Semi-Supervised Approach For Inference Serving At The Edge

Saif Eddine Khelifa*, Sihem Ouahouah§, Miloud Bagaa*, Messaoud Ahmed Ouameur * and Adlen Ksentini¶
* Université du Québec à Trois-Rivières, Trois-Rivières, QC, Canada.
Emails: {Saif.Eddine.Khelifa, miloud.bagaa, messaoud.ahmed.ouameur}@uqtr.ca
, §Aalto University, Otakaari 24, 02150 Espoo FINLAND (e-mail: sihem.ouahouah@aalto.fi),
¶EURECOM, Campus SophiaTech, France (e-mail: adlen.ksentini@eurecom.fr)

***Abstract*—Edge intelligence enables deep learning (DL) inference services to run close to end users, which is essential for latency-sensitive applications such as autonomous driving and smart surveillance. However, satisfying strict latency-oriented service level objectives (SLOs) at the edge remains challenging because inference performance is jointly influenced by the selected DL model variant, the CPU parallelism configuration, and resource interference on shared edge servers. Existing approaches have not adequately considered these three factors in a unified manner. In this paper, we address this gap by proposing a semi-supervised framework that jointly models these three dimensions to predict P90 latency and support serving decisions. Our approach represents candidate DL models through computational graphs structured as directed acyclic graphs (DAGs) and combines unsupervised structural pretraining with supervised runtime-aware graph learning. The framework is integrated into an inference-serving pipeline that filters candidate models according to user accuracy requirements, evaluates feasible thread configurations under current interference conditions, and selects the model and CPU parallelism setting that satisfy the target P90 latency constraint.**

***Index Terms*—Edge intelligence, DNN inference, DNN serving**

I. Introduction

In recent years, DL applications have expanded rapidly across many service domains, including robotics, finance, and healthcare. This growth supports the 6G vision of AI-native, user-centric services. In this vision, intelligence moves away from centralized clouds toward the network edge, closer to where data is generated and consumed [1], [2]. This shift has given rise to *edge intelligence*, which enables on-device or near-device processing to reduce end-to-end communication latency and better preserve privacy by limiting the exposure of sensitive user data.

In practice, DL models at the edge are deployed as inference services that receive application requests, such as image frames, sensor data, or text queries, and return predictions. These predictions must meet explicit performance constraints, or SLOs, defined by the application. For example, an SLO may require that 99% of requests finish within 500 ms . This is especially critical for real-time applications such as autonomous vehicles, where even small delays can affect safety [3]. As a result, satisfying latency-sensitive SLOs is particularly challenging in edge environments because edge nodes provide limited compute and memory resources, while modern deep neural networks (DNNs) remain resource-intensive. To reduce the risk of SLO violations, service providers must address a core challenge in real-time DL inference serving at the edge: meeting strict latency-oriented SLOs while maximizing user-defined accuracy, without relying on costly overprovisioning. This is difficult because both resource consumption and inference latency are jointly influenced by three key factors:

- **DL model architectures and variants:** Different DNN architectures (e.g., ResNet and MobileNet) can solve the same task, but they exhibit different trade-offs in accuracy, throughput, energy use, and inference latency.
- **Resource interference:** To improve utilization, providers often co-locate applications on the same edge server. However, co-location creates contention on shared resources, such as last-level cache and memory bandwidth, and its impact varies across DL models.
- **Parallelism knobs:** Runtime parameters, such as thread-level parallelism, also affect serving performance. Their impact is model-dependent: some DL models benefit significantly, while others gain little.

Combining these three axes creates a complex landscape of trade-offs. This, in turn, raises several important research questions, such as: which DL model should be selected under a given interference condition, at what level of parallelism should it operate, and how can these decisions be made so that strict latency-oriented SLOs are satisfied while maximizing user-defined accuracy ?

Existing work has mainly addressed this problem from two perspectives: *model-specific serving* and *model-less serving*. In model-specific serving, the user selects the DL model in advance, and the serving system only tunes the execution configuration. In contrast, model-less serving selects the most suitable DL model from a pool of candidates for serving requests. However, both perspectives primarily focus on two axes, namely resource interference and parallelism knobs, while failing to explicitly account for architecture-level differences across DL models. While, in parallel, recent studies in hardware-aware neural architecture search (HW-NAS) show that architectural differences strongly influence inference latency and lead to non-uniform runtime behavior.

