NeuRO: Inference-time Profiling and Orchestration of ML Applications at the Edge

Arshad Javeed, György Dán, Viktoria Fodor
Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
{ajaveed,gyuri,vfodor}@kth.se

***Abstract*—We address the problem of orchestrating machine learning (ML) workloads, such as latency-sensitive mobile applications and intelligent radio access network (RAN) control functions, in a heterogeneous edge cloud infrastructure, subject to inference time constraints. We introduce NeuRO, a modular profiling framework that learns to approximate ML task inference time summary statistics using data collected from edge deployments. NeuRO leverages two lightweight neural embeddings: one capturing the resource footprint of an individual ML task, and one capturing the aggregate server context induced by co-located workloads. We show the potential of NeuRO by formulating a performance-aware service placement problem subject to inference time summary statistic constraints. To solve this problem efficiently at scale, we develop a heuristic algorithm that incrementally places applications across servers in a sequential manner. For each server, the constrained optimization subproblem is solved using the barrier method, ensuring tractability despite the non-linear neural network (NN) constraints. We evaluate NeuRO on heterogeneous Kubernetes clusters using realistic ML/AI workloads. Experimental results show that NeuRO not only achieves substantially higher profiling accuracy compared to existing methods, but also improves operator revenue by up to 30%, all while satisfying diverse service-level agreements (SLAs).**

***Index Terms*—Edge Computing, Service orchestration, Performance profiling, Nonconvex optimization**

I. Introduction

Edge clouds are expected to serve as low-latency shared infrastructures for executing latency-sensitive mobile applications and intelligent radio access network (RAN) control functions [1]. As applications and RAN functions are increasingly data-driven, relying on machine learning (ML) models, efficient orchestration of ML-based applications subject to latency constraints is becoming an important requirement for edge cloud operators [2].

Efficient orchestration of ML-based services at the edge, however, is faced with several challenges. First, compute and memory availability are severely constrained compared to traditional cloud environments, compute resources may be heterogeneous and have to be shared among workloads [2], [3]. Second, ML models are known to be resource-intensive, and colocated ML models would compete for hardware resources (compute, cache, I/O, memory bus), negatively affecting their performance [4], [5]. Third, services can have very diverse latency requirements and workloads. For instance, facilitating control functions (such as traffic classification and beam forecasting) requires inference latency in the range $[1, 100]$ ms [6], while user workloads such as image classification or conversational robots may have looser latency requirements.

Existing works on cloud service orchestration typically do not take into account infrastructure heterogeneity and tend to fail at guaranteeing diverse service-level agreements (SLAs) [7]. At the same time, recent works on edge service orchestration lack an accurate and scalable application inference time model that takes into account the effect of resource contention [5], [6]. How to perform efficient service orchestration of ML workloads on a heterogeneous edge infrastructure under diverse SLAs thus remains an open challenge. In this paper, we address this challenge and make the following key contributions:

- We propose *NeuRO*, an extensible and modular NN architecture for predicting various inference times summary statistics of ML workloads, taking into account the impact of resource contention.
- We formulate NeuRO-OPT, the edge service orchestration problem under diverse SLA constraints, and show that it is NP-hard. Importantly, by incorporating NeuRO for predicting the SLAs, we obtain a novel class of optimization problems with neural network constraints.
- We propose to use the barrier method for converting the optimization problem into one that can be solved using an interior point method, and show how to approximate the Jacobians of the neural network for solving the resulting nonconvex optimization problem. To the best of our knowledge, ours is the first work that integrates a NN constraint into a classical optimization problem; thus, our solution methodology may be of broader interest.
- Our results show that *NeuRO* and NeuRO-OPT outperform state of the art methods in terms of inference time estimation accuracy and operator revenue, respectively, while meeting service SLAs.

The rest of the paper is organized as follows. In Section II we review the related work. In Section III we formulate the latency-constrained service orchestration problem. In Section IV we present the *NeuRO* performance profiler. In Section V we present the solution methodology to solve the optimization problem. We discuss the numerical results in Section VI, and conclude the paper in Section VII.

This work was supported in part by the Vinnova Center for Trustworthy Edge Computing Systems and Applications (TECoSA) and in part by the Swedish Research Council under Grant 2020-03860.

II. Related Work

There is a rich recent literature on service orchestration in edge cloud [2]. Authors in [8], [9] focus on resource allocation for services following a multi-graph structure. In [10], authors aim to minimize end-to-end service latency. Authors in [11] propose a low-complexity algorithm for optimal placement of ML applications in a cloud infrastructure. Authors in [3], [12] integrate workload prediction and service placement in the edge cloud. Service orchestration for ML workloads in the online setting was considered in [13]. However, these works on service orchestration do not consider the effect of resource contention when colocating resource-intensive applications in a shared infrastructure.

Recent works that consider the effect of resource contention have mostly focused on serverless autoscaling and request scheduling [4], [14]–[16]. Authors in [14], [15] perform pairwise performance profiling of colocated applications, but the proposed orchestrator does not scale to mixed deployments, and the approach does not take into account the load of the colocated applications. Authors in [16] build an execution time estimator for autoscaling application instances in the cloud. In [4], a binary classifier is proposed to predict whether a colocation profile would satisfy the SLA requirements. However, these works lack an explicit application inference time model to aid placement and orchestration.

Closest to our work are [5], [6]. Authors in [6] proposed a piecewise linear model to approximate the application inference time as a function of the number of colocated application instances. However, the model leads to estimation errors due to not considering application types and workloads. In [5], authors propose a hierarchical Bayesian learning approach to learn pairwise contention models. Heuristics proposed in [17], [18] do not impose hard SLAs and are known to be computationally expensive [19]. Compared to these works, we propose a neural network-based profiler that is accurate and can be adapted to changes in the ML application catalog. In addition, we show how to use the profiler for service orchestration with diverse SLA requirements in heterogeneous infrastructures.

The application of NNs to solve non-convex optimization problems has received increasing attention in recent years [20]. In [21], authors use layers of a NN to syntactically represent the optimization problem, but their approach is limited to quadratic programs. Authors in [22], [23] use backpropagation to perturb the inputs in solving an optimization problem. Similarly, the use of recurrent NNs in solving optimization problems has been explored in the literature [24]–[26]. Our work introduces a fundamentally different case of optimization, where the output of the NN itself is part of the constraints of the problem, and proposes a methodology for solving optimization problems with neural network constraints.

III. System Model and Problem Formulation

A. System Model

Infrastructure: We consider an edge cloud consisting of a set $\mathcal{S}$ of heterogeneous servers, $S = |\mathcal{S}|$. We denote the memory capacity of server $s \in \mathcal{S}$ by C_s and we denote its compute capacity (e.g., the number of CPU cores) by ρ_s. We consider that the servers are used for executing ML-based RAN network functions as well as ML-based application workloads. ML-based RAN workloads are expected to be used for, e.g., power control, spectrum sharing, traffic classification and forecasting, and optimizing antenna configurations for real-time interference management [27]–[29]. Application workloads can be diverse, including computer vision tasks offloaded from UAVs or XR devices, trajectory estimation and sensor fusion for smart city applications, as well as LLMs for conversational robots [30]. We do not consider GPUs in our model, as GPUs are expensive and would be used for RAN workloads with hard latency constraints (baseband, PHY encoding and decoding) [31]. The edge cloud offers a container as a service model for deploying the applications, and we denote the memory requirement of an instance (container) of application $a \in \mathcal{A}$ by C_a.

User requests and Service Level Agreements: There is a set of users (the RAN itself and application service providers), who can submit service requests $\mathcal{R}$. A request $r \in \mathcal{R}$ is characterized by the application type $a^r \in \mathcal{A}$, the number of instances N^r, the task arrival intensity λ^r, the payment π^r, and the latency requirement $\tau^r_{\gamma^r}$, where γ^r is the type of the SLA, chosen from the set $\Gamma = \{\gamma_1, \ldots, \gamma_T\}$ of SLA types offered by the edge cloud operator. For instance, the type of SLA may be the mean, the 75^{th} percentile, or the 95^{th} percentile of the application inference time. We consider that the operator implements load balancing, i.e., the task arrival rate is λ^r/N^r to each application instance.

Note that the SLA is in terms of a summary statistic (mean or some quantile) of the inference time per application instance. Such an SLA does not need to make assumptions about the task arrival process (e.g., interarrival time distribution, batch arrivals); instead, the users are responsible for requesting a suitable inference time in accordance with the task arrival process (which they know better than the operator, and may control) to meet their response time requirement.

