EARLY-EXIT-AWARE PARTITIONING FOR DISTRIBUTED DNN INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

An early-exit strategy allows samples to exit when a certain confidence threshold is reached at intermediate layers of the model. However, distributing early-exit networks across multiple machines introduces unique complications. Applying naive layer-wise partitioning can negate the advantages of early exits: unlike conventional model parallelism, where all inputs traverse every partition, the volume of data reaching downstream machines varies. This produces an uneven workload across the cluster. Addressing this imbalance requires jointly deciding where to place early exits and how to partition and distribute the network across nodes. In this paper, we propose a load-balanced, distributed, model-parallel inference strategy that allocates compute capacity in proportion to the inference load at each stage. We evaluate our approach across multiple CNN architectures (DenseNet-121, ResNet-50, and AlexNet) on standard benchmark datasets (ImageNet, CIFAR-100, and CIFAR-10). Experimental results demonstrate that our architecture significantly reduces inference latency and outperforms conventional model partitioning.

1 INTRODUCTION

Deep convolutional neural networks (CNNs) have achieved high accuracy across a wide range of computer vision tasks, but their increasing depth and computational complexity also make inference increasingly expensive. In a conventional CNN, every sample passes through the complete network regardless of the difficulty of the individual prediction. Consequently, computational resources may be spent processing samples that could already be classified confidently at an earlier stage of the network.

Early-exit neural networks provide a mechanism to reduce this unnecessary computation by introducing intermediate classifiers into the backbone architecture (Teerapittayanon et al., 2016; Bolukbasi et al., 2017). When an intermediate classifier produces a sufficiently confident prediction, inference can terminate without executing the remaining layers (Chen et al., 2020; Bonato & Bouganis, 2021). Samples for which the prediction is less confident continue to deeper stages of the network. Thus, the computational cost of inference varies dynamically according to the difficulty of each sample rather than requiring every sample to traverse the complete model (Rahmath P et al., 2024).

Although early exiting can reduce computation, distributing an early-exit network across multiple computing nodes introduces additional system challenges (Teerapittayanon et al., 2017; Ebrahimi et al., 2022). When prediction does not terminate at an early exit, intermediate feature representations must be serialized and transferred across the network (Zeng et al., 2020; Stahl et al., 2021). Consequently, distributed execution introduces communication, serialization, synchronization, and scheduling overheads alongside neural network computation (Laskaridis et al., 2020; Ju et al., 2021). Therefore, FLOP reductions obtained through early exiting do not automatically translate into end-to-end latency speedups (Baccarelli et al., 2021; Colocrese et al., 2024).

This work investigates the performance dynamics of distributed early-exit inference under diverse execution strategies. We systematically study how early-exit rates, batch sizes, backbone architectures, and distributed topologies interact to govern end-to-end throughput. We evaluate three distinct vision architectures: AlexNet on CIFAR-10, DenseNet on CIFAR-100, and ResNet-50 on ImageNet. Across these workloads, we evaluate single-node baselines alongside serial, concurrent and replicated distributed execution paradigms spanning up to five physical machines.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Our empirical findings reveal three key system insights: First, the latency benefits of dynamic early exits depend heavily on backbone depth and feature map dimensionality. Second, changing the early-exit rate shifts the system bottleneck between communication cost and local computation. Third, while concurrent stream replication provides substantial throughput gains and higher hardware utilization, adding compute nodes without capacity matching can induce severe queueing backpressure. To mitigate interconnect bottlenecks, we introduce *Dynamic Stage In-lining*, Building upon principles of dynamic edge-cloud task throttling (Kang et al., 2017; Laskaridis et al., 2020) and runtime GPU memory paging (Rhu et al., 2016), an adaptive runtime mechanism that caps network queueing by dynamically executing overflow micro-batches locally.

The main contributions of this work are summarized as follows:

- We propose a modular framework for distributed early-exit inference that incorporates intermediate stage replication and concurrent stage execution to eliminate pipeline bubbles across compute nodes.
- We formalize the latency dynamics and flow relations governing distributed early-exit systems, establishing capacity-matching conditions (R_k) to prevent network backpressure.
- We present a multi-workload empirical study evaluating AlexNet, DenseNet, and ResNet-50 across diverse execution topologies under bandwidth-constrained interconnects, showing that combining adjustable early-exit rates with *Dynamic Stage In-lining* converts network stalls into net speedups.

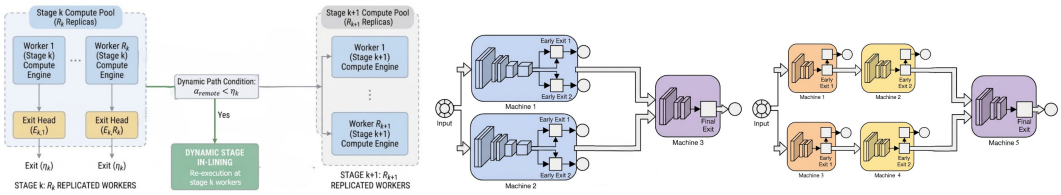
2 Related Work

Our work intersects three primary domains of research: algorithmic early-exit architectures, distributed and partitioned neural network inference, and hybrid paradigms that combine early exits with distributed processing.

