

Article

Accelerating CNN Inference via In-Network Computing in Information-Centric Networking

Kaiwei Hu ^{1,2}, Haojiang Deng ^{1,2,*} and Botao Ma ^{1,2}

¹ National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; hukw@dsp.ac.cn (K.H.); mabt@dsp.ac.cn (B.M.)

² School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

* Correspondence: denghj@dsp.ac.cn

Abstract

Although Convolutional Neural Networks (CNNs) have achieved remarkable accuracy in intelligent tasks, their increasing complexity hinders low-latency execution. While edge computing mitigates the wide-area network delays typical of cloud-based inference, it remains constrained by limited computational resources when processing complex models under high concurrency. Collaborative inference has emerged as a promising paradigm to address these limitations; however, existing approaches often struggle with rigid routing, limited scalability, and inefficient resource utilization. In this paper, we propose a novel collaborative inference acceleration mechanism that integrates In-Network Computing (INC) within an Information-Centric Networking (ICN) framework. By leveraging the name-based resolution capability of ICN, our approach dynamically harnesses underutilized computational resources across distributed INC nodes, enabling flexible layer-wise offloading that transcends the limitations of static IP paths. Furthermore, a distributed decision-making and node-selection algorithm is designed to orchestrate CNN layer assignment based on real-time network conditions and node workloads. Extensive simulations on representative models demonstrate the effectiveness of the proposed method. Specifically, for the computationally intensive VGG16 model under high concurrency, the average task completion time is reduced by 43.3% and 60.2% relative to IP-based and Edge-Cloud baselines, respectively, with a load balancing fairness index maintained above 0.86.

Keywords: information-centric networking; in-network computing; collaborative inference; task offloading; distributed scheduling



Academic Editor: Maciej Ławryńczuk

Received: 5 January 2026

Revised: 1 February 2026

Accepted: 6 February 2026

Published: 11 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The proliferation of Convolutional Neural Networks (CNNs) has revolutionized fields such as computer vision and natural language processing by enabling the learning of hierarchical feature representations [1,2]. However, the remarkable performance of CNNs comes at the cost of escalating computational complexity. Modern architectures, scaling significantly in depth and width, require billions of multiply-accumulate (MAC) operations for a single forward pass [3]. This computational burden poses a severe challenge for resource-constrained Internet of Things (IoT) devices, making standalone local inference often infeasible for latency-sensitive applications like autonomous driving and augmented reality [4].

Traditional cloud-centric inference offloads these intensive tasks to centralized data centers. While this alleviates the computational load on end devices, it introduces significant communication latency and susceptibility to wide-area network variability [5]. To mitigate these issues, Edge and Fog computing paradigms have emerged, shifting computation closer to data sources to effectively minimize transmission latency and alleviate backbone network congestion. Nevertheless, a single edge node—often limited by power and thermal constraints—can easily become a bottleneck under high request concurrency or when processing complex models [6]. Consequently, collaborative inference has gained prominence, where a CNN model is partitioned and executed cooperatively across a hierarchy of devices, edge nodes, and the cloud [7,8].

Despite the progress in collaborative inference, existing approaches face significant architectural limitations. This is particularly evident in the context of Computing-Network Convergence (CNC), an emerging paradigm that aims to seamlessly integrate ubiquitous computing resources with network infrastructure [9–11]. However, realizing CNC is fundamentally hindered by the architecture of conventional TCP/IP networks. IP networks are inherently host-centric, designed to route packets to specific physical addresses rather than to abstract computing services. This rigid coupling of service identity with physical location prevents tasks from dynamically accessing underutilized In-Network Computing (INC) nodes located slightly off the default shortest path. Furthermore, existing IP-based schemes often rely on coarse-grained partitioning strategies [12–14], failing to exploit the fine-grained data contraction characteristic of CNNs. In addition to architectural constraints, effective task scheduling in dynamic network environments remains an open challenge. Many existing solutions rely on centralized controllers (e.g., SDN-based orchestration) to optimize task placement [15]. While theoretically optimal, such centralized schemes fundamentally rely on the control plane to gather global states and disseminate flow rules. This introduces non-negligible signaling latency (typically spanning multiple Round-Trip Times), creating a hysteresis effect where scheduling decisions lag behind real-time traffic bursts.

To address these challenges, we propose a novel framework that integrates INC with Information-Centric Networking (ICN). ICN naturally aligns with the requirements of CNC by decoupling service identity from physical location, addressing the precise limitation of host-centric IP routing identified above [16]. By leveraging the Name Mapping and Resolution System (NMRS) of ICN, our framework treats model inference as a named service, enabling the dynamic discovery of available computing resources, even those off the shortest path. This integration allows inference tasks to be steered flexibly towards underutilized nodes based on real-time resource availability. The main contributions of this paper are as follows:

- We design an INC-assisted collaborative inference architecture overlaid on an ICN infrastructure. By identifying decision points where intermediate data volume contracts, the mechanism opportunistically trades communication bandwidth for distributed computing power. It leverages name-based resolution to discover and utilize idle off-path nodes, thereby overcoming the bottlenecks of fixed-path routing.
- We propose a node selection algorithm, named Distributed Load and Network-aware Scheduling (DLNS). Unlike myopic greedy strategies, DLNS evaluates candidate nodes based on a “look-ahead” segment of layers. It jointly optimizes processing, queuing, and transmission latencies using locally maintained resource tables, enabling adaptive, real-time decision-making without the overhead of global synchronization.
- We conduct a comprehensive evaluation of the proposed strategy through extensive simulations. The results demonstrate that compared with baseline schemes, our approach achieves significant advantages in both end-to-end latency and load balancing.

These findings validate the capability of our architecture to effectively harness idle in-network resources for accelerating collaborative CNN inference, thereby avoiding processing hotspots and realizing efficient load balancing.

The rest of this paper is structured as follows. Section 2 surveys existing studies on collaborative CNN inference, in-network computing, and information-centric networking. Section 3 presents the proposed ICN-based collaborative inference framework, including the system architecture, and end-to-end workflow. Section 4 introduces the system model and formulates the optimization problem. Section 5 details the in-network collaborative inference mechanism. Section 6 presents the simulation setup and evaluates the performance of the proposed approach. Finally, Section 7 summarizes our findings and discusses directions for future work.

2. Related Work

In this section, we review collaborative CNN inference in resource-constrained environments and discuss In-Network Computing and Information-Centric Networking, which collectively serve as the foundation for our proposed framework. We highlight the specific limitations of existing approaches that motivate our design.

2.1. Collaborative Inference

Collaborative inference partitions CNN execution across multiple compute nodes to mitigate resource bottlenecks on single devices. Existing studies can be broadly categorized into vertical and horizontal collaboration based on the organizational hierarchy of the computing resources.

Vertical collaborative inference offloads computation along a hierarchy spanning cloud data centers, edge servers, and end devices. This paradigm typically involves partitioning models into a sequence of segments mapped to different tiers. Kang et al. introduced Neurosurgeon, a lightweight scheduler that automatically identifies optimal partition points between mobile devices and the cloud to minimize latency and energy consumption [17]. Extending this to dynamic environments, Hu et al. proposed DADS, which adaptively adjusts partitioning strategies based on real-time bandwidth and workload fluctuations [18]. Similarly, Huang et al. developed DeePar, a three-tier architecture that coordinates inference across devices, edge nodes, and the cloud using an online scheduling algorithm [19].

Horizontal collaborative inference distributes model partitions among peer devices within the same network tier (e.g., adjacent edge nodes) to exploit aggregate local resources. In [13], CNN models are decomposed into fine-grained sub-layer components, and a matching-based scheduling strategy is applied to parallelize execution. Xu et al. formulated the offloading decision as an assignment problem, proposing a distributed mechanism to balance computational loads among collaborating peers [20].

Beyond specific model partitioning strategies, recent research has also explored task offloading from the perspectives of mobility management and wireless transmission scheduling. For instance, Shi et al. addressed the challenge of user mobility by proposing a two-timescale online resource optimization framework that dynamically selects between service migration and task rerouting to minimize system cost [21]. Similarly, Yi et al. focused on the wireless access bottleneck, designing a joint computation offloading and transmission scheduling mechanism to effectively manage multi-user channel contention for delay-sensitive applications [22].

Despite these advancements, existing frameworks face several challenges. First, partitioning decisions are often constrained by coarse granularity (e.g., splitting models into only two or three segments), which limits flexibility in adapting to heterogeneous node

capabilities. Second, task coordination frequently relies on centralized controllers, which may suffer from signaling overhead and slow response times during transient network congestion. Third, most designs operate over conventional IP networks, treating the network merely as a data transport pipe. They lack the visibility to perceive fine-grained link states (e.g., queue buildup) and the routing flexibility to steer tasks to underutilized nodes located off the default shortest path.

2.2. In-Network Computing

INC has evolved from simple packet processing to performing complex logic within network elements. With the advent of programmable data planes and hardware acceleration (e.g., FPGA-based SmartNICs), INC enables intermediate nodes (routers and switches) to execute computational tasks, thereby reducing response latency and alleviating back-haul traffic.

Recent works have explored INC for AI tasks. Hu et al. presented an architecture that distributes computational functions to host nodes equipped with virtualized execution environments [23]. Specifically for CNNs, Spina et al. proposed splitting VGG-16 models into convolutional and prediction blocks deployed across INC nodes [24]. This approach demonstrated significant reductions in network load and end-device energy consumption.

However, most INC-based inference schemes are inherently “on-path”, meaning computation is restricted to nodes strictly along the pre-determined routing path. In IP-based networks, if the shortest path becomes congested or its nodes are overloaded, the system lacks the mechanism to dynamically detour computation to idle neighbors (off-path nodes). Furthermore, existing solutions often rely on static deployment or offline profiling, lacking the runtime adaptability required for bursty inference traffic.

2.3. Information-Centric Networking

In INC scenarios, dynamic task scheduling across intermediate nodes is essential. Conventional host-centric TCP/IP architectures primarily treat the network as a passive transport pipeline, and their reliance on address-based routing limits the flexibility required for on-path task allocation in INC. The result can be higher transmission overhead, inefficient resource utilization, and constrained scalability. In contrast, ICN decouples content from physical location through name-based addressing and late binding. This architectural shift improves adaptability and scalability, making ICN well suited to distributing computational tasks efficiently within INC environments.

ICN architectures can be broadly classified by their routing and forwarding models. The first category comprises name-based routing designs such as Named Data Networking (NDN) [25] and Content-Centric Networking (CCN) [26], which use hierarchical names and integrate name resolution with packet forwarding. These designs are effective but exhibit limited compatibility with existing IP-based infrastructures. The second category adopts flat naming with a separate name-resolution system, such as a Name Mapping and Resolution System, to associate content identifiers with routable addresses. This approach offers a pragmatic balance between routing flexibility and scalability, as exemplified by MobilityFirst [27], NetInf [28], and SEANet [29]. Accordingly, our proposed mechanism builds upon this independent resolution architecture. Moreover, many ICN architectures support stateful forwarding, allowing intermediate nodes to maintain per-content state and adjust paths according to real-time network conditions. These capabilities strengthen the decision-making and collaborative functions of INC nodes and position ICN as a promising enabler for in-network computing.

2.4. Summary

Existing solutions generally treat computing and networking as separate optimization problems or are confined by the rigidity of IP routing. To address these gaps, this paper proposes a unified framework that integrates INC capabilities into an ICN substrate. Unlike prior works that rely on static IP paths or centralized scheduling, our approach leverages ICN's name resolution to expose off-path computing resources and employs a distributed, load-aware scheduling strategy. This design enables fine-grained, hop-by-hop decision-making, allowing the network to adaptively trade communication cost for computation gain in dynamic environments.

3. System Design

This section first analyzes the data contraction characteristics of CNNs and the resolution-based ICN mechanism. It subsequently details the proposed collaborative inference architecture, describing the system components and the hop-by-hop task execution workflow.

