PLUMBER: Scheduling Deep Learning Inference Pipelines on Resource-Constrained Edge Clusters

Animesh Dangwal, Ria Singh, Shruthi Unnithan, Karen Yuan, Emily Zheng, Chandra Krintz, Rich Wolski

Computer Science Dept., Univ. of California, Santa Barbara

***Abstract*—Resource-constrained edge environments exhibit dynamic network and compute conditions that can significantly degrade distributed deep learning inference when static deployment strategies are used. We present PLUMBER, an adaptive scheduler that dynamically selects pipeline decompositions and execution start times using online performance modeling to account for changing resource availability. Evaluated on a 117-node Raspberry Pi 3B+ cluster using EfficientNetB0, ResNet18, and ViT-B16 under varying network conditions and workload arrival patterns, PLUMBER consistently matches or outperforms static decomposition strategies in scheduling efficiency while improving energy efficiency. These results demonstrate that treating pipeline decomposition as a runtime scheduling decision enables robust and efficient edge inference.**

I. INTRODUCTION

Advances in machine learning (ML) have enabled increasingly sophisticated inference workloads to execute directly on resource-constrained edge devices, reducing latency, communication cost, dependence on reliable high-performance wide area network connectivity, and the cloud. Consequently, Internet of Things (IoT) and edge applications increasingly perform inference locally rather than relying on cloud offloading [1]. However, implementing efficient inference in these environments remains challenging even when edge inference algorithms are tuned for resource constrained execution. Specifically, because edge devices typically are not highly engineered resources with reliable power sources and vast internal memory systems, their available resource capacity can fluctuate (often dramatically) over time.

Unlike static data centers, edge deployments are characterized by unpredictable network bandwidth and compute variability caused by contention for "scarce" shared resources. In these dynamic environments, "one-size-fits-all" deployment strategies do not maximize delivered application performance. The best pipeline decomposition (in terms of performance) often depends on current resource conditions: balancing computation is advantageous when bandwidth is plentiful, whereas minimizing communication becomes increasingly important as bandwidth becomes constrained. Because these conditions continually change, no single decomposition strategy performs well across all deployment scenarios. Adaptive scheduling is therefore required to continuously select the decomposition best suited to the current operating environment.

Existing approaches improve edge inference through model optimization (e.g., pruning, quantization, and compression) [2] and distributed model partitioning and pipelining [3], [4], [5], [6]. However, these approaches generally assume static resource conditions or edge–server architectures [7]. Other studies show that shared power infrastructure and concurrent device usage can cause CPU throttling and significant performance variability in resource-constrained edge clusters [8]. Existing approaches do not adapt pipeline decompositions online to such variability, which requires predicting candidate performance under different conditions.

To address this challenge, we present PLUMBER, an adaptive scheduler for distributed deep learning (DL) inference on resource-constrained edge clusters. Rather than relying on a fixed deployment configuration, PLUMBER dynamically selects pipeline decompositions based on current network conditions, background load, workload characteristics, and request arrival patterns. Specifically, we make the following contributions:

Adaptive Pipeline Scheduling: We develop an online scheduling framework to select between computation- and communication-optimized decompositions using a pipeline-aware performance model, reducing profiling overhead by more than 95%.

Workload Compaction: We develop an energy-aware scheduling strategy that dynamically balances and consolidates inference workloads across edge devices, improving cluster throughput while reducing estimated energy consumption by limiting unnecessary device activity.

Experimental Evaluation: We evaluate PLUMBER on a 117-node Raspberry Pi 3B+ cluster using EfficientNetB0 [9], ResNet18 [10], and ViT-B16 [11], representing efficient convolutional, residual convolutional, and transformer-based architectures. Across realistic Azure Functions workload traces [12] and network bandwidths ranging from 2–240 Mbps, we show improved scheduling and energy efficiency over static decomposition.

