CAP: Optimal Pricing of Machine Learning Workloads at the Edge under Resource Contention

Siying Chen, Arshad Javeed, Viktoria Fodor, György Dán
Department of Network and Systems Engineering and Digital Futures
KTH Royal Institute of Technology, Sweden
{siyingch,ajaveed,vfodor,gyuri}@kth.se

***Abstract*—We study the problem of joint pricing and orchestration of machine learning (ML) workloads on edge cloud infrastructures, where heterogeneous applications with strict inference time requirements compete for limited resources. Pricing plays a dual role in this setting: it directly determines the revenue of the edge operator, while it indirectly shapes the effective workload and, through resource contention, the performance of the ML applications. We consider a dynamic population of users with ML workloads, and we develop Contention-aware Pricing (*CAP*), a teletraffic theory-based static pricing strategy that explicitly accounts for demand elasticity and latency constraints. While conceptually simple, *CAP* maximizes the revenue of the operator while ensuring that inference-time requirements are met under resource contention. Our experimental results show that optimal pricing can significantly improve system performance: *CAP* achieves up to 40% higher revenue compared to baselines, while consistently satisfying strict service level agreements in terms of inference time.**

***Index Terms*—Edge computing, pricing, teletraffic theory**

I. Introduction

Edge clouds in radio access networks (RANs) are expected to provide computing capability at the network edge, making it possible to execute computationally demanding applications with low delays. These applications may directly support RAN operation, such as traffic classification, beam forecasting, and optimizing RAN configurations [1], [2], [3]. They could also support external users staying in the vicinity of the edge server, like autonomous vehicle fleets, eXtended reality users [4].

Efficient orchestration of applications at the network edge is inherently difficult because applications compete for limited computing and memory resources, and their performance is therefore tightly coupled [5]. As shown in [6], [7], co-located applications can significantly affect each other's inference times. At the same time their requirements vary widely. For example, RAN control functions such as traffic classification and beam prediction require inference latencies below 100 ms [8], while user workloads may tolerate substantially longer delays. The heterogeneous workloads and service requirements, together with the unintended interdependency, make admission control and resource allocation challenging.

In this setting, pricing has the potential to become a fundamental control mechanism [9]. By influencing which applications are served and how many instances are deployed, prices shape the actual workload at the edge server. Higher prices increase the revenue per application but reduce demand from price-sensitive users, while lower prices increase load and potential revenue, but also the level of congestion. This trade-off is particularly complex in multi-service edge systems, where pricing must implicitly prioritize applications that generate higher value relative to the resources they consume.

Designing optimal pricing schemes is inherently challenging because application performance depends on which other applications are admitted, and demand depends on prices that are set under uncertainty about these performance interactions. Moreover, edge platforms typically support a large and diverse application catalog, with users requesting different application types and varying numbers of instances. Pricing must therefore jointly manage congestion, performance interference, and demand composition in real time.

In this paper we address this challenging problem and make the following main contributions.

- We propose Contention-aware Pricing (*CAP*), a static pricing scheme that maximizes the revenue of the service provider when serving dynamically arriving ML workloads with diverse service quality requirements.
- We develop a low complexity Markovian model of the edge computing system that takes contention for resources into account, and we use the model for finding the optimal price.
- We propose to use a pairwise linear model of application contention that is aligned with the assumptions of the Markovian system model.
- We demonstrate with numerical results and with real deployments of ML models that pricing is an efficient tool for reaching an application mix that maximizes the revenue while meeting inference time requirements.

The rest of the paper is organized as follows. In Section II, we review the related work. In Section III, we introduce the system model and give the problem formulation. In Section IV we present a Markovian model for revenue optimization. In Section V we demonstrate revenue optimization under a given application contention model. We show numerical results in Section VI, and we conclude the paper in Section VII.

II. Related Work

Our work extends earlier results on edge orchestration, pricing, and workload characterization. Edge cloud technologies are reviewed in [10]. Most of the proposed solutions

ISBN 978-3-903176-82-9 © 2026 IFIP

orchestrate a fixed set of application instances. In [11] the authors aim to minimize end-to-end service latency. The authors in [12] propose a low-complexity algorithm for optimal placement of ML applications. Service placement with the help of workload prediction is considered in [13], [14]. Metaheuristics are proposed in [15], however, no hard SLAs are imposed. Service orchestration for ML workloads in the online setting is considered in [16]. These works assume fixed resources for the deployed instances, and do not consider the involved effects of colocation.

Pricing is proved to be an effective tool to control access to shared resources in multi-service environments [17], [18]. Static pricing is considered in [9], [19], showing that it closely matches the revenue of the dynamic one. Pricing where the customers willingness to pay depends on their waiting time is proposed in [20], [21] for single and multi server systems. Pricing that depends on the holding time and deterministic per-task pricing are compared in [22], and it is shown that in equilibrium the two solutions are economically identical. These works all build on the Markovian modeling of the system, with exponential holding times. Pricing schemes considering general holding times and identical resource requirements per class are proposed in [23]. It is shown that the revenue has a single local maximum, which can be found with standard gradient descent methods. Pricing based on experienced system load is designed in [24], [25]. The advantage of these solutions is that they do not need the detailed modeling of the system state and or to be aware of the application mix on the server. However, none of these pricing schemes considers the effects of contention for resources among the co-located applications. To do so, optimal pricing would require the careful profiling of the inference times affected by the resource contention [26].

Measurement based pairwise profiling of ML applications is presented in [27], [28], but the proposed methods do not scale to larger application catalogs. In [6], a binary classifier is proposed to predict whether a co-location profile would satisfy the SLA requirements. These works lack an explicit application inference time model to aid placement and orchestration. Two workload characterization solutions that can aid the pricing based control of coexisting workloads are [8], [7]. Authors in [8] propose a piecewise linear model to approximate the application inference time as a function of the number of co-located application instances. However, the model does not handle the diversity of the application types and consequently, leads to estimation errors. Instead, in [7] the pairwise profiling of application types is combined with an additive model to estimate the inference times of arbitrary application mixes. This simplicity of the inference time estimation supports well the model based pricing proposed in this paper.