Early-Exit Neural Networks. Dynamic early exiting was pioneered by Teerapittayanon et al. (2016) through *BranchyNet*, which introduced auxiliary intermediate classification heads into feed-forward backbones to allow confident samples to terminate early. Bolukbasi et al. (2017) formalized adaptive inference by modeling early exiting as an optimal stopping problem subject to resource constraints. Subsequent studies expanded early-exit training and decision policies: Chen et al. (2020) developed joint learning frameworks that optimize prediction and exit criteria simultaneously, while Bonato & Bouganis (2021) proposed class-specific exit strategies to specialize intermediate heads based on semantic difficulty. Beyond computer vision backbones, early-exit strategies have been adapted to specialized domains such as graph neural networks for hyperspectral image classification (Haseena Rahmath & Chaurasia, 2023). Comprehensive taxonomy reviews by Rahmath P et al. (2024) classify state-of-the-art exit policies, highlighting that while algorithmic FLOP reductions are well-established on single nodes, their runtime translation into distributed latency speedups remains tightly constrained by system overheads.

Distributed and Partitioned DNN Inference. Distributing deep neural network inference across multiple computing nodes is widely studied as a technique to overcome hardware memory and compute constraints. Kang et al. (2017) introduced *Neurosurgeon*, establishing fine-grained layer-wise partitioning between edge devices and the cloud based on dynamic network conditions and layer execution costs. Building on collaborative execution, Zeng et al. (2020) presented *CoEdge*, an adaptive workload partitioning scheme over heterogeneous edge devices. Similarly, Stahl et al. (2021) proposed *DeeperThings*, executing fully distributed CNN inference on highly constrained edge nodes via spatially partitioned feature maps. While these frameworks optimize spatial or layer-wise pipeline parallelism, they generally assume static model architectures where every sample traverses all layers, leaving system pipelines vulnerable to interconnect bottlenecks when processing high-resolution input tensors.

Distributed Early-Exit and Edge-Cloud Co-Inference. Recent literature has increasingly focused on combining model partitioning with dynamic early exits to optimize distributed inference. Teerapittayanon et al. (2017) extended early exiting to distributed hierarchies, introducing *DDNN* across cloud, edge, and end devices. Laskaridis et al. (2020) developed *SPINN*, a synergistic progressive inference engine that dynamically balances local early execution against cloud offloading under



(a) System architecture (b) Multi-Stream-3 (c) Multi-Stream-5

Figure 1: Overview of the distributed early-exit pipeline architecture.

time-varying wireless conditions. Similarly, Ju et al. (2021) formulated dynamic early-exit scheduling using contextual bandits to optimize latency in volatile edge environments, while Baccarelli et al. (2021) introduced *Learning-in-the-Fog* (LiFo) to minimize energy consumption in delay-critical IoT deployments.

More recently, Ebrahimi et al. (2022) evaluated the structural trade-offs of combining layer partitioning with early exit heads, and Ilhan et al. (2024) introduced *EENet* to adaptively optimize multi-exit execution. Specialized edge topologies have also been explored by Peng et al. (2024) with *DistrEE* for edge device clusters, Colocrese et al. (2024) for model-distributed inference across edge networks, and Bajpai & Hanawal (2025) through an early-exit-based clustering approach for mobile edge-cloud systems.

While recent distributed early-exit systems like *Apparate* (Dai et al., 2024) run early-exited samples through the full backbone to preserve batch structures, our system executes true early exits, pruning terminated tensors immediately to eliminate redundant downstream compute. Furthermore, unlike predictive scheduling approaches (Iyer & Gandhi, 2025) that buffer intermediate activation fragments to reconstruct static micro-batch sizes, our downstream nodes are completely batch-size agnostic, executing dynamically on incoming payloads of arbitrary size ($N \in \{4, 7, 397\}$). By decoupling execution from rigid batching and pairing true early exits with *Dynamic Stage In-lining* (α_{remote}) and VRAM weight swapping, our framework balances inter-node communication against local GPU capacity to maximize hardware utilization and overall pipeline throughput.

Unlike prior works that primarily focus on high-level wireless task offloading policies, static partition cuts, or edge scheduling heuristics, our work systematically addresses the low-level system bottlenecks of distributed early-exit pipelines. By explicitly modeling activation serialization, micro-batch size interactions, multi-stream capacity matching (R_k), and interconnect queue backpressure, our framework introduces *Dynamic Stage In-lining* with load-factor capping (α_{remote}) and on-demand VRAM weight swapping to eliminate live interconnect stalls in asynchronous distributed pipelines.