Motivated by this limitation, we introduce a new inference serving approach designed to bridge this gap through a semi-supervised learning framework that jointly considers these three axes. Following the HW-NAS methodology, we transform DL models into directed acyclic graphs (DAGs) and

use these graph representations to train the framework. Within this framework, a Graph Encoder is used to learn topological features from unlabeled DL models, while GraphSAGE's are applied to labeled DL models to capture parallelism and interference signals. These learned representations are then combined to predict tail latency for real-time applications, with a particular focus on the 90th-percentile latency (P90). The proposed semi-supervised framework is further integrated into a custom pipeline that selects the most suitable DL model from a candidate pool for serving edge-application requests. In addition, to reflect the resource limitations of edge environments, we evaluate the approach in CPU-based, resource-constrained settings.

Finally, the paper is organized as follows: first, a Background and Related Work section, which introduces the key concepts and prior studies needed to highlight the novelty of our contribution, second, a Framework Design section, where we describe our pipeline and semi-supervised framework, third, a Results section, which evaluates the effectiveness of our approach, and finally, a Conclusion section, which summarizes our findings and outlines future directions.

II. Background and Related Works

Inference Serving : In the previous section, we explained that inference serving at the edge is challenging because it requires navigating three interdependent axes, *parallelism*, *model variants*, and *interference*, all of which directly affect inference latency. Existing works address this problem under the umberall of inference serving systems implementing it's diffrent compenents. Some focus on *model selection* in inference serving, that is, choosing the most suitable model variant for a request under latency and resource constraints. Others study *model switching*, where the serving system dynamically changes the deployed model at runtime to maintain service objectives under changing workloads [4]. Other approaches optimize request-level configurations, such as input resolution, to improve the latency–accuracy trade-off [5]. Although inference-serving systems involve many design dimensions, the literature can, in broad terms, be divided into *model-specific* and *model-less*, following the distinction discussed in both INFaaS [6] and IRIS [7]. However, some works do not fully fit either category. Unlike the broader model-less setting, where the DL variants is said to be generated infintly, some works [8] assume a predefined and finite set of DNN variants or configurations already available in the system, while the user specifies only high-level objectives such as SLO requirements. To characterize these studies more precisely, we classify them as a third category, which we call *constrained model-less*.

In the **constrained model-less** category, InfAdapter [8] proactively selects a subset of model variants and CPU allocations by solving an Integer Linear Programming problem to maximize a weighted accuracy–cost objective under user-defined SLOs. Similarly, RAMSIS [9] and MIDIA [10] both operate over a predefined, finite pool of already available DNN models and rely on workload-aware optimization to decide how models should be selected or deployed, RAMSIS through Markov Decision process-based model selection and MIDIA through adaptive deployment and decision-interval optimization. SneakPeek [11] also operates over a registered set of model variants and uses Bayesian estimation for joint model selection and scheduling under user-defined SLOs. Likewise, AdaptSwitch [12] and ModelSwitching [4] dynamically switch among a limited number of pre-trained models according to user-defined SLO tuples. EdgeAdaptor [5] and HISPER [13] jointly adapt request configurations (e.g., image resolution) and select models from a pool of DL variants to satisfy user-defined SLOs. Finally, SuperServe [14] introduces a different formulation by grouping DL models within a single SuperNet, given a task and a latency SLO, the system navigates the Pareto-optimal frontier of subnets within that SuperNet.

In **model-sepcific cateogry**, works from the same author like SynergAI [15] and IRIS [7] proposes a model-specific, where SynergAI focuses on diffrent hardware archtiectures such as ARM, X86 and map discrete infernece engines with specific DL models to a set of works deployed on multiple hardware archtiectures using predictive orchestrattion. In the same fashion IRIS provides a model-specific soltuion where use picks a DL model and engine and by using predictive scheduling IRIS allocate the inference serving application to suitable node minmizng CPU subject to user-defined SLO. Moreover, IRIS also provides a constrained model-less approach where finit set of DL models are chosen automatically and user submit the task and SLO. Another approach in inference serving [16] optimizes the deployment and startup behavior of the exact model instance already selected by the user (e.g., BERT-Base) through layer-wise caching and Direct-Host-Access for a user-specified DNN instance. Another, model-sepcific work is Clipper [17] considred as one of the first inference serving system where the author creates a model abstraction and selection layerse designed for developers to deploy and ensemble specific model to provide low-latency predictions across different DL frameworks (e.g., Pytorch, TensorFlow). Finally, INFaas [6] is the only fully model-less approach since it claims to be the first system that automatically generates the model pool itself (hundreds of variants) from a single user-provided model for certain user-defined SLO.