III. System Model

We consider an edge server in the RAN with memory capacity C, dedicated for executing ML applications from an application catalog $\mathcal{A}$. The memory requirement of an instance of application $a \in \mathcal{A}$ is C_a.

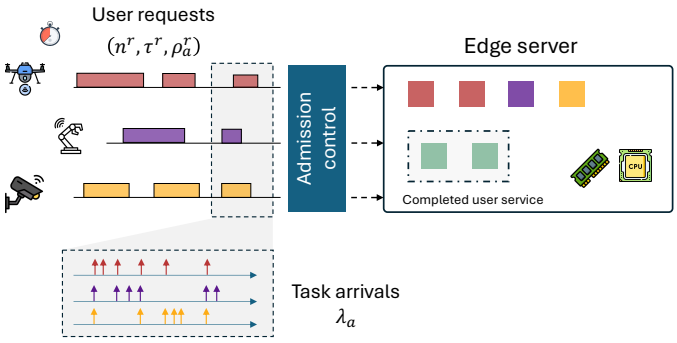


Fig. 1: System model. Users arrive to the service area of the edge server. If they accept the price and are accepted by the system, they stay for some time (dwell time), and during their stay they submit a stream of tasks to be served by the requested application instances.

A. User request and service model

Users of type a arrive to the service area of the edge server according to a Poisson process with intensity γ_a^+. A service request of a user of type a is a tuple $r = (a, n_a^r, \tau^r, \rho_a^r)$, where $a \in \mathcal{A}$ is the application type, $n_a^r \in \mathcal{N}_a^r$ is the required number of instances of application a, τ^r is the 95^{th} percentile of the application inference time (service level agreement (SLA)), and ρ_a^r is the price the user is willing to pay per application instance. We assume that the number of instances requested is drawn independently from a distribution f_a.

If a user request is accepted, the user remains in the service area, and submits a stream of tasks to be served by the installed application instances, as shown in Fig. 1. We consider that the dwell times of the users are exponentially distributed with mean $1/\mu_a$. We consider that the arrival rate of the stream of tasks is application type specific. This assumption is not restrictive, as one can define one application type for each task arrival intensity for a particular application.

B. Pricing model and admission control

For each accepted service request, the user is charged the price $u_a \in (0, u_a^+]$ per application instance. The operator sets the price based on the estimated values of the user arrival intensities for each application type γ_a^+, the mean dwell times $1/\mu_a$, and the distribution of the willingness of the users to pay, $\Pr\{u_a \leq \rho_a^r\}$. We denote the vector of prices by $\mathbf{u}$. The price is independent of the actual dwell time of the user, which is motivated by a preference to stating the price upfront, as the evolution of the load of the system and the dwell time are not yet known [22]. The price is also independent of the number of requests already in service, for transparency.

Users not willing to pay the price set by the operator leave immediately (a form of balking), which leads to an effective arrival rate

$$\gamma_a(u_a) = \gamma_a^+ \Pr\{u_a \leq \rho_a^r\}. \quad (1)$$

We consider that the effective arrival rate is continuous and strictly decreasing in u_a, which is reasonable for rational users.

If the user is willing to pay the announced price, i.e., $\rho_a^r \geq u_a$, the operator accepts the user request r for an application a if the inference time requirements and capacity limits would not be violated. Otherwise, the request is blocked.

We consider that the state of the edge server at each point of time is given by the vector of the number of instances deployed for each application type, that is, $\mathbf{n} = \{n_a\}_{a=1}^{A}$. The state determines the performance of the server, that is, the task inference times. A state is feasible if (i) the inference time constraint τ^r holds for each instance of application $a \in \mathcal{A}$, that is, $t_a < \tau^r$, and (ii) the capacity limit is satisfied, that is, $\sum_{a\in\mathcal{A}} n_a C_a \leq C$. We denote the set of feasible states by $\mathcal{S}$.

The operator is aware of the application requirements and consequently of the set $\mathcal{S}$ of feasible states, and blocks requests that would lead to an infeasible state. Let us denote the steady state probability that the operator blocks a user request for n_a^r instances of application a to avoid SLA or memory capacity violations by $P_{\text{block}}^{a,n_a^r}(\mathbf{u})$. Note that the blocking probability depends on the price vector $\mathbf{u}$, since the effective arrival rate to the edge server depends on the prices of all applications according to (1).

We can see that the pricing involves a tradeoff, as a higher price helps mitigate congestion and leads to fewer but high paying users, while a lower price attracts more lower paying users, potentially leading to higher resource congestion [9].

C. Problem Formulation

We consider that the operator wants to maximize its revenue from serving user requests. Since the deployed application instances contend for system resources, the operator shapes the arrival of requests via pricing, and performs admission control to satisfy the SLAs. For a given price vector $\mathbf{u}$ and corresponding blocking probability $P_{\text{block}}^{a,n_a^r}(\mathbf{u})$, the operator's revenue per time unit can be expressed as

$$J(\mathbf{u}) = \sum_{a\in\mathcal{A}} \sum_{n_a^r \in \mathcal{N}_a^r} \gamma_a(u_a) f_a(n_a^r)(1 - P_{\text{block}}^{a,n_a^r}(\mathbf{u})) n_a^r u_a, \quad (2)$$

and the objective of the operator is to maximize its revenue by finding the optimal prices,

$$\mathbf{u}^* = \arg\max_{\mathbf{u}} J(\mathbf{u}). \quad (3)$$

Note that the blocking probability $P_{\text{block}}^{a,n_a^r}(\mathbf{u})$ in (2) is not known a prior. In what follows, we propose methodologies for computing $P_{\text{block}}^{a,n_a^r}$ and for solving (3).