3 Preliminaries: Early-Exit Neural Networks

An early-exit neural network (EENN) augments a standard feedforward backbone architecture by inserting auxiliary classification heads at intermediate layers. Let $x \in \mathcal{X}$ denote an input tensor propagated through a backbone model decomposed into K sequential compute stages. Intermediate classification heads E_k ($k \in \{1, \ldots, K-1\}$) are attached at intermediate feature representations, producing prediction vectors $\hat{y}_k = E_k(x)$ and corresponding confidence scores $c_k(x) = \max \text{softmax}(\hat{y}_k)$.

Dynamic inference is governed by confidence thresholding. At stage k, if the confidence score satisfies $c_k(x) \geq \tau_k$ for a predefined threshold $\tau_k \in [0, 1]$, inference terminates immediately and prediction $\hat{y}_k$ is returned. Unresolved samples where $c_k(x) < \tau_k$ are forwarded to stage $k+1$. Samples failing all intermediate thresholds ($\tau_1, \ldots, \tau_{K-1}$) propagate to the final classifier $E_{\text{final}} = E_K$. The empirical early-exit rate $\eta_k \in [0, 1]$ measures the fraction of workload samples terminating at stage k. Elevating η_k reduces downstream Floating Point Operations (FLOPs); however, in distributed topologies, exit rates also dictate inter-node data volume, as non-exiting samples require network serialization while exiting samples terminate local execution.

4 System Methodology and Theoretical Framework

To characterize and optimize distributed early-exit execution, we formalize the system dynamics governing execution latency, network transfer overheads, and capacity matching across our proposed distributed execution paradigms.

4.1 Taxonomy of Distributed Execution Paradigms

To isolate algorithmic computational savings from communication overheads, we categorize distributed early-exit execution across six distinct operational paradigms. The baseline execution profile includes two single-node paradigms: a *Single-Node Baseline* baseline ($\tau_k = 1.0, \forall k$) evaluating full-model computation without early termination to establish execution latency, and a *Single-Node Early Exit* adaptive baseline isolating raw algorithmic FLOP reductions without network overhead.

For distributed topologies, the *Sequential Pipeline* (*Sequential-2* and *Sequential-3*) distributes backbone stages serially across $M \in \{2, 3\}$ nodes, where stage k transmits unresolved intermediate tensors over the interconnect to Node $k+1$. To eliminate serial dependency stalls, the *Replicated Multi-Stream* (*Multi-Stream-3* and *Multi-Stream-5*) topologies decouples intermediate classifiers (E_1, E_2) across parallel worker nodes, asynchronously streaming unresolved feature maps to the final stage node, preventing intermediate queueing bottlenecks and maximizing steady-state throughput.

4.2 Sequential Pipeline Dynamics

Consider a backbone network partitioned serially across M compute nodes. In a distributed inference pipeline without early exits, every workload traverses the full network depth. Consequently, every node processes an identical workload volume, making steady-state throughput tightly bounded by the un-overlapped summation of stage computation and network transfer times.

Introducing sequential early exits reduces this baseline cost: because a fraction η_k of samples terminates at intermediate classifiers within stage $k \in \{1, \ldots, M\}$, the volume of data transmitted to Stage $k+1$ shrinks. This reduction lowers both the communication overhead $T_{\text{comm}}^{(k \to k+1)}$ and the downstream computational load $T_{\text{comp}}^{(k+1)}$, allowing sequential early-exit inference to outperform standard non-early-exit distributed baselines. Because exiting samples terminate execution at stage k, they require zero network transmission and stage k immediately proceeds to process the next incoming workload. Conversely, the remaining $(1 - \eta_k)$ fraction of unresolved activation tensors must be serialized and transmitted across the interconnect.

However, in this sequential model, downstream stage $k+1$ experiences severe structural idleness. Because stage $k+1$ depends directly on the output stream of stage k, whenever a sample or batch exits early at stage k, no intermediate data is transmitted. This creates dynamic starvation gaps, leaving stage $k+1$ completely idle until an unresolved tensor is eventually forwarded.

When a non-exiting workload propagates across all stages, the end-to-end processing latency T_{seq} accumulates as:

$$T_{\text{seq}} = \sum_{k=1}^{M} T_{\text{comp}}^{(k)} + \sum_{k=1}^{M-1} T_{\text{comm}}^{(k \to k+1)} \tag{1}$$

Equation 1 highlights the core paradox of sequential early-exit pipelines: while early exits reduce net compute and transmission volume compared to standard pipelines, they introduce unpredictable idle cycles downstream, while network transfer latency T_{comm} remains an additive penalty on non-exiting paths.