HW-NAS performance prediction: investigates how architectural differences among DL models affect key performance metrics such as latency, energy consumption, and throughput. A recent trend in this area is to represent DL models as direct acyclic graphs (DAGs) and encode them into graph embeddings that are processed by GNNs to estimate performance metrics. Existing studies have employed several GNN backbones(ex., GraphSage [18] ,GAT [19]), while others have proposed specialized architectures [20], [21] to better capture node-, edge-, and graph-level execution characteristics. In addition, semi-supervised approaches have been introduced [22] to reduce the need for large amounts of labeled profiling data, which is particularly valuable in cloud-edge environments where DNN profiling is costly.

At the intersection of HW-NAS performance prediction and inference serving, our work addresses a constrained model-

less serving problem in which the system jointly selects a DL model variant and a CPU thread configuration under runtime interference. Unlike conventional inference-serving approaches that rely mainly on model-level descriptors, resource availability, or execution configurations, our method represents each DL model as a computational graph and combines this architecture-level information with interference descriptors and intra-op/inter-op parallelism controls. While inspired by HW-NAS performance prediction, our approach integrates architecture-aware learning into an SLO-driven serving decision process. It also distinguishes the effects of CPU parallelism by incorporating intra-op and inter-op thread information into the GNN-based latency modeling process. This allows latency to be modeled as a joint function of model architecture, interference, and CPU parallelism.

III. Framework Design

A. Pipeline Overview

Our constrained model-less approach operates over a finite set of DL models extracted from the timm library.[1] In addition, the proposed method/gap is agnostic to the inference serving platform itself, which means that it can be applied to different components of the serving pipeline, including resource management, model selection, and model switching. As a proof of concept for the importance of the three considered axes in inference serving, we focus on one of the most critical phases of the serving process: the addition of a new worker to handle an incoming set of user requests. This process is implemented through the pipeline illustrated in Figure 1.

The pipeline begins with the preparation of a DL model pool. First, models are retrieved from Hugging Face storage ①, then converted into DAGs, and finally stored in a persistent volume. At this stage, the system organizes the candidate models according to metadata extracted from storage, particularly their accuracy ②. The system then waits for the user to submit a task together with a target SLO ③. In the current proof of concept, the task is fixed to image classification, although the approach can be extended to other tasks. The user specifies two SLO constraints: accuracy and tail latency (P90). The accuracy constraint is handled through a simple heuristic filtering step, where the system compares the SLO accuracy of the user against the TOP-1 accuracy of avaiable DL and extracts the names of the top 10 eligible models, when such number exist ③.

Next, a graph retrieval and filtering phase uses these model names to fetch the corresponding precomputed graph representations. These graph data are combined with system interference metrics, such as CPU usage, available L3 cache, and related runtime indicators ④. Based on the same resource information, the system also estimates the number of available CPU cores and constructs a grid of feasable candidate thread configurations, including both intra-op and inter-op thread settings ⑤. The candidate threads, the interference metrics, and the embedded graph representation are then fed into our semi-supervised framework ⑥. The framework outputs a set of tuples of the form (DL model, thread configuration, predicted tail latency). From this set ⑦, the system selects the best DL model x thread configuration whichl satisfies the target tail-latency constraint and maximize accuracy ⑧. Finally, the selected worker is instantiated to receive inference requests from the end-user application and return predictions and monitoring system based on our Telegraf [2] agent collects infromations about achieved Tail-Latency and ressource metrics ⑨⑩⑪.

B. Semi-Supervised Framework

We propose a two-phase semi-supervised framework for predicting tail latency from DL computation graphs. The dataset contains 1000 DL models, including CNNs, transformer-based architectures, hybrid models, and Mamba-inspired vision models. Each model is represented as a graph whose nodes correspond to operators and whose edges represent data dependencies. The models are evaluated under a multi-stream inference setting with a 60 ms latency constraint. To prevent data leakage, we use family-wise splits, since the same graph topology may appear under different thread configurations and interference conditions.