IV. Revenue optimization with the Markovian service model

Under the assumption of a Poisson user arrival process and exponential dwell times the evolution of the state of the edge server can be described by a continuous time Markov chain of dimension A, with the state given by $\mathbf{n}$, the vector of number of instances per application type. The resulting system is a finite state blocking system, and thus the system has a unique steady-state probability distribution. Let $\mathbf{N}$ be the vector of all the enumerated states $\mathbf{n}$, $\mathbf{Q}$ the the state transition intensity matrix, and $\mathbf{0}$ the zero vector. The state probabilities in steady-state satisfy the matrix equation $P(\mathbf{N})\mathbf{Q} = \mathbf{0}$, or the respective local and global balance equations.

A. State probabilities in steady-state

Note that since users request multiple instances simultaneously, instances are installed and then removed in batches, leading to a batch-arrival, batch-departure process. The resulting system is a Markovian multi-class blocking system [29, Chapter 18]. To calculate the steady state probabilities, we build on the following theorem on blocking systems with convex admission policy, that is, with a convex set of feasible states.

Theorem 1. *Let us assume that the set of feasible states $\mathcal{S}$ is convex. Let $P(\mathbf{n})$ be the probability of state $\mathbf{n}$. Then, the stationary distribution of the system has the product form*

$$P(\mathbf{n}) = \frac{1}{G} \prod_{a\in\mathcal{A}} P_a(n_a), \quad (4)$$

where $P_a(n_a)$ is the one-dimensional state probability and $G = \sum_{\mathbf{m}\in\mathcal{S}} \prod_{a\in\mathcal{A}} P_a(m_a)$ is the normalization constant obtained by summing over the feasible state space, ensuring that $\sum_{\mathbf{n}\in\mathcal{S}} P(\mathbf{n}) = 1$.

Proof. See [30], [29, Chapter 18]. □

B. Calculating the steady-state probabilities

To calculate the state probabilities according to (4), we need the one-dimensional state probability $P_a(n_a)$, where n_a is the number of instances of application a. To calculate the steady-state probabilities, we propose to follow the Kaufman-Roberts method, originally developed for bandwidth allocation in finite-capacity systems [31], [32], [33].

Recall that among users requesting application a, the distribution of the number n_a^r of requested application instances is $f_a(n_a^r)$, i.e., the arrival intensity of users requesting n_a^r instances of application a is

$$\gamma_{a,n_a^r} = \gamma_a f_a(n_a^r). \quad (5)$$

We introduce the state vector $\mathbf{m}_a = (m_{a,1}, m_{a,2}, \ldots, m_{a,n_a^r}, \ldots)$, where m_{a,n_a^r} denotes the number of users under service running n_a^r instances of application a. The total number of installed instances of application a is then given by

$$n_a = \sum_{n_a^r \in \mathcal{N}_a^r} n_a^r \, m_{a,n_a^r} = \mathbf{n}_a^{r\top} \mathbf{m}_a, \quad (6)$$

where $\boldsymbol{n}_a^r$ is a vector collecting the elements of $\mathcal{N}_a^r$, and n_a is upper bounded by the capacity constraint C_a.

Let $p_a(\mathbf{m}_a)$ denote the steady-state probability of state $\mathbf{m}_a$. Then the one-dimensional probability is

$$P_a(n_a) = \sum_{\boldsymbol{n}_a^{r\top}\mathbf{m}_a = n_a} p_a(\mathbf{m}_a). \quad (7)$$

The local balance equation associated with users requesting n_a^r instances of application a is given by

$$\gamma_{a,n_a^r}\, p_a(\mathbf{m}_a - \mathbf{e}_{n_a^r}) = m_{a,n_a^r}\mu_a\, p_a(\mathbf{m}_a), \tag{8}$$

where $\mathbf{e}_{n_a^r}$ denotes the n_a^r-th unit vector. Multiplying both sides of (8) with n_a^r and summing them up over $\mathcal{N}_a^r$, we get

$$\sum_{n_a^r \in \mathcal{N}_a^r} \gamma_{a,n_a^r} n_a^r P_a(n_a - n_a^r)\mathbb{I}_{\{n_a \geq n_a^r\}} = \mu_a \sum_{n_a^r \in \mathcal{N}_a^r} m_{a,n_a^r} n_a^r P_a(n_a) \tag{9}$$

Since $n_a = \sum_{n_a^r \in \mathcal{N}_a^r} m_{a,n_a^r} n_a^r$, $P_a(n_a)$ can be expressed as [31]

$$\frac{1}{n_a \mu_a} \sum_{n_a^r \in \mathcal{N}_a^r} \gamma_{a,n_a^r} n_a^r P_a(n_a - n_a^r)\mathbb{I}_{\{n_a \geq n_a^r\}} = P_a(n_a). \tag{10}$$

This result allows to calculate the state probabilities iteratively. We first assume $\tilde{P}_a(0) = 1$. Then compute the quantities $\tilde{P}_a(n_a)$ iteratively using (10) and normalize them to obtain the probabilities

$$P_a(n_a) = \frac{\tilde{P}_a(n_a)}{\sum_{i=0}^{C_a} \tilde{P}_a(i)} \tag{11}$$

For further details see [33, Section 11.1.2].

The steady state probabilities of the Markovian system are thus computed based on the boundaries of the feasible state space $\mathcal{S}$, the independence property in (4), and the Kaufman–Roberts method (10)-(11).

C. Calculating the state and blocking probabilities

The probability of blocking a request depends on both the application type and the number of requested instances. Let us denote the subset of feasible states $\mathcal{S}$ in which deploying an additional n_a^r instances of application a would cause the system to violate the feasibility constraints by $\mathcal{B}^{a,n_a^r}$. We can then obtain the blocking probability by summing the steady-state probabilities over the set of blocking states,

$$P_{\text{block}}^{a,n_a^r} = \sum_{\mathbf{n} \in \mathcal{B}^{a,n_a^r}} P(\mathbf{n}). \tag{12}$$

Remark 1. *The blocking probability is insensitive to the distribution of the dwell time, an insensitivity property that holds for many blocking systems. See for example [31].*

Remark 2. *The complexity of calculating the blocking probabilities is exponential in the size of the application catalog $\mathcal{A}$. For large application catalogs approximations with decreased complexity are available in [29].*

D. Revenue optimization

The optimal price vector $\mathbf{u}^*$ is the solution of (2)-(3), where both the blocking probability (12) and the effective rate $\gamma_a(u_a)$ depend on the price vector. Due to the iterative calculation of the state probabilities, we cannot prove the existence of a single local maximum of $J(\mathbf{u})$ in (3) in general. However, results are available for specific cases.