B. Problem Formulation

We consider that the edge cloud operator wants to maximize its revenue by serving a subset of a set $\mathcal{R} = \{(a^r, N^r, \lambda^r, \tau^r_{\gamma^r}, \pi^r)\}_{r=1}^{R}$ of service requests, while ensuring that the corresponding SLAs are satisfied. We denote by $z^r \in \{0, 1\}$ the decision of the operator whether request r is to be served, and we denote the placement decision $\mathbf{n} = (n^r_s)_{r,s \in \mathcal{R} \times \mathcal{S}}$, where $n^r_s \in \mathbb{Z}^+$ is the number of application instances of a^r provisioned for request r on server s. Note that the operator must provision a total of n^r instances of application a^r to serve request r, i.e. $\sum_{s \in \mathcal{S}} n^r_s = N^r$ if $z^r = 1$ in order to claim the revenue π^r. Let $\boldsymbol{\tau} = [\tau^1_{\gamma^1}, \tau^2_{\gamma^2}, \ldots, \tau^R_{\gamma^R}]^T$ denote the the γ^r inference time requirement of the requests. The resulting optimization problem can be formulated as

$$\text{(NeuRO-OPT)} \quad \max_{\mathbf{z}} \sum_{r \in \mathcal{R}} z^r \pi^r \tag{1}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} n_s^r C_{a^r} \leq C_s, \ \forall s \in \mathcal{S} \tag{2}$$

$$\mathbf{z}_s \odot \mathbf{t}(\mathbf{z}_s) \leq \boldsymbol{\tau}, \ \forall r \in \mathcal{R}, s \in \mathcal{S} \tag{3}$$

$$\sum_{s \in \mathcal{S}} n_s^r = z^r N^r, \ \forall s \in \mathcal{S}, \tag{4}$$

where $\odot$ represents the dot product, and $\mathbf{t}(\mathbf{z}_s)$ in (3) is an oracle that determines the γ^r inference time summary statistics for each request $r \in \mathcal{R}$, given the placements on server s. Constraint (2) ensures that each application instance is allocated its minimum resource requirement and that the total resource allocation does not exceed the available resource capacity on any of the servers $s \in \mathcal{S}$, (3) enforces that the inference time SLA is met for each admitted request r (i.e., when $z^r = 1$), and (4) ensures that the requested number of application instances is provisioned for each admitted request r, i.e., $\sum_s n_s^r = N^r$ if $z^r = 1$ and 0 otherwise.

Solving NeuRO-OPT is challenging in two ways. First, the inference times in (3) are not known a priori. Thus, formulating the problem requires an accurate predictor of inference times. Second, even if the inference time model $\mathbf{t}$ was given in closed form, the problem would be computationally hard, as we show next.

Proposition 1. *Problem (NeuRO-OPT) is NP-hard.*

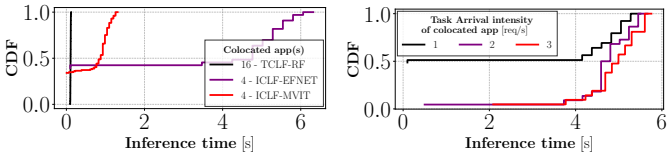
Proof. For $S = 1$, the NeuRO-OPT problem together with constraint (2) corresponds to a KNAPSACK problem, which is known to be NP-hard. Constraints (3) and (4) impose additional nonlinear constraints, and consequently, NeuRO-OPT is a multidimensional KNAPSACK problem. □

In what follows, we address the above two challenges. We propose *NeuRO*, a neural network approximator for predicting the inference time summary statistics of ML applications as a function of the colocated workloads. We then propose a methodology for solving the resulting optimization problem (1) including the neural network constraints.

IV. PERFORMANCE PROFILING

A main challenge in orchestrating machine learning applications subject to inference time constraints is that the inference time is heavily affected by resource contention. Meeting inference time constraints thus requires a model that captures the impact of contention as a function of service orchestration decisions.

Table I shows illustrative examples of ML workloads, including traditional (random forest) and deep ML models (convolutional and graph convolutional networks, LLMs, LSTMs) and the frameworks they are implemented in. To illustrate the impact of contention on the inference time, Fig. 1a shows the CDF of the inference time of the TEXT-BERT application when colocated with different sets of applications and with applications with similar task arrival intensities (we detail the evaluation methodology in Sec. VI). We also observe that the inference time varies drastically depending on the load of the contending application (Fig.1b). Importantly, the impact of these factors on different summary statistics (mean and different quantiles) is highly nonlinear, especially when an application contends with a mix of different types of applications, which is often the case in practice.



(a) Impact of colocating different applications.

(b) Impact of load of colocated application (ICLF-MVIT).

Fig. 1: CDF of inference time of TEXT-BERT application under various deployments.

TABLE I: Benchmark ML apps considered for profiling.

Application	Description	Framework
KNet-GRU	Deep Kalman network [27]	PyTorch
TCLF-RF	Random forest for traffic classification [29]	Scikit-learn
TSTR-LSTM	Traffic steering in 5G ORAN [28]	TensorFlow
TCLF-GCN	Internet traffic classification [32]	PyTorch
TEXT-TBERT	Text classification using tiny BERT [33]	PyTorch
TEXT-BERT	Text classification using BERT [34]	PyTorch
ICLF-MNET	Image classification using MobileNetV3 [35]	PyTorch
ICLF-EFNET	Image classification using EfficientNetV2 [36]	PyTorch
ICLF-MVIT	Image classification using vision Transformer [37]	PyTorch

Previous work has proposed piecewise linear approximation [6] and a composite performance model based on pairwise characterization [5], but these approaches fall short in terms of prediction accuracy and scalability, as we show in Sec. VI

A. NeuRO Architecture

We propose NeuRO, a novel neural network approximator of ML application inference time that captures the nonlinear effects of resource contention, can estimate different SLA targets simultaneously, and can be effectively adapted to changes in the ML application catalog. For simplicity, we omit the server index s when describing the architecture.

We represent an application deployment ψ^i by the tuple $\psi^i = (a^i, n^i, \bar{\lambda}^i)$, where $a^i \in \mathcal{A}$, n^i and $\bar{\lambda}^i$ denote the application type, the number of instances deployed on the server, and the avg. task arrival intensity, respectively. We call the set of application deployments on a server the server context, and denote it by $\Psi = \{(a^i, n^i, \bar{\lambda}^i)\}_{i=1}^I$. Furthermore, we denote the server context from the perspective of application deployment ψ^i by $\Psi^{-\psi^i} = \Psi \setminus \{\psi^i\}$. *NeuRO*, described next, can be used to estimate the inference time summary statistics of an application deployment $\psi^i \in \Psi$ given its context $\Psi^{-\psi^i}$, for server contexts of arbitrary size I, with state of the art accuracy (as we show in Sec. VI).

NeuRO consists of four modules: an embedding layer, a performance predictor, a contention feature extractor, and an inference time regressor, as illustrated in Fig. 2. The key tenet of *NeuRO* is that it learns independent latent embeddings of application types, which it uses to create latent representations of deployments. It then leverages a permutation-invariant transformation to create the server context based on an arbitrary number of deployments, and uses it for estimating the summary statistics of a particular deployment. A key

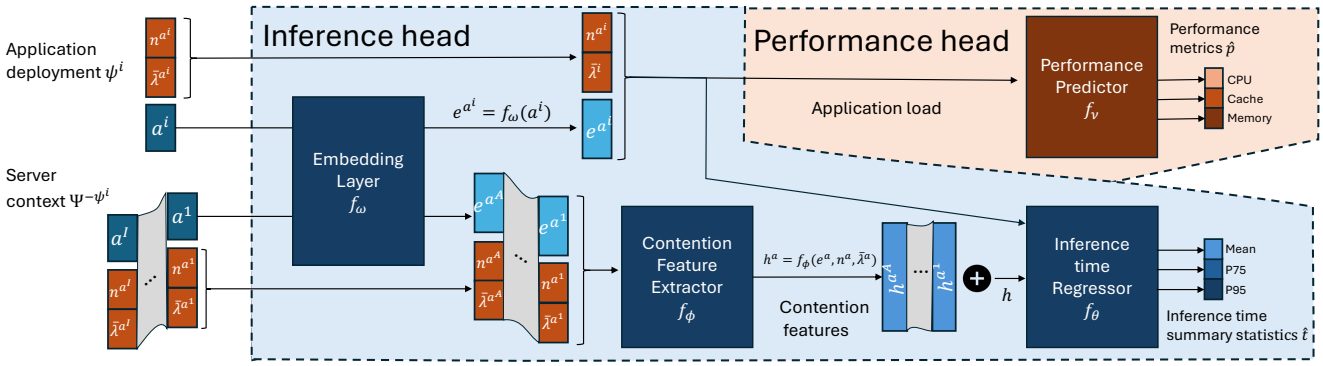


Fig. 2: *NeuRO* application performance predictor architecture. *NeuRO* uses a modular architecture comprising the inference head and performance head. During the training phase, both heads are trained jointly. In the inference phase, the performance head is detached, and the inference head is used for estimating the application inference times.

feature of *NeuRO* is that its architecture allows it to adapt to the changes in the application catalog, by introducing new embedding vectors to the `Embedding Layer`. We now describe each component in detail.

