Regarding model compression and memory efficiency, Sharmila and Nagapadma [12] achieved a 92.23% reduction in size using a QAE. In this study, the application of Post-Training Quantization (PTQ) and ONNX optimization achieved up to 89% reduction in file size without requiring complex retraining processes. Furthermore, the proposed system can process over ten million inferences per second at a batch size of 4096, compared to the throughput of 55,000 samples per second achieved by the LUCID [7] model.

3. Classification of DrDoS Attacks

DNS Distributed Reflection and Amplification (DrDoS) attacks represent a significant security threat that leverages the DNS infrastructure to cause service disruption and network congestion. By exploiting open DNS resolvers and the inherent discrepancies in response sizes within the DNS protocol, these attacks aim to overwhelm targeted systems with massive volumes of spoofed traffic [15]. DrDoS DNS attacks can be categorized into four primary classifications based on [16]:

- DNS Record Type Employed
- Amplification Factor
- Targeted DNS Infrastructure
- Attack Vector

3.1. Classification Based on the DNS Record Type Employed

Attackers utilize various DNS record types to maximize the reflection and amplification effects. While ANY queries generate high-volume traffic by producing large responses that include all available DNS records, attacks based on TXT records allow for massive responses to be derived from small queries due to the length of text-based entries. Similarly, attacks leveraging MX and NS records result in large responses containing lists of multiple mail servers or authoritative name servers. Furthermore, AAAA records increase the response size due to the length of IPv6 addresses, while CNAME records can indirectly escalate the traffic load by triggering additional recursive queries.

3.2. Classification Based on the Amplification Factor

The potential impact of an attack is determined by the amplification factor, which denotes the ratio of the response size received to the size of the submitted query. Although low amplification factor attacks generate responses with lower volumes, their impact is compounded by a high frequency of queries. In medium amplification factor attacks, the responses are larger, making the attack's impact more pronounced. Finally, high amplification factor attacks enable attackers to elicit exceptionally large responses using minimal query data.

3.3. Classification Based on the Targeted DNS Infrastructure

DrDoS attacks can target various components of the DNS infrastructure within a network. Based on these targets, attack types are classified as Recursive Resolver attacks,

which utilize open resolvers as reflectors. Alternatively, authoritative name servers may be targeted directly to induce resource exhaustion. Vulnerabilities in DNSSEC are also frequently exploited, as its validation processes require intensive computational resources.

3.4. Classification Based on the Attack Vector

The diversity of techniques employed by attackers allows for classification by attack vector. Single-vector DrDoS attacks are straightforward operations involving a single type of query. Conversely, multi-vector DrDoS attacks, which combine different query types, exhibit a more complex and high-volume structure.

4. Experimental Setup

In this study, three deep learning models were developed and trained to detect DNS-based reflection attacks (DrDoS), a specific variant of Distributed Denial-of-Service (DDoS) attacks. For the model training phase, the DrDoS subset of the CIC-DDoS-2019 dataset [17] was utilized. The implementation was executed in a Google Colab A100 GPU environment. To ensure reproducibility of the experimental results, the dataset train/test partitioning, neural network weight initializations and the exact software environment, including library dependencies, package versions and hardware profiling parameters, are exhaustively cataloged in the Supplementary Materials. The systematic research workflow followed during this process is illustrated in Figure 1.

4.1. Dataset Selection and Characteristics

The CIC-DDoS-2019 dataset is a comprehensive repository derived from real-world network traffic, encompassing diverse types of DDoS attacks. Since this study focuses specifically on DNS-based reflection and amplification (DrDoS) attacks, the relevant subset of the dataset was employed. An analysis of the class distribution in the original DrDoS_DNS file revealed 3402 Benign traffic samples and 5,071,011 DrDoS attack samples. This substantial class imbalance represents a critical factor that must be addressed during the model training process.

4.2. Data Pre-Processing Stage

Data preprocessing is vital to model success, transforming raw network captures into a machine-learning-ready format. The target day of the DrDoS attack in the CIC-DDoS-2019 dataset contains a massive volume of attack packets but a severely limited representation of normal traffic. To establish a robust background baseline of normal network behavior, we gathered and consolidated all available benign packets across the other recording days of the official CIC-DDoS-2019 dataset, yielding a total of 113,828 benign samples. Despite this consolidation, the ratio of benign to DrDoS packets remained heavily imbalanced at approximately 1:44. To prevent biased learning while fully retaining our valuable normal traffic, down-sampling was applied exclusively to the majority DrDoS attack class. All 113,828 benign samples were preserved, while the massive DrDoS attack pool was uniformly and randomly subsampled to 227,656 samples. This established a controlled 1:2 (Benign to DrDoS) ratio, resulting in a balanced benchmark dataset of 341,484 packets. Because this uniform down-sampling of the homogeneous attack class was executed without parameterized transformations or cross-sample dependencies, it does not introduce any data leakage across the subsequent train–test partition.

During the analysis of the features, the cardinality of each column was examined. Identifying fields such as ‘Flow ID’, ‘Source IP’, ‘Destination IP’, and ‘Protocol’ that exhibit low diversity and could negatively impact generalization were removed, along with the ‘SimilarHTTP’ column, which could not be converted into a numerical format.