For each model graph k, we define its graph structure, adjacency matrix, node features, interference descriptor, and intra-op/inter-op thread controls as follows:

$$\begin{aligned} &G_k = (V_k, E_k), \qquad A_k \in \{0,1\}^{N_k \times N_k}, \\ &X_k \in \mathbb{R}^{N_k \times d_x}, \qquad \iota_k \in \mathbb{R}^{d_\iota}, \qquad \tau_k^{\text{intra}}, \tau_k^{\text{inter}} \in \mathbb{N}. \end{aligned} \tag{1}$$

intra-op threads τ_k^{intra}, inter-op threads τ_k^{inter} and interference ι_k affect execution at different levels, we do not combine them into a single runtime vector. Intra-op threads mainly influence operator-level execution, inter-op threads affect graph-level parallelism across independent operators, and interference can impact both local operator behavior and global execution capacity. Therefore, we define separate node-level and graph-level context vectors, as shown in (2).

$$\begin{aligned} c_k^{\text{node}} &= [\, \tau_k^{\text{intra}} \| \iota_k \,], \\ c_k^{\text{graph}} &= [\, \tau_k^{\text{inter}} \| \iota_k \,]. \end{aligned} \tag{2}$$

Overall, the framework consists of three architectural components: (i) an unsupervised graph autoencoder that learns a topology-aware structural prior, (ii) a supervised GraphSAGE regressor conditioned on runtime factors, and (iii) a prediction head that combines both embeddings with graph-level runtime context. Together, these three components are presented to the framework's two training phases: Phase 1, which performs unsupervised training, and Phase 2, which carries out the full semi-supervised training.

a) Phase I: Unsupervised structural pretraining.: In the first phase, we train our first compenent graph autoencoder (GAE) on the unlabeled subset of DL graphs. The encoder is a four-layer GraphSAGE network with batch normalization, ReLU activations, and dropout. Given the node features X_k and adjacency matrix A_k, the encoder produces node embeddings as defined in (3).