The `Embedding Layer` is used to generate a low-dimensional application embedding vector $e^a = f_\omega(a) \in \mathbb{R}^{d_E}$ for each application type $a \in \mathcal{A}$. This allows us to leverage information pooling, whereby the subsequent blocks in the architecture may adapt to applications of a similar type.

The `Contention Feature-Extractor` creates the contention features due to server context $\Psi^{\psi^{-i}}$, based on the application embedding vectors of deployed applications. Since the server context $\Psi^{\psi^{-i}}$ for application deployment i may consist of an arbitrary number of deployments of an application $a \in \mathcal{A}$, the `Contention Feature-Extractor` considers the aggregate load of each of the applications to generate the application contention features, i.e., $h^a = f_\phi(e^a, n^a, \bar{\lambda}^a) \in \mathbb{R}^{d_C}$, where $n^a = \sum_{\psi'=(a',n',\bar{\lambda}') \in \Psi^{-\psi^i}} \mathbb{1}_{\{a=a'\}} n^{a'}$ and $\bar{\lambda}^a = \frac{1}{n^a} \sum_{\psi'=(a',n',\bar{\lambda}')} \mathbb{1}_{\{a=a'\}} \bar{\lambda}'$. Note that the application contention features are generated per application type, which means that the `Contention Feature-Extractor` can handle an arbitrary number of application deployments in the server context. The ordering of applications should not affect the performance; hence, we have to ensure that the feature representation is permutation-invariant with respect to the context $\Psi^{-\psi^i}$. Thus, we adopt DeepSets [38], a state of the art neural network architecture for set inputs, and we use summation as the permutation-invariant operator to aggregate the application contention features $h = \sum_{a \in \mathcal{A}} h^a$.

The `Inference time Regressor` is used for predicting the inference time summary statistics for application deployment ψ^i. It takes as input the aggregate application contention feature h, the application embedding e^{a^i} of the deployment ψ^i, as well as its workload $((n^{a^i}, \bar{\lambda}^i))$. It then predicts

$$\hat{\mathbf{t}}(\psi^i; \Psi^{-\psi^i}) = \overline{\Gamma}^{a^i} e^{f_\theta\left(f_\omega(a^i), n^i, \lambda^i; \sum_{a \in \mathcal{A}} f_\phi\left(f_\omega(a), n^a, \bar{\lambda}^a\right)\right)}, \tag{5}$$

where we use exp as the output activation function, motivated by the observation that the inference time statistics are non-negative (we show in Section VI that this choice improves accuracy, compared to predicting the inference time directly).

The fourth module is the `Performance Predictor`, which is used only when learning the application type embeddings. The rationale for this module is that different applications make use of the available hardware resources (CPU, memory bus, L1, L2, and L3 caches) to a different extent [4], [39]. For deployment ψ^i this module takes the application embedding e^{a^i} and learns $\hat{p}(\psi^i) = f_\nu\Big(f_\omega(a^i), n^i, \lambda^i\Big)$.

For example, on Linux, one can use the `perf` utility to monitor `cpu-clock` (`CC`), `cache-misses` (`CM`), `page-faults` (`PF`) as application-agnostic performance metrics, i.e., one could define $p =< \texttt{CC}, \texttt{CM}, \texttt{PF} >$. We use the performance predictor to force the embedding layer to learn application embeddings e^a that capture the hardware performance signatures of the application types.

B. Training using Data Augmentation and Zero Forcing

Training *NeuRO* requires collecting a dataset $\mathcal{D} = \{(\Psi_d, \mathbf{t}_d, \mathbf{p}_d)\}_{d=1}^D$ consisting of server contexts along with the measured application inference time statistics and the system performance metrics. Achieving high accuracy requires a large dataset, but creating each data point requires a profiling experiment to be performed, which makes training resource-intensive. We propose two approaches for improving training efficiency and for making *NeuRO* amenable to be used in the constraint in NeuRO-OPT.

First, we propose to augment the dataset by aggregating the metrics of application deployments of the same type (Fig. 3). Consider a data point $d \in \mathcal{D}$ with two application deployments of the same type as part of the server context, i.e., let $\psi^i, \psi^j \in \Psi_d$, such that $a^i = a^j = a$. Then, we create a synthetic application deployment $\psi^{i,j}$ by distributing the load among the instances $\psi^{i,j} = \left(a, n^i + n^j, \frac{\bar{\lambda}^i + \bar{\lambda}^j}{2}\right)$. Similarly, we aggregate the inference and performance metrics for the synthetic deployment as $\mathbf{t}^{i,j} = \frac{\mathbf{t}^i + \mathbf{t}^j}{2}$ and $\mathbf{p}^{i,j} = \frac{\mathbf{p}^i + \mathbf{p}^j}{2}$. Thus, the augmented data point d' may be represented as $d' =$

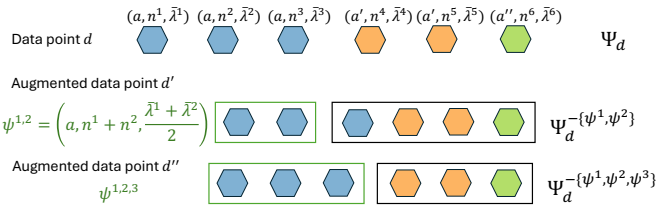


Fig. 3: Data augmentation via aggregating application deployments. Data point d is used to create two synthetic samples.

$\left(\psi^{i,j} \cup \{\Psi_d \backslash \{\psi^i, \psi^j\}\}, \mathbf{t}_{d'}, \mathbf{p}_{d'}\right)$. This increases the number of data points for a given number of profiling experiments, and provides good accuracy if load balancing is used among instances (as we show in Section VI).

Second, we augment the training dataset with synthetic data points that ensure in (5) that $\hat{\mathbf{t}}((a^i, 0, \bar{\lambda}^i); \Psi^{-\psi^i}) = \mathbf{0}$, i.e., if there is no deployed instance on a server then the SLA is met. We leverage this zero-prediction property in alleviating the nonlinearity when solving the optimization problem (discussed in Sec. V-A).

During the training phase, the inference head and the performance heads are trained simultaneously using a mean-squared-error (MSE) loss on the two heads. During the inference phase, the performance head is detached and only the inference time predictor is used.

V. Barrier-based Neural Constraint Optimization

The edge operator can use *NeuRO* for predicting the inference times summary statistics for any server context Ψ, and thus, can use it for formulating the NeuRO-OPT problem. NeuRO-OPT is, however, not only nonconvex and NP-hard, but the presence of the nonlinear neural network-based latency constraint (3) makes the NeuRO-OPT problem nontrivial to solve via techniques such as branch-and-bound. Linear relaxation of the integer variables could be used to obtain a feasible relaxed solution in a computationally efficient way, but dependent rounding schemes, such as Pipage rounding, are ineffective at handling nonconvex constraints [40].

To solve NeuRO-OPT, we propose a linear relaxation-based approach that decouples the multi-server optimization problem into a sequence of single-server optimization problems, with intermediate rounding steps to make the rounding process more efficient (Sec. V-A). To address the neural constraints in the single-server optimization problems, we exploit the fact that the neural network approximator in (5) may be viewed as a twice differentiable function to solve each instance of the relaxed problem by applying a barrier method (Sec. V-B).