3.1. Background and Motivation

3.1.1. Motivation for Collaborative Inference

CNNs typically adopt a hierarchical architecture consisting of an input layer, an output layer, and hidden layers. The model processes input data layer by layer through a sequence of operators (e.g., convolutional layers, nonlinear activation layers such as ReLU, pooling layers, and fully connected layers), thereby progressively extracting multi-scale features ranging from low-level textures to high-level semantics. During inference, the activation tensors produced by different layers often vary substantially in spatial resolution and channel dimension, leading to significant differences in activation volume across layers. To quantify the data and compute characteristics of individual layers, we profile AlexNet [3] on a Dell PowerEdge server using PyTorch 2.6.0 under an inference setting with batch size 1 and an input resolution of $224 \times 224 \times 3$. For each layer, we record the output activation size and the average per-layer processing latency. To ensure statistical stability, all measurements are obtained over 1000 iterations. Figure 1 compares the output activation sizes and average processing latencies across layers. The results show that the activation volume can shrink markedly after certain layers; for instance, the input of the pool2 layer contains approximately 140k elements, whereas its output has only about 32.4k elements. Consequently, transmitting intermediate results after such layers can substantially reduce link load and, in turn, communication latency.

Motivated by the hierarchical structure of CNNs and the potential contraction of inter-layer activation volumes, we consider enabling collaborative inference across multiple resource-constrained INC nodes. As illustrated in Figure 2, an end device first generates inference tasks for a complete CNN, and the model is then partitioned into multiple partial CNN segments that are executed along a cooperative inference chain formed by multiple INC nodes (e.g., INC node 1 and INC node 2), with the cloud optionally participating when additional resources are needed. During execution, inference progresses hop by hop by transmitting intermediate activation tensors as intermediate results between adjacent nodes, and the final inference output is returned to the end device. By adaptively trading off computation cost against communication cost, the system selects appropriate partitioning and forwarding points, thereby effectively reducing end-to-end inference latency while leveraging otherwise idle in-network compute resources.

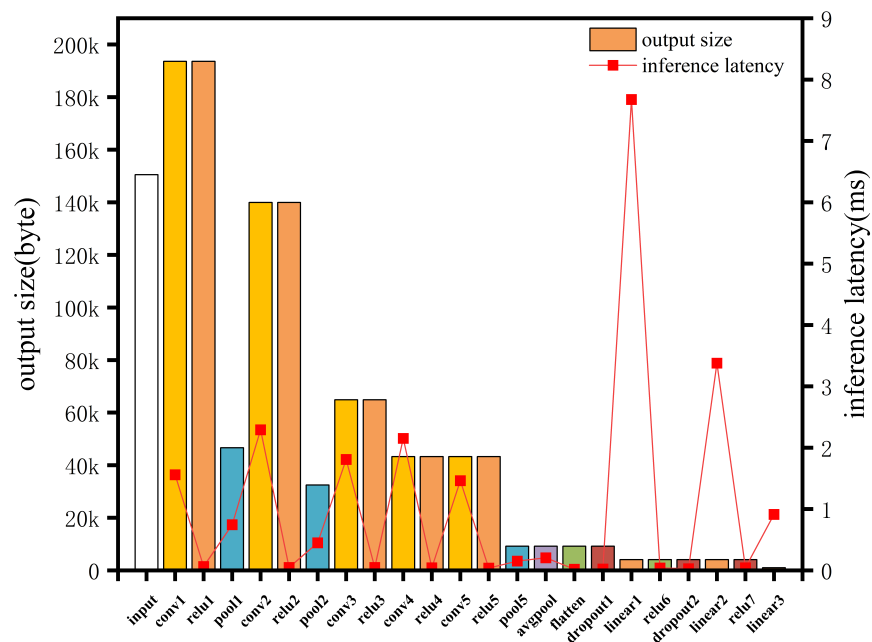


Figure 1. Per-layer output size and inference time of AlexNet (Bar colors denote layer types).

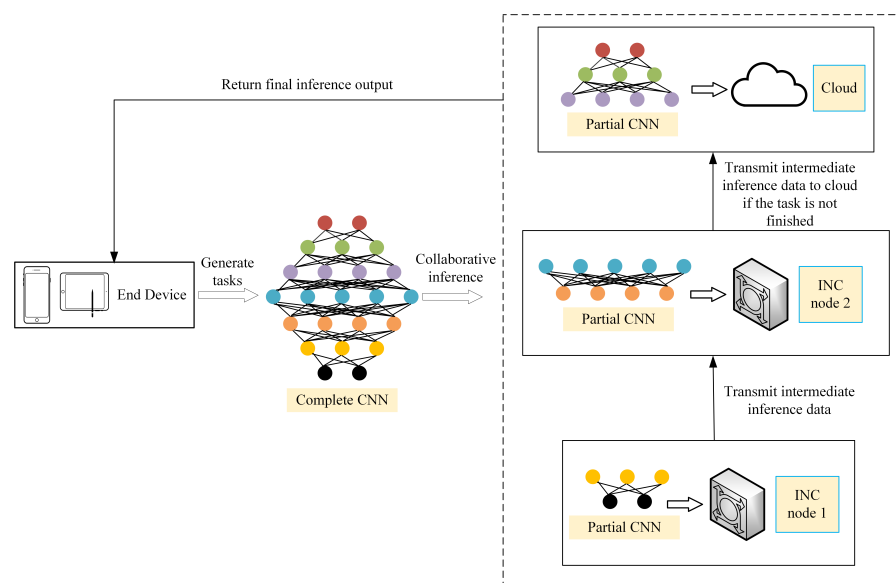


Figure 2. Illustration of INC-assisted collaborative CNN inference.

3.1.2. Name Resolution-Based ICN

As discussed in Section 1, to ensure compatibility with existing network infrastructures, the proposed in-network collaborative inference scheme is built upon a name resolution-based ICN architecture [29]. The architecture assigns a unique and invariant identifier (ID) to each network entity, such as hosts, devices, data, and services, and integrates an NMRS to maintain the mapping between IDs and IP addresses [30]. NMRSs can define multiple service tiers according to measurement metrics such as latency and distance and can be organized on demand into a hierarchical structure. From the perspective of administrative scope, the NMRS is categorized into two types. The Global Name Mapping and Resolution System (GNMRS) maintains global identity-to-address mappings and provides resolution services for all users at the global scale, thereby ensuring service accessibility. In contrast, the Local Name Mapping and Resolution System (LNMRS) is deployed at the network edge and consists of distributed resolve nodes, offering nearby users deterministic, on-site

resolution with bounded latency. Figure 3 illustrates an example NMRS architecture, depicting a three-tier resolution-domain hierarchy in which each domain provides a different class of resolution service; notably, both the number of tiers and the tiering criteria can be adjusted to meet practical deployment requirements. In Figure 3, the labels L1, L2, and L3 denote three resolution tiers from coarse-grained/global to fine-grained/local resolution. The annotated values (e.g., 50 ms, 25 ms, and 5 ms) represent illustrative upper bounds on the name-resolution latency provided by each tier, corresponding to different service classes. The circles labeled “Resolve Node” indicate NMRS resolution servers deployed in each tier, while the black dots labeled “Entities” represent named network entities (e.g., hosts, devices, data, and services) whose invariant IDs are resolved to IP addresses. The dashed lines indicate hierarchical relationships between tiers and the possible forwarding paths of resolution queries across tiers.

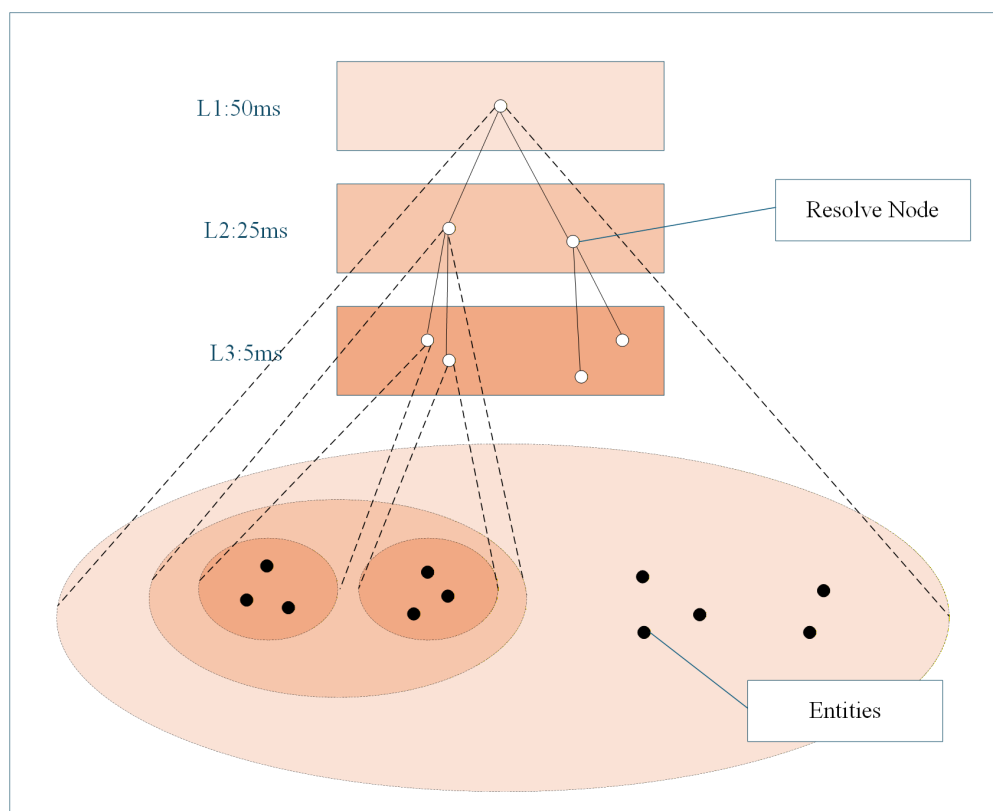


Figure 3. An example of hierarchical name resolution system.

With the aid of an NMRS, an ICN node can resolve the network address (NA) corresponding to the ID carried in a packet and then dynamically select and rewrite the destination address field according to service requirements and network conditions, thereby enabling flexible readdressing and routing adaptation. In our previous work, we abstracted requirements such as in-network caching [30], multipath transmission [31], and multicast [32] into network services and assigned each service a corresponding ID. By performing addressing and routing via IDs, we validated the feasibility of this approach. Inspired by these results, this paper similarly treats in-network collaborative inference as an in-network service. Specifically, ICN routers are deployed on in-network nodes with computation capability, and the associated service ID is registered with the resolution system so that the network can dynamically discover available computing resources. The ID/IP integrated ICN reconstruction module is then leveraged to update forwarding addresses on demand without changing the ID, thereby steering collaborative computation tasks to suitable nodes outside the default transmission path for cooperative processing.

3.2. Architecture

This paper proposes an ICN-based, distributed, hop-by-hop decision framework for in-network collaborative inference to accelerate CNN inference tasks. As illustrated in Figure 4, the ICN overlay consists of user devices (D1, D2), wireless access nodes (A1, A2), and multiple computation-capable INC nodes (N1–N10), with the cloud serving as a fall-back compute endpoint. An inference request carries a Model Service ID (MSID) and may traverse the default forwarding path or be opportunistically detoured to idle INC nodes for cooperative execution. In Figure 4, pending CNN tasks are buffered at INC nodes, task flow arrows depict hop-by-hop forwarding/execution of CNN partitions, and register/query arrows depict MSID-based name resolution. The architecture dynamically discovers and exploits idle INC nodes beyond the default forwarding path to reduce end-to-end inference latency and balance workload while preserving stable reachability under path deviations.

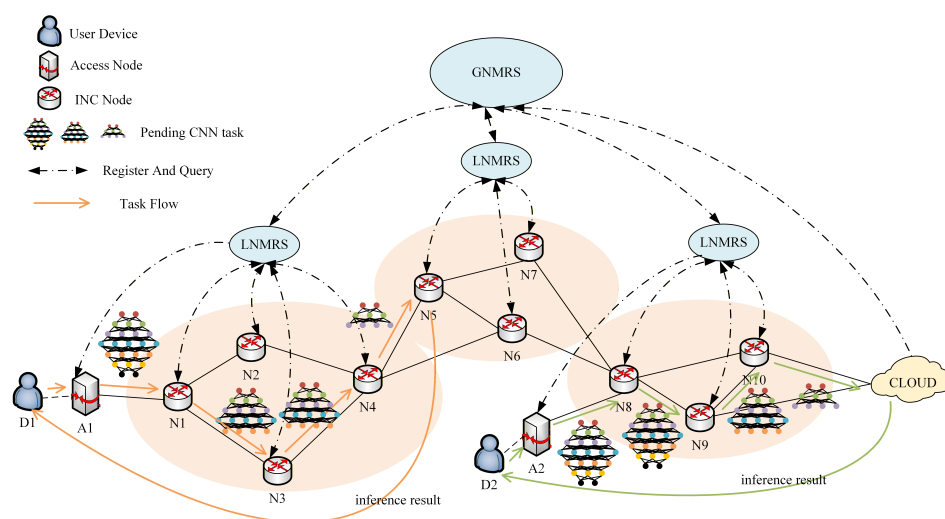


Figure 4. An overview of INC-assisted collaborative CNN inference based on an ICN overlay network.