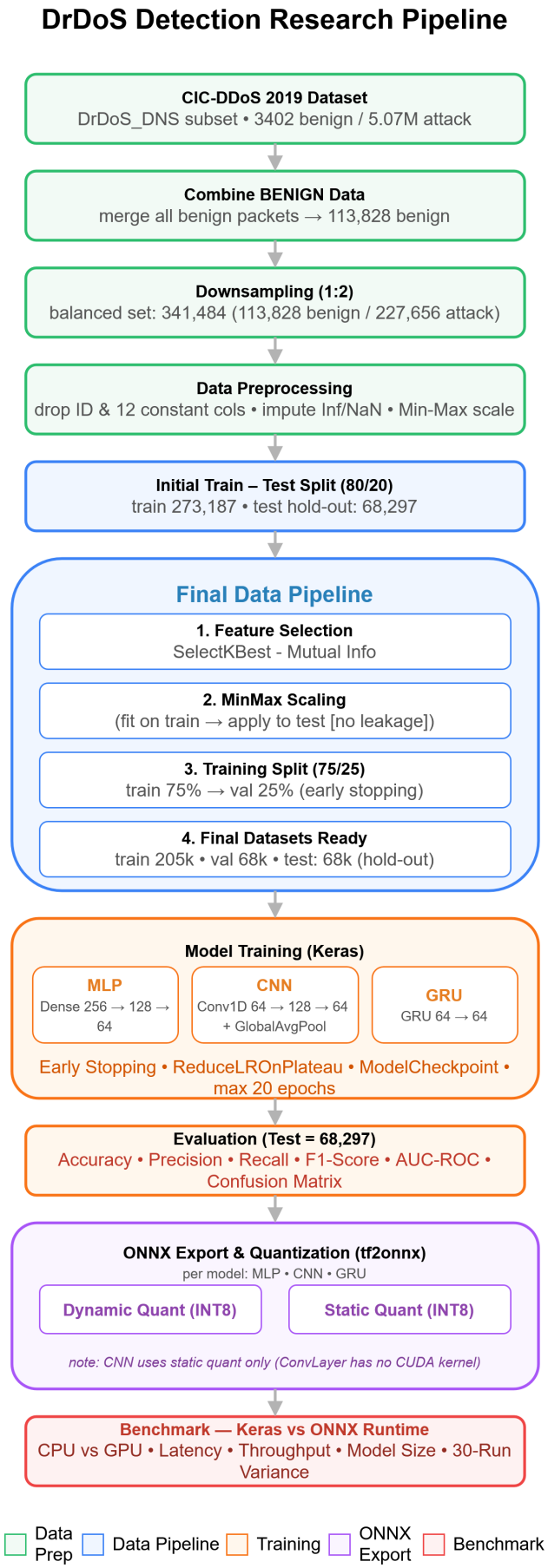

Figure 1. Research Workflow.

In the data cleaning process, missing values were managed using SimpleImputer by replacing them with the mean values of the respective columns, while infinite values were replaced with NaN. Additionally, twelve columns containing constant values with zero variance were eliminated (e.g., Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, FIN Flag Count, PSH Flag Count, ECE Flag Count, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, and Bwd Avg Bulk Rate). These steps enabled the model to produce more stable and accurate results.

Data scaling is a critical step to prevent machine learning models from being adversely affected by large discrepancies in the range of feature values. It is essential to perform the scaling stage after the data splitting process to avoid data leakage. If scaling is conducted on the entire dataset, the statistics of the training set would influence the test and validation sets, potentially leading to a biased evaluation of real-world performance [18]. In this study, the dataset was first partitioned into 80% training and 20% test sets; subsequently, Min-Max scaling was applied using only the statistics derived from the training set. The learned scaling parameters were then applied to the test set, thereby preventing leakage. Min-Max scaling was preferred as it maps input data to a fixed range, providing the limited value interval required by the quantization algorithm.

4.3. Feature Selection

A filter-based feature selection method was implemented to enhance model performance, reduce computational complexity, and improve interpretability. In this study, the Mutual Information method was chosen for its capacity to capture non-linear relationships between variables. In machine learning-based systems developed for DrDoS detection, accurate feature selection plays a critical role in ensuring model generalizability and consistent performance across diverse network environments. During the feature selection process, it was observed that rate-based features such as Flow Bytes/s, Flow Packets/s, and Fwd Packets/s achieved high scores [19].

To prevent data leakage, the SelectKBest algorithm was fitted exclusively on the training data, and the learned transformation was subsequently applied to the test data. To analyze processing delay, specifically inference delay, we selected the maximum feature set based on Mutual Information scores. These features are presented in Table 1.

Table 1. Features Ranked by Mutual Information Score.

Category	Features
Packet Size Features	Min Packet Length, Fwd Packet Length Min, Avg Fwd Segment Size, Fwd Packet Length Mean, Average Packet Size, Packet Length Mean, Fwd Packet Length Max,
Flow Statistics	Inbound, Flow Bytes/s, Flow Packets/s, Fwd Packets/s

5. Model Performance Metrics

The metrics utilized to evaluate the performance of a classification model are predicated on the correlation between the model's predictions and the ground truth labels. Within this framework, four fundamental measurement parameters are established:

True Positive (TP): Represents the count of instances that are positive (e.g., an attack) and are correctly identified as positive by the model.

True Negative (TN): Denotes the number of instances that are negative (e.g., normal behavior) and are accurately classified as negative by the model.

False Positive (FP): Refers to instances that are negative but are erroneously categorized as positive by the model.

False Negative (FN): Refers to instances that are positive but are erroneously categorized as negative by the model.

The following evaluation metrics are derived utilizing these four primary parameters:

5.1. Accuracy

This metric is defined as the ratio of correctly predicted instances to the total number of samples within the dataset.

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (1)$$

5.2. Precision

This metric indicates the proportion of instances classified as positive that are truly positive.

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (2)$$

5.3. Recall (Sensitivity)

This metric measures the proportion of actual positive instances that were correctly identified by the model.

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (3)$$

5.4. F1-Score

Defined as the harmonic mean of precision and recall, this metric provides a more robust performance measure, particularly in the context of imbalanced datasets.

$$\text{F1 Score} = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall}) \quad (4)$$

Evaluating these metrics in conjunction facilitates a comprehensive analysis of attack detection performance, moving beyond a singular focus on overall accuracy.

6. Used Deep Learning Models and Architectures

Following feature selection, we evaluated three structurally distinct deep learning models trained on the prepared balanced dataset:

- Multi-Layer Perceptron (MLP): A foundational feedforward baseline designed to capture non-linear relationships within aggregated, tabular statistical features without inherent spatial or temporal awareness.
- 1D Convolutional Neural Network (1D-CNN): A spatial feature extraction baseline to capture localized patterns and structural relationships.
- Gated Recurrent Unit (GRU) network: A lightweight sequential baseline designed to capture temporal and flow dependencies.

These architectures were selected for their capacity to learn non-linear relationships, making them highly effective for modeling complex, high-volume, and time-sensitive DDoS attacks, such as DrDoS. Prior literature supports the efficacy of these neural networks in detecting DDoS and DNS-based reflection attacks, noting that they yield superior accuracy and F1-scores compared to classical machine learning methods [20–22].

To facilitate model training and real-time monitoring, the dataset was partitioned using stratified sampling into a 60% training set, a 20% validation set, and a 20% test (hold-out) set. To improve generalization and prevent overfitting, Dropout and Layer Normalization layers were integrated after the hidden layers across these models. Dropout reduces over-reliance on specific features by randomly deactivating a set proportion of neuronal connections during training, while Layer Normalization stabilizes the training dynamics at each layer.