4.3 Asynchronous Pipelining and Latency Hiding

To eliminate the structural idle periods and dynamic starvation gaps inherent to sequential execution, our proposed concurrent execution paradigms decouple processing stages into independent asynchronous streams. By continuously streaming micro-batches across compute nodes, the asynchronous pipeline fills the downstream idle gaps created by upstream early exits. As a result, net-

work transfer time $T_{\text{comm}}^{(k\to k+1)}$ and downstream compute duration $T_{\text{comp}}^{(k+1)}$ overlap directly behind active computation at preceding stages $T_{\text{comp}}^{(k)}$.

In steady-state asynchronous execution, effective step latency T_{step} is no longer governed by additive summation, but by the maximum duration among parallel operational phases:

$$T_{\text{step}} = \max\left(T_{\text{comp}}^{(k)},\ T_{\text{comm}}^{(k\to k+1)} + T_{\text{comp}}^{(k+1)}\right) \tag{2}$$

When $T_{\text{comp}}^{(k)} \geq T_{\text{comm}}^{(k\to k+1)} + T_{\text{comp}}^{(k+1)}$, communication latency and downstream evaluation are fully concealed within the compute window of stage k. By converting what would otherwise be downstream idle time into useful parallel processing windows, asynchronous execution achieves near-zero network overhead and maximizes pipeline throughput.

4.4 Queueing Bottlenecks, Backpressure, and Capacity Matching

Replicating intermediate processing stages increases compute capacity but alters inter-stage arrival rates. Let R_k denote the number of parallel worker instances assigned to stage k. The processing service capacity μ_k for stage k scales as $\mu_k = R_k/T_{\text{comp}}^{(k)}$. However, the data arrival rate λ_{k+1} entering downstream stage $k{+}1$ depends on the non-exiting sample stream from stage k, establishing the flow relation:

$$\lambda_{k+1} = (1-\eta_k)\cdot\frac{R_k}{T_{\text{comp}}^{(k)}} \tag{3}$$

A critical queueing imbalance occurs if the arrival rate λ_{k+1} exceeds the downstream service capacity μ_{k+1}, inclusive of communication ingestion overhead:

$$(1-\eta_k)\cdot\frac{R_k}{T_{\text{comp}}^{(k)}} > \frac{R_{k+1}}{T_{\text{comm}}^{(k\to k+1)} + T_{\text{comp}}^{(k+1)}} \tag{4}$$

When Inequality 4 holds, activation tensors accumulate rapidly in input buffers at node $k+1$. Upon reaching buffer capacity, upstream nodes experience backpressure, halting stage k and introducing queuing delays T_{queue} that degrade end-to-end throughput.

Optimal pipeline stability requires balancing worker allocations (R_k, R_{k+1}) against early-exit rates (η_k) to achieve steady-state capacity matching:

$$(1-\eta_k)\cdot\frac{R_k}{T_{\text{comp}}^{(k)}} \approx \frac{R_{k+1}}{T_{\text{comm}}^{(k\to k+1)} + T_{\text{comp}}^{(k+1)}} \tag{5}$$

Equation 5 defines the core theoretical principle of distributed early-exit inference: maximum efficiency requires jointly tuning exit thresholds (η_k) and worker replication factors (R_k) to equalize inter-stage flow rates, thereby concealing communication latency without triggering downstream queue congestion.

4.5 Stage-Exit Assignment and Conditional Evaluation

Model partitioning cuts are aligned strictly with early-exit boundary locations to prevent redundant serialization across network boundaries. Furthermore, intra-stage exit heads are evaluated conditionally based on single-machine profiling. If a distributed node contains multiple co-located early-exit heads, all heads are evaluated sequentially only when their combined inference overhead is lower than local backbone execution time. For lightweight backbones where intermediate classifier overhead exceeds local latency improvements, non-performing intra-stage exit heads are disabled, retaining only the terminal classifier of that stage to preserve local execution efficiency.

4.6 Mitigating Communication Bottlenecks via Dynamic Stage In-lining

When early-exit rates drop below capacity-matching thresholds ($\eta < 50\%$) or when processing high-dimensional inputs over bandwidth-constrained interconnects, asynchronous concurrent pipelines become severely communication-bound. In this regime, the transmission delay of non-exiting activation tensors across the network (T_{comm}) exceeds the computational execution window of local

stages (T_{comp}). As a result, non-exiting activation tensors rapidly accumulate in outgoing serialization buffers, creating network stalls that force preceding compute workers to idle and degrading pipeline concurrency.