Remark 3. *For low system load with $P_{block}^{a,n_a^r} \approx 0$, the operator revenue $J(\mathbf{u})$ is a concave function, and the optimal price vector is given by $u_a = \frac{u_a^+}{2}$ for all a.*

Remark 4. *Prices $u_a = 0$ and $u_a = u_a^+$ for all a result in $J(\mathbf{u}) = 0$.*

Remark 5. *In the special case when $n_a^r = 1$ for all users, the admission is constrained only by the capacity C and $\gamma_a(u_a)$ is strictly decreasing in u_a, our problem is identical to that of [23], where the existence of a single local maximum of* (3) *is proved.*

Similar result cannot be derived for the general case we consider. Still, motivated by Remark 5 we will apply gradient ascent to find a suboptimal price for (3).

Finally, the following observation extends the applicability of the results to profit optimization.

Remark 6. *If the cost of serving one application instance of a request is linear in the dwell time $t_a^r \sim Exp(\mu_a)$, then the revenue optimization also leads to profit optimization. Consider cost per request $v_a^1 + v_a^2 t_a^r$. Then, the profit per time unit can be derived from* (2)*, by substituting the last term u_a with $u_a - v_a^1 - v_a^2 \frac{1}{\mu_a}$.*

V. Revenue optimization under the PERX inference time model

The revenue optimization requires the operator to know the feasible set of states $\mathbf{n} \in \mathcal{S}$, that is, the states where neither the infrence time limits not the memory limits are violated. Applications co-located on the same server, however, typically interfere with each other, depending on the server architecture and the memory and CPU access characteristics of the ML models. Therefore, the inference time of an application depends on all the other application instances running on the server.

Below we present PERX [7], an inference time model that is particularly suitable for the revenue optimization, as it fulfills the convexity requirements of Theorem 1. Then we show how to derive the feasible state space $\mathcal{S}$.

A. Application inference time model

PERX is based on the pair-wise characterization of the inference times, and an additive model for larger application sets, ensuring the scalability of the profiling, as the size of the application catalog increases. It is shown to be efficient for the orchestration of fixed set of application requests [7].

The inference time of application a is estimated using the composite model

$$t_a = \bar{t}_a + \sum_{a' \in \mathcal{A}} [t_{a,a'}(n_{a'}) - \bar{t}_a)], \tag{13}$$

where $\bar{t}_a$ denotes the 95^{th} percentile of the inference time of application a when deployed in isolation, and $t_{a,a'}(n_{a'})$ is the

95^{th} percentile of the inference of the same application when co-located with $n_{a'}$ instances of application a'. Indeed, if zero instances of application a' are deployed then $t_{a,a'}(0) = \bar{t}_a$.

The function $t_{a,a'}(n_{a'})$ is approximated based on measurement data, using a piecewise linear model

$$t_{a,a'}(n_{a'}) = \bar{t}_a + \mathbf{1}_{\{n_{a'} \leq \eta_{a'}\}} \left(\alpha^{(1)}_{a,a'} \, n_{a'} + \beta^{(1)}_{a,a'} \right) + \mathbf{1}_{\{n_{a'} > \eta_{a'}\}} \left(\alpha^{(2)}_{a,a'} \, n_{a'} + \beta^{(2)}_{a,a'} \right), \quad (14)$$

where $\eta_{a'}$ is the breakpoint, $\alpha^{(i)}_{a,a'}$ and $\beta^{(i)}_{a,a'}$ are slopes and intercepts. The curve is typically convex, with $\alpha^{(1)}_{a,a'} < \alpha^{(2)}_{a,a'}$.

B. The space of feasible states under PERX

As discussed, the state of the computing server is given by the vector of the number of instances deployed for each application type. A state is feasible if the inference time constraint τ_a holds for each $a \in \mathcal{A}$, that is, $t_a < \tau_a$, where t_a is given by (13), and $\sum_a n_a C_a \leq C$.

As a consequence of the piecewise linear approximation of the inference times, and the pairwise contention model, we can represent the boundary of the feasible states with the help of several hyperplanes, in the form of the matrix equation

$$\mathbf{B}\mathbf{n}^+ < \mathbf{0}. \quad (15)$$

As an example, for the specific case of two contending applications a and a'

$$\mathbf{n}^+ = [n_a, n_{a'}, 1]^T$$

$$\mathbf{B_a} = \begin{bmatrix} \alpha^{(1)}_{a,a} & \alpha^{(1)}_{a,a'} & -\alpha^{(1)}_{a,a} + \beta^{(1)}_{a,a} + \beta^{(1)}_{a,a'} - \kappa_a \\ \alpha^{(1)}_{a,a} & \alpha^{(2)}_{a,a'} & -\alpha^{(1)}_{a,a} + \beta^{(1)}_{a,a} + \beta^{(2)}_{a,a'} - \kappa_a \\ \alpha^{(2)}_{a,a} & \alpha^{(1)}_{a,a'} & -\alpha^{(2)}_{a,a} + \beta^{(2)}_{a,a} + \beta^{(1)}_{a,a'} - \kappa_a \\ \alpha^{(2)}_{a,a} & \alpha^{(2)}_{a,a'} & -\alpha^{(2)}_{a,a} + \beta^{(2)}_{a,a} + \beta^{(2)}_{a,a'} - \kappa_a \end{bmatrix}$$

$$\mathbf{B_{a'}} = \begin{bmatrix} \alpha^{(1)}_{a',a} & \alpha^{(1)}_{a',a'} & -\alpha^{(1)}_{a',a'} + \beta^{(1)}_{a',a} + \beta^{(1)}_{a',a'} - \kappa_{a'} \\ \alpha^{(1)}_{a',a} & \alpha^{(2)}_{a',a'} & -\alpha^{(2)}_{a',a'} + \beta^{(1)}_{a',a} + \beta^{(2)}_{a',a'} - \kappa_{a'} \\ \alpha^{(2)}_{a',a} & \alpha^{(1)}_{a',a'} & -\alpha^{(1)}_{a',a'} + \beta^{(2)}_{a',a} + \beta^{(1)}_{a',a'} - \kappa_{a'} \\ \alpha^{(2)}_{a',a} & \alpha^{(2)}_{a',a'} & -\alpha^{(2)}_{a',a'} + \beta^{(2)}_{a',a} + \beta^{(2)}_{a',a'} - \kappa_{a'} \end{bmatrix}$$