A. Linear Relaxation and Problem Decoupling

Linear relaxation: We introduce the relaxed decision variables $\tilde{\mathbf{z}} \in [0,1]^{R \times S}$, such that $\tilde{n}_s^r = \tilde{z}_s^r N^r \in \mathbb{R}_{\geq 0}$ denotes the fractional number of application instances placed on server s for serving request r. Let $\tilde{\mathbf{z}}_s = [\tilde{z}_s^1, \tilde{z}_s^2, \ldots, \tilde{z}_s^R]^\top$ denote the fractional deployment on server s. We have $\sum_{s \in \mathcal{S}} \tilde{z}_s^r \leq 1, \ \forall r \in \mathcal{R}$, which implies $\sum_{s \in \mathcal{S}} \tilde{n}_s^r \leq N^r$. We thus obtain the relaxed optimization problem

$$\text{(NeuRO-OPT-CON)} \quad \max_{\tilde{\mathbf{z}}} \sum_{r \in \mathcal{R}} \tilde{z}_s^r \pi^r \tag{6}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \tilde{n}_s^r C_{a^r} \leq C_s, \ \forall s \in \mathcal{S} \tag{7}$$

$$\hat{t}_{\gamma^r}(\tilde{\psi}_s^r; \tilde{\Psi}^{-\psi_s^r}) \leq \tau_{\gamma^r}^r, \ \forall r \in \mathcal{R}, \forall s \in \mathcal{S} \tag{8}$$

$$\tilde{z}_s^r \leq 1, \quad \forall r \in \mathcal{R}. \tag{9}$$

Observe that (8) is analogous to (3), except that it is now a function of the fractional deployment $\tilde{n}_s^r = \tilde{z}_s^r N^r$. Importantly, note that we omitted z^r on the LHS of (8), since we have that $\hat{t}_{\gamma^r}((a^r, 0, \bar{\lambda}^r); \cdot) = 0$ (zero prediction property of *NeuRO*).

Decoupling: To find a feasible solution such that each $\tilde{z}_s^r N^r$ is integer, we propose to decouple the NeuRO-OPT-CON problem into a sequence of optimization problems for each server s and round each $\tilde{\mathbf{z}}_s$ before the subsequent instances of the optimization problem are solved. Thus, each instance of NeuRO-OPT-CON is expressed as

(NeuRO-OPT-CON-INS)

$$\max_{\tilde{\mathbf{z}}_s} \sum_{r \in \mathcal{R}} \tilde{z}_s^r \pi^r - \sum_{r \in \mathcal{R}} (\tilde{z}_s^r + \hat{z}_s^r)(1 - (\tilde{z}_s^r + \hat{z}_s^r)) \tag{10}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} \tilde{n}_s^r C_{a^r} \leq C_s \tag{11}$$

$$\hat{t}_{\gamma^r}(\tilde{\psi}_s^r, \tilde{\Psi}^{-\tilde{\psi}_s^r}) \leq \tau_{\gamma^r}^r, \ \forall r \in \mathcal{R} \tag{12}$$

$$\tilde{z}_s^r \leq 1 - \hat{z}_s^r, \ \forall r \in \mathcal{R} \tag{13}$$

$$\tilde{z}_s^r \geq 0, \ \forall r \in \mathcal{R}, \tag{14}$$

where $\hat{z}_s^r = \sum_{s' < s} \tilde{z}_{s'}^r$. The second term in the objective function (10) is a penalty term that encourages the cumulative value of $\tilde{z}^r$ to be 1, i.e., it favors requests to be fully served. Constraints (13) and (14) are coupling constraints that ensure that (9) is satisfied.

B. Solving NeuRO-OPT-CON

Note that NeuRO-OPT-CON-INS still contains neural constraints and is thus hard to solve using classical methods. To solve the problem, we propose to introduce barrier functions replacing the inequality constraints (11) & (12) and impose (13) and (14) on the decision variables, i.e., we obtain

(NeuRO-OPT-IPS)

$$\max_{\tilde{z}_s} \mathcal{B}(\tilde{\mathbf{z}}_\mathbf{s}) = \max_{\tilde{z}_s} \sum_{r \in \mathcal{R}} \tilde{z}_s^r \pi^r - \sum_{r \in \mathcal{R}} (\tilde{z}_s^r + \hat{z}_s^r)(1 - (\tilde{z}_s^r + \hat{z}_s^r)) - \alpha \log\left(g_1(\tilde{n}_s)\right) - \sum_{r \in \mathcal{R}} \beta_r \log\left(g_2(\tilde{z}_s^r)\right) \tag{15}$$

where α and $\boldsymbol{\beta} = [\beta^1, \beta^2, \ldots, \beta^R]^\top$ are parameters, $g_1(\tilde{n}_s) = -\sum_{r \in \mathcal{R}} \tilde{n}_s^r C_{a^r} + C_s$ and $g_2(\tilde{z}_s^r) = -\hat{t}_{\gamma^r}(\tilde{\psi}^r, \tilde{\Psi}^{-\psi^r}) + \tau_{\gamma^r}^r$, and $\tilde{\psi}^r = (a^r, \tilde{z}^r N^r, \bar{\lambda}^r)$.

The advantage of incorporating the neural constraint in the objective is that we can solve NeuRO-OPT-IPS using the interior point method, if we can express the Jacobian of the original neural constraint.

Let $f(\tilde{z}_s) = \sum_{r\in\mathcal{R}}(\tilde{z}_s^r + \hat{z}_s^r)(1 - (\tilde{z}_s^r + \hat{z}_s^r))$, and δ and $\boldsymbol{\zeta} = [\zeta^1, \zeta^2, \ldots, \zeta^R]^\top$ denote the complementarity variables, so that $\alpha = \delta g_1(\tilde{n}_s)$ and $\beta^r = \zeta^r g_2(\tilde{z}_s)$, and let $g_2(\tilde{z}_s) = [g_2(\tilde{z}_s^1), g_2(\tilde{z}_s^2), \ldots, g_2(\tilde{z}_s^R)]$. Let J_1 and J_2 represent the Jacobians of g_1 and g_2, respectively. Then,

$$\nabla_{\tilde{z}_s}\mathcal{B} = \Pi - (1+\hat{\mathbf{z}}_s)\odot(1-\hat{\mathbf{z}}_s) - \delta J_1 \odot \nabla_{\tilde{z}_s}\tilde{n}_s - \mathrm{diag}(\zeta)J_2 \tag{16}$$

$$J_1 = \mathbf{1}_{R\times 1} \tag{17}$$

$$J_2 = [\nabla_{\tilde{z}_s} g_2(\tilde{z}_s^1), \nabla_{\tilde{z}_s} g_2(\tilde{z}_s^2), \ldots \nabla_{\tilde{z}_s} g_2(\tilde{z}_s^R)]^\top, \tag{18}$$

where $\Pi = [\pi^1, \pi^2, \ldots \pi^R]^\top$ and $\odot$ represents the dot product. In hindsight, when predicting the inference time statistics for instance a^r, we observe the benefit of not appending $\tilde{n}^{a^r} - 1$ instances as part of the server context for the request r. This allows us to compute the gradients of the decision variable $\tilde{z}^r$. Thus, the elements of the Jacobian $\nabla_{\tilde{z}_s} g_1(\tilde{z}_s^r)$ can be expressed as

$$\frac{\partial \hat{t}_{\gamma^r}(\tilde{\psi}^r; \tilde{\Psi}^{-\tilde{\psi}^r})}{\partial \tilde{z}_s^r} = \frac{\partial f_\theta}{\partial \tilde{n}_s^r}\frac{\partial \tilde{n}_s^r}{\partial z_s^r}\hat{t}_{\gamma^r} = \frac{\partial f_\theta}{\partial \tilde{n}_s^r} N^r \hat{t}_{\gamma^r}, \tag{19}$$

$$\frac{\partial \hat{t}_{\gamma^r}(\tilde{\psi}^r; \tilde{\Psi}^{-\tilde{\psi}^r})}{\partial \tilde{z}_s^{r'}} = \begin{cases} \dfrac{\partial f_\theta}{\partial f_\phi}\dfrac{\partial f_\phi}{\partial \tilde{n}_s^{r'}} N^{r'}\hat{t}_{\gamma^r} & \text{if } r' \in \tilde{\Psi}^{-\tilde{\psi}_s^r} \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

In (19) and (20), we apply chain rule to compute the partial derivatives of the neural network constraint, where f_θ, f_ϕ are the `Inference-time-Regressor` and `Contention Feature-Extractor` NNs, respectively, as detailed in Sec. IV-A. Given the smoothness of the predictor, the partial derivatives $\frac{\partial f_\theta}{\partial \tilde{n}_s^r}$ and $\frac{\partial f_\theta}{\partial f_\phi}\frac{\partial f_\phi}{\partial \tilde{n}_s^{r'}}$ can be approximated via auto-differentiation, which is supported by most ML frameworks (e.g., `PyTorch` & `TensorFlow`). Using (16) - (20), the step sizes ($\mathbf{d}$) for the interior point method are given by

$$\begin{bmatrix} \mathbf{d}_{\tilde{\mathbf{z}}_\mathbf{s}} \\ d_\delta \\ \mathbf{d}_{\boldsymbol{\zeta}} \end{bmatrix} = \begin{bmatrix} H(\tilde{z}_s) & J_1 \odot \nabla_{\tilde{z}_s}\tilde{n}_s & J_2 \\ \delta J_1 \odot \nabla_{\tilde{z}_s}\tilde{n}_s & g_2(\tilde{n}_s) & \mathbf{0} \\ \mathrm{diag}(\boldsymbol{\zeta})J_2 & 0 & g_2(\tilde{z}_s) \end{bmatrix}^{-1} \begin{bmatrix} \Pi - (1+\hat{\mathbf{z}}_s)\odot(1-\hat{\mathbf{z}}_s) - \delta J_1 \odot \nabla_{\tilde{z}_s}\tilde{n}_s - \mathrm{diag}(\boldsymbol{\zeta})J_2 \\ -\delta g_1(\tilde{n}_s) \\ -\mathrm{diag}(\boldsymbol{\zeta})g_2(\tilde{z}_s) \end{bmatrix}, \tag{21}$$

where $H(\tilde{z}_s)$ denotes the Hessian of (15).