To eliminate these network-bound bottlenecks without requiring dedicated high-speed hardware, we propose and evaluate *Dynamic Stage In-lining* (Figure 1a). This runtime mechanism dynamically adapts model execution boundaries based on an offload threshold, α_{remote}, which defines the maximum allowable fraction of batch samples permitted to be offloaded to remote pipeline stages. During Stage k inference, if the proportion of non-exiting samples exceeds α_{remote}, the local worker caps remote transfers at this threshold. Excess activation tensors are retained locally, and the corresponding samples are appended to a deferred local processing queue. Upon completing Stage k feature extraction for the batch sequence, the node dynamically unloads Stage k model weights from VRAM and loads Stage k+1 parameters to execute deferred inference locally. Empirically, enforcing a remote load factor α_{remote} acts as an adaptive software valve that bounds network traffic to match interconnect capacity.

5 Experiments and Analysis

To validate our theoretical formulation and demonstrate the operational efficiency of asynchronous distributed early-exit execution, we evaluate single-machine baselines, serial pipeline bottlenecks, multi-stream replication, and dynamic stage in-lining.

5.1 Experimental Setup and System Environment

Workloads and Models. We evaluate three vision architectures covering diverse model depths, feature map dimensionalities, and computational densities: AlexNet on CIFAR-10 (32×32), DenseNet on CIFAR-100 (32×32), and ResNet-50 on ImageNet (224×224). Each model is augmented with two auxiliary classification heads (E_1, E_2).

Deployment Paradigms. Experiments are conducted across six execution configurations: (1) *Single-Node Baseline*, a single-node upper-bound baseline where intermediate classifiers are evaluated but all inferences route to the final exit; (2) *Single-Node Early Exit*, an adaptive single-node baseline terminating samples via confidence thresholds; (3) *Sequential-2*, a serial 2-node pipeline partitioning intermediate exits to Node 1 and final execution to Node 2; (4) *Sequential-3*, a serial 3-node pipeline partitioning E_1 to Node 1, E_2 to Node 2, and final execution to Node 3; (5) *Multi-Stream-3* (Figure 1b), an asynchronous replicated multistream topology deploying co-located E_1/E_2 classifiers across Nodes 1 and 2, which concurrently process micro-batches and stream non-exiting features to Node 3; and (6) *Multi-Stream-5* (Figure 1c), a replicated multi-stream topology assigning two parallel workers to E_1 (Nodes 1, 3), two to E_2 (Nodes 2, 4), and remaining samples passed to E_5 (Node 5).

Hardware Environment and Sweep Parameters. All nodes are equipped with NVIDIA Quadro RTX 4000 GPUs (8 GB VRAM) interconnected via a dedicated 10 Gbps Ethernet fabric. For AlexNet and DenseNet, batch sizes span $B \in \{8, 16, 32, 64, 128, 256, 512, 1024\}$; for ResNet-50, $B \in \{8, 16, 32\}$ due to GPU memory constraints at larger batch sizes.

Table 1: Classification accuracy across early-exit target rates (η). Target rates selected for downstream system latency evaluations reflect the optimal accuracy-compute trade-off point for each architecture.

Setup	$\eta = 0\%$	$\eta \approx 30\%$	$\eta \approx 50\%$	$\eta \approx 80\%$	$\eta \approx 90\%$
AlexNet CIFAR-10	84%	83%	83%	**82%**	77%
DenseNet CIFAR-100	76%	76%	76%	**74%**	70%
ResNet-50 ImageNet	83% / 96%	77% / 93%	**76% / 92%**	62% / 85%	51% / 77%

5.2 SINGLE-NODE ALGORITHMIC CHARACTERIZATION

We establish single-node baselines by comparing standard execution against early-exit evaluation. For lightweight models like AlexNet on CIFAR-10, early exits yield negligible speedup. The FLOP savings from skipping deeper layers are neutralized by the kernel launch overheads of evaluating intermediate heads on low-resolution (32×32) feature maps.

In contrast, deeper architectures exhibit dramatic latency reductions under *Single-Node Early Exit* execution. DenseNet achieves its maximum runtime improvement at $B = 1024$, achieving a $1.62\times$ speedup relative to full-model execution. At $B = 8$, DenseNet exhibits a minor degradation $0.88\times$ caused by GPU under-utilization and launch overheads. For ResNet-50 on ImageNet, intermediate exits consistently accelerate inference across all batch sizes, yielding a $1.56\times$ speedup at $B = 32$. These findings confirm that computationally dense backbones extract substantial algorithmic savings from dynamic early exit.

Guided by these single-node profile results, we tailor node-level exit head activations for distributed deployment. Because intra-node execution on lightweight models yields no net speedup from intermediate terminations, we consolidate node-local processing by evaluating only the final exit head present on that specific machine, suppressing intermediate head evaluations to avoid unnecessary kernel overheads. Conversely, for compute-heavy architectures—where multi-exit execution yields substantial speedups across batch sizes—all intermediate exit heads are retained locally.