We consider in-network collaborative inference as a service identified by MSID. To ensure engineering feasibility and support flexible version management, we adopt a flat naming mechanism [30] where the MSID is defined as a globally unique, location-independent, fixed-length (e.g., 160-bit) identifier. This ID is generated by computing the hash value of the model’s unique metadata, including the model name, version tag, and input specification, so that distinct model versions are resolved as independent services. To facilitate multi-instance sharing, the architecture utilizes a “One-ID-to-Many-NAs” mapping schema within the resolution system. As shown in Figure 4, the two-level name-resolution system consists of a GNMRS for global resolution and an LNMRS within each domain. The LNMRS maintains the mapping between a single MSID and the dynamic set of NAs corresponding to all accessible INC nodes hosting that model version. This design allows the system to leverage the deterministic-latency resolution capability of LNMRS to discover optimal off-path resources and maintain stable service reachability despite network dynamics or routing deviations [33]. Crucially, this dynamic registration mechanism effectively addresses the practical challenges of large-scale deployment, where preloading every CNN model on all INC nodes is unrealistic due to storage limitations and maintenance costs. By ensuring that only nodes possessing the specific model register the corresponding MSID, our architecture naturally accommodates sparse model placement.

The detailed in-network collaborative inference procedure is described as follows:

- INC nodes that host the corresponding CNN inference models deploy applications such as the ICN protocol stack and a resource monitoring module. Upon initialization,

each node computes the MSID based on the loaded model's metadata and registers the mapping between its NA and the MSID with the NMRS. To ensure mapping freshness, these entries are associated with a Time-To-Live (TTL) and are maintained via periodic heartbeats; if a node switches models or goes offline, it explicitly sends a request to update or deregister the mapping. Figure 4 illustrates how this mechanism is deployed in three example resolution domains.

- An end device initiates a CNN inference task and sends the request over the wireless access link to the nearest access node. In the example, devices D1 and D2 send their tasks to access nodes A1 and A2, respectively.
- After receiving an inference request, the access node parses the required model type and encapsulates it with an ICN header to form an ICN packet. It then queries the LNMRS using the MSID to obtain a list of available candidate INC nodes (i.e., the set of NAs mapped to that MSID) and, according to the node selection policy, chooses an appropriate INC node as the next hop for network-collaborative inference. The NA field in the packet is updated to the NA of the selected INC node and the packet is forwarded. For instance, A1 and A2 forward the tasks to N1 and N8, respectively.
- Upon receiving the task request, the selected INC node likewise queries the LNMRS based on the MSID to obtain the candidate node list and decides—according to its current status and the candidate information—whether to process the next partition of the CNN task locally or to forward it to the next INC node. If it decides to process locally, the INC node enqueues the task in its local task queue and executes it; once processing is completed, it triggers the next decision. If it chooses to forward to the next node, it updates the NA field in the packet to the NA of the chosen node and forwards the packet. In the example, N1, N4, N9, and N10 process subsequent layers locally, whereas N3 and N8 forward the task to other nodes because a better next hop is available.
- If there is no available candidate INC node and the current node cannot continue processing the task, it modifies the NA in the packet to the cloud NA and forwards the packet according to its local routing table. When the task reaches the next INC node, the above procedure can still be executed to enable collaborative inference.
- After an INC node completes inference, the result is sent back to the initiating device; for example, N5 returns the output to D1. If the final layers of the model are executed in the cloud, the cloud returns the final result to the requester, as shown in the D2 workflow in Figure 4.

Note that the candidate node set obtained by querying the NMRS is maintained locally in the form of a resource-state information table. The resource monitoring module on an INC node periodically collects network, load, and computing information between the node and candidate next-hop nodes and updates the table for use by the node selection policy. When the topology changes (e.g., nodes join or leave), the LNMRS notifies nodes subscribing to the relevant MSID, prompting them to update the candidate node list. The name-resolution lookup is triggered only when a node first receives a task for a given MSID, or when the local cache expires or an LNMRS change notification is received. For subsequent tasks with the same MSID, the node selects the next hop directly based on the locally maintained table, so the resolution latency has a negligible impact on the end-to-end inference latency.

3.3. Security and Privacy Considerations

Off-path collaborative inference can expand the attack surface compared to strictly on-path processing. To improve practical deployability, we discuss how our architecture can mitigate security and privacy risks from three perspectives:

- **Trust Management via NMRS:** In our architecture, NMRS is not only a service discovery mechanism but can also serve as a trust and policy enforcement point within an administrative domain. During the service registration phase (Section 3.2), the NMRS can enforce authentication and authorization so that only verified INC nodes with valid cryptographic credentials are allowed to register their MSIDs. Consequently, the candidate set returned by LNMRS (and thus the nodes selected by the scheduler) can be restricted to authenticated, operator-approved off-path nodes, mitigating rogue-node participation and registration abuse.
- **Data Privacy and Reduced Exposure:** The transmitted content in our system consists of intermediate feature maps rather than raw input data. Compared to raw inputs, intermediate representations can reduce direct exposure of sensitive information, especially when data contraction occurs and only contracted features are forwarded. However, intermediate features do not provide a complete privacy guarantee under advanced inference attacks; therefore, for sensitive applications, the ICN-based packet format is compatible with payload encryption to keep content opaque to intermediate forwarders.
- **Content Integrity:** To protect integrity against tampering by compromised nodes, packets carrying task requests and inference results can be cryptographically signed, enabling consumers (or downstream INC nodes) to verify authenticity and integrity. This object-level integrity protection complements the off-path processing workflow by allowing detection of malicious modification of inference outputs.

We emphasize that a full-fledged security protocol design and empirical security evaluation are beyond the scope of this paper; nonetheless, the above discussion clarifies the key risks and practical mitigation directions for operator-managed deployments.

4. System Model and Problem Formulation

This section establishes the system model by abstracting the heterogeneous INC network and CNN inference tasks. It subsequently formulates the latency minimization problem as a mixed-integer linear program (MILP), incorporating processing, queuing, and transmission constraints. Table 1 summarizes the notation used throughout this paper.

Table 1. Summary of key symbols.

Symbol	Description
$\mathcal{N}, n$	Set of computing-capable nodes (INC nodes and cloud) and its cardinality, with $ \mathcal{N} = n$
$\mathcal{G}_{\mathcal{N}} = (\mathcal{N}, \mathcal{E})$	Network graph with nodes $\mathcal{N}$ and links $\mathcal{E}$
j	A computing node index in $\mathcal{N}$
$\mathcal{E}$	Set of directed links between computing nodes
$B_{jj'}$	Bandwidth of link $(j, j') \in \mathcal{E}$
$t_{jj'}$	Propagation delay of link $(j, j') \in \mathcal{E}$
c_j	Computation capability of node j
$\mathcal{D}, d$	Set of end devices and its cardinality, with $ \mathcal{D} = d$
$\mathcal{T}, T$	Set of inference tasks and its cardinality, with $ \mathcal{T} = T$
k	A task index in $\mathcal{T}$
$\mathcal{G}^k = (\mathcal{L}^k, \mathcal{V}^k)$	CNN graph with edges $\mathcal{V}$ and layers $\mathcal{L}$ for task k
i	A layer index in $\mathcal{L}^{(k)}$
Q_j	Queue length of node j
w_i	Workload of layer i
s_i	Input feature size of layer i
x_{ij}	Binary variable denoting whether layer i assigns to a node j
$y_{ij'}^i$	Binary variable denoting whether the input of layer i traverses link (j, j')

4.1. Network and Inference Tasks

The in-network infrastructure is heterogeneous and consists of two categories of nodes. Specifically, conventional routing nodes operate as pure packet forwarders without native computing support. In contrast, INC nodes are computation-capable network elements, such as smart switches, programmable routers, and FPGA-based appliances, which can execute in-network processing functions. INC nodes are interconnected, and a subset of them is equipped with dedicated backhaul connectivity to a centralized cloud platform. The cloud is modeled as a virtually elastic computing resource with abundant processing capacity, while its access bandwidth to the in-network infrastructure is comparatively constrained.

To reduce modeling and computational complexity and focus on computation offloading and resource-allocation decisions, we abstract the physical network into a logical topology consisting of computing nodes only. Specifically, we refer to all INC nodes together with the cloud as computing nodes and denote the set by $\mathcal{N} = \{N_1, N_2, \dots, N_n, N_{cloud}\}$. Forwarding-only routers in the physical network are not explicitly included in $\mathcal{N}$ and are treated as transparent transit elements; their impact is captured through the effective characteristics of logical links between computing nodes. Accordingly, the overall network is modeled as a directed graph $\mathcal{G}_{\mathcal{N}} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{E}$ denotes the set of directed logical communication links. A directed logical link from node j to node j' is denoted by $e_{j,j'}$ and characterized by an effective bandwidth $B_{j,j'}$ and a propagation delay $t_{j,j'}$. The computational capability of node $j \in \mathcal{N}$ is denoted by c_j .

The set of end devices is denoted by $\mathcal{D} = \{D_1, D_2, \dots, D_d\}$. These IoT end devices, like smartphones, tablets, surveillance cameras, and wearables, are geographically distributed and access the network via wireless last-hop links (e.g., Wi-Fi). Each device associates with a nearby access INC node to transmit and receive data over a shared wireless channel.

End devices execute a set of CNN inference jobs, denoted by $\mathcal{T} = \{t_1, t_2, \dots, t_T\}$. For any job $k \in \mathcal{T}$, inference corresponds to a forward pass of a CNN. We abstract the CNN as a linear graph $\mathcal{G}^k = (\mathcal{L}^k, \mathcal{V}^k)$, where $\mathcal{L}^k = \{l_1^k, l_2^k, \dots, l_L^k\}$ is the ordered set of layers and l_1^k and l_L^k represent the input and output layers, respectively. An edge $(l_i^k, l_{i+1}^k) \in \mathcal{V}^k$ captures the data dependency that the computation of l_{i+1}^k consumes the output of l_i^k , implying that l_i^k must be executed beforehand. This abstraction facilitates subsequent layer partitioning, offloading decisions, and end-to-end latency analysis. For brevity, we omit the superscript k in the rest of the paper when focusing on a single task.

4.2. Inference Delay Model

The latency of transmitting the raw input data from the end device to the access node is fixed and does not vary with our decisions, so it is outside the scope of our optimization and can be ignored. A task can be partitioned into multiple subtasks, where each subtask consists of a contiguous segment of layers and is executed either on an INC node or in the cloud. If a subtask is assigned to a computing node $j \in \mathcal{N}$, the node may not be able to process it immediately and will enqueue it in a local FIFO execution queue. The queue capacity Q_j is proportional to the node's computational capability [13].

We model the execution placement with a binary decision variable $x_{ij} \in \{0, 1\}$, which indicates whether layer i is executed on computing node j . Each layer is executed at exactly one node:

$$\sum_{j \in \mathcal{N}} x_{ij} = 1, \quad \forall i \quad (1)$$

Assume that layer i requires w_i floating-point operations. Then, the processing delay of layer i can be expressed as

$$T_{\text{process}}^i = \sum_{j \in \mathcal{N}} x_{ij} \frac{w_i}{c_j} \quad (2)$$

Since layer i may have to wait in the local FIFO queue after arriving at computing node j , its queuing delay can be expressed as

$$T_{\text{queue}}^i = \sum_{j \in \mathcal{N}} x_{ij} \sum_{\ell \in \mathcal{Q}_j, \ell \neq i} \frac{w_\ell}{c_j} \quad (3)$$

In a linear computation graph of a CNN, the execution of layer i depends on the feature maps produced by layer $i - 1$. In a distributed collaborative inference setting, layers $i - 1$ and i may be placed on the same node or scheduled to different nodes for computation. In the latter case, the intermediate data transfer between the two layers may traverse multiple INC nodes that only forward packets and do not perform any computation.