To analyze the performance differential between the Keras and ONNX implementations, throughput was measured across various batch sizes. The throughput gap becomes significant starting at a batch size of 1024. Because this trend remains consistent across Keras, ONNX, and quantized configurations for all evaluated models, a batch size of 1024 was adopted for this study.

6.1. Multi-Layer Perceptron Model

MLP is a feed-forward artificial neural network model characterized by one or more hidden layers situated between the input and output layers. As illustrated in Figure 2, the developed MLP model consists of an input layer, three hidden layers, and an output layer. The hidden layers comprise 256, 128, and 64 neurons, respectively, with each layer employing the ReLU activation function. Following the first two hidden layers, both Layer Normalization and a 30% Dropout rate were applied. A 20% Dropout rate was implemented after the third hidden layer. Finally, a Softmax activation function was utilized in the output layer to calculate class probabilities.

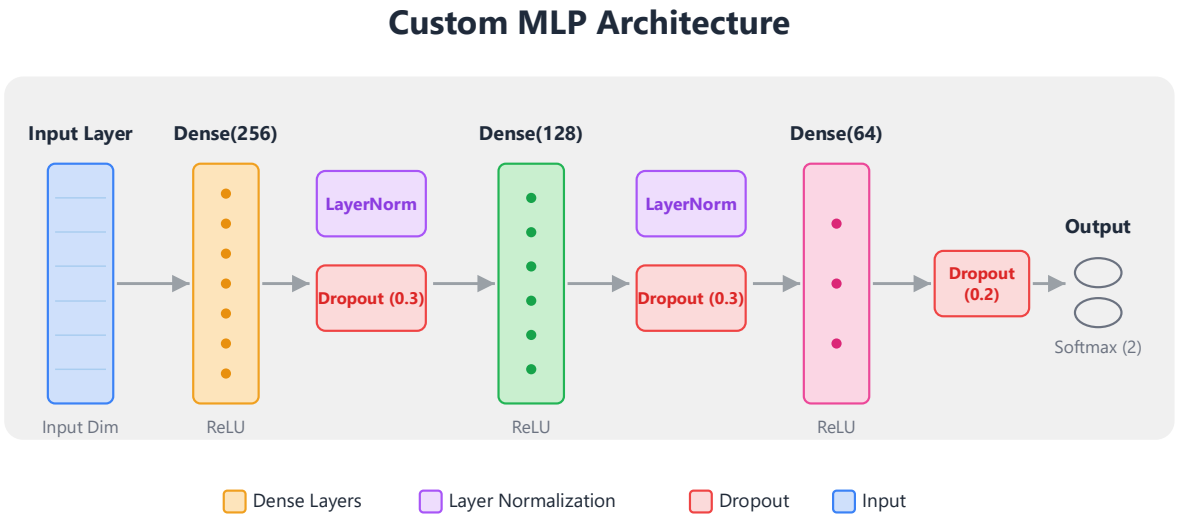

Figure 2. MLP Model Architecture.

The model was compiled using the Adam optimization algorithm and the sparse categorical cross-entropy loss function, with accuracy serving as the primary performance metric. The training process was supported by an early stopping mechanism monitored against the validation loss. Furthermore, ReduceLROnPlateau was implemented to decrease the learning rate upon stagnation of the validation loss, and ModelCheckpoint was utilized to record the optimal weights. The training phase was capped at a maximum of 20 epochs.

6.2. 1D Convolutional Neural Network (1D-CNN)

CNN is a deep learning architecture characterized by shared filter weights that slide across the input to detect recurring local structures with relatively few parameters. In this study, a one-dimensional (1D) variant was adopted to convolve the filters along the ordered feature vector of each network-flow sample.

As illustrated in Figure 3, the developed CNN model consists of an input layer, three stacked one-dimensional convolutional (Conv1D) layers, a Global Average Pooling layer, a fully connected layer, and an output layer. The Conv1D layers comprise 64, 128, and 64 filters with a kernel size of 3, respectively, with each layer employing the ReLU activation function. Following the first two convolutional layers, both Layer Normalization and a 30% Dropout rate were applied. A 20% Dropout rate was implemented after the third

convolutional layer. Next, a Global Average Pooling layer was utilized to condense the resulting feature maps. The condensed representation was passed to a fully connected layer of 64 neurons employing the ReLU activation function, followed by an additional 20% Dropout rate. Finally, a Softmax activation function was utilized in the output layer to calculate class probabilities.

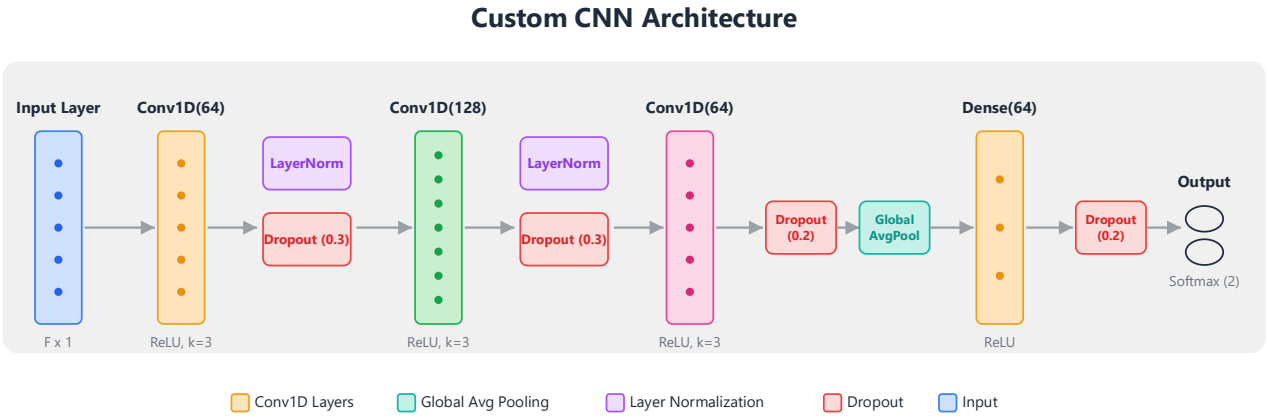

Figure 3. CNN Model Architecture.

6.3. Gated Recurrent Unit (GRU) Network

GRU is a recurrent neural network model designed to capture sequential dependencies via an internal hidden state regulated by an update gate and a reset gate, thereby mitigating the vanishing-gradient problem. In this study, the feature vector of each sample is presented to the network as a sequence to model the relationships among successive features.