II. PLUMBER

PLUMBER is an online scheduler for distributed DL inference that dynamically selects pipeline decompositions based on predicted execution costs under current resource conditions. It partitions DL models into sequential stages to distribute inference and allow multiple inference requests to be processed simultaneously, as shown in Figure 1. For each arriving job, PLUMBER evaluates candidate decompositions by estimating compute throttling, network contention, and device utilization, ultimately

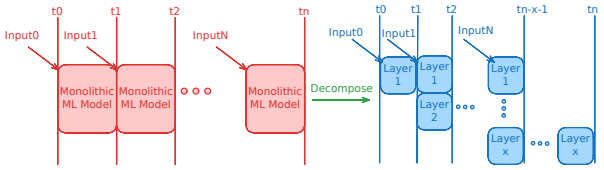


Fig. 1: A monolithic DL model is partitioned into sequential pipeline stages that execute on separate edge devices. Different pipeline decompositions correspond to different layer partitionings serving as candidate schedules by PLUMBER.

selecting the schedule to maximize throughput while minimizing overall energy usage.

PLUMBER partitions DL models at layer boundaries, allowing it to evaluate alternative organizations online without modifying the underlying model representation. To do so, PLUMBER maintains state information for every device in the cluster (e.g., assigned job and estimated execution progress). This allows PLUMBER to estimate background load, emulate network bandwidth sharing, and predict performance degradation caused by power-driven CPU throttling due to shared power infrastructure.

As formalized in Algorithm 1, PLUMBER evaluates each candidate decomposition θ using three metrics: local throughput (th_{local}), global cluster throughput (th_{global}), and estimated energy consumption (E). We estimate energy using the time devices spend at full 1.4GHz clock speed (T_{active}) versus throttled or idle states at 0.6GHz (T_{idle}). Using power estimates of $P_{active} \approx 5.1W$ and $P_{idle} \approx 1.9W$ [13], we compute $E = P_{active}T_{active} + P_{idle}T_{idle}$. These times are estimated using `vcgencmd` performance counters, avoiding direct power measurement.

For each arriving job, PLUMBER evaluates valid pipeline decompositions at candidate start times determined by the estimated completion times of running jobs. It evaluates these candidates in parallel across the scheduler's available CPU cores, stopping when the estimated waiting cost outweighs any potential throughput benefit. PLUMBER selects the decomposition and start time that maximize an equal-weight score combining normalized local throughput, global throughput, and estimated energy consumption.

A. Estimating Throttling Overhead

To quantify instability caused by CPU throttling due to uneven power distribution from shared power supplies, we follow the methodology described in [8]. We use matrix multiply workloads to construct a piece-wise (PW) linear regression model to predict the throttling factor tf based on background load. We define tf as the ratio of time spent under CPU throttling ($T_{throttled}$), measured via hardware counters (`vcgencmd`), to total execution time (T_{total}): $tf = T_{throttled}/T_{total}$

Algorithm 1 Dynamic Pipeline Scheduler (PLUMBER)

Input: New job J (Model, N inputs, arrival time t)
Output: Selected decomposition θ^* and execution delay d^*

$valid_delays \leftarrow [0] \cup \{\text{Active task times} > t\}$
$\Theta \leftarrow \{\text{CandidateDecompositions(J, N)}\}$
for each valid decomposition $\theta \in \Theta$ **do**
  for each valid delay $d \in valid_delays$ **do**
    $cluster \leftarrow \text{BackgroundLoadInfo}(d)$
    $(T_{active}, T_{idle}) \leftarrow \text{Simulate}(\theta, J, cluster)$
    $E \leftarrow P_{active} * T_{active} + P_{idle} * T_{idle}$
    $th_{local} \leftarrow \text{Throughput}(N, J)$
    $th_{global} \leftarrow \text{Throughput}(N, cluster)$
    if $th_{global} < prev_th_{global}$ **then**
      break
    end if
    $prev_th_{global} \leftarrow th_{global}$
  end for
end for
$\theta^*, d^* \leftarrow \arg\max_{\theta,d} (\text{Score}(-E, th_{global}, th_{local}))$
return $\{\theta^*, d^*\}$

Extending this approach to DL pipelines naively, requires profiling every possible layer combination under varying throttling states. To avoid this, PLUMBER assumes that tf for a DL model's forward pass is proportional to the matrix multiplication workloads used to derive the PW model, given the majority of their computation consists of floating point operations.