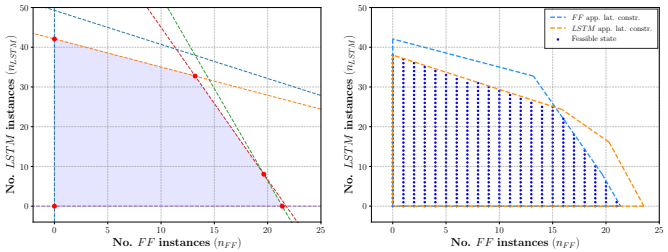
$$\mathbf{B_0} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\mathbf{B_c} = \begin{bmatrix} C_a & C_{a'} & -C \end{bmatrix}$$

and $\mathbf{B} = [\mathbf{B_a}; \mathbf{B_{a'}}; \mathbf{B_0}; \mathbf{B_c}]$, $\kappa_a = \tau_a + \bar{t}_a$, $\kappa_a = \tau_{a'} + \bar{t}_{a'}$.

The above matrix equation can be extended for $A > 2$. The length of vector $\mathbf{n}^+$ is $A + 1$, and the size of matrix $\mathbf{B}$ is $A \cdot 2^A \times (A+1)$. For an application catalog of reasonable size (15) provides a feasible way to evaluate the space of feasible states, but this result shows the importance of our modeling decision that an application type is defined by the requested ML model but not by the number of requested instances. Fig. 2 shows an example of how the final boundary is obtained for two contending applications.

Remark 7. *The boundaries for a single application form a convex set. The final set of feasible states is the intersection of these convex sets, and thus convex. Consequently, the assumption for the product form solution in Theorem 1 holds.*



(a) FF app. latency constraints (b) Feasible states

Fig. 2: Boundary defined by hyperplanes considering two contending applications (FF and $LSTM$).

Application	Memory Requirement (C_a) [MB]	Max. Payment (u_a^+) [$]	Latency Requirement [ms]
FF	700	10	300
LSTM	1,000	9	250
CNN	500	8	200
RF	400	7	50

TABLE I: Parameters of the application specific user requests.

VI. Experimental Results

A. Evaluation Methodology

We performed experiments to evaluate the effect of pricing on the operator revenue. We used a Dell R7515 PowerEdge server with $C = 32$ GB of RAM as the edge server. The application catalog consists of light-weight ML models $\mathcal{A} = \{FF, LSTM, CNN, RF\}$ as in [7], and we deployed the applications as Docker containers with memory requirement C_a shown in Table I.

For each user request, the task arrivals follow a Possion process, representative of Machine-type Communications (MTC) and Ultra Reliable and Low Latency Communications (URLLC) traffic at the edge [34]. The average task arrival intensity $\bar{\lambda}_a$ is uniformly distributed on $\{2, 4, \ldots, 8\}$ tasks/sec and $f_a(n_a^r)$ is uniform on $\{1, 2 \ldots, 10\}$, to account for low load and high load scenarios [35], [36].

The arrival intensities of the user requests γ_a^+ are uniformly distributed on $\{2.5, 5, 10, \ldots, 30\}$ reqs/hr. The dwell times for the user requests follow an exponential distribution with rate $\mu_a = 5$ reqs/hr as in [25]. We consider that the price preference of the users is uniformly distributed, that is, the effective arrival rate is

$$\gamma_a(u_a) = \gamma_a^+ \left(1 - \frac{u_a}{u_a^+}\right), \quad u_a \in (0, u_a^+]. \quad (16)$$

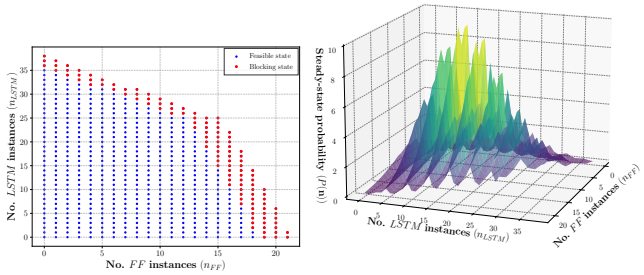
Table I shows the rest of the user request parameters.

We use the inference time results from [7] to determine the feasible region, and to compute the blocking probability in (12) for *CAP*. These can be calculated exactly for the considered catalog of four applications. To solve the unconstrained revenue optimization problem (3) we approximate the derivatives and the Hessian using the method of finite differences and use the SciPy [37] implementation of the L-BFGS algorithm with the parameters in Table II.