As illustrated in Figure 4, the developed GRU model consists of an input layer, two stacked GRU layers, a fully connected layer, and an output layer. The GRU layers comprise 64 units each, with the first layer returning its complete output sequence and the second layer returning only the final hidden state as a fixed-length summary vector. This vector is passed to a fully connected layer of 64 neurons employing the ReLU activation function. Following the first GRU layer, both Layer Normalization and a 30% Dropout rate were applied. A 20% Dropout rate was implemented after both the second GRU layer and the fully connected layer. Finally, a Softmax activation function was utilized in the output layer to calculate class probabilities.

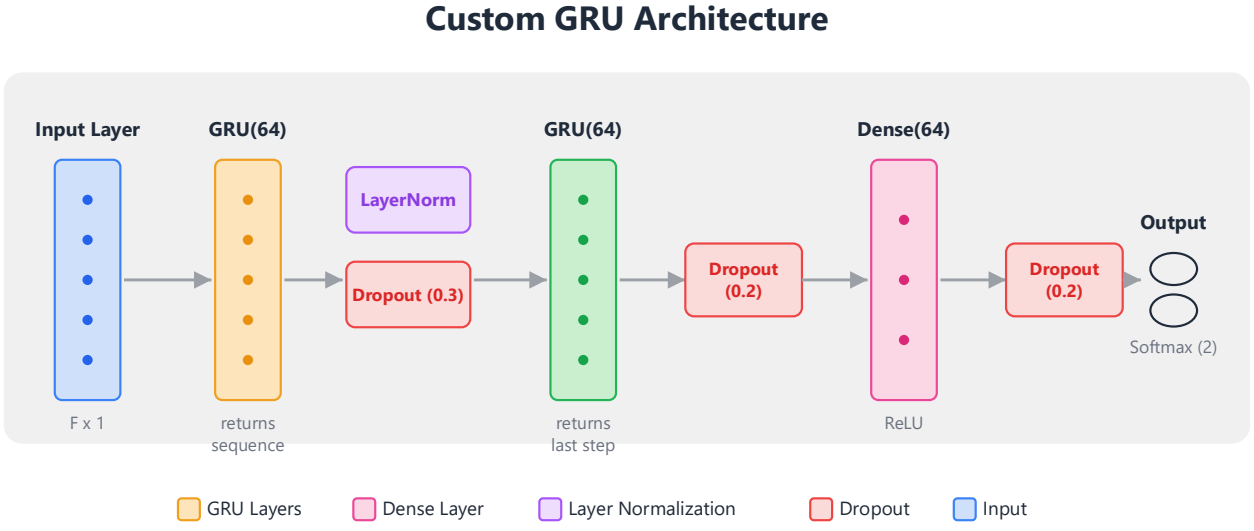

Figure 4. GRU Model Architecture.

7. ONNX and Quantization

To improve operational efficiency, all trained TensorFlow/Keras models were converted to the Open Neural Network Exchange (ONNX) format. To ensure compatibility, the models were first encapsulated within the Functional API and structured as standard tf.keras.Model objects. Applying quantization to the converted ONNX models significantly reduced both memory footprint and inference latency. Conversely, auxiliary optimization steps, specifically model simplification via ONNX Simplifier v0.6.5 and independent graph-level optimization, provided no measurable improvements in this study.

Both dynamic and static quantization configurations were evaluated. Under dynamic quantization, only the model weights were converted to INT8, while activations remained in FP32 format at runtime. In static quantization, both weights and activations were converted to INT8, enabling the use of optimized operators such as QLinearMatMul for hardware-accelerated inference. However, static quantization led to a marginally higher degradation in accuracy than the dynamic method.

Furthermore, execution compatibility varied across hardware platforms. While the multilayer perceptron (MLP) and gated recurrent unit (GRU) architectures supported dynamic quantization on both the CPU and GPU, the convolutional neural network (CNN) could not be evaluated under dynamic quantization on the GPU. This limitation arises because dynamic quantization maps fully connected and recurrent layers to the MatMulInteger operator, which is supported by both CPU and GPU execution providers.

For convolutional layers, however, dynamic quantization generates the ConvInteger operator, which ONNX Runtime (v1.27.0) implements exclusively in its CPU execution provider, lacking a corresponding CUDA kernel. GPU-accelerated INT8 convolution requires a static Quantize–DeQuantize (QDQ) pipeline with vendor-specific kernels (cuDNN/TensorRT), as dynamic activation scaling is incompatible with these optimized kernels. Consequently, a dynamically quantized CNN cannot be executed on the GPU. This restriction aligns with ONNX Runtime guidelines, which recommend static quantization for convolutional models and dynamic quantization for MatMul-based models. Therefore, CNN's INT8 GPU performance was evaluated using static quantization only.

8. Experimental Results

This section presents experimental results evaluating ONNX conversion and both static and dynamic quantization. This evaluation is applied to three DrDoS detection architectures: MLP, CNN, and GRU. We measure training and inference performance across CPU and GPU environments by systematically varying batch sizes. Although individual model metrics differ, their performance trends are highly consistent. For clarity, we analyze these shared trends using the MLP model as a representative case. The Supplementary Materials contains detailed results for the CNN and GRU models.

8.1. Reference Model Performance Test Results

To evaluate model efficacy, testing was performed on a down-sampled, balanced dataset. Table 2 presents the performance metrics for all three developed architectures: the Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Gated Recurrent Unit (GRU). To ensure statistical robustness and consistency, independent training iterations were executed for each architecture, with an optimal decision threshold uniquely determined during each run.

As presented in Table 2, the Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Gated Recurrent Unit (GRU) models all record classification metrics exceeding 0.9970 and Area Under the Curve (AUC) values of at least 0.9997. The CNN architecture demonstrates the lowest training loss alongside the highest training accuracy

and test recall. Conversely, the MLP architecture achieves the highest test accuracy and test precision, while obtaining identical performance to the CNN in both test F1-score and AUC. In contrast, the GRU model consistently yields the lowest performance metrics across all evaluation criteria, resulting in the highest training loss and the lowest test accuracy. Notably, for all three architectures, both the validation and test accuracies exceed the respective training accuracies, indicating stable model generalization.

Table 2. Comparison of Three Deep Learning Models Metrics (Keras).

Metric	MLP	CNN	GRU
Train Accuracy	0.9976	0.9980	0.9970
Validation Accuracy	0.9983	0.9984	0.9977
Train Loss	0.0108	0.0093	0.0129
Test Accuracy	0.9985	0.9984	0.9977
Test Precision	0.9993	0.9991	0.9987
Test Recall	0.9984	0.9985	0.9979
Test F1-score	0.9988	0.9988	0.9983
Test AUC	0.9999	0.9999	0.9997

Figure 5 illustrates a progressive decrease in both training and validation loss across epochs for the MLP model. Furthermore, Figure 6 demonstrates that the model's ROC curve consistently trends towards the upper-left corner.