We define T_{best} as the execution time without CPU throttling. Under throttling, CPU frequency drops from 1.4 GHz to 0.6 GHz. This implies the additional penalty period (T_{added}) is roughly half of the measured throttled time: $T_{throttled} \approx 2 \cdot T_{added}$. That is, the additional overhead T_{added} relative to T_{best} runs at approximately $\frac{1}{2}$ the clock speed. As such, we can estimate it by dividing $T_{throttled}$ by 2. We have empirically verified that this relationship holds, using cluster profiling.

Given that $T_{total} \approx T_{best} + T_{added}$, we can express the total runtime in terms of only T_{best} and tf:

$$T_{total} \approx T_{best} \left(\frac{2}{2 - tf} \right) \tag{1}$$

This derivation allows us to estimate the runtime for any arbitrary model split using only its baseline latency (T_{best}) and the current environmental tf (from the PW model), bypassing the need for exhaustive re-profiling.

B. DL Model Decomposition and Profiling

To support online scheduling, PLUMBER must rapidly evaluate candidate pipeline decompositions under changing network conditions and compute contention. As stated above, exhaustively profiling every possible layer partition is impractical for realistic deep learning models.

For example, for ViT-B16 model, with only three partitions, each with a minimum of 10% FLOP allocations per stage, yields more than 2,500 candidate decompositions for each available network bandwidth state.

To make online scheduling tractable, PLUMBER restricts candidate decompositions using three constraints. First, each pipeline stage must fit within the available device memory (we use 600MB in our study). Second, partition boundaries must preserve simple tensor interfaces to avoid complex model rewiring or developer intervention. Finally, we restrict pipelines to have between three and six stages. Fewer than three stages exceeds memory limits for large models like ViT-B16, whereas more than six stages produce partitions whose communication overhead outweighs any additional parallelism in ResNet18 and EfficientNetB0.

Given these constraints, PLUMBER evaluates two complementary decomposition strategies:

- **Compute-Balanced (COMP SPLIT):** Distributes FLOPs evenly across layers to balance the computational load.
- **Communication-Optimized (N/W SPLIT):** Selects partitions to minimize the size of transferred tensors, reducing communication overhead.

All pipelines execute synchronously, eliminating intermediate buffering and providing predictable memory consumption within device limitations. Under synchronous execution, pipeline throughput is limited by the slowest stage, making highly unbalanced decompositions undesirable. Thus, PLUMBER only considers these two decomposition strategies.

Applying these constraints reduces the profiling space from more than 2,500 candidate decompositions to 108 baseline measurements (3 models × 18 valid configurations per strategy), a reduction of $> 95\%$. We collect these baseline latencies during warm-up and combine them with the analytical performance model to evaluate candidate schedules online.

III. EVALUATION

We evaluate PLUMBER under realistic edge deployment conditions to answer four research questions.

- RQ1: Are the performance models sufficiently accurate to support online scheduling?
- RQ2: Does PLUMBER improve end-to-end scheduling performance relative to static decomposition strategies across diverse workloads and conditions?
- RQ3: How do environmental conditions, including network bandwidth, background load, and model architecture, influence the scheduling decisions made by PLUMBER?
- RQ4: How robust is PLUMBER to changes in workload composition and arrival patterns?

To answer these questions, we describe our experiments, hardware platform, workloads, baselines, and evaluation.

A. Experimental Setup

We evaluate PLUMBER on a 117-node Raspberry Pi 3B+ cluster under varying network bandwidth, background load, workload composition, and arrival patterns. Each device has a 4×1.4GHz ARM processor, 1GB memory, and communicates over Ethernet with a maximum available bandwidth of 240 Mbps. The devices are organized into three subclusters of 39 devices, where each subcluster shares a common power distribution unit (PDU). This shared power infrastructure results in power-induced CPU throttling for devices when executing compute-intensive workloads concurrently. To emulate a range of edge deployment environments, we limit available bandwidth to 240, 30, 10, and 2 Mbps, and equally share available bandwidth among simultaneously communicating devices.