We consider two baseline pricing schemes for comparison. The first baseline, *Posted-price* is a dynamic pricing scheme

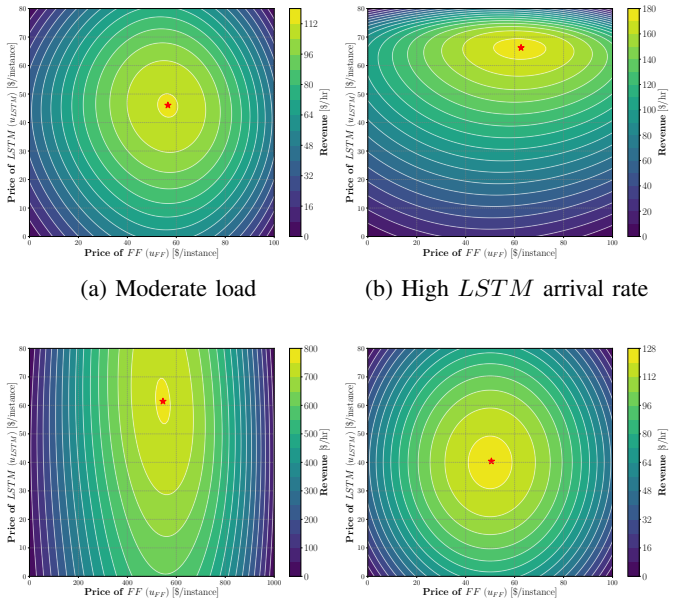
Parameter	Value
gtol	10^{-4}
tol	10^{-6}
Max. line search steps per iteration	20
Gradient approx. method	2-point method

TABLE II: Optimization parameters for *CAP*



(a) State space (b) Steady-state probabilities

Fig. 3: Feasible state space and steady state distribution of a mix of FF and $LSTM$ users.



(a) Moderate load (b) High $LSTM$ arrival rate

(c) High FF price upper bound (d) Short dwell times

Fig. 4: Optimal price of FF and $LSTM$ applications under different workloads.

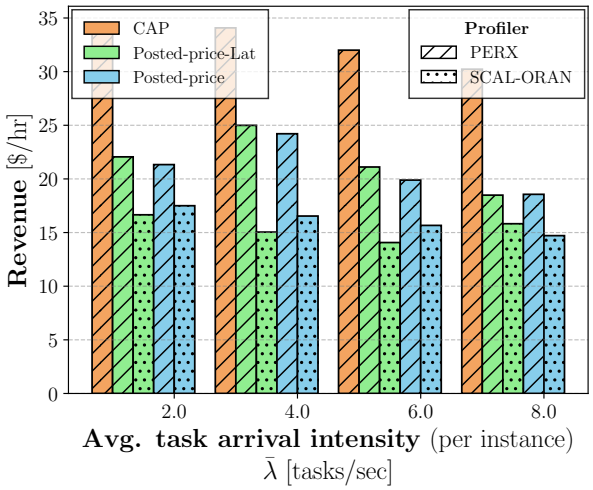
proposed in [24] that considers the memory utilization to adapt the application prices. The second baseline is *Posted-price-Lat*, which is a variant of *Posted-price* that computes the admissible number of instances for each application type such that the latency constraint is ensured, and adapts the pricing based on the number of already accepted instances. For both *Posted-price* and *Posted-price-Lat*, the SLA constraints are imposed prior to admitting new users to ensure feasibility. It is worth noting that these baselines do not consider the future load imposed by the user requests, that is, they use a myopic strategy to admit the user requests based on the instantaneous load. We use these baselines to demonstrate the significance of long term planning based on expected load as done in *CAP*. We implement both baselines using two different inference profilers, *PERX* [7], as described in Section V and *Scal-ORAN* [38] which uses conservative inference time estimates of co-located applications as shown in [7], [26]. We use *Scal-ORAN* to demonstrate the impact of profiler accuracy on the resource utilization and the operator revenue.

B. Revenue Characterization

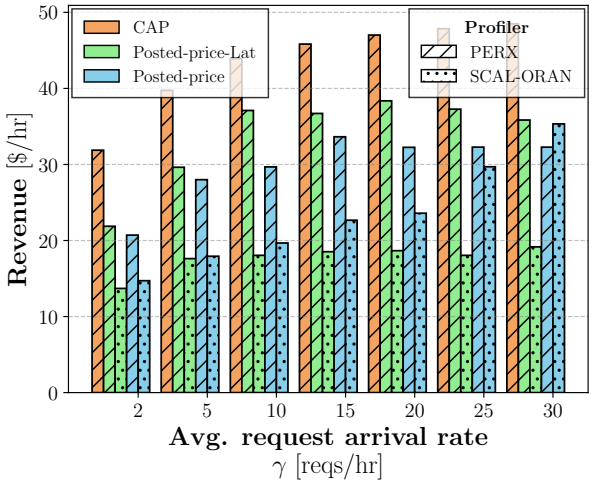
We first demonstrate the effect of pricing on the revenue in a simple scenario with two application types, LSTM and FF. We consider a server that can serve a maximum of 18 LSTM instances, or a maximum of 39 FF instances. Fig. 3a shows the feasible state space. As an example, the red dots indicate the set of blocking states for FF application requests requiring 3 instances, $\mathcal{B}^{FF,3}$. The region of blocking states lies along the boundary of the feasible region and depends on the particular application type and the requested number of instances.

Fig. 3b shows the steady-state probability distribution under the effective arrival and departure intensities $\gamma^{+}_{\text{FF}} = 0.4$, $\gamma^{+}_{\text{LSTM}} = 0.3$, $\mu_{\text{FF}} = 0.15$, $\mu_{\text{LSTM}} = 0.07$. The distribution of the requested number of instances is $f_{FF}(n^{r}_{\text{FF}}) = [0.05,\ 0.4,\ 0.5,\ 0.05]$ and $f_{LSTM}(n^{r}_{\text{LSTM}}) = [0.02,\ 0.03,\ 0.1,\ 0.8,\ 0.05]$. Note that the steady-state distribution is non-monotonic and highly varying due to the batch arrival and batch departure processes. Using Monte Carlo simulation to validate the numerical results, the Kullback–Leibler divergence is 0.289 nats between the analytical and empirical distributions, which shows the precision of the analytic method.