[1] https://huggingface.co/timm

[2] https://www.influxdata.com/time-series-platform/telegraf/

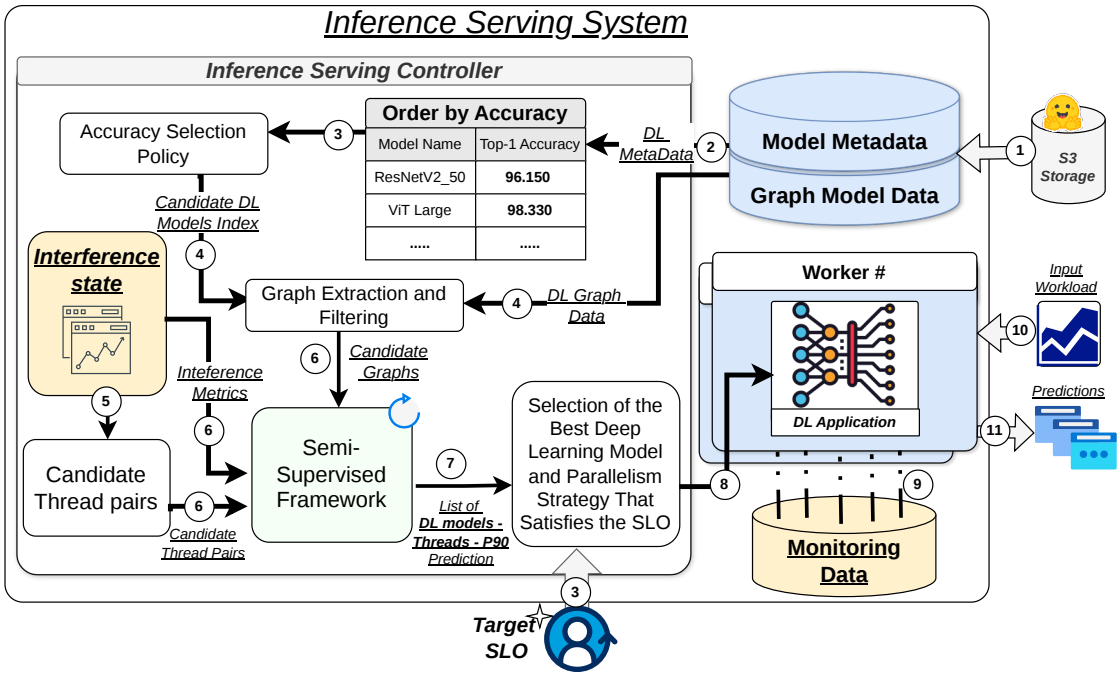

Fig. 1: Pipeline Overview explaining our approach

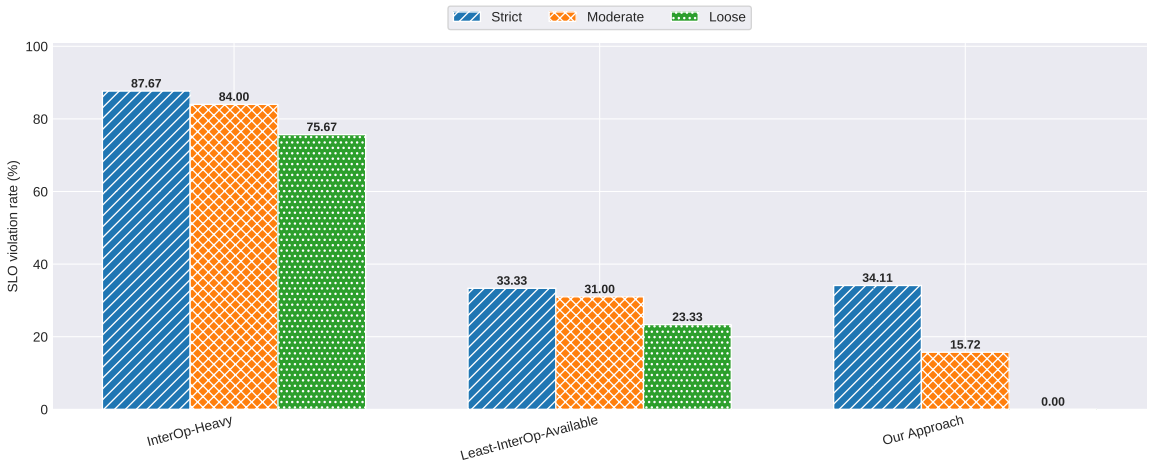

Fig. 2: Figure shows SLO Violations of our approach and baselines

$$Z_k = \mathrm{GAEEnc}(X_k, A_k), \qquad Z_k \in \mathbb{R}^{N_k \times d_h}. \tag{3}$$

A graph-level structural embedding is then obtained by sum pooling the node embeddings, as shown in (4).

$$e_k^{\mathrm{GAE}} = \mathrm{SUMPOOL}(Z_k), \qquad e_k^{\mathrm{GAE}} \in \mathbb{R}^{d_h}. \tag{4}$$

On the other hand, to reconstruct the graph structure, we use a *dot-product decoder*. For any pair of nodes i and j, the reconstructed edge probability is given by (5).

$$\hat{A}_{k,ij} = \sigma\left(z_{k,i}^{\top} z_{k,j}\right). \tag{5}$$

Finally, the GAE is trained with a binary cross-entropy reconstruction objective over observed edges and sampled negative edges, as written in (6).

$$\mathcal{L}_{\mathrm{GAE}} = -\sum_{(i,j)\in E_k^{+}} \log \hat{A}_{k,ij} - \sum_{(i,j)\in E_k^{-}} \log\left(1 - \hat{A}_{k,ij}\right). \tag{6}$$

where E_k^{+} denotes the set of positive edges and E_k^{-} denotes the set of sampled negative edges. After this phase, the GAE encoder is frozen, and the embedding e_k^{GAE} from (4) is used as a structural prior for the semi-supervised Phase.

b) Phase II: Semi-supervised phase.: In the second phase, we train a supervised GraphSAGE-based regressor on the labeled subset of DL models. This regressor learns task-specific graph representations from the same DL computation graph used in the unsupervised phase. Let $H_k^{(0)} = X_k$ denote the initial node-feature matrix of graph G_k. For each

supervised layer $t \in \{1, 2\}$, the raw node representation is computed as

$$\widetilde{H}_k^{(t)} = \Phi_{\text{sup}}^{(t)} \left(H_k^{(t-1)}, A_k \right), \qquad H_k^{(0)} = X_k. \tag{7}$$

Here, $\Phi_{\text{sup}}^{(t)}$ denotes the t-th supervised GraphSAGE layer, A_k is the adjacency matrix of graph G_k, and $\widetilde{H}_k^{(t)}$ is the raw node representation produced before runtime conditioning.