C. Sequential Application Placement and Rounding Algorithm

Algorithm 1 shows the pseudocode of the proposed sequential application placement algorithm for solving NeuRO-OPT, based on NeuRO-OPT-IPS. Given a set of user requests $\mathcal{R}$ and a set of servers $\mathcal{S}$, we first sort the servers in increasing order of their capacity ρ_s (Line 3). Sorting in increasing order is motivated by that any feasible solution on a server with lower compute capacity would be feasible on a server with higher compute capacity. Thus, by having $\rho_s < \rho_{s'} < \cdots < \rho_{s''}$ we explore a larger solution space. Starting with the server with the lowest compute capacity, for each server s we obtain a relaxed solution $\tilde{\mathbf{z}}_s$ by solving NeuRO-OPT-IPS given in (15).

Algorithm 1 Sequential Application Placement & Rounding (SAPR) Algorithm

Require: Set of user requests $\mathcal{R}$, set of servers $\mathcal{S}$
1: Initialize $\mathcal{R}_s = \mathcal{R}$, $\bar{\mathbf{z}} = \mathbf{0}$
2: ▷ **Rank servers**
3: $\mathcal{S}_\rho \leftarrow \{s, s', \ldots, s'' \mid \rho_s < \rho_{s'} < \cdots < \rho_{s''}\}$
4: **for each** $s \in \mathcal{S}_\rho$ **do**
5: ▷ **Obtain $\tilde{z}_s$ by solving NeuRO-OPT-IPS**
6: ▷ **Reduce partial resource allocations**
7: Collect partially served requests, $\mathcal{X} = \{r \in \mathcal{R}_s : 0 < \sum_{s'\le s} \tilde{z}_{s'}^r < 1\}$
8: $\tilde{\mathbf{x}}_\mathbf{s} \leftarrow \mathrm{RRC}(\tilde{\mathbf{z}}, s, \mathcal{X})$
9: $\tilde{z}_s^r \leftarrow \tilde{x}_s^r,\ \forall r \in \mathcal{X}$
10: ▷ **Round decision variables to the feasible integer deployments**
11: $\tilde{\mathbf{z}}_s = [\tilde{z}_s^r, \tilde{z}_s^{r'}, \ldots, \tilde{z}_s^{r''}]$, $\mathbf{N} = [N^r, N^{r'}, \ldots, N^{r''}]$ such that $\pi^r \tilde{z}_s^r > \pi^{r'}\tilde{z}_s^{r'} > \cdots > \pi^{r''}\tilde{z}_s^{r''}$
12: $\bar{\mathbf{z}}_s \leftarrow \mathrm{RFS}(\tilde{\mathbf{z}}_s, \mathbf{N})$
13: ▷ **Eliminate served requests**
14: $\tilde{\mathcal{R}} = \left\{r \in \mathcal{R}_s : \sum_{s'\le s} \bar{z}_s^r < 1\right\}$
15: $\mathcal{R}_s = \tilde{\mathcal{R}}$
16: **end for**
17: **return** $\bar{\mathbf{z}}$

We then round $\tilde{\mathbf{z}}_s$ to a feasible integer allocation $\bar{\mathbf{z}}_s$, such that $\bar{z}_s^r N^r \in \mathbb{Z}^+$.

When there are many requests, there may be many partial allocations. Therefore, we propose a two-step process for rounding, consisting of placement coalescing (Algorithm 2) and the rounding itself (Algorithm 3). Algorithm 2 finds pairs of requests of the same application type a and SLA type γ, and reallocates resources from the request with higher computational load to that with lower computational load (Algorithm 2, Lines 7 - 9). Next, we round the placements so that each $\bar{z}_s^r N^r$ leads to a feasible integer deployment. We sort the fractional allocations considering the earned revenue (Line 11) and attempt to round the solution up if $\bar{z}_s^r N^r$ yields a feasible solution, and round down otherwise (Algorithm 3 Lines 3 - 6). We then update the set of requests left to be placed (i.e., partially allocated requests, Line 14) and we continue with the next server.

To show the correctness of the algorithm, we next prove that the proposed resource coalescing algorithm does not create infeasible placements.

Proposition 2. *Algorithm 2 produces a feasible solution, that is, the solution $\bar{\mathbf{z}}$ satisfies constraints* (11) *and* (12)*.*

Proof. Let $\tilde{\mathbf{z}}_s$ be the fractional solution obtained by solving NeuRO-OPT-IPS (Line 5 Algorithm 1), and let for user requests r, r' hold $a^r = a^{r'} = a$, $\gamma^r = \gamma^{r'}$, $\tau^r \ge \tau^{r'}$ and $\bar{\lambda}^r \le \bar{\lambda}^{r'}$. Since $a^r = a^{r'}$, let $C_{a^r} = C_{a^{r'}} = C_a$.

We first consider the capacity constraint. Since $\tilde{\mathbf{z}}_s$ is a fea-

Algorithm 2 Residual Resource Coalescing (RRC) Algorithm

Require: Placement matrix $\tilde{\mathbf{z}}$, Server index s, Partially served requests $\mathcal{X}$

Initialize $\tilde{\mathbf{x}} = \tilde{\mathbf{z}}$
for $r \in \mathcal{X}$ **do**
  for $r' \in \mathcal{X}_{>r}$ **do**
    if $\sum_{s' \leq s} \tilde{x}^r_s < 1$ and $a^r = a^{r'}$ and $\gamma^r = \gamma^{r'}$ and $\tau^r \geq \tau^{r'}$ and $\bar{\lambda}^r \leq \bar{\lambda}^{r'}$ **then**
      Instances needed for r, $m^r = N^r(1 - \sum_{s' \leq s} \tilde{x}^r_{s'})$
      Instances allocated for r', $p^{r'} = N^{r'} \tilde{x}^r_s$
      Let $v = \min(m^r, p^{r'})$
      $\tilde{x}^r_s \leftarrow \tilde{x}^r_s + \frac{v}{N^r}$
      $\tilde{x}^{r'}_s \leftarrow \tilde{x}^{r'}_s - \frac{v}{N^{r'}}$
    end if
  end for
end for
return $\tilde{\mathbf{x}}$

Algorithm 3 Round Fractional Solution (RFS) Algorithm

Require: Fractional solution $\tilde{\mathbf{z}}$, no. of requested instances $\mathbf{N}$

Initialize $\epsilon << 1$, $\bar{\mathbf{z}} = \tilde{\mathbf{z}}$
for $i = 1$ to R_s **do**
  $\bar{z}^{(i)} \leftarrow \left\lceil \frac{\tilde{z}^{(i)}_s}{1/N^{(i)}} \right\rceil 1/N^{(i)}$
  while $\bar{z}^{(i)}$ is not feasible w.r.t (11) & (12) **do**
    $\bar{z}^{(i)} \leftarrow \left\lfloor \frac{\tilde{z}^{(i)}_s - \epsilon}{1/N^{(i)}} \right\rfloor 1/N^{(i)}$
  end while
end for
return $\bar{\mathbf{z}}$

sible solution, we have that $\tilde{\mathbf{z}}_s$ satisfies the capacity constraint,

$$\sum_{q \in \mathcal{X} \setminus \{r, r'\}} C_{a^q} \tilde{z}^q_s N^q + C_a \tilde{z}^r_s N^r + C_a \tilde{z}^{r'}_s N^{r'} \leq C_s \quad (22)$$