Fig. 4 shows the revenue as a function of the price, for four different server loads and user price preferences. It also marks the optimal price vector for each case. The distribution of the requested number of instances is as before. Fig. 4a is the baseline with moderate load, with parameters $\gamma^{+}_{FF} = 1.2$, $\gamma^{+}_{LSTM} = 0.6$, $u^{+}_{FF} = 100$, $u^{+}_{LSTM} = 80$, $\mu_{FF} = 0.15$, and $\mu_{LSTM} = 0.07$. The optimal prices are slightly above $\frac{u_{max}}{2}$, to keep the server utilized while avoiding high blocking probability. In accordance with Remark 3, the revenue at the borders of the acceptable price range is zero. Fig. 4b shows the revenue for the case when the arrival rate of LSTM requests is increased to $\gamma^{+}_{LSTM} = 4$, leading to an increase of the optimal LSTM price. The effect of user preferences is evaluated on Fig. 4c, where the maximum acceptable price for FF applications is increased to $u^{+}_{FF} = 1000$. The figure shows that this makes the revenue more sensitive to the FF price, compared to the LSTM price. Finally, Fig. 4d shows a scenario with low dwell times, $\mu_{FF} = 1.5$ and $\mu_{LSTM} = 0.7$, and consequently low load and nearly zero blocking probability. The optimal price is at the middle of the acceptable price range, according to Remark 3.

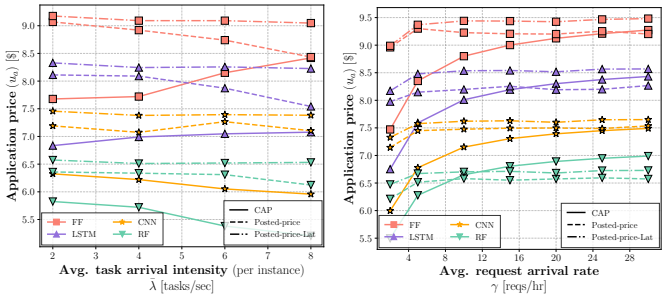


(a) Impact of task arrival rate



(b) Impact of request arrival rate

Fig. 5: Hourly operator revenue J vs. system load.



(a) Impact of task arrival rate (b) Impact of request arrival rate

Fig. 6: Operator pricing strategy $\mathbf{u}^*$ vs. system load (with *PERX* profiler).

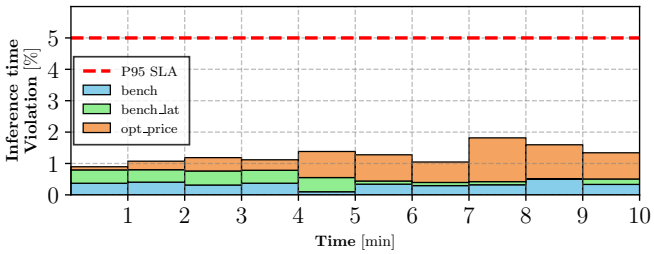


Fig. 7: Percentage of task latency violations over time. The dashed line (red) marks the SLA target.

C. Optimal pricing

Next we evaluate CAP compared to baselines, following the evaluation methodology given in Section VI-A, based on real deployment of a mix of ML workloads.

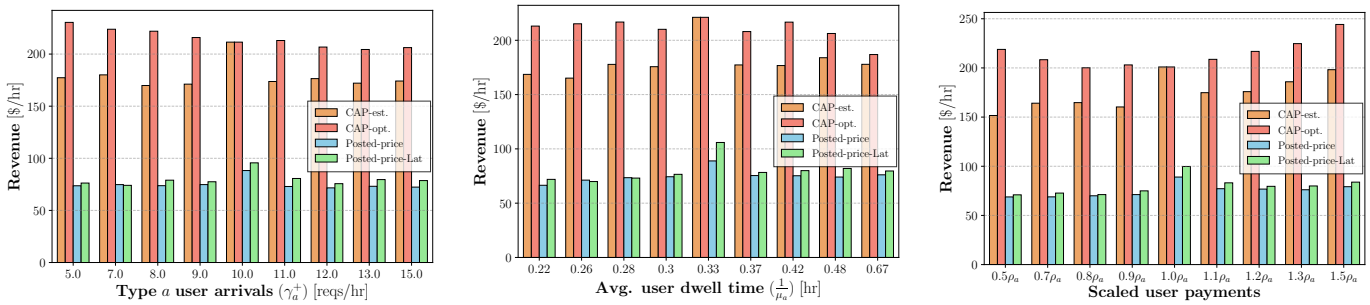
1) Operator Revenue: Fig. 5a shows the average hourly revenue of the operator as a function of the average task arrival intensity per application instance. *CAP* consistently results in a higher operator revenue by up to 40% compared to the baselines, since its pricing can keep free resources for the presumptive high-revenue users. The baseline *Posted-price* achieves lowest revenue as its dynamic pricing policy does not take blocking due to the SLA constraints into account. For both baselines, the conservative inference time model *Scal-ORAN* overestimates the effects of resource contention, consequently resulting it lower resource utilization and operator revenue.

Fig. 5b shows the average hourly revenue of the operator as a function of the arrival rate of the user requests. The initial increase in the arrival rate of the user requests has a positive effect on the operator revenue, as the operator is able to admit profitable user requests. However, the operator revenue saturates due to the limited system capacity. *CAP* consistently achieves up to 40% higher revenue compared to the baselines.

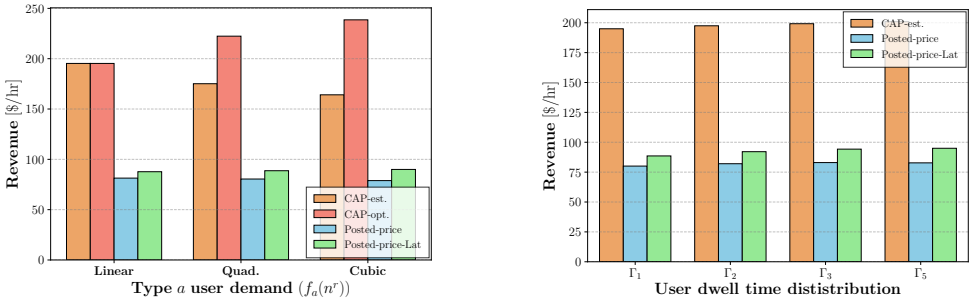
2) Pricing Strategy: Fig. 6a provides insight into the operation of the pricing policies using the *PERX* profiler, showing the price for each application type as the load changes. The figure shows that *CAP* increases the price of the FF and $LSTM$ applications, reflecting that FF and $LSTM$ are the most contending applications [7]. On the contrary, *Posted-price* adjusts the price of these applications based on their memory consumption. Fig. 6b shows that all pricing policies increase the application prices with the arrival intensity, serving as a form of demand shaping.