We evaluate PLUMBER using three representative deep learning image classification models: EfficientNetB0, ResNet18, and ViT-B16. Models are exported to ONNX [14], quantized, and partitioned into inference pipelines of three to six stages according to decomposition constraints described in Section II. Each pipeline processes batches of ten 1 MB images. The worst case overhead introduced by PLUMBER is $\approx$ 3s per job, when running on a 4×2.4GHz Apple Mac M2 machine, 28GB of memory. Model decompositions can take up to 30s but are performed offline as valid constraints and models are known a priori, as described in Section II-B.

To capture both regular and bursty edge workloads, we evaluate two arrival processes. We implement a Uniform arrival process representing periodic sensing applications with approximately constant inter-arrival times, and we use inter-arrival times from an Azure Functions trace to represent bursty serverless workloads exhibiting realistic temporal variability. Unless otherwise noted, we uniformly distributed workload composition across the three models.

We compare PLUMBER against two static decomposition strategies: *Only Compute* always selects the compute-balanced decomposition (COMP SPLIT), and *Only Communication* always selects the communication-optimized decomposition (N/W SPLIT). In contrast, PLUMBER evaluates both strategies for each arriving job and dynamically selects the decomposition predicted to maximize overall scheduling and energy efficiency given current cluster conditions. All three schedulers use the same underlying execution framework described in Section II and differ only in their selection strategy.

We evaluate whether adaptive pipeline decomposition improves scheduling efficiency and energy consumption relative to static compute-balanced (Only Compute) and communication-optimized (Only Communication) strategies. All schedulers execute identical workloads using Uniform and Azure Functions arrival traces across bandwidths of 2–240 Mbps. We report scheduling

Model	Avg Latency (s)	Standard Deviation (s)	Pipeline-Aware				Pipeline-Unaware			
			Worst RMSE (s)	Avg RMSE (s)	Worst MAE (s)	Avg MAE (s)	Worst RMSE (s)	Avg RMSE (s)	Worst MAE (s)	Avg MAE (s)
EfficientNetB0	3.998s	0.365s	**0.795s**	**0.572s**	**0.655s**	**0.448s**	3.264s	2.412s	3.241s	2.387s
ResNet18	2.982s	0.123s	**0.652s**	**0.466s**	**0.520s**	**0.374s**	1.879s	1.371s	1.876s	1.365s
ViT_B16	22.294s	4.124s	**5.525s**	**4.476s**	**4.668s**	**3.791s**	23.655s	20.178s	22.619s	19.125s

TABLE I: Comparison of latency estimation error for Pipeline-Aware vs. Pipeline-Unaware estimations. Results show average and worst-case RMSE and MAE. Average latency and standard deviation is reported for non-throttled execution. Pipeline-Aware estimations (boldface) are significantly closer to measured latencies across background load variations, compared to Pipeline-Unaware, validating our performance model.

efficiency (η) and relative energy consumption (E_{rel}), both normalized against an ideal execution without CPU throttling. Their ratio, η/E_{rel}, captures throughput relative to energy cost.

B. Validating the Performance Model

Accurate scheduling decisions require reliable estimates of the execution time of candidate pipeline decompositions under changing resource conditions. Rather than empirically profiling every possible decomposition under every combination of network and power contention, PLUMBER predicts execution latency using the analytical performance model described in Section II-A. This experiment evaluates whether those predictions are sufficiently accurate to support online scheduling.

We compare two prediction approaches. Pipeline-Aware estimates the changing background load experienced throughout pipeline execution as different stages start and end. In contrast, Pipeline-Unaware estimation assumes that background load remains constant for the duration of an inference pipeline. Both approaches use the same regression model to estimate throttling effects; they differ only in whether they account for the temporal interference effects during pipeline execution. Prediction accuracy is evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) across varying levels of background load and 30 repeated experiments.

Table I summarizes the errors for EfficientNetB0, ResNet18, and ViT-B16. Across all three models, Pipeline-Aware, consistently produces substantially lower prediction error than the Pipeline-Unaware baseline. The improvement is particularly pronounced for ViT-B16, where average RMSE decreases from 20.18s to 4.48s. This demonstrates the degree to which performance interference impacts prediction. Similar improvements are observed for RMSE and MAE across all evaluated models.