5.3 TWO-STAGE DISTRIBUTED TOPOLOGIES: SEQUENTIAL-2 VS. MULTI-STREAM-3

To evaluate latency hiding under single-hop network communication, we compare sequential execution against replicated multi-stream across target early-exit rates $\eta \in \{30\%, 50\%, 80\%, 90\%\}$. As detailed in Table 1, operating points are selected based on accuracy retention: $\eta \approx 80\%$ represents the primary operating regime for AlexNet (82.0% accuracy) and DenseNet (74.0% accuracy), whereas $\eta \approx 50\%$ is optimal for ResNet-50 (76.0% Top-1 accuracy). Analyzing the operational differentials across these architectures reveals how memory bandwidth, feature map dimensionalities, and serialization overheads dictate asynchronous latency hiding.

AlexNet Execution Dynamics. For AlexNet, the interaction between batch size scaling and network serialization dictates distinct performance inflection points. In low-exit regimes ($\eta \approx 30\%$), where heavy network serialization would typically choke the pipeline, our system deploys *Dynamic Stage In-lining* with a maximum remote load factor of $\alpha_{\text{remote}} = 0.50$, achieving a $1.50\times$ speedup. At a moderate early-exit rate of $\eta \approx 50\%$, *Multi-Stream-3* achieves peak speedups of $1.25\times$ over serial execution at intermediate batch sizes ($B \in \{32, 64\}$ Table 2). At these intermediate batch sizes, the compute window of the first stage is wide enough to completely overlap the network transfer time of non-exiting tensors without saturating local queue buffers. However, at extreme batch sizes ($B = 1024$) increased volume of serialized activation tensors creates mild network and buffer contention, eroding concurrency benefits and reducing the overall performance gain to a marginal $1.05\times$ speedup. As the early-exit rate is elevated to $\eta \approx 80\%$, the volume of data requiring network transmission drops significantly. By empirically balancing communication and computation overheads within our system setup and considering hardware utilization, an early-exit rate of $\eta \approx 80\%$ is identified as the optimal operational threshold for minimizing latency, and idle periods. This reduction in inter-node traffic expands the latency-hiding window, driving peak speedups to $2.18\times$ ($B = 32$). Pushing the exit rate further to $\eta \approx 90\%$ strips away nearly all downstream tensor traffic, maximizing concurrency and pushing peak speedups to $3.70\times$ ($B = 256$).

DenseNet Execution Dynamics. At $\eta \approx 30\%$, *Dynamic Stage In-lining* with a maximum remote load factor of $\alpha_{\text{remote}} = 0.50$, achieves a $2\times$ speedup. At $\eta \approx 50\%$, optimal speedups of $1.98\times$ are concentrated at $B = 32$. At $B = 8$, lowest speedup is seen with $1.5\times$ due to GPU kernel launch overheads underutilizing parallel streaming multiprocessors. As reported in Table 2, at higher termination targets ($\eta \approx 80\%$ and $\eta \approx 90\%$) , intermediate to large batch configurations ($B \in \{64, 128, 256, 512\}$) consistently yield robust speedups ranging from $2.0\times$ to $2.3\times$, hitting a maximum acceleration of $2.64\times$ at $B = 1024$. Because DenseNet's internal feature maps are compact yet computationally dense, the replicated multi-stream architecture successfully sustains high pipeline throughput across extended execution runs without triggering memory-induced backpressure.

Table 2: System throughput (samples per second) across early-exit target rates (η) under varying batch sizes and architectures. Measurements highlight the scalability and dynamic trade-offs between multi-exit models and batching strategies. Topologies are labeled Sequential (seq) or Multi-Stream (ms); the suffix is the number of nodes.

Setup	Batch Size	Architecture	$\eta \approx 50\%$	$\eta \approx 80\%$	$\eta \approx 90\%$
AlexNet CIFAR-10	8	seq-2	3560.47	5061.80	5144.35
		ms-3	4408.98	11109.63	12302.85
		seq-3	-	-	4295.42
		ms-5	-	-	12342.63
	32	seq-2	3560.16	7284.70	8184.65
		ms-3	4437.74	15869.49	26730.82
		seq-3	-	-	4306.71
		ms-5	-	-	19025.88
	1024	seq-2	3291.10	6458.78	8606.59
		ms-3	3471.26	13921.37	28475.43
		seq-3	-	-	4039.91
		ms-5	-	-	25168.63
DenseNet CIFAR-100	8	seq-2	850.46	1001.41	1441.73
		ms-3	1282.83	1621.41	2419.70
		seq-3	-	-	2447.19
		ms-5	-	-	4544.71
	32	seq-2	1681.65	2139.28	2285.68
		ms-3	3326.79	4801.09	5145.14
		seq-3	-	-	2817.16
		ms-5	-	-	5611.23
	1024	seq-2	1175.26	1914.74	2279.59
		ms-3	1984.75	5051.58	5605.51
		seq-3	-	-	2583.26
		ms-5	-	-	3634.30
ResNet-50 ImageNet	8	seq-2	125.36	320.15	493.56
		ms-3	112.52	283.58	742.18
		seq-3	-	-	400.90
		ms-5	-	-	1334.30
	16	seq-2	118.56	322.94	505.99
		ms-3	108.66	278.85	740.59
		seq-3	-	-	417.69
		ms-5	-	-	1378.48
	32	seq-2	106.99	294.66	510.46
		ms-3	70.33	262.01	708.11
		seq-3	-	-	426.15
		ms-5	-	-	1345.91