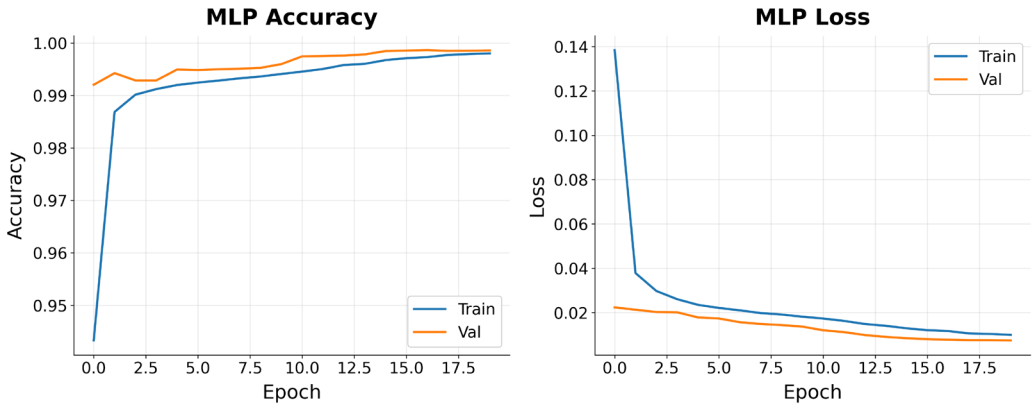

Figure 5. Validation-Training Loss and Accuracy Graphs for the MLP Model.

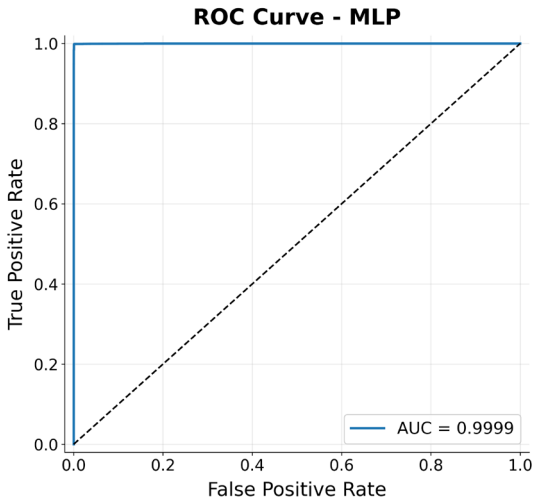

Figure 6. Test ROC Curve for the MLP Model.

The classification distribution for the MLP model is detailed in the confusion matrix in Figure 7. Within the test dataset, 22,739 of the 22,766 benign traffic instances were correctly classified as True Negatives (TN), while twenty-seven instances were mislabeled as False Positives (FP). Regarding the Attack class, 45,469 out of 45,531 samples were identified as True Positives (TP), with only sixty-two samples recorded as False Negatives (FN).

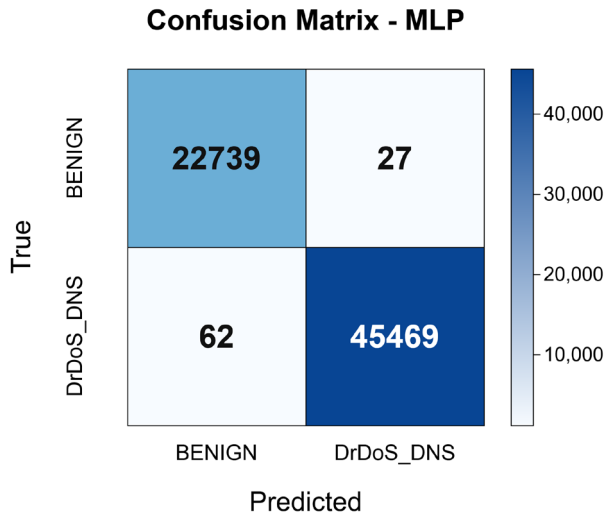

Figure 7. Confusion Matrix for the MLP Model.

8.2. ONNX Conversion and Inference Performance Comparison

The impact of converting the Keras-trained models to the ONNX format on inference performance was investigated. Baseline Keras models were compared against ONNX models across various precision levels (FP32, dynamic quantization, and static quantization) on both CPU and GPU hardware. Table 3 details the average inference latency per sample and the model size for the MLP under these configurations. Because the CNN and GRU models exhibited similar general performance trends, their corresponding results are provided in the Supplementary Materials, and the remainder of this analysis uses the MLP model as the primary reference.

Table 3. MLP Model Comparison Summary.

Model	Environment	Inference Time (µs)	File Size (MB)
Keras	CPU	4.2715	0.592
Keras	GPU	2.5726	0.592
ONNX	CPU	0.7476	0.1879
ONNX	GPU	0.0895	0.1879
Dynamic Quantized ONNX	CPU	0.6517	0.063
Dynamic Quantized ONNX	GPU	0.8989	0.063
Static Quantized ONNX	CPU	2.3041	0.0869
Static Quantized ONNX	GPU	0.2056	0.0869

The conversion and quantization processes did not result in any degradation of the model's classification capability. Experimental results showed that Accuracy, Precision, Recall, F1-Score and AUC remained stable at approximately 0.99 across all formats. These findings demonstrate that the model's predictive efficacy was preserved during the compression process.

In contrast, models converted to the ONNX format achieved a significant reduction in inference latency compared to the original Keras model. The top-performing ONNX-GPU configuration for the MLP model (89.5 nanoseconds) executed inference approximately 47 times faster than the Keras-CPU configuration (4.2715 microseconds). The initial Keras

model size of 0.592 MB decreased to 0.063 MB under Dynamic Quantization, representing a reduction of over 89%.

8.3. Batch Size Selection

Figure 8 illustrates the variation in the total training duration required for 20 epochs, categorized by batch size and hardware type. The data obtained indicates that an increase in batch size leads to a reduction in training time across both hardware configurations.