To model the transmission latency required by layer i , we introduce a binary variable $y_{jj'}^i$ indicating whether the input of layer i traverses the link formed by nodes j and j' . Accordingly, the transmission latency of layer i can be expressed as

$$T_{\text{transmission}}^i = \sum_{j \in \mathcal{N}} \sum_{j' \in \mathcal{N}} y_{jj'}^i \left(\frac{s_i}{B_{jj'}} + t_{jj'} \right), \quad \forall i \geq 2 \quad (4)$$

where s_i denotes the input data size (i.e., the intermediate feature maps) of layer i .

Therefore, the end-to-end inference delay of a task is given by the sum of the processing delay, queuing delay, and inter-layer transmission delay across all layers. Since the uplink latency of transmitting the raw input from the end device to the access node is fixed, it is excluded from the objective.

$$T_{\text{total}} = \sum_{i=1}^L \left(T_{\text{process}}^i + T_{\text{queue}}^i \right) + \sum_{i=2}^L T_{\text{transmission}}^i \quad (5)$$

4.3. Problem Formulation

This paper aims to accelerate CNN inference by leveraging the collaborative inference capability of in-network computing nodes. Therefore, we formulate the optimization objective as minimizing the average task completion time.

Let Ω denote the feasible decision space of the collaborative inference problem. In general, the placement and routing decisions can be defined per task $k \in \mathcal{T}$. Specifically, let $X = \{x_{ij} \mid i \in [1, L], j \in \mathcal{N}\}$ and $Y = \{y_{jj'}^i \mid i \in [2, L], (j, j') \in \mathcal{E}\}$, and denote $\mathbf{d} = (X, Y)$. Then the feasible set is

$$\Omega \triangleq \{\mathbf{d} \mid \text{Constraints C1–C4 are satisfied}\}. \quad (6)$$

The optimization objective is a mapping $F : \Omega \rightarrow \mathbb{R}_{\geq 0}$ defined by

$$F(\mathbf{d}) \triangleq \bar{T} = \frac{1}{T} \sum_{k \in \mathcal{T}} T_{\text{total}}^k, \quad (7)$$

where $T_{\text{total}}^k \in \mathbb{R}_{\geq 0}$ is given in Equation (5).

The corresponding optimization problem is formulated as follows:

$$\min_{\{x_{ij}, y_{jj'}^i\}} \bar{T} = \min_{\{x_{ij}, y_{jj'}^i\}} \frac{1}{T} \sum_{k \in \mathcal{T}} T_{\text{total}}^k \quad (8)$$

and subject to

$$C1 : x_{ij} \in \{0, 1\}, \quad \forall i \in [1, L], \forall j \in \mathcal{N} \quad (9)$$

$$C2 : y_{jj'}^i \in \{0, 1\}, \quad \forall i \in [2, L], \forall (j, j') \in \mathcal{E} \quad (10)$$

$$C3 : \sum_{j \in \mathcal{N}} x_{ij} = 1, \quad \forall i \in [1, L] \quad (11)$$

$$C4 : \sum_{j' \in \mathcal{N}} y_{vj'}^i - \sum_{j' \in \mathcal{N}} y_{j'v}^i = x_{(i-1)v} - x_{iv}, \quad \forall i \in [2, L], \forall v \in \mathcal{N} \quad (12)$$

Constraints C1–C3 define the layer placement decisions. Specifically, C1 enforces the binary nature of x_{ij} , indicating whether layer i is executed on computing node j , and C3 guarantees that each layer is assigned to exactly one node in $\mathcal{N}$. Constraint C2 defines the routing decisions using binary variables $y_{jj'}^i$, which indicate whether the intermediate feature maps required by layer i traverse the directed link (j, j') . Finally, C4 couples placement and routing by imposing flow conservation for each layer transition $i - 1 \rightarrow i$: it ensures that a unit flow is generated at the node executing layer $i - 1$, absorbed at the node executing layer i , and conserved at intermediate forwarding nodes, thereby guaranteeing the feasibility of transmitting intermediate activations between consecutive layers.

This optimization problem can be formulated as an MILP problem. Its solution complexity grows exponentially with the number of tasks, the model depth, and the scale of network nodes. In a dynamic network environment, where node states and task queues change rapidly, a centralized solver would incur prohibitive computational latency and communication overhead, making it difficult to meet real-time requirements. Therefore, this paper proposes a distributed online optimization mechanism that decomposes the original problem into two stages: decision-point identification and node selection, and enables each INC node to make independent decisions based on its local, real-time state. By sacrificing part of the global optimality, this decomposition strategy substantially reduces decision complexity, thereby achieving efficient and feasible real-time solutions in dynamic distributed settings. Experimental results show that compared with baseline approaches, the proposed mechanism effectively reduces end-to-end inference latency.

5. In-Network Collaborative Inference Mechanism

This section details the distributed collaborative inference mechanism. It first introduces a lightweight decision-point identification method based on data contraction to minimize overhead. Subsequently, it presents the DLNS algorithm, which jointly optimizes processing and transmission latencies for real-time task steering.

5.1. Decision-Point Identification Mechanism

To enable low-overhead online decisions in a hop-by-hop collaborative inference chain, we introduce a lightweight decision-point identification mechanism. The key idea is to trigger the node-selection decision only at layer boundaries where forwarding the intermediate activations can potentially reduce communication cost, i.e., the intermediate data volume becomes smaller than the data currently carried by the task packet.

Consider a task packet arriving at an INC node with an input activation tensor of size s_{in} . Let s_{i+1} denote the output feature size after executing layer i . We define layer i as a potential decision point if its output size is smaller than the current packet input, i.e.,

$$\mathcal{P}(s_{\text{in}}) = \{i \mid s_{i+1} < s_{\text{in}}, i \in [1, L - 1]\}. \quad (13)$$

Intuitively, once the computation reaches such a layer boundary, transmitting the intermediate result (s_{i+1}) is cheaper than transmitting the original input (s_{in}), which makes forwarding more attractive. Figure 5 illustrates the identified decision points along the layer-wise forward pass, where the red markers denote potential split points (i.e., decision

points) at which the activation volume contracts notably. The numbers (1–10) annotate the indices of intermediate activation tensors along the forward path, where 1 denotes the original input activation and $k + 1$ denotes the output activation after executing layer k (equivalently, the input to layer $k + 1$).

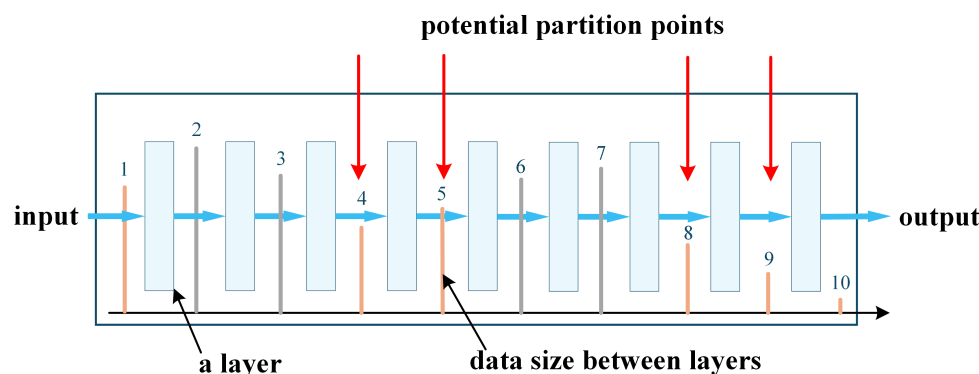


Figure 5. Identifying potential decision points of a CNN.

When an INC node receives a task packet, it (i) records the current input size s_{in} from the packet payload, (ii) parses the MSID and the current layer index carried in the packet header, and (iii) looks up a pre-profiled per-layer feature-size table $\{s_i\}_{i=1}^L$ for the corresponding model. The node then scans subsequent layers and constructs the ordered set $\mathcal{P}(s_{in})$ according to Equation (13). This operation is lightweight, with time complexity linear in the number of remaining layers (see Algorithm 1).

The node executes the CNN layers sequentially. Each time the computation reaches a layer $i \in \mathcal{P}(s_{in})$, it triggers a decision event and invokes the node-selection strategy. If the strategy chooses local execution, the node continues processing the next layers without partitioning and waits for the next decision point. Otherwise, if the strategy chooses forwarding, the node partitions the model at layer i : it finalizes the execution up to layer i , encapsulates the output activation tensor as the new packet payload, updates the packet metadata, and forwards the intermediate result to the selected next-hop INC node. In this way, decisions are made only at a small set of informative layer boundaries, reducing decision overhead while preserving the opportunity to trade computation for communication whenever intermediate activations shrink.

Algorithm 1 Decision-point identification

Input: Current packet input size s_{in} ; current layer index p (the next layer to execute); profiled per-layer output sizes $\{o_\ell\}_{\ell=1}^L$ for the target model

Output: Ordered set of decision points $\mathcal{S}$

Initialize: $\bar{s} \leftarrow s_{in}$ and $\mathcal{S} \leftarrow \emptyset$

- 1: **for** $\ell = p, p + 1, \dots, L - 1$ **do**
 - 2: **if** $o_\ell < \bar{s}$ **then**
 - 3: Insert ℓ into $\mathcal{S}$
 - 4: **end if**
 - 5: **end for**
 - 6: **return** $\mathcal{S}$
-

We acknowledge that the formulation in Section 4 abstracts a CNN as a linear layer graph for tractable hop-by-hop partitioning and latency modeling. Many modern architectures (e.g., ResNet/Inception-style networks) exhibit directed acyclic graph (DAG) structures with skip connections and parallel branches. In the current mechanism, a practical way to apply our decision-point identification is to treat each multi-branch block (e.g., a residual block or an inception module) as an atomic macro-layer, such that the global

model becomes a sequence of macro-layers. Decision points can then be identified at macro-layer boundaries based on the contraction of inter-block activation volume, and DLNS can operate on these boundaries in the same hop-by-hop manner. We emphasize that this coarse-grained abstraction does not yet exploit fine-grained intra-block parallelism; extending the framework to explicitly model fork-join dependencies and schedule parallel branches is an important direction for future work.

5.1.1. Discussion on Decision-Point Identification Criteria

We emphasize that the proposed data contraction rule is a lightweight pre-screening mechanism for pruning candidate partitioning layers, rather than a replacement of the final offloading decision. Its role is to trigger the node-selection procedure only at layer boundaries where forwarding is potentially beneficial from a communication perspective, i.e., when the intermediate activation size becomes smaller than the payload currently carried by the task packet.

A fixed-interval strategy (e.g., selecting every k layers) is agnostic to the layer-wise activation-size dynamics of CNNs. In practice, intermediate feature-map volumes fluctuate substantially across layers and may exhibit sharp expansions and contractions. As a result, fixed-interval partitioning can trigger decisions at boundaries with large activations, which, according to our transmission model, may lead to unnecessarily high transmission latency.

One may consider identifying decision points using a more comprehensive metric that jointly accounts for computation cost and communication cost. While such a composite rule can be informative, evaluating it at every layer boundary would require frequent acquisition of time-varying network and system states (e.g., bandwidth and remote queuing conditions) and would substantially increase signaling and control-plane overhead.

Therefore, we adopt a two-stage design to balance decision quality and runtime efficiency: the first stage uses the data-contraction rule as a low-cost filter to reduce the number of decision events, and the second stage (DLNS in Section 5.2) performs the detailed evaluation by explicitly aggregating transmission, queuing, and processing latency components (Equation (15) and Algorithm 2). This hierarchical design preserves the ability to optimize end-to-end inference latency while keeping online decision-making lightweight.