The increased prediction accuracy translates directly into improved scheduling decisions. Using the Pipeline-Aware estimator increases scheduling efficiency by at least 63% while reducing energy consumption by at minimum 46% relative to the Pipeline-Unaware scheduler.

These results demonstrate that Pipeline-Aware estimation is sufficiently accurate for online scheduling, allowing PLUMBER to evaluate candidate decompositions analytically rather than through exhaustive runtime profiling. That is, the scheduler need only evaluate candidate decompositions, rather than execute or profile exhaustively, making online adaptive scheduling computationally practical.

C. End-to-End Scheduling Performance

We next evaluate the scheduling efficiency of PLUMBER under realistic edge deployments. Our goal is to determine whether adaptive pipeline decomposition results in higher scheduling efficiency and lower energy consumption than static decomposition strategies. We compare PLUMBER against two baseline schedulers that select either compute-optimized (Only Compute) or communication-optimized (Only Communication) decompositions, regardless of current operating conditions.

Each scheduler executes identical workloads under both Uniform and Azure Functions arrival traces for different available network bandwidths between 2 and 240 Mbps. Performance is evaluated using Scheduling Efficiency (η), which measures achieved throughput relative to the theoretical best-case schedule, and Relative Energy Efficiency ($\mathcal{E}_{rel}$), which quantifies total energy consumption relative to the same idealized execution. An ideal schedule in this case, is the theoretical schedule if devices never throttle. Note, because η and $\mathcal{E}_{rel}$ are unitless, their ratio is a benefit-to-cost ratio, comparable across all experiments (each having a potentially different time scale).

Table II summarizes the results. Across all evaluated workloads and network conditions, PLUMBER consistently achieves the highest measured scheduling efficiency (largest η values). It also achieves the largest benefit-to-cost ratios (shown in boldface in the table). Note, the non-adapting strategies perform well near the extremes with respect to network bandwidth, but neither is best across the full range of conditions. These results demonstrate that dynamically selecting pipeline decompositions allows PLUMBER to adapt to changing resource availability without sacrificing either throughput or energy efficiency. Across all evaluated deployment conditions, PLUMBER remains within approximately 5%

Arrival Type	Avail B/W (Mbps)	PLUMBER			Only Compute			Only Communication		
		$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$
Azure Functions Trace	240	0.42	0.94	**2.24**	0.42	0.94	**2.24**	0.79	0.53	0.67
	30	0.47	0.99	**2.11**	0.47	0.98	2.09	0.5	0.86	1.72
	10	0.47	1.0	**2.13**	0.5	0.99	1.98	0.49	0.99	2.02
	2	0.46	1.0	**2.17**	0.47	1.0	2.13	0.46	1.0	**2.17**
Uniform	240	0.4	0.96	**2.4**	0.4	0.96	**2.4**	0.94	0.62	0.66
	30	0.63	0.99	**1.57**	0.96	0.99	1.03	0.72	0.85	1.18
	10	0.59	1.0	**1.69**	1.21	1.0	0.83	0.62	0.95	1.53
	2	0.73	1.0	**1.37**	1.42	1.0	0.7	0.73	1.0	**1.37**

TABLE II: Evaluation of relative energy consumption ($\mathcal{E}_{rel}$) and scheduling throughput(η) across varying bandwidth availability. Best benefit-to-cost ratios are shown in boldface.

of the no-throttling reference, and remains within 1% for moderate and low bandwidth conditions.

Another interesting observation is that several measured $\mathcal{E}_{rel}$ values are <1. This result does not imply the strategies exceed theoretical optimum. Rather, the theoretical reference assumes only the absence of power-induced throttling and does not account for workload compaction. By consolidating computation onto fewer devices when appropriate, PLUMBER reduces both active and idle energy, yielding lower energy consumption than the reference schedule despite operating under realistic resource contention.

Overall, adaptive pipeline decomposition enables PLUMBER to consistently outperform static decomposition strategies across diverse edge deployment conditions, achieving high scheduling efficiency while reducing energy consumption versus static decomposition strategies. Static strategies perform well only when deployment conditions suit their optimization objective, whereas PLUMBER consistently delivers high scheduling efficiency and low energy consumption across diverse workloads, arrival patterns, and network environments.