ResNet-50 Execution Dynamics. For ResNet-50 operating on ImageNet, performance is heavily constrained by spatial resolution scaling. The large 224×224 input resolution yields high-dimensional intermediate activation tensors that quickly saturate the bandwidth limits of a commodity 10 Gbps Ethernet fabric, shifting the system into a severely communication-bound regime. At $\eta \approx 50\%$—which yields the optimal accuracy trade-off for ResNet-50— Table 2 reveals that *Multi-Stream-3* suffers a performance degradation compared to *Sequential-2*. The volume of non-exiting activation tensors transmitted across the network remains high enough that communication latency T_{comm} completely dominates local stage execution time T_{comp}, resulting in degraded performance ($0.66\times$ to $0.92\times$). To reconcile this tension and recover performance at the optimal $\eta \approx 50\%$ threshold, our architecture deploys *Dynamic Stage In-lining* . By dynamically executing downstream stages locally during network saturation, in-lining bypasses the interconnect bottleneck entirely achieves a $3.98\times$ speedup, enabling low-latency inference without sacrificing accuracy. While increasing the exit rate to $\eta \approx 80\%$ and $\eta \approx 90\%$ acts as a software-level bandwidth relief valve. At $\eta \approx 90\%$ communication bottleneck is overcome, achieving a $1.5\times$ speedup.

"Note that Table 2 reports steady-state system throughputs under static multi-stream routing ($\eta \in \{50\%, 80\%, 90\%\}$). Because Dynamic Stage In-lining is an adaptive runtime mechanism triggered specifically under severe serialization bottlenecks, its performance recoveries are evaluated under dynamic traffic bursts and reported in Section 5.3 rather than static benchmarking. To keep the table compact, representative batch sizes are reported for AlexNet and DenseNet "

5.4 THREE-STAGE DISTRIBUTED TOPOLOGIES: SEQUENTIAL-3 VS. MULTI-STREAM-5

To evaluate multi-hop pipeline scaling, we benchmark a serial 3-node pipeline against a 5-node replicated multi-stream execution. Over commodity Ethernet interconnects, cumulative serialization latency across two network hops (Node 1 $\rightarrow$ Node 2 $\rightarrow$ Node 3) creates severe pipeline bubbles unless non-exiting data volume is strictly minimized, requiring high early-exit rates ($\eta \approx 90\%$) to maintain capacity matching. 5.

Under $\eta \approx 90\%$, multi-stream worker replication in *Multi-Stream-5* successfully absorbs incoming traffic bursts and eliminates downstream queueing bottlenecks across all backbones. For AlexNet, multi-stream replication achieves a peak speedup of $6.23\times$ at $B = 1024$. DenseNet achieves a peak speedup of $1.99\times$ at $B = 32$. Finally, ResNet-50 reaches its peak speedup of $3.33\times$ at $B = 8$. These results confirm that allocating parallel worker instances ($R_k > 1$) effectively prevents backpressure stalls even under multi-hop interconnect constraints.

6 DISCUSSION

Our findings show that in distributed environments, early exiting acts primarily as a *network bandwidth management protocol*. By terminating inference early, it converts network serialization delay (T_{comm}) into local compute (T_{comp}), serving as an adaptive congestion control mechanism.

Scaling and Communication Barriers. Scaling compute workers ($R_k > 1$) yields diminishing returns unless inter-stage data transfer is strictly bounded. As modeled in Inequality 4, when arrival rates exceed ingestion capacity ($\lambda_{k+1} > \mu_{k+1}$), backpressure reduces asynchronous execution to synchronous stalls. Maintaining high exit rates ($\eta \geq 80\%$) or enforcing Dynamic Stage In-lining is essential to maintain pipeline concurrency over commodity interconnects.

Limitations and Future Directions. While our study focuses on CNN backbones over Ethernet topologies, high-performance deployments may leverage ultra-low latency fabrics such as InfiniBand. In such environments, T_{comm} shrinks drastically, allowing optimal capacity matching ($\lambda_{k+1} \approx \mu_{k+1}$) to be achieved at lower early-exit rates ($\eta \approx 30\%$–50%). Extending these multi-stream topologies to large language models (LLMs) and autoregressive decoder stages presents a promising direction for future research.