5.1.2. Discussion on Dynamic Adaptability

It is important to note that while our current implementation utilizes a pre-profiled look-up table to minimize runtime overhead for static CNNs, this design does not inherently limit adaptability to dynamic models. For scenarios involving variable input resolutions or dynamic computational graphs, the static look-up table can be replaced by a lightweight shape inference module. This module analytically calculates the output tensor dimensions (s_{i+1}) in real time based on the operator metadata (e.g., kernel size, stride, and padding) and the current input properties. Since the core logic of our decision-point identification (Algorithm 1) relies solely on the relative comparison of data volumes (i.e., $s_{i+1} < s_{in}$), it remains effective regardless of whether the output size is retrieved from a profile or computed on-the-fly. In cases where shapes cannot be fully determined ahead of execution (e.g., data-dependent control flow), the node can directly measure the realized output tensor size after executing the operator and apply the same comparison rule to trigger a decision event.

5.2. Node Selection Strategy

When the computation reaches an identified decision point, the current INC node must choose between continuing the remaining layers locally and forwarding the intermediate activations to another INC node (or the cloud) for further processing. We adopt a distributed load- and network-aware selection strategy (Algorithm 2) that enables lightweight online

decisions without global coordination. The decision is made using a locally maintained resource-state table, which is periodically refreshed by the resource monitoring module. This table aggregates the candidate nodes resolved by the LNMRS together with their reported computing capability and queue status, as well as the measured link conditions between the current node and each candidate.

To avoid myopic decisions at a single layer, we evaluate a candidate node based on the expected time to execute a contiguous segment of layers until the next decision point (or the end of the model). The rationale for this segment-based setting lies in the data flow characteristics of CNNs defined in Section 5.1. Since decision points are identified by data contraction, the intermediate layers between two consecutive decision points typically involve data expansion or constant data volume. Consequently, forwarding data within such intervals would likely incur higher transmission costs than processing locally. Therefore, we treat the contiguous layers between decision points as an atomic execution block. This strategy prevents the scheduler from initiating expensive transmissions during data-expansion phases while avoiding the excessive signaling overhead associated with global full-model look-ahead. Specifically, when the current decision is triggered at layer boundary i and the next layer to execute is $p = i + 1$, the node computes the next decision point $\ell \in \mathcal{S}$ with $\ell \geq p$ (obtained from Algorithm 1); if no such point exists, set $\ell = L$. The workload of the segment is

$$W_{p:\ell} = \sum_{r=p}^{\ell} w_r, \quad (14)$$

and the payload size to be transmitted (if forwarded) equals the current intermediate activation size, denoted by s_{in} .

For a candidate next-hop node j' , we approximate the time to finish the next segment of layers $p, \dots, \ell$ on j' by summing the communication, queuing, and processing components:

$$\hat{T}(j \rightarrow j') = \mathbb{1}_{j' \neq j} \left(\frac{s_{\text{in}}}{B_{jj'}} + t_{jj'} \right) + \hat{T}_{\text{queue}}(j') + \frac{W_{p:\ell}}{c_{j'}}. \quad (15)$$

Here, the first term accounts for shipping the current payload of size s_{in} from node j to j' over link (j, j') and it vanishes when $j' = j$. The queuing term $\hat{T}_{\text{queue}}(j')$ is obtained from the local queue information of j' ; in practice, it can be estimated as the remaining workload already enqueued at j' divided by $c_{j'}$, which is consistent with Equation (3). The last term is the execution time of layers $p-\ell$ on j' with computing capability $c_{j'}$.

The next-hop node is selected by minimizing the above estimated latency:

$$N_{\text{next}} = \arg \min_{j' \in \mathcal{M} \cup \{j\}} \hat{T}(j \rightarrow j'), \quad (16)$$

where $\mathcal{M}$ is the candidate set returned by LNMRS for the target MSID. If multiple nodes tie, we prefer the local node to avoid unnecessary transmissions. If all candidate nodes are infeasible (e.g., queue full or unreachable), the packet is forwarded to the cloud following the default routing behavior described in Section 3.

Complexity analysis. This strategy is fully distributed; each node performs the computation in Equation (15) using only local measurements and cached candidate information, requiring no global synchronization. The complexity is $O(|\mathcal{M}|)$ per decision event, and decision events occur only at the sparse decision points identified by Algorithm 1, keeping the online overhead small.

Algorithm 2 Distributed load- and network-aware scheduling (DLNS)

Input: current node j ; next layer index p ; payload size s_{in} ; decision-point set $\mathcal{S}$; candidate set $\mathcal{M}$ from LNMRS; profiled per-layer FLOPs $\{w_i\}$; local table $\{c_{j'}, \hat{T}_{queue}(j'), B_{jj'}, t_{jj'}\}$

Output: selected node N_{next}

Initialize: $\mathcal{C} \leftarrow \mathcal{M}$; $best \leftarrow +\infty$; $N_{next} \leftarrow \emptyset$

```

1: if  $j$  is an INC node then
2:    $\mathcal{C}.add(j)$ 
3: end if
4: if  $\exists r \in \mathcal{S}$  and  $r \geq p$  then
5:    $\ell \leftarrow \min\{r \in \mathcal{S} \mid r \geq p\}$ 
6: else
7:    $\ell \leftarrow L$ 
8: end if
9:  $W_{p:\ell} \leftarrow \sum_{r=p}^{\ell} w_r$ 
10: for each  $j' \in \mathcal{C}$  do
11:   if  $j'$  is infeasible then
12:     continue
13:   end if
14:   if  $j' = j$  then
15:      $T_{comm} \leftarrow 0$ 
16:   else
17:      $T_{comm} \leftarrow \frac{s_{in}}{B_{jj'}} + t_{jj'}$ 
18:   end if
19:    $T_{proc} \leftarrow \frac{W_{p:\ell}}{c_{j'}}$ 
20:    $T \leftarrow T_{comm} + \hat{T}_{queue}(j') + T_{proc}$ 
21:   if  $T < best$  then
22:      $best \leftarrow T$ 
23:      $N_{next} \leftarrow j'$ 
24:   else if  $T = best$  and  $j' = j$  then
25:      $N_{next} \leftarrow j$ 
26:   end if
27: end for
28: if  $N_{next} = \emptyset$  then
29:    $N_{next} \leftarrow N_{cloud}$ 
30: end if
31: return  $N_{next}$ 

```

6. Experiments and Analysis

This section describes the simulation setup and reports the results for the proposed mechanism, benchmarking it against several representative baseline schemes. We then analyze and discuss the observations to validate the effectiveness and advantages of our approach under diverse workload conditions.

6.1. Experimental Setup

The proposed in-network collaborative inference architecture and its DLNS scheduler were implemented in the ns-3 discrete-event simulator [34]. It is important to note that our simulation is trace-driven. To ensure the simulation reflects real-world hardware performance, the computation latency and data volume parameters for each INC node are strictly mapped from the real-hardware profiling data collected from the Dell PowerEdge server, as detailed in Section 3.1.1. This approach allows us to evaluate the network-wide scheduling performance with realistic processing constraints. The GEANT2 topology was used as the experimental network graph, where each numbered circle in Figure 6 represents a router node and each edge denotes a link with heterogeneous bandwidth and propagation delay. An ICN router was instantiated on every router node so that the ICN overlay covered

the entire topology. As illustrated in Figure 6, ten end devices (D0–D9, shown as blue squares) were connected to selected router nodes to generate inference requests. In addition, a cloud endpoint was attached to the topology via a dedicated backhaul link (as labeled “CLOUD” in Figure 6) to serve as the fallback compute destination when in-network resources are insufficient.

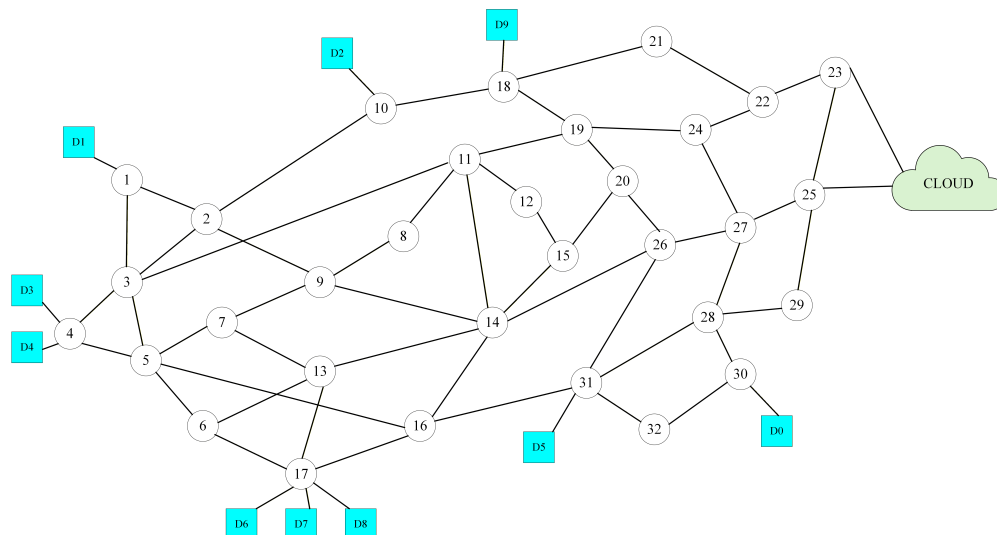


Figure 6. Simulation topology.

Link bandwidths were configured to capture heterogeneous network conditions by sampling the rate of each link from a uniform distribution $\mathcal{U}[50, 150]$ Mbps, and the sampled rates were kept constant throughout each simulation run. Likewise, the propagation delay of each link was drawn from $\mathcal{U}[1, 5]$ ms and remained unchanged within a run. We further introduced a cloud server as a fallback execution site. The cloud was connected to the topology via a dedicated link with bandwidth $B_{\text{cloud}} = 200$ Mbps and propagation delay $d_{\text{cloud}} = 50$ ms.

We evaluated three CNN models with increasing complexity, namely, NiN, AlexNet, and VGG16. To parameterize the proposed decision mechanism, we conducted offline profiling using PyTorch (batch size = 1) and recorded the per-layer baseline computation latency t_i^{base} together with the corresponding output feature size S_i (in bytes). Each latency value was obtained by averaging over 1000 inference runs.

In the simulation, executing block i on an INC node j took

$$t_{i,j} = \frac{t_i^{\text{base}}}{\kappa_j}, \quad (17)$$

where κ_j denotes the compute coefficient of INC node j . Compute heterogeneity followed a three-tier model across the network: 20% high-performance nodes with $\kappa_j \sim \mathcal{U}[1.5, 2.0]$, 60% standard nodes with $\kappa_j \sim \mathcal{U}[0.9, 1.1]$, and 20% low-performance nodes with $\kappa_j \sim \mathcal{U}[0.5, 0.7]$. This specific ratio was selected to approximate a Normal Distribution of computational capabilities commonly observed in heterogeneous wide-area networks, where the majority of infrastructure consists of standard commercial off-the-shelf (COTS) devices, flanked by smaller fractions of upgraded high-performance clusters and legacy constrained devices. This setting ensures a realistic representation of hardware variance without introducing extreme bias. Due to node heterogeneity, each INC node j has a different queue capacity. We implement the queue constraint using a workload-based metric: an INC node is considered infeasible if the estimated queued workload exceeds its node-specific limit, i.e.,

$$W_j(t) + \hat{t}_{\text{serve},j}(i) > W_j^{\max}, \quad (18)$$

where $W_j(t)$ is computed as the sum of the estimated processing time of all pending blocks at INC node j .

The cloud compute capacity was set to $4\times$ the baseline. To validate the system in a concrete intelligent perception scenario, we simulate an Intelligent Traffic Monitoring application. The input workload is derived from the Berkeley DeepDrive (BDD100K) driving dataset [35], which contains diverse video sequences collected from vehicle-mounted cameras. In our simulation, end devices (D0–D9) act as roadside smart cameras or connected vehicles performing real-time object detection/recognition tasks. We sampled frames from these traffic videos to generate a realistic stream of inference requests, reflecting the latency-sensitive nature of surveillance and autonomous driving applications. Task arrivals were modeled as an independent Poisson process with mean rate λ at each end device. This assumption is consistent with typical simulation settings in recent distributed inference studies [13,20], allowing for theoretical tractability. While real-world traffic exhibits burstiness, we address this by evaluating the system under high-intensity arrival rates (up to $\lambda = 100$ tasks/s), which effectively simulates the peak load conditions typical of bursty traffic periods. Furthermore, the proposed DLNS algorithm explicitly incorporates queuing delay (T_{queue}) to handle such congestion dynamics.