D. Impact of Resource Conditions

We next examine why these improvements occur by analyzing how available network bandwidth and model architecture influence decomposition selection. Specifically, we investigate the resource conditions under which compute-balanced and communication-optimized decompositions become preferable.

Figure 2 compares throughput across decomposition strategies and network bandwidths for all three models, with a maximum observed standard deviation of 0.1 images/s. As expected, the preferred decomposition depends on the relative cost of computation and communication.

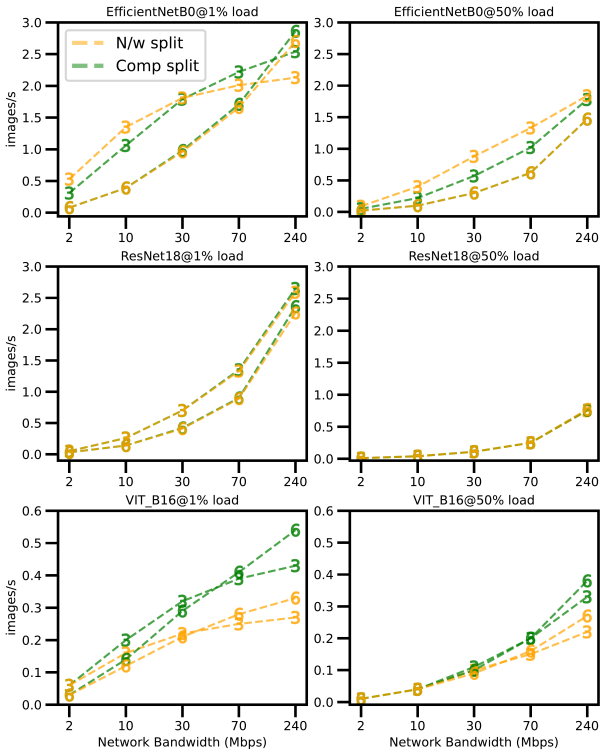


Fig. 2: Throughput across varying network bandwidths and background loads. Each row represents a different model and columns differentiate low (1%) and high (50%) subcluster background load. Orange lines denote communication splits, and green lines denote compute splits. Markers indicate the number of splits (3 or 6). The points represent throughput of the most recent job and higher throughput indicates better decomposition choice.

At high network bandwidth (240 Mbps), communication latency contributes only a small fraction of total execution time. Under these conditions, compute-balanced decompositions consistently outperform communication-optimized decompositions because they distribute computation more evenly across participating devices, reducing pipeline imbalance and improving utilization. The additional communication introduced is largely hidden by the available network capacity.

The crossover point between the two decomposition strategies is not fixed but depends on the computational characteristics of the deployed model. EfficientNetB0, ResNet18, and ViT-B16 exhibit different ratios of computation to communication, causing each architecture to transition between preferred decomposition strategies at different bandwidths.

These results demonstrate that the preferred decomposition depends on model-specific computation-to-communication ratios and available bandwidth. Consequently, a static decomposition cannot consistently optimize performance across models and operating condi-

Arrival Type	Avail B/W	PLUMBER			Only Compute			Only Communication		
		$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$	$\mathcal{E}_{rel}$	η	$\eta/\mathcal{E}_{rel}$
Azure Functions Trace	(Mbps) 2	0.67	1.0	**1.49**	1.22	0.89	0.73	0.68	0.99	1.46
Uniform	2	0.66	1.0	**1.52**	1.41	0.81	0.57	0.81	0.93	1.15

TABLE III: Evaluation of relative energy consumption ($\mathcal{E}_{rel}$) and scheduling throughput(η) when ViT-B16 has a higher probability of selection than other models in the workload. Here 2 Mbps has non-trivial schedules due to increased ViT-B16 jobs.

tions. PLUMBER addresses this variability by evaluating alternative decompositions online rather than relying on fixed partitioning heuristics.