Let $m^r = N^r(1 - \sum_{s' \leq s} \tilde{x}^r_{s'})$ denote the resource needed by request and $p^{r'} = N^{r'} \tilde{x}^r_s$ be the resource allocated to request r' and let $v = \max(m^r, p^{r'})$. Then, adding and subtracting $C_a v$, we get

$$\sum_{q \in \mathcal{X} \setminus \{r, r'\}} C_{a^q} \tilde{z}^q_s N^q + C_a(\tilde{z}^r_s + \frac{v}{N^r}) N^r + C_a(\tilde{z}^{r\prime}_s - \frac{v}{N^{r'}}) N^{r'} \leq C_s, \quad (23)$$

Let us now look at the latency constraints. Relocating the resource from the request r' to r satisfies the latency constraint for r due to the assumption that $\tau^r_{\gamma^r} \geq \tau^{r'}_{\gamma^{r'}}$ and $\gamma^r = \gamma^{r'}$. Note that if $v = p^{r''}$, we have $\hat{t}_{\gamma^r}((a^{r'}, 0, \bar{\lambda}^{r'}); \cdot) = 0 \leq \tau^{r'}_{\gamma^{r'}}$ (zero prediction property). Since $\bar{\lambda}^r \leq \bar{\lambda}^{r'}$, resource contention is reduced for the other colocated instances, ensuring that the latency constraints are met. □

VI. Numerical Results

We evaluated *NeuRO* and the proposed solution to NeuRO-OPT using realistic ML workloads and compared them against the state-of-the-art.

A. Evaluation Methodology

We consider $S \in \{1, 2, 3, 5, 7, 10\}$ servers, representative of edge cloud deployments [41], [42]. We consider two types of servers with $C_s = 32$GB of RAM: i) `EPYC64`: Dell PowerEdge-R7515 server with 1 × AMD EPYC 7543P CPU with 64 cores ($\rho_{\texttt{EPYC64}} = 64$), and ii) `XEON32`: Dell PowerEdge-R750xs server with 2 × Intel Xeon Silver 4314 CPUs with 16 cores each ($\rho_{\texttt{XEON32}} = 32$).

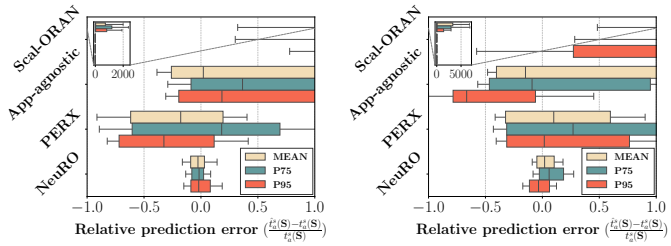
The application catalog (`app-cat-v2`, unless otherwise noted) consists of 9 highly distinct ML models, and is a mix of real-time inference (`RT`) and edge (`E`) applications, shown in Table II. For the `RT` applications, users may request $N^r \in \{2, \ldots, 10\}$ application instances with task arrival intensity λ^r uniformly distributed on $[20, 100]$ tasks/sec. For `E` applications, requests are for $N^r \in \{1, \ldots, 3\}$ application instances with task arrival rate λ^r uniformly distributed on $[1, 3]$ tasks/sec. For both application types, we consider that the task arrival process is Poisson, which is representative of mMTC and uRLLC traffic in edge computing [43]–[45]. We consider the mean (`M`), 75^{th} (`P75`), and 95^{th} (`P95`) percentiles for the user SLA targets. We consider a linear pricing scheme as in [46] and assume that the payment offered by the users π^r is proportional to the requested workload $\pi^r = \bar{\pi}_{a^r} \sigma(\gamma^r) \left(1 + 0.15 \times \left(\frac{\lambda^r}{100} + \frac{N^r}{5}\right)\right)$, where $\bar{\pi}_{a^r}$ is the base payment for the application a^r (Table II), and we let $\sigma(\text{P95}) = 1.1\sigma(\text{P75}) = 1.375\sigma(\text{M})$ to make the monetary value depend on the requested SLA type. The user latency requirements $\tau^r_{\gamma^r}$ are uniformly chosen within the respective ranges, and the SLA type γ^r is randomly chosen from $\{\texttt{P95}, \texttt{P75}, \texttt{M}\}$ with equal probability for each user request.

We used 4 lightweight NNs to implement *NeuRO*. The `Embedding Layer` uses $d_E = 5$ as the embedding dimension. We found that using a lower embedding dimension does not allow us to extend the application catalog. The `Contention Feature-Extractor` and the `Inferencetime Regressor` use a NN with 2 hidden layers with 50 and 25 neurons. The `Contention Feature-Extractor` produces a latent representation in $d_C = 5$ dimensions. We use Gaussian Error Linear Unit (GELU) as the activation function for all the hidden layers. Our choice is motivated by the fact that GELU is differentiable at the origin, while ReLU makes numerical approximations to the gradient around the origin. We use IPOpt to solve each instance of the NeuRO-OPT-IPS problem and provide explicit Jacobians of the objective and constraint functions to the optimizer.

We consider three baselines for comparison. The first

Catalog		Use case	Application	Memory Requirement C_a [MB]	Base Payment $\bar{\pi}_a$ [$]	Latency Requirement τ_a [ms]
app-cat-v2	app-cat-v1	RT	KNet-GRU	300	3	[5, 25]
		RT	TCLF-RF	300	3	[20, 75]
		RT	TSTR-LSTM	1,000	6	[200, 600]
		E	TCLF-GCN	500	6	[1,000, 5,000]
		E	ICLF-MNET	800	7	[1,000, 5,000]
		E	TEXT-BERT	600	8	[1,000, 5,000]
		E	TEXT-TBERT	700	7	[1,000, 3,000]
		E	ICLF-EFNET	1,000	5	[1,000, 5,000]
		E	ICLF-MVIT	500	5	[1,000, 5,000]

TABLE II: User request parameters per application type.



(a) EPYC64 (b) XEON32

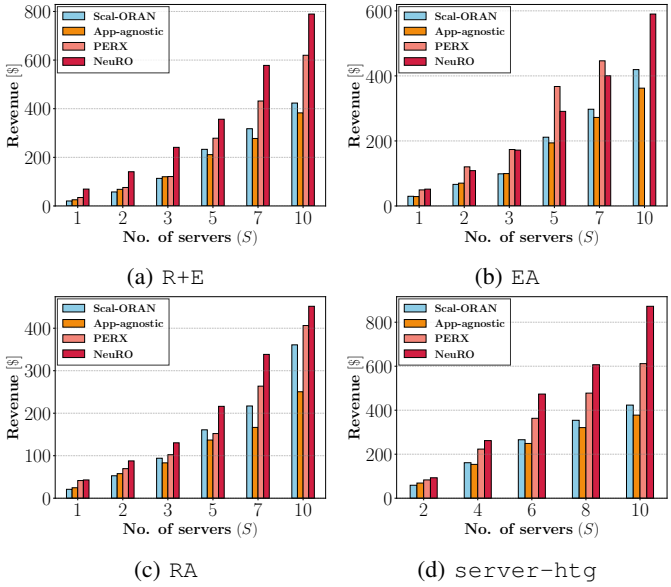
Fig. 4: Profiler prediction accuracy on two server types.

baseline is Scal-ORAN [6], which ensures that the average inference time requirement for a given server placement meets the latency requirement. The second baseline is PERX [5], which uses a hierarchical Bayesian learning approach for pairwise application profiling, on which it builds a composite model. Third, we consider App-agnostic as a variant of Scal-ORAN, which assumes that all applications contend in the same way. We formulate the orchestration with these three baselines as integer optimization problems and solve them via a branch-and-bound (BB) solver, as in [5], [6].

For training *NeuRO*, we collected a dataset of application inference time summary statistics from 700 random application deployments on each of the two server types. Since PERX involves pairwise profiling of the applications, we performed 1,296 experiments by systematically colocating the applications in a pairwise manner for $\bar{\lambda} \in \{1, 2, 3, 4\}$ tasks/sec as the load and $n \in \{2, 4, 8, 16\}$ as the number of instances for the colocated instances. For the validation dataset, we collected a dataset of 800 data points gathered from random application deployments on a Kubernetes cluster. We performed the profiling for both server types (EPYC64 and XEON32) independently, resulting in two performance models.