We categorize the baseline schemes into two groups: architecture-level and algorithm-level. Architecture-level baselines include: Cloud-only (CO), which fully offloads tasks to the cloud; Dynamic Edge-Cloud (DEC) [17], which dynamically partitions DNN inference between the access node and the cloud while bypassing intermediate routers; IP-based On-Path INC (IP-INC) [36], which restricts execution to nodes strictly along the IP shortest path, preventing off-path optimization; and the proposed ICN-based INC (ICN-INC), which leverages the NMRS to exploit off-path resources. To evaluate node-selection policies within the ICN-INC architecture, we compare four algorithm-level baselines: Random Selection (RAND), which selects nodes uniformly at random; Greedy Transmission (GREEDY-TX), a heuristic prioritizing locality and transmission latency; Least Workload (LWL), which minimizes estimated queuing delay; and DLNS, our proposed algorithm that jointly optimizes processing, queuing, and transmission latencies. All results represent the average of ten replications, with error bars denoting the standard deviation.

6.2. Simulation Results

We primarily investigate system performance under varying workload intensities by increasing the mean task arrival rate λ at each device. In addition, since our optimization objective is to minimize inference latency, we focus our evaluation on latency-related metrics.

6.2.1. Architecture-Level Performance Comparison

We first evaluate the advantages of the proposed ICN-INC architecture compared to the baseline architectures (CO, DEC, and IP-INC). We analyze the performance in terms of Average Task Completion Time and Average Transmission Time across the different models.

Average Task Completion Time. This metric measures the end-to-end latency from the task initiation at the end device to the reception of the final result. It directly reflects the quality of experience (QoE) for time-sensitive applications. Figure 7 shows the average task completion time for the three CNN models as the task arrival rate increases. As expected, the latency of all schemes rises with the workload due to network congestion, yet ICN-INC consistently achieves the lowest latency irrespective of the model complexity, with an overall average improvement of approximately 35% compared to the next best baseline across the simulated range. Specifically, the Cloud-only approach exhibits the highest baseline latency due to the significant propagation delay and limited bandwidth of the

backhaul link. The DEC scheme performs well in the low-load regime, particularly for lightweight models like NiN, as the access edge node can sufficiently handle the inference tasks. However, its performance degrades sharply as the arrival rate increases or for computation-intensive models like VGG16, where the single access node quickly becomes saturated, forcing subsequent tasks to be offloaded to the remote cloud. Similarly, IP-INC achieves performance comparable to ICN-INC under light traffic conditions when on-path nodes are underutilized. However, as the load intensifies, the rigid shortest-path routing causes on-path nodes to become congestion hotspots, leading to excessive queuing delays. In contrast, ICN-INC demonstrates superior scalability and robustness. By leveraging the NMRS-based service resolution, the system dynamically steers tasks to idle computing resources located off the default path, effectively avoiding the hotspots encountered by IP-INC and the resource exhaustion seen in DEC. This advantage is most pronounced in the VGG16 scenario, where ICN-INC significantly reduces the processing bottleneck through effective network-wide load balancing, achieving a latency reduction of approximately 43.3% and 60.2% compared to IP-INC and DEC, respectively, at the highest arrival rate ($\lambda = 100$).

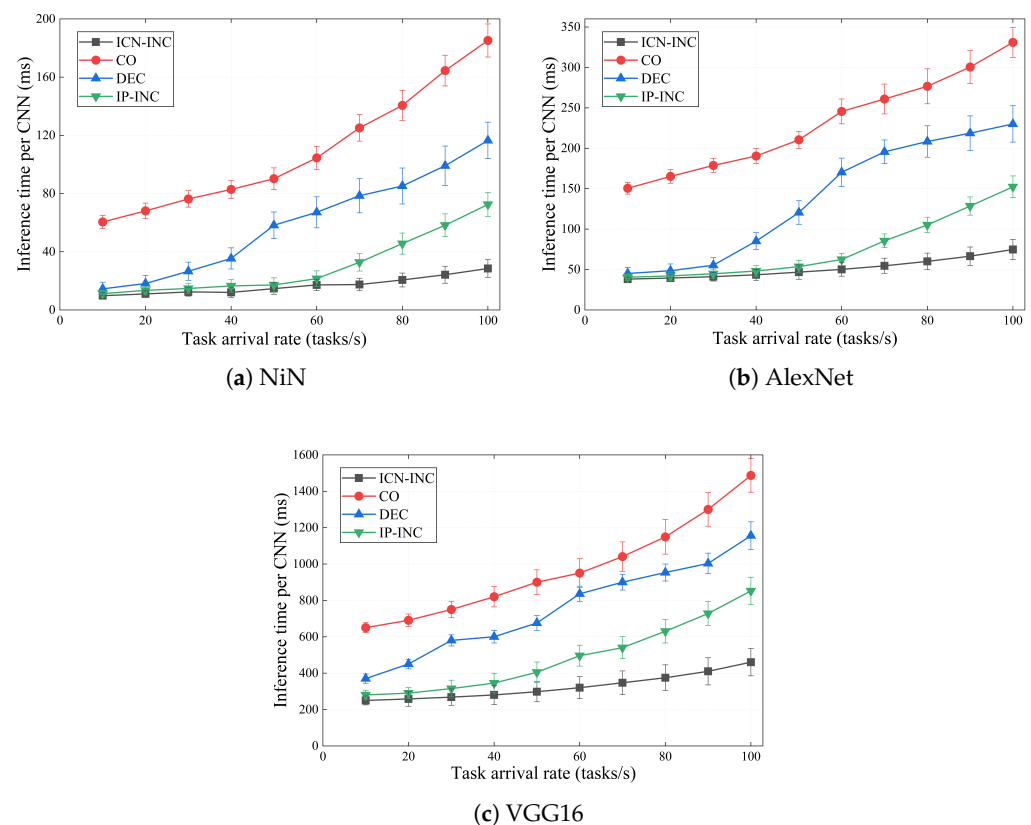


Figure 7. Task completion time performance of different architectures with different task arrival rate for (a) NiN, (b) AlexNet, and (c) VGG16.

Average Task Transmission Time. This metric isolates the network communication overhead involved in transferring data between nodes. It serves as an indicator of how effectively the architecture minimizes data movement by transmitting compact intermediate results instead of raw data. Figure 8 shows the average task transmission time for the three CNN models as the task arrival rate increases. The results show that Cloud-only incurs the highest transmission cost as it requires transmitting raw input data over long-haul links, consistently exceeding 350 ms for the VGG16 model. DEC exhibits a distinct performance pattern. Under light workloads, it achieves minimal transmission latency by

processing tasks locally at the access edge node. However, this advantage is short-lived; as the workload increases and the single edge node becomes saturated, DEC is forced to offload the majority of tasks to the remote cloud. Consequently, its transmission time rises sharply, surging to over 310 ms at $\lambda = 100$, approaching the high overhead of the Cloud-only approach.

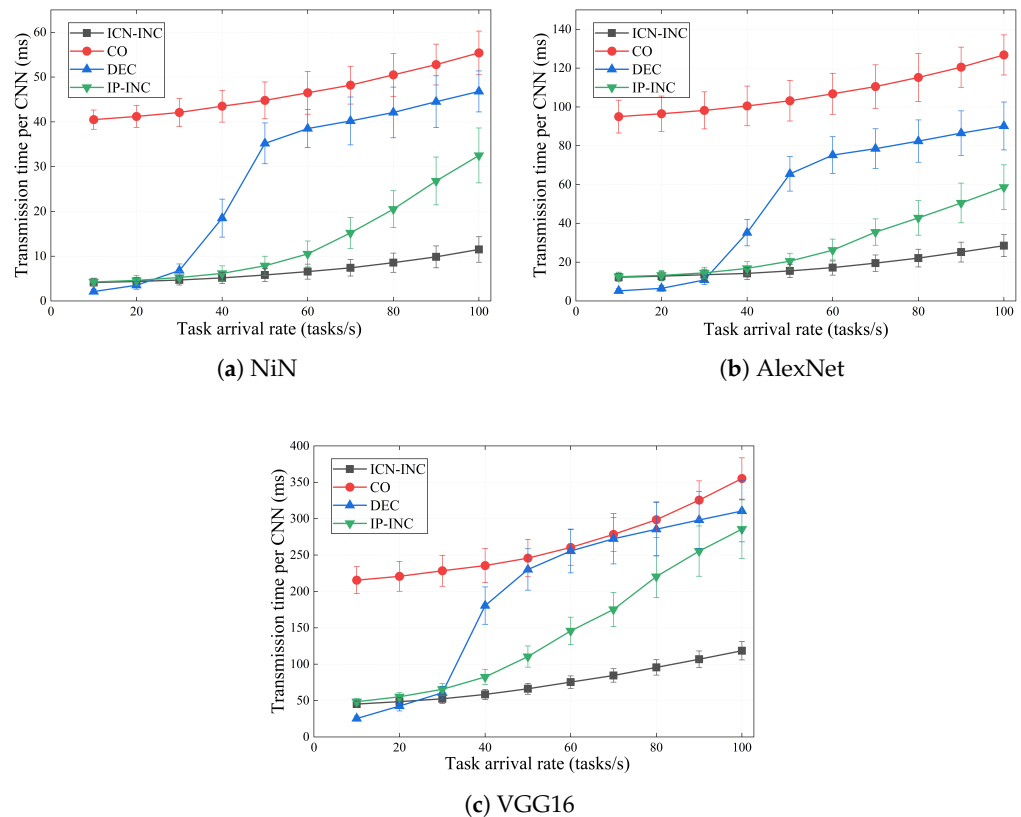


Figure 8. Task transmission time performance of different architectures with different task arrival rate for (a) NiN, (b) AlexNet, and (c) VGG16.

In contrast, both IP-INC and ICN-INC leverage the proposed decision-point identification mechanism to reduce data volume by transmitting contracted intermediate feature maps. However, a significant performance gap emerges between them under high-load conditions, particularly for data-intensive models like VGG16. When on-path nodes are overloaded, IP-INC is forced to forward data further downstream along the congested shortest path towards the remote cloud, leading to a rapid accumulation of transmission latency over multiple hops, reaching a peak of approximately 315 ms. ICN-INC, conversely, demonstrates superior stability. Although it may slightly exceed DEC's transmission time under negligible loads (due to forwarding tasks to neighbors), it utilizes the DLNS algorithm to proactively identify topologically closer off-path neighbors during congestion. By prioritizing nearby nodes with higher available bandwidth, ICN-INC effectively mitigates the exponential rise in transmission time observed in IP-INC and DEC, thereby maintaining the most robust and efficient communication performance as network contention intensifies. Specifically, for the VGG16 scenario under peak load, ICN-INC limits the transmission time to roughly 118 ms, representing a substantial reduction of 58.5% compared to IP-INC and 61.8% compared to DEC.

6.2.2. Algorithm-Level Performance Comparison

Having validated the architecture, we now evaluate the proposed DLNS algorithm against baselines (RAND, GREEDY-TX, LWL). Our analysis focuses on three aspects: Average Task Completion Time for overall efficiency, a Latency Breakdown to identify gain sources, and Jain's Fairness Index to assess load balancing effectiveness.

Average Task Completion Time. Figure 9 illustrates the latency performance of different node selection strategies. The results confirm that DLNS consistently outperforms the baselines across all workload intensities and model types. Specifically, GREEDY-TX exhibits a distinct saturation behavior; while it performs comparably to DLNS under light traffic, its latency explodes under high loads due to severe queuing at the nearest nodes. For instance, in the computation-intensive VGG16 scenario at $\lambda = 100$, GREEDY-TX reaches a latency of approx. 1285 ms, whereas DLNS maintains it at 460 ms, representing a massive 64.2% reduction. This indicates that blindly selecting proximal nodes leads to local hotspots. Conversely, LWL effectively avoids queuing but suffers from excessive transmission overhead by selecting distant underutilized nodes. Consequently, although LWL outperforms GREEDY-TX in the high-load VGG16 regime, it still lags behind DLNS by roughly 34.7% due to the unoptimized transmission path. The advantage of DLNS is equally evident in lightweight models, for NiN at peak load, DLNS achieves a $1.7\times$ speedup compared to the suboptimal solution (LWL), proving its ability to jointly optimize computation and transmission resources to minimize end-to-end latency.