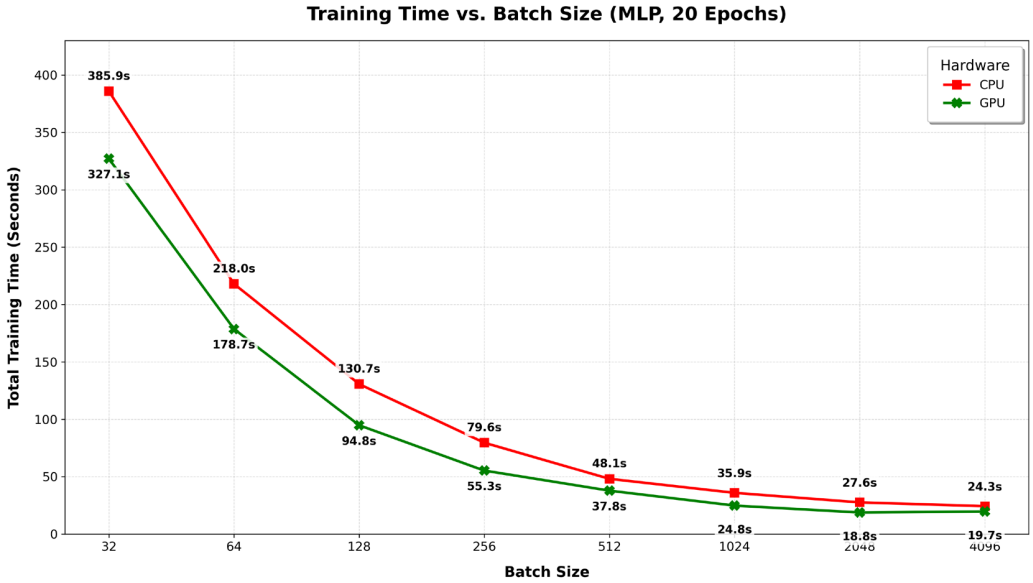

Figure 8. Training Time vs. Batch Size.

- On the CPU, the training duration, which was measured at 385.9 s with a batch size of thirty-two, decreased by 93.7% to 24.3 s when the batch size was scaled to 4096.
- Similarly, the initial duration of 327.1 s on the GPU dropped to 19.7 s at a batch size of 4096.

Figure 9 illustrates the throughput values representing the number of inferences per second achieved by various model formats and hardware configurations. According to the analysis results:

- The highest throughput was attained by the standard ONNX-GPU configuration. This setup exhibited an exponential performance increase as the batch size expanded, exceeding 1.0×10^7 inferences per second at a batch size of 4096.
- On the GPU, the statically quantized ONNX model demonstrated the second-highest performance. However, it was observed to fall behind the other ONNX variants at smaller batch sizes.
- The throughput of Keras-based models (both CPU and GPU) remained limited despite the increase in batch size, consistently performing below models utilizing the ONNX runtime.

Figure 10 presents the results of the latency analysis, which defines the average time elapsed to process a single instance. This metric holds critical significance, especially for real-time applications.

- At smaller batch sizes (32 and 64), a substantial performance disparity was recorded between the model formats. With a batch size of thirty-two, the average latency of Keras models was approximately 350 microseconds, whereas all ONNX variants (ONNX Standard CPU/GPU, ONNX Dynamic Quant CPU/GPU, ONNX Static Quant CPU/GPU) maintained similar values below 50 microseconds.

- As the batch size increased to 256 and beyond, the latency of Keras models declined owing to augmented parallel processing capabilities, eventually converging toward the performance levels of the ONNX-based models.

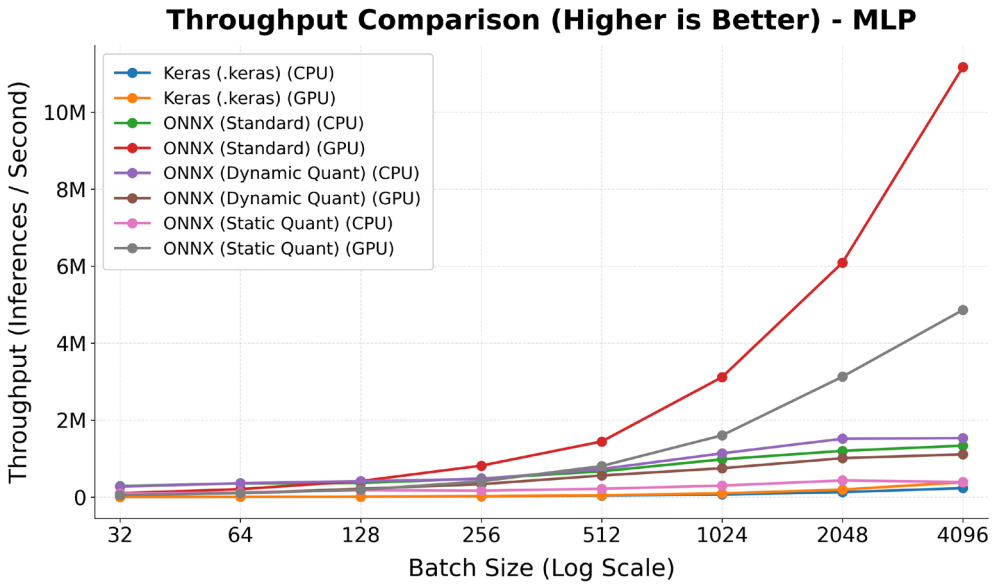

Figure 9. Throughput vs. Batch Size.

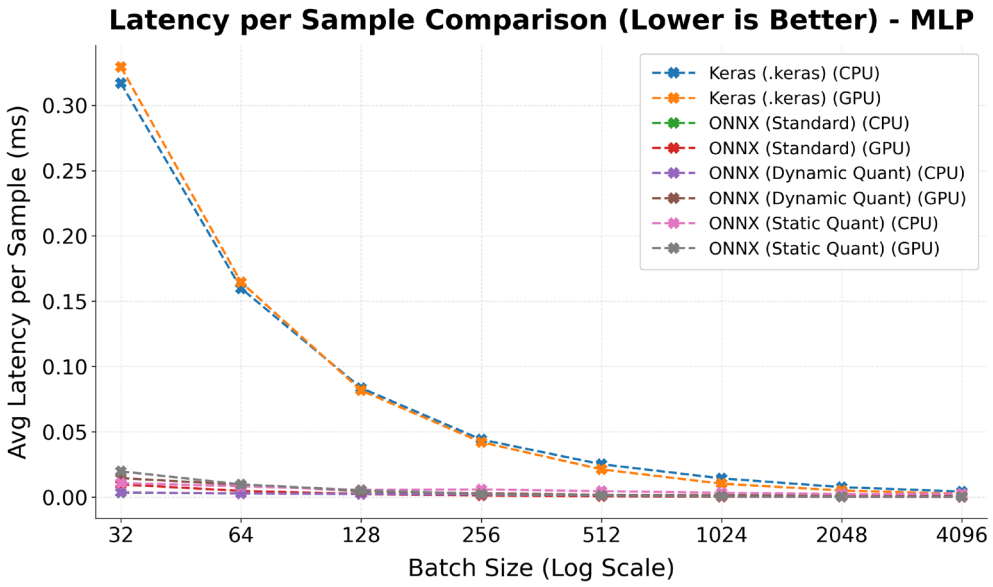

Figure 10. Latency vs. Batch Size.