To inject runtime effects into the supervised graph representation, we use the context vectors defined in (2). We condition each supervised graph layer using FiLM with the node-level context vector c_k^{node}. For each layer t, a learnable projection $\Psi^{(t)}$ maps c_k^{node} into two FiLM vectors:

$$(\gamma_k^{(t)}, \beta_k^{(t)}) = \Psi^{(t)} \left(c_k^{\text{node}} \right). \tag{8}$$

In (8), $\gamma_k^{(t)}$ and $\beta_k^{(t)}$ are feature-wise scaling and shifting vectors, respectively. These vectors are shared across all nodes of graph G_k and are applied to each node representation. The FiLM-conditioned representation of node v is then given by

$$\bar{H}_{k,v}^{(t)} = \widetilde{H}_{k,v}^{(t)} \odot \left(1 + \alpha \tanh(\gamma_k^{(t)}) \right) + \alpha \beta_k^{(t)}, \qquad v \in V_k. \tag{9}$$

where $\odot$ denotes element-wise multiplication, and α is a small scaling factor that controls the strength of FiLM conditioning. After modulation, the node representations are passed through ReLU and dropout:

$$H_k^{(t)} = \text{Dropout} \left(\text{ReLU} \left(\bar{H}_k^{(t)} \right) \right), \qquad t \in \{1, 2\}. \tag{10}$$

After the two supervised GraphSAGE layers, we obtain a graph-level embedding by applying sum pooling to the final node representations:

$$e_k^{\text{GNN}} = \text{SUMPOOL} \left(H_k^{(2)} \right), \qquad e_k^{\text{GNN}} \in \mathbb{R}^{d_h}. \tag{11}$$

The supervised embedding e_k^{GNN} captures task-specific graph features conditioned by intra-op parallelism and interference. We then combine it with output of autoencoder and the graph-level runtime context c_k^{graph} defined as:

$$v_k = \left[e_k^{\text{GAE}} \| e_k^{\text{GNN}} \| c_k^{\text{graph}} \right]. \tag{12}$$

This representation is passed to a two-layer MLP regressor to predict the tail latency:

$$\hat{y}_k = \text{MLP}(v_k). \tag{13}$$

Finally, the supervised phase is trained using the Huber loss between the predicted tail latency $\hat{y}_k$ and the measured tail latency y_k. For a batch of B labeled samples, the loss is defined as

$$\mathcal{L}_{\text{sup}} = \frac{1}{B} \sum_{k=1}^{B} \ell_\delta \left(y_k - \hat{y}_k \right). \tag{14}$$

$$\ell_\delta(r) = \begin{cases} \frac{1}{2} r^2, & \text{if } |r| \le \delta, \\ \delta \left(|r| - \frac{1}{2} \delta \right), & \text{otherwise.} \end{cases} \tag{15}$$

Here, $r = y_k - \hat{y}_k$ denotes the prediction residual, and δ controls the transition point between the quadratic and linear regions of the Huber loss.