E. Robustness to Workload Composition

Real-world edge deployments rarely execute a single application in isolation. Instead, workloads evolve over time as new applications are introduced, request rates fluctuate, and the relative frequency of different inference tasks changes. We therefore evaluate the robustness of PLUMBER under varying workload compositions and arrival patterns to determine whether adaptive pipeline decomposition continues to provide benefits when workload characteristics differ from those used to derive the scheduling model.

To evaluate robustness to workload composition, we vary the relative frequency of EfficientNetB0, ResNet18, and ViT-B16 requests while maintaining constant total workload volume. We evaluate both Uniform and Azure Functions arrival patterns. Table III presents results for a ViT-B16-heavy workload at 2 Mbps. PLUMBER maintains higher scheduling efficiency and a better throughput-to-energy ratio than either static strategy, demonstrating that its benefits extend beyond the uniformly distributed workload used in Table II

The benefits of adaptation become particularly apparent as workload composition shifts toward models with different computational and communication requirements, for example, a higher frequency of ViT-B16 inference tasks. Because the preferred decomposition varies across model architectures, changes in workload composition also change the resource demands placed on the cluster. Static strategies cannot account for these changing demands. In contrast, PLUMBER continuously evaluates candidate decompositions for each arriving inference request, allowing scheduling decisions to naturally adapt as the workload evolves. For 2 Mbps, in Table III, the communication-optimized strategy fails to match the performance of PLUMBER, contrasting with Table II.

These results show that PLUMBER does not rely on a fixed workload mix or arrival pattern to achieve high performance. Instead, adaptive decomposition enables the scheduler to respond to changing application demands while maintaining consistently high throughput and energy efficiency across diverse deployment scenarios. PLUMBER remains effective across changing workload compositions and arrival patterns without requiring workload-specific tuning.

IV. CONCLUSIONS

We presented PLUMBER, an adaptive scheduler for distributed deep learning inference on resource-constrained edge clusters. By combining online performance modeling with dynamic pipeline decomposition and start-time selection, PLUMBER adapts to changing network conditions, compute contention, and workload demands. Evaluation on a 117-node Raspberry Pi 3B+ cluster demonstrates improved scheduling and energy efficiency over static decomposition strategies across diverse operating conditions. This research is supported in part by an Oracle gift and NSF awards CNS-2444318 and CNS-2107101.

REFERENCES

[1] W. Wang, K. Li, B. Ji, X. Liu, J. Yu, and Q. Wu, "A survey of ai inference technologies for on-device systems," *IEEE Internet of Things Journal*, 2025.

[2] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Intl. Conference on Learning Representations*, 2016.

[3] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: Dnn model slicing for visual analytics containers at the edge," in *ACM Symposium on Edge Computing*, 2019.

[4] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. Beerel, S. Crago, and J. Walters, "Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices," in *Euromicro Conference on Digital System Design*, 2022.

[5] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, P. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *ACM symposium on operating systems principles*, 2019.

[6] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: agile offloading with explainable ai," in *Conference on Mobile Computing And Networking*, 2022.

[7] Q. Liang, P. Shenoy, and D. Irwin, "Ai on the edge: Rethinking ai-based iot applications using specialized edge architectures," in *Intl. Symposium on Workload Characterization*, 2020.

[8] A. N. Dangwal, R. Singh, S. Unnithan, K. Yuan, E. Zheng, C. Krintz, and R. Wolski, "Reliable unreliability in edge deployments," in *Conference On Distributed Computing In Smart Systems And The Internet Of Things (DCOSS)*, June 2026.

[9] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Intl. conference on machine learning*, 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition*, 2016.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *Intl. Conference on Learning Representations*, 2020.

[12] F. Romero, G. Chaudhry, I. Goiri, P. Gopa, P. Batum, N. Yadwadkar, R. Fonseca, C. Kozyrakis, and R. Bianchini, "Faa$t: A transparent auto-scaling cache for serverless applications," in *ACM Symposium on cloud computing*, 2021.

[13] J. Geerling. (2026) Power consumption benchmarks. https://pidramble.com/wiki/benchmarks/power-consumption. Raspberry Pi Dramble. Accessed: 2026-07-30.

[14] ONNX, "Onnx runtime," https://onnxruntime.ai/, 2021, v1.19.2.