9. Discussion and Conclusions

Real-time detection of DrDoS attacks poses severe network security threats and generates devastating impacts through techniques such as DNS Amplification. While traditional detection systems provide high accuracy rates, their substantial computational costs and sluggish inference times have been identified as primary performance bottlenecks that hinder instantaneous response. This research presents a comprehensive, systematic comparison of different compilers (ONNX) and quantization configurations across multiple architectures (MLP, CNN, GRU) on a unified reference platform. Throughout the study, we systematically analyzed not only model accuracy but also runtime efficiency across various hardware architectures (CPU/GPU) and batch sizes.

To evaluate our optimized models under conditions that mirror actual high-throughput network operations, we focus on batch-averaged latency configurations rather than single-sample B = 1 processing. In realistic, high-speed line-rate environments, packet-processing engines do not typically execute tasks in a strictly sequential, single-packet blocking mode. Modern frameworks like DPDK or SmartNIC architectures natively leverage burst-based queueing (processing packet batches, e.g., $B \geq 32$) to minimize CPU interrupt overheads and optimize packet-processing rings. Consequently, evaluating batch-average execution time represents a highly practical operational paradigm for network security engineers designing systems to handle bursty, high-throughput line-rate traffic.

Experimental data indicate that standard ONNX conversion preserved the core classification metrics, including accuracy, precision, recall, and F1-score, yielding performance identical to the original Keras model. Under dynamic quantization, classification metrics experienced a statistically negligible decline in both CPU and GPU environments. These results demonstrate that the model maintains its performance after being reduced to 8-bit integer (INT8) precision, resulting in almost no drop in accuracy across both CPU and GPU environments.

The analysis of training durations reveals a clear inverse relationship between batch size and training time. At a batch size of thirty-two, the training duration was measured at 385.9 s on the CPU and 327.1 s on the GPU, representing only a marginal improvement. This performance convergence at small batch sizes can be explained by Kernel Launch Overhead. Consequently, at a batch size of 4096, training durations dropped to 24.3 s on the CPU and 19.7 s on the GPU. Although both CNN and GRU models exhibit a consistent performance trend, the execution disparity between the CPU and GPU is more pronounced in the CNN model than in the alternative architectures.

To evaluate model efficiency in real-time applications, the impact of varying batch sizes on throughput was analyzed across different architectural and execution configurations. Architectural compatibility varied during optimization; although the multilayer perceptron (MLP) and gated recurrent unit (GRU) architectures supported dynamic quantization on both the CPU and GPU, the convolutional neural network (CNN) could not be evaluated under dynamic quantization on the GPU. This constraint arises because dynamic activation scaling is incompatible with these optimized GPU kernels, preventing dynamically quantized CNNs from executing on the hardware.

Regarding throughput performance, small batch sizes (32–128) consistently resulted in low processing rates across all models due to the dominance of data transfer overhead. Conversely, increasing the batch size to 512 and above induced a distinct acceleration in GPU throughput, particularly for the ONNX GPU and ONNX Static Quant GPU configurations. However, this throughput scaling for the CNN model decelerated near a batch size of 2048, which is likely clear evidence of GPU on-chip or cache memory limitations. In contrast, the remaining evaluation methods maintained steady and comparable throughput across all models between batch sizes of 32 and 4096, demonstrating that batch size modifications have a negligible impact on their performance.

Ultimately, the combination of ONNX Runtime and the CUDAExecutionProvider (GPU) achieved peak performance, delivering over ten million inferences per second at a batch size of 4096. These findings underscore that leveraging large batch sizes alongside GPU-based acceleration is critical to mitigating processing bottlenecks in high-traffic network environments.

The impact of different batch sizes on the average latency per sample was also examined. As the batch size increases, the parallel processing capacity of the GPU allows fixed costs to be distributed across thousands of samples, resulting in amortization. ONNX

models, especially the static quantized versions, exhibited a lower and more stable latency profile even at small batch sizes.

While the optimization pipelines evaluated in this study demonstrate significant performance acceleration, it is important to establish that all absolute execution times and latency ceilings documented in this work are fundamentally hardware-dependent. The benchmarking suite was executed on an enterprise-grade computing stack comprising an Intel Xeon processor and an NVIDIA A100 Tensor Core GPU. Deploying these compiled and quantized architectures on alternative execution environments such as commodity host CPUs, standard user workstations, or resource-constrained edge gateways will yield higher absolute execution latencies. However, the relative speedup trends are expected to remain consistent across different computing platforms.

In conclusion, this research presents a systematic comparison of compilers (ONNX) and quantization configurations across multiple architectures (MLP, CNN, GRU) on a unified reference platform. The findings guide cybersecurity experts in designing operational systems and developing low-latency, scalable defense architectures.

To build upon these findings, our future work will focus on performing an end-to-end hardware-in-the-loop system evaluation, integrating our optimized models with kernel-level frameworks like eBPF/XDP and hardware-accelerated interfaces like DPDK to empirically measure overall system latency under high-bandwidth, live traffic conditions.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/sym18071187/s1. Figure S1: Validation-Training Loss and Accuracy Graphs for the CNN Model, Figure S2: Validation-Training Loss and Accuracy Graphs for the GRU Model, Figure S3: Test ROC Curve for the CNN Model, Figure S4: Test ROC Curve for the GRU Model, Figure S5: Confusion Matrix for the CNN Model, Figure S6: Confusion Matrix for the GRU Model, Figure S7: Training Time vs Batch Size (CNN), Figure S8: Training Time vs Batch Size (GRU), Figure S9: Throughput vs Batch Size (CNN), Figure S10: Throughput vs Batch Size (GRU), Figure S11: Latency vs Batch Size (CNN), Figure S12: Latency vs Batch Size (GRU), Table S1: CNN Model Comparison Summary, Table S2: GRU Model Comparison Summary, LIBRARY VERSIONS (reproducibility).