B. Profiler Performance

We begin by evaluating the accuracy of *NeuRO*. Fig. 4 plots the normalized prediction error for three inference time summary statistics for each of the server classes. We observe that *NeuRO* provides the most accurate prediction of the application inference times, with relative prediction errors concentrated around zero. We also observe that PERX tends to underestimate the inference time statistics on EPCY64 servers (Fig. 4a), which can potentially lead to SLA violations. On the contrary, Scal-ORAN has a tendency to significantly overestimate the application inference times as a consequence of averaging, e.g. the relative prediction error of the KNet-GRU (with an inference time of 5 ms) application when colocated with a few instances of ICLF-MVIT application (with an inference time of 10 s) would be $\frac{\frac{0.005+10}{2}}{0.005} \approx 1,000$. As a consequence, Scal-ORAN would not colocate applications with dissimilar inference times, potentially leading to under-utilization of resources. Finally, the App-agnostic model has a tendency to overestimate the application inference time summary statistics on both server types (Fig. 4a & 4b).



(a) R+E (b) EA

(c) RA (d) server-htg

Fig. 5: Revenue vs. no. of servers S under different scenarios.

C. Service Orchestration Performance

We first consider a homogeneous edge cloud consisting of EPYC64 servers, and consider three scenarios for the user requests. In scenario R+E there is an equal proportion of RT and E user requests, while in scenarios EA and RA the proportion of E to RT user requests is 7 : 3 and 3 : 7, respectively.

Fig. 5(a)-(c) shows the operator revenue as a function of the number of servers S, for the three scenarios, for $R = 15S$ requests. In scenarios R+E and RA (Figs. 5a and Figs. 5c), we observe that *NeuRO* consistently outperforms PERX by up to 30%, and the other baselines by up to 50 %. In the case of scenario EA (Fig. 5b), PERX achieves a marginally higher revenue than *NeuRO*, but as we will later observe, PERX violates the user SLAs in scenario EA. This is explained by the fact that PERX has a tendency to underestimate inference time metrics, as seen in Fig. 4a. We note that Scal-ORAN and App-agnostic under-utilize the resources, achieving much lower revenue as a consequence of overestimating the application inference times. It is worth noting that the baseline PERX was unable to find a feasible solution for $S = 10$ (Fig.5b) with a time cap of 250 mins, whereas *NeuRO* found the solution in about 33 mins.

We also consider a heterogeneous edge cloud (scenario server-htg) consisting of $S \in \{2, 4, 6, 8, 10\}$ servers, half of which are EPYC64 and half XEON32. Fig. 5d shows the corresponding results for the R+E scenario, and shows that *NeuRO* handles heterogeneity well, and outperforms PERX and Scal-ORAN by up to 30% and 60%, respectively.

To provide more insight into the resulting placements, Fig. 6 shows the average number of application requests served per application type for $S = 5$ homogeneous servers under the three scenarios. We observe that *NeuRO* admits more edge applications than the other baselines, which explains the

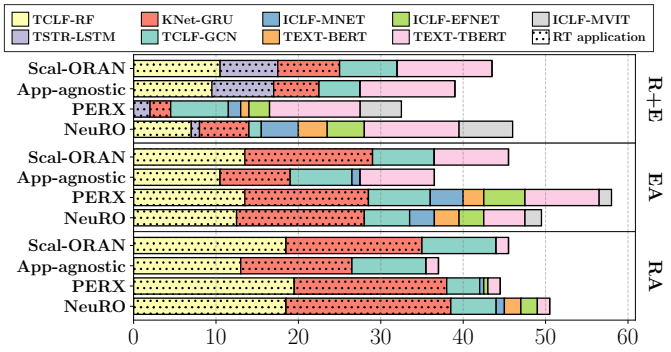


Fig. 6: Avg. no. of served applications per application type for $S = 5$ under different scenarios.

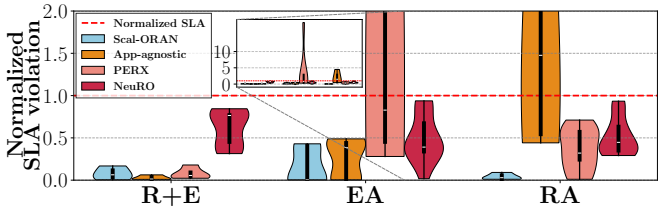


Fig. 7: Normalized SLA metric violation for $S = 3$ (server `EPYC64`). The dashed line marks the normalized SLA target.

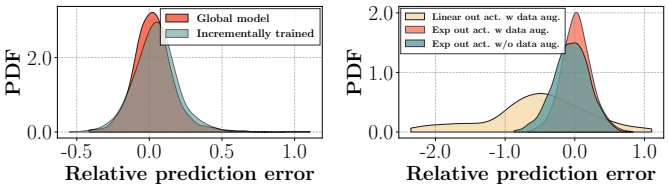
higher operator revenue. Furthermore, we observe that Scal-ORAN and App-agnostic fail to admit ICLF-EFNET, ICLF-MNET, and ICLF-MVIT applications, while *NeuRO* manages to colocate these with other `RT` applications. This is a direct consequence of the higher accuracy of the *NeuRO* profiler.

D. Experimental Validation of SLA Conformance

To evaluate the ability of the orchestrator in meeting the user SLAs, we deployed the respective placements on a Kubernetes cluster and exposed them to user tasks over a period of 3 minutes, measuring the inference time statistics. Fig. 7 shows the normalized SLA violations, that is, the ratio of the observed inference time statistic to the user SLA requirement (`M`, `P75`, `P95`) for $S = 3$ homogeneous servers, i.e., a value of 1.0 or below implies that the SLA was met. We observe that *NeuRO* satisfies the SLAs under all scenarios. On the contrary, PERX violates the SLAs, e.g., in the `EA` scenario where it achieved marginally higher revenue than *NeuRO* (Fig. 5b). We also observe that App-agnostic results in SLA violations in the `RA` scenario as the application-agnostic contention model leads to an underestimation of the application inference time statistics of `RT` applications when colocated with the `E` applications. Scal-ORAN, on the other hand, results in the least SLA violations due to the tendency to overestimate the inference time statistics (Fig. 5b).

E. Incremental Training and Ablation Study

To show the adaptability of NeuRO, we show how it allows to extend the application catalog at low computational cost, without affecting prediction accuracy. For the evaluation, we



(a) Predictor adaptability (b) Learning performance

Fig. 8: Adaptability and ablation study for NeuRO.

collected a dataset comprising 300 random deployments for the application catalog `app-cat-v1` (dataset `data-cat1`) and an additional dataset using 400 random deployments with the catalog `app-cat-v2` (dataset `data-cat2`), and a dataset using 100 random deployments as the validation set.

We first trained *NeuRO* on `data-cat1` for 15 epochs. We then expanded the `Embedding-Layer` by introducing three embedding vectors corresponding to the new applications in `data-cat2`, without further training (incrementally trained model). We also trained a model from scratch on `data-cat2` for 15 epochs (global model). Fig.8a shows the relative prediction errors of the global model and the incrementally trained model. We observe that the performance of the incrementally trained model is comparable to that of the global model, which indicates that *NeuRO* can use previously learned embeddings to adapt to new application types, i.e., it needs no access to the whole application catalog a priori, making it a promising candidate for practical edge deployments.

Finally, we assess the impact of the data augmentation technique (Sec. IV-B) and that of exponential activation in (5). The data augmented training procedure uses one third of the `data-cat2` dataset; the non-augmented training uses the entire dataset. Fig. 8b shows the relative prediction error of the three approaches. We observe that linear activation results in a lower prediction accuracy due to naive prediction errors, including negative predictions (a relative prediction error less than -1). We also observe that the training procedure with data augmentation achieves the highest prediction accuracy, despite using only a third of the dataset, thus reducing the number of profiling experiments to be performed.

VII. Conclusion

We proposed *NeuRO*, a novel ML application profiling framework that uses a modular NN architecture to learn non-linear performance models of contending ML applications. We show how *NeuRO* can be used for service orchestration under SLA constraints, resulting in a novel integer programming problem with NN constraints. We developed a computationally tractable methodology for dealing with NN constraints, based on the barrier method and using a novel decoupled rounding scheme that scales to large problem sizes. Our numerical results show that *NeuRO* outperforms existing methods in terms of profiling accuracy and operator revenue, and can be adapted as the application catalog changes. The authors have provided public access to the source code at [47].

REFERENCES

[1] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-Computing-Enabled Smart Cities: A Comprehensive Survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 200–10 232, 2020.

[2] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures – A Technology Review," in *Proc. of International Conference on Future Internet of Things and Cloud*, 2015, pp. 379–386.