IV. Experimental Setup and Preliminary Results

A. Experimental Setup

In our experimental setup, we use an x86_64 server equipped with an Intel Xeon E5-2697 v4 processor running at 2.30 GHz, exposing 72 logical CPUs (i.e., 36 physical cores with 2 hardware threads per core). On this server, we deploy the inference-serving stack, including TorchServe and our custom inference-serving controller (Figure 1), and we also run a Telegraf agent to collect runtime data. In addition, we use MLPerf LoadGen on a separate VM to emulate an edge client and generate a multistream request pattern, which is representative of latency-sensitive real-time applications such as autonomous vehicles. During execution, we collect tail-latency and CPU usage to evaluate system behavior and quantify SLO violations. These SLO violations are measured against loose, moderate,and strict latency targets. For comparison, we use two fixed baselines defined at the thread-decomposition level rather than as CPU-quota allocation policies. The first baseline, InterOp-Heavy inspired from [8], fixes the smallest available intra-op setting and the largest available inter-op setting in the action space, thereby prioritizing parallelism across independent operations and concurrent requests. The second baseline, is the inverse of the prior Least-InterOp-Available, fixes the smallest available inter-op setting and the largest available intra-op setting supported by the action space, thereby favoring stronger per-operation parallelism and lower orchestration overhead. These two baselines are intentionally used as static, fixed-budget threading policies, allowing us to compare our adaptive policy against two opposite and literature-grounded thread-organization strategies under the same nominal parallelism budget.

B. SLO violation

We start the experiment by fixing the accuracy SLO and varying the tail-latency constraint across three levels: strict, moderate, and loose. For each interference state and each tail-latency constraint, our method evaluates the candidate DL models together with all feasible intra-op and inter-op thread configurations derived from the available CPU resources. Based on the predicted P90 latency, our framework selects the model–thread pair expected to satisfy the target latency SLO while maximizing accuracy. The selected worker is then deployed and tested using a multi-stream request pattern generated by MLPerf LoadGen, which validates the decision under the corresponding interference condition. SLO violations are calculated by comparing the measured P90 latency of the deployed worker against the target latency constraint. If the measured P90 latency exceeds the target SLO, the run is counted as a violation. The final SLO violation rate is computed as the percentage of violating runs over the total number of evaluated runs for each policy and latency-SLO level. The results of the experiement are plotted as a bar chart in the Figure 2. As shown in the Figure The superiority of the proposed approach stems from modeling tail latency

as a joint function of *architecture*, *runtime interference*, and *parallelism*, whereas the two baselines rely on fixed thread-decomposition strategies and therefore cannot adapt to the coupled effect of model structure and system contention. By jointly reasoning over these three axes, the method can select a threading configuration that is better aligned with both the computational characteristics of the deployed model and the instantaneous interference level of the host CPU. This advantage is directly reflected in the updated results, where *Our Approach* achieves SLO-violation rates of 34.11%, 15.72%, and 0.00% under strict, moderate, and loose targets, respectively, compared with 33.33%, 31.00%, and 23.33% for *Least-InterOp-Available*, and 87.67%, 84.00%, and 75.67% for *InterOp-Heavy*. Although the gain is marginal under the strict target, the substantial reduction in violations under the moderate and loose targets shows that the proposed three-axis formulation avoids the persistent mismatch introduced by static policies. Therefore, these updated results further confirm that simultaneously accounting for model architecture, runtime interference, and thread-level parallelism yields the most robust and effective strategy for minimizing tail-latency violations across operating conditions.

V. Conclusion

Our preliminary results show that the proposed constrained model-less serving approach can reduce P90-latency SLO violation rates compared with fixed thread-decomposition baselines, especially under moderate and loose latency targets. These results suggest that jointly considering parallelism, runtime interference, and model architecture can improve latency-aware serving decisions. While this study focuses on image-classification DL models and one component of the serving pipeline, it provides a basis for future work on broader workloads, additional hardware platforms, and end-to-end inference-serving systems.

References

[1] K. B. Letaief, Shi *et al.*, "Edge artificial intelligence for 6g: Vision, enabling technologies, and applications," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 1, pp. 5–36, 2022.

[2] Q. He, J. Lin, H. Fang, X. Wang, M. Huang, X. Yi, and K. Yu, "Integrating iot and 6g: Applications of edge intelligence, challenges, and future directions," *IEEE Transactions on Services Computing*, vol. 18, no. 4, pp. 2471–2488, 2025.