Author Contributions: Conceptualization, İ.Ö.; methodology, İ.Ö.; investigation, S.E. and A.G.; validation, S.E. and A.G.; software, S.E. and A.G.; writing—original draft preparation, S.E., A.G. and Ö.Ö.; writing—review and editing, İ.Ö.; supervision, İ.Ö. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Doruk İletişim ve Otomasyon Sanayi ve Ticaret A.Ş. (DORUKNET) under funding number DRK.BGDT.DDOS.001.

Data Availability Statement: The original contributions presented in this study are included in the article/Supplementary Materials. Further inquiries can be directed to the corresponding author.

Acknowledgments: This material is based upon work supported by Doruk İletişim ve Otomasyon Sanayi ve Ticaret A.Ş. (DORUKNET). The authors gratefully acknowledge this support and take responsibility for the contents of this report. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Doruk İletişim ve Otomasyon Sanayi ve Ticaret A.Ş. (DORUKNET).

Conflicts of Interest: The authors declare that this study received funding from Doruk İletişim ve Otomasyon Sanayi ve Ticaret A.Ş. (DORUKNET). The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article, or the decision to submit it for publication.

References

1. Shohan, N.J.; Tanbhir, G.; Elahi, F.; Ullah, A.; Sakib, N. Enhancing network security: A hybrid approach for detection and mitigation of distributed denial-of-service attacks using machine learning. In *Proceedings of the International Conference on Advanced Network Technologies and Intelligent Computing*; Springer: Cham, Switzerland, 2023; pp. 81–95.
2. Dummer, S.H.; Rath, S. A Retrospective on DDoS Trends in 2023 and Actionable Strategies for 2024. 2024. Available online: https://www.akamai.com/blog/security/a-retrospective-on-ddos-trends-in-2023 (accessed on 5 January 2025).
3. Yoachimik, O. Cloudflare DDoS Threat Report for 2022 Q4. 2023. Available online: https://blog.cloudflare.com/ddos-threat-report-2022-q4 (accessed on 20 January 2023).
4. Jakkani, A.K. Real-time network traffic analysis and anomaly detection to enhance network security and performance: Machine learning approaches. *J. Electron. Comput. Netw. Appl. Math.* **2024**, *4*, 32–44. [CrossRef]
5. Jain, A.; Bhattacharya, S.; Masuda, M.; Sharma, V.; Wang, Y. Efficient execution of quantized deep learning models: A compiler approach. *arXiv* **2020**, arXiv:2006.10226.
6. Wu, H.; Judd, P.; Zhang, X.; Isaev, M.; Micikevicius, P. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv* **2020**, arXiv:2004.09602.
7. Doriguzzi-Corin, R.; Millar, S.; Scott-Hayward, S.; Martínez-Del-Rincón, J.; Siracusa, D. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 876–889. [CrossRef]
8. Ibrahim, A.J.; Répás, S.R.; Bektaş, N. Feature-Optimized Machine Learning Approaches for Enhanced DDoS Attack Detection and Mitigation. *Computers* **2025**, *14*, 472. [CrossRef]
9. Hernandez, D.V.; Lai, Y.-K.; Ignatius, H.T.N. Real-time DDoS detection in high-speed networks: A deep learning approach with multivariate time series. *Electronics* **2025**, *14*, 2673. [CrossRef]
10. Sever, M.; Öğüt, S. A performance study depending on execution times of various frameworks in machine learning inference. In *Proceedings of the 2021 15th Turkish National Software Engineering Symposium (UYMS)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5.
11. Alemayehu, M.; Ghanem, M.C.; Kheddar, H.; Dunsin, D.; Kerrache, C.A.; Rathee, G. Real-Time DDoS Detection in Industrial IoT Using Proximal Policy Optimisation and Deep Reinforcement Learning. *Preprints* **2026**. [CrossRef]
12. Sharmila, B.S.; Nagapadma, R. Quantized autoencoder (QAE) intrusion detection system for anomaly detection in resource-constrained IoT devices using RT-IoT2022 dataset. *Cybersecurity* **2023**, *6*, 41. [CrossRef]
13. Khantouchi, R.; Gasmi, I.; Ferrag, M.A. Eye-Net: A Low-Complexity Distributed Denial of Service Attack-Detection System Based on Multilayer Perceptron. *J. Sens. Actuator Netw.* **2024**, *13*, 45. [CrossRef]
14. Nkoom, M.; Commey, D.; Alsenani, Y.; Hounsinou, S.G.; Crosby, G.V. Federated DDoS Detection with Clustered Quantization-Aware Training Models for IoRT. In *Proceedings of the 2026 IEEE 23rd Consumer Communications & Networking Conference (CCNC)*; IEEE: Piscataway, NJ, USA, 2026.
15. Özçelik, İ.; Brooks, R. *Distributed Denial of Service Attacks: Real-World Detection and Mitigation*; CRC Press: Boca Raton, FL, USA, 2020.
16. Anagnostopoulos, M.; Kambourakis, G.; Kopanos, P.; Louloudakis, G.; Gritzalis, S. DNS amplification attack revisited. *Comput. Secur.* **2013**, *39*, 475–485. [CrossRef]
17. Sharafaldin, I.; Lashkari, A.H.; Hakak, S.; Ghorbani, A.A. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *Proceedings of the 2019 International Carnahan Conference on Security Technology (ICCST)*; IEEE: Piscataway, NJ, USA, 2019.
18. Bouke, M.A.; Abdullah, A. An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability. *Expert Syst. Appl.* **2023**, *230*, 120715. [CrossRef]
19. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and other botnets. *Computer* **2017**, *50*, 80–84. [CrossRef]
20. Mekala, S.; Dasari, K.; Katta, D. DNS DDoS Amplification Attack Detection Using Multi-Layer Perceptron Classification Algorithm. In *Proceedings of the 2024 IEEE 3rd World Conference on Applied Intelligence and Computing (AIC)*; IEEE: Piscataway, NJ, USA, 2024; pp. 1355–1360.
21. Qazi, E.U.; Almorjan, A.; Zia, T. A one-dimensional convolutional neural network (1D-CNN) based deep learning system for network intrusion detection. *Appl. Sci.* **2022**, *12*, 7986. [CrossRef]
22. Alshdadi, A.A.; Almazroi, A.A.; Alsolami, E.; Ayub, N.; Lytras, M.D. Enhanced IoT security for DDOS attack detection: Split attention-based ResNeXt-GRU ensembler approach. *IEEE Access* **2024**, *12*, 112368–112380. [CrossRef]