[3] Q. Liu, S. Huang, J. Opadere, and T. Han, "An Edge Network Orchestrator for Mobile Augmented Reality," in *Proc. of IEEE INFOCOM*, 2018, pp. 756–764.

[4] S. Li, W. Wang, J. Yang, G. Chen, and D. Lu, "Golgi: Performance-Aware, Resource-Efficient Function Scheduling for Serverless Computing," in *Proc. of ACM SoCC*, 2023, p. 32–47.

[5] A. Javeed, V. Fodor, and G. Dán, "PERX: Energy-aware O-RAN Service Orchestration with Pairwise Performance Profiling," in *Proc. of IEEE INFOCOM Workshop NG-OPERA*, 2025.

[6] S. Maxenti, S. D'Oro, L. Bonati, M. Polese, A. Capone, and T. Melodia, "ScalO-RAN: Energy-aware Network Intelligence Scaling in Open RAN," in *Proc. of IEEE INFOCOM*, 2024, pp. 891–900.

[7] M. Ghorbian, M. Ghobaei-Arani, and R. Asadolahpour-Karimi, "Function Placement Approaches in Serverless Computing: A Survey," *Journal of Systems Architecture*, vol. 157, 2024.

[8] F. Mungari, C. Puligheddu, A. Garcia-Saavedra, and C. F. Chiasserini, "OREO: O-RAN intElligence Orchestration of xApp-based network services," in *Proc. of IEEE INFOCOM*, 2024, pp. 71–80.

[9] ——, "O-RAN Intelligence Orchestration Framework for Quality-Driven xApp Deployment and Sharing," *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 4811–4828, 2025.

[10] A. Das, S. Imai, S. Patterson, and M. P. Wittie, "Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement," in *Proc. of IEEE CCGRID*, 2020, pp. 41–50.

[11] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN," in *Proc. of IEEE INFOCOM*, 2022, pp. 270–279.

[12] K. Ali and M. Jammal, "Proactive VNF Scaling and Placement in 5G O-RAN Using ML," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 174–186, 2024.

[13] T. Ouyang, K. Zhao, X. Zhang, Z. Zhou, and X. Chen, "Dynamic Edge-centric Resource Provisioning for Online and Offline Services Colocation," in *Proc. of IEEE INFOCOM*, 2023, pp. 1–10.

[14] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, and S. Bagchi, "ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs," in *Proc. of USENIX OSDI*, 2022.

[15] H. Tian, S. Li, A. Wang, W. Wang, T. Wu, and H. Yang, "Owl: Performance-aware Scheduling for Resource-efficient Function-as-a-service Cloud," in *Proc. of ACM SoCC*, 2022, p. 78–93.

[16] A. Singhvi, A. Balasubramanian, K. Houck, M. D. Shaikh, S. Venkataraman, and A. Akella, "Atoll: A Scalable Low-Latency Serverless Platform," in *Proc. of the ACM SoCC*, 2021, p. 138–152.

[17] Y. Mao, X. Shang, and Y. Yang, "Ant Colony based Online Learning Algorithm for Service Function Chain Deployment," in *Proc. of IEEE INFOCOM*, 2023, pp. 1–10.

[18] W. Xia and L. Shen, "Joint Resource Allocation at Edge Cloud based on Ant Colony Optimization and Genetic Algorithm," *Wireless Personal Communications*, vol. 117, no. 2, pp. 355–386, 2021.

[19] S. T. Ng and Y. Zhang, "Optimizing Construction Time and Cost Using Ant Colony Optimization Approach," *Journal of construction engineering and management*, vol. 134, no. 9, pp. 721–728, 2008.

[20] X.-S. Zhang, *Neural Networks in Optimization*. Springer Science & Business Media, 2013, vol. 46.

[21] B. Amos and J. Z. Kolter, "OptNet: Differentiable Optimization as a Layer in Neural Networks," in *Proc. of ICML*, Aug. 2017, pp. 136–145.

[22] L. Zhang, F. Wang, T. Sun, and B. Xu, "A constrained optimization method based on bp neural network," *Neural Computing and Applications*, vol. 29, no. 2, pp. 413–421, 2018.

[23] Y. Xia, H. Leung, and J. Wang, "A Projection Neural Network and its Application to Constrained Optimization Problems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 4, pp. 447–458, 2002.

[24] Y. Xia, G. Feng, and J. Wang, "A Novel Recurrent Neural Network for Solving Nonlinear Optimization Problems With Inequality Constraints," *IEEE Transactions on Neural Networks*, vol. 19, no. 8, pp. 1340–1353, 2008.

[25] X.-B. Liang and J. Wang, "A recurrent neural network for nonlinear optimization with a continuously differentiable objective function and bound constraints," *IEEE Transactions on Neural Networks*, vol. 11, no. 6, pp. 1251–1262, 2000.

[26] Y. Xia and J. Wang, "A recurrent neural network for nonlinear convex optimization subject to nonlinear inequality constraints," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1385–1394, 2004.

[27] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. G. van Sloun, and Y. C. Eldar, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics," *IEEE Transactions on Signal Processing*, vol. 70, pp. 1532–1547, 2022.

[28] F. Kavehmadavani, V.-D. Nguyen, T. X. Vu, and S. Chatzinotas, "Intelligent Traffic Steering in Beyond 5G Open RAN Based on LSTM Traffic Prediction," *IEEE Transactions on Wireless Communications*, vol. 22, no. 11, pp. 7727–7742, 2023.

[29] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust Smartphone App Identification via Encrypted Network Traffic Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2018.

[30] J. J. Bird, A. Ekárt, and D. R. Faria, "Chatbot Interaction with Artificial Intelligence: human data augmentation with T5 and language transformer ensemble for text classification," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 4, pp. 3129–3144, 2023.

[31] L. L. Schiavo, J. A. Ayala-Romero, A. Garcia-Saavedra, M. Fiore, and X. Costa-Perez, "YinYangRAN: Resource Multiplexing in GPU-Accelerated Virtualized RANs," in *Proc. of IEEE INFOCOM*, 2024, pp. 721–730.

[32] S. Xu, J. Han, Y. Liu, H. Liu, and Y. Bai, "Few-shot Traffic Classification based on Autoencoder and Deep Graph Convolutional Networks," *Scientific Reports*, vol. 15, no. 1, p. 8995, 2025.

[33] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for Natural Language Understanding," in *Proc. of EMNLP*, 2020, pp. 4163–4174.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. of ACL Anthology*, 2019.

[35] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. of IEEE ICCV*, 2019, pp. 1314–1324.

[36] M. Tan and Q. Le, "EfficientNetV2: Smaller Models and Faster Training," in *Proc. of PMLR ICML*, vol. 139, 2021, pp. 10 096–10 106.

[37] S. Mehta and M. Rastegari, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," in *Proc. of ICLR*, 2022.

[38] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[39] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing Serverless Platforms with Serverlessbench," in *Proc. of ACM SoCC*, 2020, p. 30–44.

[40] A. A. Ageev and M. I. Sviridenko, "Pipage Rounding: A New Method of Constructing Algorithms with Proven Performance Guarantee," *Journal of Combinatorial Optimization*, vol. 8, no. 3, pp. 307–328, 2004.

[41] S. Jošilo and G. Dán, "Selfish Decentralized Computation Offloading for Mobile Cloud Computing in Dense Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 207–220, 2019.

[42] S. Jošilo and G. Dán, "A Game Theoretic Analysis of Selfish Mobile Computation Offloading," in *IEEE INFOCOM*, 2017, pp. 1–9.

[43] B. Li, T. Patel, S. Samsi, V. Gadepally, and D. Tiwari, "MISO: Exploiting Multi-instance GPU Capability on Multi-tenant GPU Clusters," in *ACM SoCC*, 2022, p. 173–189.

[44] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "ColO-RAN: Developing Machine Learning-Based xApps for Open RAN Closed-Loop Control on Programmable Experimental Platforms," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5787–5800, 2023.

[45] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "{Heterogeneity-Aware} Cluster Scheduling Policies for Deep Learning Workloads," in *Proc. of USENIX OSDI*, 2020.

[46] F. Tütüncüoğlu, A. Ben-Ameur, G. Dán, A. Araldo, and T. Chahed, "Dynamic Time-of-Use Pricing for Serverless Edge Computing with Generalized Hidden Parameter Markov Decision Processes," in *Proc. of IEEE ICDCS*, 2024, pp. 668–679.

[47] A. Javeed. [Online]. Available: https://github.com/arshad171/neuro.git