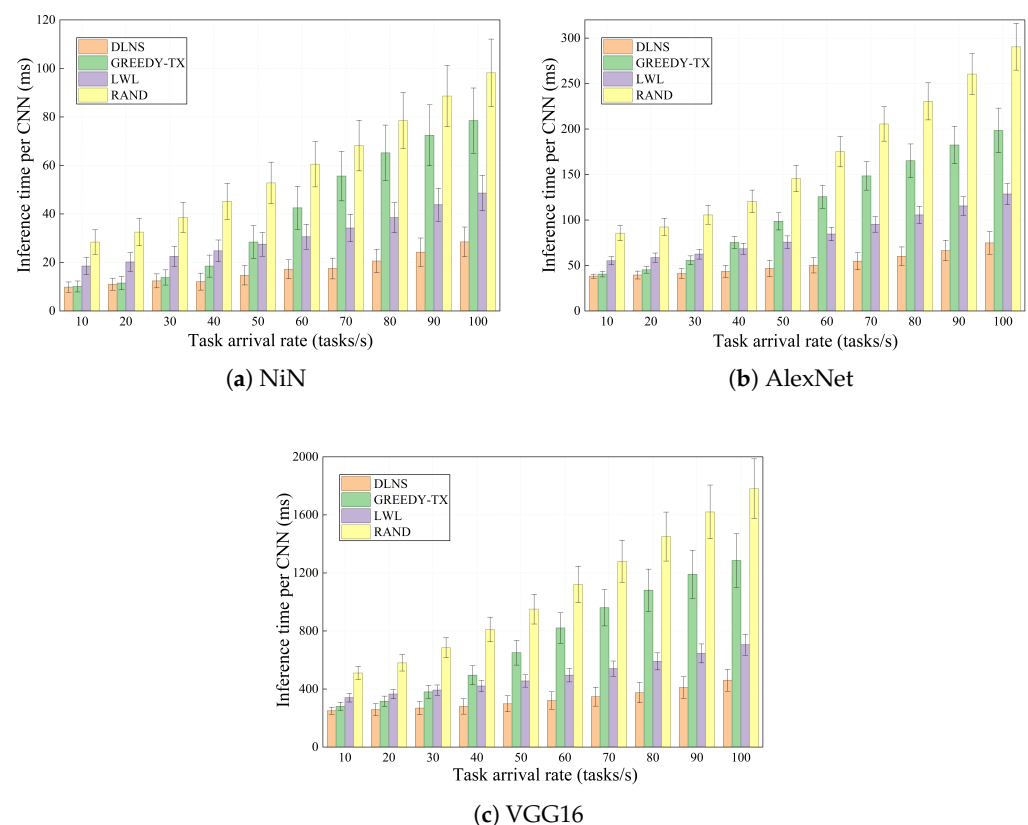


Figure 9. Task completion time performance of different algorithms with different task arrival rate for (a) NiN, (b) AlexNet, and (c) VGG16.

Improvement. Figure 10 illustrates the performance improvement of DLNS over baseline schemes across three key metrics. For total inference time (Figure 10a), DLNS achieves a consistent speedup, outperforming LWL by approximately $1.5\times$ for VGG16. The inferior performance of RAND is attributed to its oblivious selection strategy, which inevitably

leads to high latency by frequently assigning tasks to topologically distant or computationally saturated nodes without utilizing any network state information. The overall gain of DLNS is primarily driven by transmission efficiency (Figure 10b), where it demonstrates a massive $2.9\times$ improvement over LWL for VGG16, as LWL's neglect of network proximity leads to excessive data transfer costs. Regarding queuing time (Figure 10c), LWL achieves competitive performance, occasionally even surpassing DLNS, owing to its exclusive focus on minimizing server load. However, this minor concession allows DLNS to optimize the global objective. Specifically, DLNS distinguishes itself by resolving the catastrophic congestion of GREEDY-TX with an $11.5\times$ reduction in queuing for AlexNet, thereby striking the optimal balance between transmission and queuing to minimize the end-to-end delay.

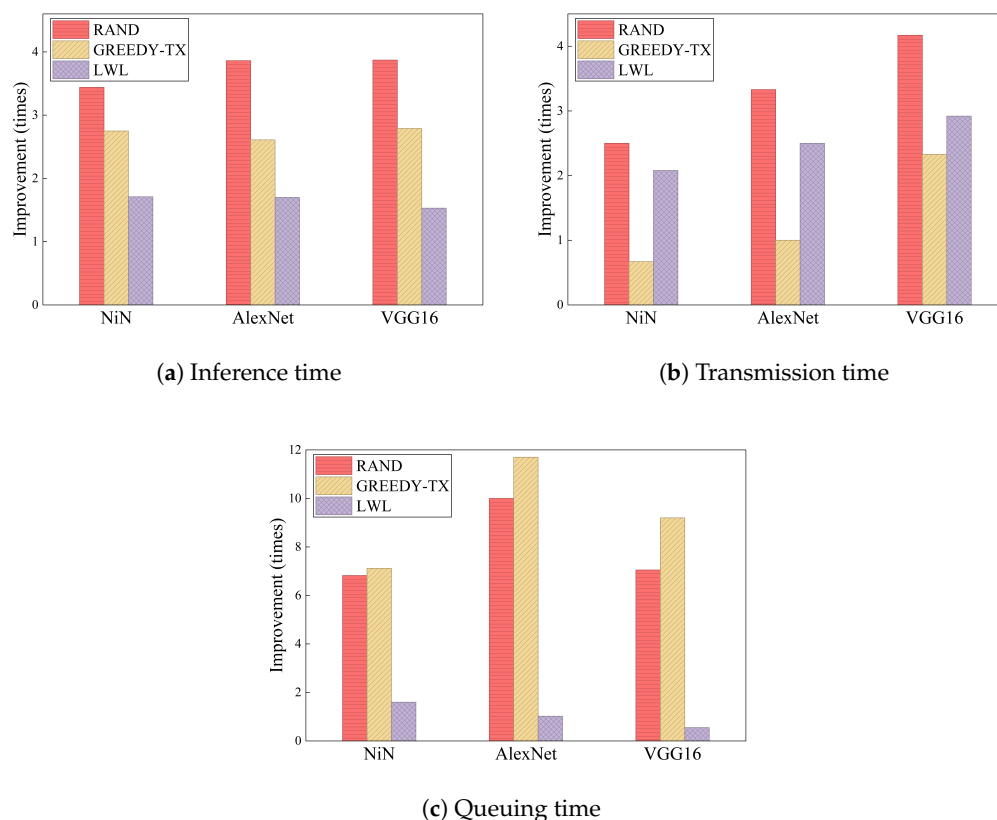


Figure 10. Improvement of DLNS against the other schemes in terms of (a) total inference time, (b) transmission time, and (c) queuing time.

Load balancing performance. We employ Jain's Fairness Index (JFI) to quantify the system's ability to effectively utilize network-wide resources and avoid hotspots. Unlike simple task counting, we define the load based on the actual computation time to account for the heterogeneity of CNN models. The index is defined as $\mathcal{J} = (\sum W_k)^2 / (N \cdot \sum W_k^2)$, where N denotes the total number of INC nodes and W_k represents the accumulated processing time of node k . The value of $\mathcal{J}$ ranges from $1/N$ to 1, with 1 indicating a perfectly balanced distribution. Figure 11 illustrates the JFI performance for the VGG16 benchmark as workload intensity increases. As expected, LWL consistently achieves the highest fairness index (>0.96) across all arrival rates, serving as the theoretical ceiling due to its explicit variance-minimization objective. Crucially, the proposed DLNS demonstrates remarkable robustness, maintaining a JFI above 0.86 under different task arrival rates. This confirms that DLNS successfully prevents hotspot formation by proactively offloading tasks to underutilized INC nodes, thereby achieving near-optimal load balancing. In stark contrast, Greedy-Tx exhibits a sharp decline in fairness as load increases, dropping to

0.18 at peak load. This collapse confirms that selecting INC nodes solely based on proximity leads to severe polarization, where a few nearest nodes are overwhelmed while others remain idle. It is worth noting that although DLNS yields a marginally lower JFI than LWL, this represents a strategic trade-off. By sacrificing a fraction of load uniformity to prioritize topologically closer and higher-performance INC nodes, DLNS achieves superior total inference latency, avoiding the excessive transmission costs often incurred by the strict load-balancing logic of LWL.

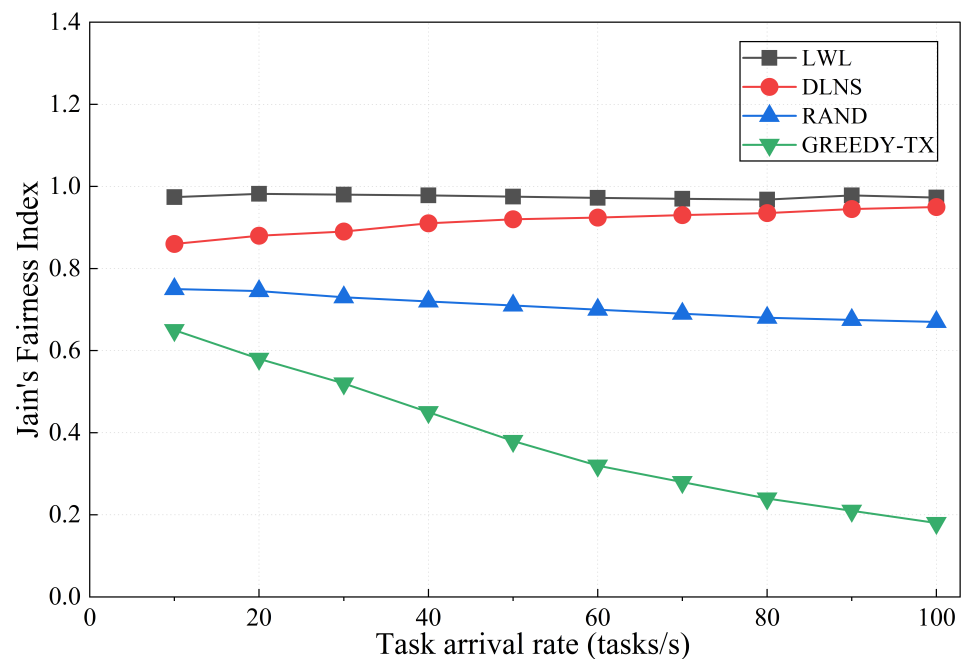


Figure 11. Comparison of Jain's Fairness Index under varying workload intensities.

6.2.3. Energy Efficiency Analysis

Energy consumption is a critical concern for network infrastructure, directly impacting operational costs and carbon footprint. Although our simulation primarily focuses on latency metrics, the proposed INC-assisted architecture inherently improves system-wide energy efficiency through two key mechanisms:

- **Reduction of Data Movement:** In traditional cloud-centric inference, raw data must traverse multiple hops across the core network, consuming forwarding energy at every intermediate device. In contrast, our mechanism enables proximal computing, processing tasks at INC nodes near the source. Furthermore, the decision-point identification strategy (described in Section 5.1) ensures that only contracted intermediate features—which are significantly smaller than the raw input—are transmitted. As demonstrated in Figure 8, our approach drastically reduces the total task transmission time (e.g., by 61.8% for VGG16 under peak load). Since data transmission is a major consumer of energy in network interfaces, this reduction in traffic volume directly translates to lower aggregate energy consumption.
- **Trade-off between Computing and Transmission:** While executing tasks on INC nodes incurs computational energy costs, this is effectively offset by the substantial savings in long-haul transmission energy. Previous studies [23] have indicated that in-network processing is often more energy-efficient than transmitting large payloads to centralized data centers, especially for deep learning tasks where layer-wise execution progressively reduces data volume. By optimizing the trade-off between local process-

ing and transmission, the proposed DLNS algorithm minimizes the total active time of network resources, thereby enhancing overall energy efficiency.