7 CONCLUSION

We formalized the system dynamics, latency-hiding limits, and queueing mechanics of distributed early-exit neural network inference. Empirical evaluations across three CNN architectures and six deployment paradigms demonstrate that FLOP reductions only yield end-to-end speedups when communication costs are hidden behind compute windows. Replicated topologies (*Multi-Stream-3* and *Multi-Stream-5*) achieved up to $6.23\times$ speedups over serial baselines. To resolve link saturation under low exit rates, our proposed *Dynamic Stage In-lining* mechanism caps remote transfers and processes excess micro-batches locally, converting network-bound $0.66\times$ slowdowns into $3.98\times$ net speedups. Ultimately, co-designing confidence thresholds, worker replication, and dynamic memory in-lining provides a scalable framework for distributed inference engines.

AI USE STATEMENT

In this work, we used generative AI tools to aid in editing and polishing the manuscript, including improving clarity, formality, and academic style of author-written text. We also used generative AI tools for retrieval and discovery, including identifying potentially related research works based on research ideas and descriptions provided by the authors.

For research execution, generative AI tools was used for programming assistance to generate code implementing an algorithm developed by the authors. The research idea, methodology, and algorithm were developed entirely by the authors. The AI-generated code was reviewed, adapted, and tested by the authors for correctness. All experiments were executed by the authors, and the authors are responsible for the analysis, interpretation, and conclusions presented in this work.

We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, clean and reformat dataset, support qualitative and thematic data analysis, interpret results. The remaining required-disclosure tasks are not applicable to this work.

We have reviewed all AI-assisted work and take full responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

REFERENCES

Enzo Baccarelli, Michele Scarpiniti, Alireza Momenzadeh, and Sima Sarv Ahrabi. Learning-in-the-fog (lifo): Deep learning meets fog computing for the minimum-energy distributed early-exit of inference in delay-critical iot realms. *IEEE Access*, 9:25716–25757, 2021.

Divya Jyoti Bajpai and Manjesh Kumar Hanawal. Distributed inference on mobile edge and cloud: An early exit based clustering approach. In *ICC 2025-IEEE International Conference on Communications*, pp. 5425–5430. IEEE, 2025.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International conference on machine learning*, pp. 527–536. PMLR, 2017.

Vanderlei Bonato and Christos-Savvas Bouganis. Class-specific early exit design methodology for convolutional neural networks. *Applied Soft Computing*, 107:107316, 2021.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *International conference on machine learning*, pp. 1520–1530. PMLR, 2020.

Marco Colocrese, Erdem Koyuncu, and Hulya Seferoglu. Early-exit meets model-distributed inference at edge networks. In *2024 IEEE 30th International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 39–44. IEEE, 2024.

Yinwei Dai, Rui Pan, Anand Iyer, Kai Li, and Ravi Netravali. Apparate: Rethinking early exits to tame latency-throughput tensions in ml serving. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 607–623, 2024.

Maryam Ebrahimi, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. Combining dnn partitioning and early exit. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, pp. 25–30, 2022.

P Haseena Rahmath and Kuldeep Chaurasia. Adaptive early-exit inference in graph neural networks based hyperspectral image classification. In *International Conference on Intelligent Systems Design and Applications*, pp. 444–453. Springer, 2023.

Fatih Ilhan, Ka-Ho Chow, Sihao Hu, Tiansheng Huang, Selim Tekin, Wenqi Wei, Yanzhao Wu, Myungjin Lee, Ramana Kompella, Hugo Latapie, et al. Adaptive deep neural network inference optimization with eenet. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 1362–1371. IEEE, 2024.

Anand PADMANABHA Iyer and Swapnil Sunilkumar Gandhi. Deep neural networks (dnn) inference using practical early exit networks, December 30 2025. US Patent 12,511,219.

Weiyu Ju, Wei Bao, Liming Ge, and Dong Yuan. Dynamic early exit scheduling for deep neural network inference through contextual bandits. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 823–832, 2021.

Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. Spinn: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pp. 1–15, 2020.

Xian Peng, Xin Wu, Lianming Xu, Li Wang, and Aiguo Fei. Distree: Distributed early exit of deep neural network inference on edge devices. In *GLOBECOM 2024-2024 IEEE Global Communications Conference*, pp. 3116–3121. IEEE, 2024.

Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto G Pacheco, and Rodrigo S Couto. Early-exit deep neural network-a comprehensive survey. *ACM Computing Surveys*, 57(3): 1–37, 2024.

Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13. IEEE, 2016.

Rafael Stahl, Alexander Hoffman, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. Deeperthings: Fully distributed cnn inference on resource-constrained edge devices. *International Journal of Parallel Programming*, 49(4):600–624, 2021.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*, pp. 2464–2469. IEEE, 2016.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pp. 328–339. IEEE, 2017.

Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices. *IEEE/ACM Transactions on Networking*, 29(2):595–608, 2020.