[3] V. J. Reddi, C. Cheng *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[4] J. Zhang, S. Elnikety *et al.*, "Model-Switching: Dealing with fluctuating workloads in Machine-Learning-as-a-Service systems," in *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, Jul. 2020. [Online]. Available: https://www.usenix.org/conference/hotcloud20/presentation/zhang

[5] K. Zhao, Z. Zhou *et al.*, "Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5870–5886, 2023. [Online]. Available: https://doi.org/10.1109/TMC.2022.3189186

[6] F. Romero *et al.*, "Infaas: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 397–411. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/romero

[7] A. Ferikoglou, P. Chrysomeris *et al.*, "Iris: Interference and resource aware predictive orchestration for ml inference serving," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. IEEE, Jul. 2023, pp. 1–12. [Online]. Available: https://doi.org/10.1109/CLOUD60044.2023.00021

[8] M. Salmani, S. Ghafouri, A. Sanaee, K. Razavi *et al.*, "Reconciling high accuracy, cost-efficiency, and low latency of inference serving systems," in *Proceedings of the 3rd Workshop on Machine Learning and Systems*, ser. EuroMLSys '23. ACM, May 2023. [Online]. Available: https://doi.org/10.1145/3578356.3592578

[9] D. Mendoza, F. Romero, and C. Trippel, "Model selection for latency-critical inference serving," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24. ACM, Apr. 2024. [Online]. Available: https://doi.org/10.1145/3627703.3629565

[10] H. Sun, Z. Qian, and A. Zhu, "Bridging the prediction-decision gap: Enhancing model deployment and online service request forecasting in edge inference systems," in *2025 IEEE/ACM International Symposium on Quality of Service (IWQoS)*. IEEE, Jul. 2025, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IWQoS65803.2025.11143526

[11] J. Wolfrath, D. Frink, and A. Chandra, "Sneakpeek: Data-aware model selection and scheduling for inference serving on the edge," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '25. ACM, 2025. [Online]. Available: https://doi.org/10.1145/3772052.3772217

[12] T. Knorst, M. G. Jordan, G. Korol, M. B. Rutzig, and A. C. S. Beck, "Adaptive model switching for dynamic inference serving in the iot-edge-cloud continuum," in *2025 XV Symposium on Computing Systems Engineering (SBESC)*. IEEE, Nov. 2025. [Online]. Available: https://doi.org/10.1109/SBESC68008.2025.11288829

[13] H. Huang, J. Liang, and G. Min, "Joint dnn model deployment, selection, and configuration for heterogeneous inference services toward edge intelligence," *IEEE Transactions on Mobile Computing*, vol. 24, no. 11, pp. 12 726–12 741, Nov. 2025. [Online]. Available: https://doi.org/10.1109/TMC.2025.3586793

[14] A. Khare, D. Garg *et al.*, "Superserve: Fine-grained inference serving for unpredictable workloads," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Apr. 2025. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/khare

[15] F. Stathopoulou, A. Ferikoglou, M. Katsaragakis *et al.*, "Synergai: Edge-to-cloud synergy for architecture-driven high-performance orchestration for ai inference," *arXiv preprint arXiv:2509.12252*, Sep. 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2509.12252

[16] J. Jeong and J. Ahn, "An efficient dnn model serving system using layer-wise caching and direct-host-access," *ACM Transactions on Computer Systems*, vol. 44, no. 1, Jan. 2026. [Online]. Available: https://doi.org/10.1145/3774909

[17] D. Crankshaw, X. Wang *et al.*, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Mar. 2017, pp. 613–627. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[18] Y. Chai, Tripathy *et al.*, "Perfsage: Generalized inference performance predictor for arbitrary deep learning models on edge devices," *arXiv preprint arXiv:2301.10999*, 2023. [Online]. Available: https://arxiv.org/abs/2301.10999

[19] Y. Gao, X. Gu, Zhang *et al.*, "Runtime performance prediction for deep learning models with graph neural network," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2023, pp. 368–380.

[20] Z. Wang, Pengfei Yang, Linwei Hu, Bowen Zhang *et al.*, "Slapp: Subgraph–level attention–based performance prediction for deep learning models," *Neural Networks*, vol. 170, pp. 285–297, 2024.

[21] X. Zhao, J. Sun, Zhang *et al.*, "Perfseer: an efficient and accurate deep learning models performance predictor," in *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, ser. IJCAI '25, 2025. [Online]. Available: https://doi.org/10.24963/ijcai.2025/793

[22] K. P. Selvam and M. Brorsson, "Can semi-supervised learning improve prediction of deep learning model resource consumption?" in *Workshop on Machine Learning for Systems (MLSys), in conjunction with NeurIPS 2023*, 2023, poster (no page numbers). [Online]. Available: https://openreview.net/forum?id=C4nDgK47OJ