7. Conclusions

To address the challenges of high inference latency and inefficient resource utilization encountered by existing edge-cloud collaboration schemes, this paper proposes an INC-assisted collaborative inference framework overlaid on an ICN architecture. By integrating a decision-point identification mechanism with ICN's name-based resolution, the proposed framework enables fine-grained, hop-by-hop task offloading that dynamically exploits underutilized off-path computing resources beyond the limitations of rigid IP routing. We further design a Distributed Load and Network-aware Scheduling algorithm, in which each INC node independently determines the optimal next-hop execution node based on a look-ahead estimation of processing and transmission latencies. Under dynamic network conditions, the proposed approach effectively trades off communication costs for distributed computational gains. Simulation results demonstrate that the proposed framework significantly reduces end-to-end inference latency while maintaining robust performance under high concurrency. In typical scenarios, our approach reduces the average task completion time by approximately 35% compared to the best-performing baseline architectures. Moreover, the DLNS algorithm achieves superior load balancing, maintaining a Jain's Fairness Index above 0.86 even under peak traffic loads, thereby avoiding the local hotspots common in greedy strategies. In summary, the proposed mechanism offers a collaborative inference solution that balances low latency, high adaptability, and distributed scalability, making it well-suited for deploying complex AI services in heterogeneous wide-area networks.

Despite the promising performance of the proposed framework, several open issues remain for future exploration. First, the current mechanism primarily targets CNN models with linear topologies (e.g., AlexNet, VGG). However, modern deep learning models, such as ResNet and Inception, feature complex DAG structures with parallel branches. The current sequential offloading strategy does not fully exploit the potential for parallel execution across multiple nodes. Second, the DLNS algorithm makes decisions based on instantaneous snapshots of network states. In highly volatile networks, this lack of predictive modeling for traffic trends might lead to suboptimal decisions during sudden bursts. Future work will extend the framework to support DAG-based model partitioning and incorporate lightweight traffic prediction techniques [37] to further enhance the robustness and throughput of collaborative inference. Third, while this work validates architectural scalability via large-scale simulations, we plan to implement a small-scale hardware prototype using programmable data planes (e.g., P4-based SmartNICs) and high-performance packet processing frameworks (e.g., DPDK) to evaluate protocol-stack overheads and validate the system in real-world environments. This prototype will also enable a fair end-to-end comparison with practical inference serving/orchestration stacks (e.g., Kubernetes-based edge scheduling, NVIDIA Triton, and Ray Serve).

Author Contributions: Conceptualization, K.H., H.D. and B.M.; Methodology, K.H. and H.D.; Data curation, K.H.; Formal analysis, K.H. and H.D.; Funding acquisition, H.D.; Investigation, K.H., H.D. and B.M.; Project administration, H.D.; Resources, K.H.; Software, K.H.; Supervision, H.D.; Validation, K.H. and H.D.; Visualization, K.H.; Writing—original draft, K.H.; Writing—review & editing, K.H. and H.D. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Independent Deployment Project of the Institute of Acoustics, Chinese Academy of Sciences: Research on Secure Transmission Technology in Open Network Environments (Project No. E555180401).

Data Availability Statement: The original data presented in this study are included within the article. Additional data can be obtained from the first author or corresponding author on reasonable request.

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

References

1. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. <https://doi.org/10.1186/s40537-021-00444-8>.
2. Sahu, M.; Dash, R. A survey on deep learning: Convolution neural network (CNN). In *Intelligent and Cloud Computing: Proceedings of ICICC 2019*; Springer: Singapore, 2020; Volume 2, pp. 317–325.
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. <https://doi.org/10.1145/3065386>.
4. Son, J.; Buyya, R. Priority-aware VM allocation and network bandwidth provisioning in software-defined networking (SDN)-enabled clouds. *IEEE Trans. Sustain. Comput.* **2018**, *4*, 17–28. <https://doi.org/10.1109/TSUSC.2018.2842074>.
5. 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. <https://doi.org/10.1016/j.sysarc.2019.02.009>.
6. Xu, Z.; Zhao, L.; Liang, W.; Rana, O.F.; Zhou, P.; Xia, Q.; Xu, W.; Wu, G. Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 799–814. <https://doi.org/10.1109/TPDS.2020.3032443>.
7. Ren, W.Q.; Qu, Y.B.; Dong, C.; Jing, Y.Q.; Sun, H.; Wu, Q.H.; Guo, S. A survey on collaborative DNN inference for edge intelligence. *Mach. Intell. Res.* **2023**, *20*, 370–395. <https://doi.org/10.1007/s11633-022-1391-7>.
8. Ng, N.; Souza, A.; Diggavi, S.; Suri, N.; Abdelzaher, T.; Towsley, D.; Shenoy, P. Collaborative Inference in Resource-Constrained Edge Networks: Challenges and Opportunities. In Proceedings of the MILCOM 2024–2024 IEEE Military Communications Conference (MILCOM), Washington, DC, USA, 28 October–1 November 2024; IEEE: New York, NY, USA, 2024; pp. 1–6.
9. Zheng, X.; Zhang, W.; Hu, C.; Zhu, L.; Zhang, C. Cloud-Edge-End Collaborative Inference in Mobile Networks: Challenges and Solutions. *IEEE Netw.* **2025**, *39*, 90–96. <https://doi.org/10.1109/MNET.2025.3533581>.
10. Huang, X.; Lei, B.; Wei, M.; Ji, G.; Lv, H. Task value aware optimization of routing for computing power network. In Proceedings of the 2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Beijing, China, 14–16 June 2023; IEEE: New York, NY, USA, 2023; pp. 1–6.
11. Tang, S.; Yu, Y.; Wang, H.; Wang, G.; Chen, W.; Xu, Z.; Guo, S.; Gao, W. A survey on scheduling techniques in computing and network convergence. *IEEE Commun. Surv. Tutor.* **2023**, *26*, 160–195.
12. Xue, M.; Wu, H.; Li, R.; Xu, M.; Jiao, P. EosDNN: An efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments. *IEEE Trans. Green Commun. Netw.* **2021**, *6*, 248–264. <https://doi.org/10.1109/TGCN.2021.3111731>.
13. Mohammed, T.; Joe-Wong, C.; Babbar, R.; Di Francesco, M. Distributed inference acceleration with adaptive DNN partitioning and offloading. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; IEEE: New York, NY, USA, 2020; pp. 854–863. <https://doi.org/10.1109/INFOCOM41043.2020.9155237>.
14. Yang, L.; Zheng, C.; Shen, X.; Xie, G. Ofpcnn: On-demand fine-grained partitioning for cnn inference acceleration in heterogeneous devices. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 3090–3103. <https://doi.org/10.1109/TPDS.2023.3321755>.
15. Mohajer, A.; Hajipour, J.; Leung, V.C. Dynamic offloading in mobile edge computing with traffic-aware network slicing and adaptive TD3 strategy. *IEEE Commun. Lett.* **2024**, *29*, 95–99. <https://doi.org/10.1109/LCOMM.2024.3501956>.
16. Amadeo, M.; Ruggeri, G. Exploring In-Network Computing with Information-Centric Networking: Review and Research Opportunities. *Future Internet* **2025**, *17*, 42. <https://doi.org/10.3390/fi17010042>.
17. Kang, Y.; Hauswald, J.; Gao, C.; Rovinski, A.; Mudge, T.; Mars, J.; Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Comput. Archit. News* **2017**, *45*, 615–629. <https://doi.org/10.1145/3093337.3037698>.
18. Hu, C.; Bao, W.; Wang, D.; Liu, F. Dynamic adaptive DNN surgery for inference acceleration on the edge. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; IEEE: New York, NY, USA, 2019; pp. 1423–1431. <https://doi.org/10.1109/INFOCOM.2019.8737614>.
19. Huang, Y.; Wang, F.; Wang, F.; Liu, J. DeePar: A hybrid device-edge-cloud execution framework for mobile deep learning applications. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Paris, France, 29 April–2 May 2019; IEEE: New York, NY, USA, 2019; pp. 892–897. <https://doi.org/10.1109/INFCOMW.2019.8845240>.
20. Xu, Y.; Mohammed, T.; Di Francesco, M.; Fischione, C. Distributed assignment with load balancing for DNN inference at the edge. *IEEE Internet Things J.* **2022**, *10*, 1053–1065. <https://doi.org/10.1109/JIOT.2022.3205410>.

21. Shi, Y.; Yi, C.; Wang, R.; Wu, Q.; Chen, B.; Cai, J. Service migration or task rerouting: A two-timescale online resource optimization for MEC. *IEEE Trans. Wirel. Commun.* **2023**, *23*, 1503–1519.
22. Yi, C.; Cai, J.; Su, Z. A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. *IEEE Trans. Mob. Comput.* **2019**, *19*, 29–43.
23. Hu, N.; Tian, Z.; Du, X.; Guizani, M. An energy-efficient in-network computing paradigm for 6G. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 1722–1733. <https://doi.org/10.1109/TGCN.2021.3099804>.
24. Spina, M.; Lebron, D.S.; Schwarzmann, S.; Guerzoni, R.; Trivisonno, R.; Iera, A.; De Rango, F.; Karetos, G.; Zinner, T.; Corujo, D. In-Network Computing and Split-AI in 6G: Enablers and Proof-of-Concept Studies. In Proceedings of the 2024 3rd International Conference on 6G Networking (6GNet), Paris, France, 21–24 October 2024; IEEE: New York, NY, USA, 2024; pp. 135–143. <https://doi.org/10.1109/6GNet63182.2024.10765651>.
25. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Claffy, K.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named data networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. <https://doi.org/10.1145/2656877.2656887>.
26. Jacobson, V.; Smetters, D.K.; Thornton, J.D.; Plass, M.F.; Briggs, N.H.; Braynard, R.L. Networking named content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, Rome, Italy 1–4 December 2009; pp. 1–12. <https://doi.org/10.1145/1658939.1658941>.
27. Raychaudhuri, D.; Nagaraja, K.; Venkataramani, A. Mobilityfirst: A robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **2012**, *16*, 2–13. <https://doi.org/10.1145/2412096.2412098>.
28. Dannewitz, C.; Kutscher, D.; Ohlman, B.; Farrell, S.; Ahlgren, B.; Karl, H. Network of information (netinf)—an information-centric networking architecture. *Comput. Commun.* **2013**, *36*, 721–735. <https://doi.org/10.1016/j.comcom.2013.01.009>.
29. Wang, J.; Chen, G.; You, J.; Sun, P. Seanet: Architecture and technologies of an on-site, elastic, autonomous network. *J. Netw. New Media* **2020**, *6*, 1–8.
30. Dang, S.; Han, R. An In-Network Cooperative Storage Schema Based on Neighbor Offloading in a Programmable Data Plane. *Future Internet* **2021**, *14*, 18. <https://doi.org/10.3390/fi14010018>.
31. Xu, Y.; Ni, H.; Zhu, X. A Novel Multipath Transmission Scheme for Information-Centric Networking. *Future Internet* **2023**, *15*, 80.
32. Li, B.; Wang, J. An identifier and locator decoupled multicast approach (ILDm) based on ICN. *Appl. Sci.* **2021**, *11*, 578.
33. Li, J.; Sheng, Y.; Deng, H. Two optimization algorithms for name-resolution server placement in information-centric networking. *Appl. Sci.* **2020**, *10*, 3588.
34. Riley, G.F.; Henderson, T.R. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 15–34.
35. Yu, F.; Chen, H.; Wang, X.; Xian, W.; Chen, Y.; Liu, F.; Madhavan, V.; Darrell, T. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv* **2018**, arXiv:1805.04687.
36. Yang, R.; Wang, Z.; Yang, Y.; Wang, S. A DRL-Based Algorithm for DNN Partition, Subtask Offloading and Resource Allocation in Multi-Hop Computing Nodes with Cloud. *IET Commun.* **2025**, *19*, e70048.
37. Le, V. A.; Le Nguyen, P.; Ji, Y. Deep convolutional LSTM network-based traffic matrix prediction with partial information. In Proceedings of the 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Washington, DC, USA, 8–12 April 2019; IEEE: New York, NY, USA, 2019; pp. 261–269.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.