3) SLA Conformance: Fig. 7 shows the probability of SLA violations for the three pricing schemes. To evaluate the actual SLA violations, we had to scale down the user interarrival and dwell times. The rate of arrival and departure of users were set to $\gamma_a^+ = 10$ reqs/min and $\mu_a = 2$ reqs/min, respectively, and the task arrival rate was $\bar{\lambda}_a = 6.0$ tasks/sec. Fig. 7 shows the percentage of task latency violations within time slots of 1 minute over a period of 10 minutes. The results show that the user SLAs are satisfied under all pricing policies.

4) Sensitivity Analysis: We evaluate the robustness of *CAP* by evaluating the sensitivity of the operator's revenue to noisy estimates of the model parameters. In Fig. 8 we compare the revenue by simulating the user arrivals for the FF and $LSTM$ applications over a time horizon of $1,000$ hrs when the operator applies pricing optimized for the estimated model



(a) Impact of the user arrival rate. (b) Impact of the average dwell time. (c) Impact of the offered price.



(d) Impact of the resource demand. (e) Impact of the user dwell time distribution.

Fig. 8: Sensitivity analysis of the operator revenue to misestimation of the model parameters. Estimated parameters are $\gamma_a = 10$ reqs/hr and $1/\mu_a = 0.33$ hr.

parameters (*CAP-est*) instead of the optimal prices (*CAP-opt*). We assume that the operator estimates the user arrivals and departures to be $\gamma_a^+ = 10$ reqs/hr and $\mu_a = 3$ reqs/hr, respectively. We scale the continuous parameters (γ_a^+, μ_a, and ρ_a) by a scaling factor $s \in \{0.5, 0.7, \ldots, 1.5\}$. We vary the user demand by varying the order of decay of the demand curve of application instances ($f_a(n^r)$). While we keep the assumption of Poisson arrival process, which is often justified for large user population, we use a gamma distribution with a scale parameter $s \in \{1, 2, \ldots, 5\}$ to emulate different distributions for the user dwell times. In Figs 8a - 8d we observe that indeed, *CAP-est* yields a lower operator revenue than *CAP-opt* as the model parameters deviate from the operator estimates. That is, the operators should maintain accurate estimates of these parameters and tune the price accordingly. Despite the revenue loss, *CAP* consistently results in a higher revenue than the baselines. Finally, Fig. 8e shows the revenue for different dwell time distributions and demonstrates that the revenue earned by *CAP* is insensitive to the dwell time distribution of the users, consistent with Remark 1.

VII. Conclusion

We proposed *CAP*, a novel pricing scheme for delay sensitive machine learning workloads sharing computing resources at the edge. *CAP* is based on a Markovian model of the state of the server and relies on a scalable solution to derive the steady state probabilities. It incorporates inference time models that account for the effect of resource contention, are accurate and can be easily extended to cover larger application catalogs. Our results show that *CAP* outperforms existing pricing policies, as it determines the price taking into account both the set of the feasible system states as well as the diversity of the applications. Interesting directions of future work include the design of state aggregation for the Markov chain representation to improve scalability, the use of more accurate inference time predictors [26] and the application of model-based learning for pricing methods.

Acknowledgement

The work was partly funded by the Swedish Research Council through project DICE-6G (2024-06586) and by KTH Digital Futures.

References

[1] B. Brik, K. Boutiba, and A. Ksentini, "Deep Learning for B5G Open Radio Access Network: Evolution, Survey, Case Studies, and Challenges," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 228–250, 2022.

[2] A. Giannopoulos, S. Spantideas, N. Kapsalis, P. Gkonis, L. Sarakis, C. Capsalis, M. Vecchio, and P. Trakadas, "Supporting Intelligence in Disaggregated Open Radio Access Networks: Architectural Principles, AI/ML Workflow, and Use Cases," *IEEE Access*, vol. 10, pp. 39580–39595, 2022.

[3] Y. Cao, S.-Y. Lien, Y.-C. Liang, K.-C. Chen, and X. Shen, "User Access Control in Open Radio Access Networks: A Federated Deep Reinforcement Learning Approach," *IEEE Transactions on Wireless Communications*, vol. 21, no. 6, pp. 3721–3736, 2022.

[4] R. Singh, R. Sukapuram, and S. Chakraborty, "A Survey of Mobility-aware Multi-access Edge Computing: Challenges, Use cases and Future Directions," *Ad Hoc Networks*, vol. 140, p. 103044, 2023.

[5] F. Tütüncüoglu and G. Dán, "Joint resource management and pricing for task offloading in serverless edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 6, pp. 7438–7452, 2024.

[6] 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*, p. 32–47, 2023.

[7] 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*, pp. 1–8, 2025.

[8] 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*, pp. 891–900, 2024.

[9] I. Paschalidis and J. Tsitsiklis, "Congestion-dependent Pricing of Network Services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 171–184, 2000.

[10] 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*, pp. 379–386, 2015.

[11] 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*, pp. 41–50, 2020.

[12] 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*, pp. 270–279, 2022.

[13] 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.

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

[15] 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.

[16] 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*, pp. 1–10, 2023.

[17] F. Tütüncüoglu and G. Dán, " Optimal Service Caching and Pricing in Edge Computing: A Bayesian Gaussian Process Bandit Approach ," *IEEE Transactions on Mobile Computing*, vol. 23, pp. 705–718, Jan. 2024.

[18] F. Tütüncüoglu and G. Dán, "Raptor: Rate-adaptive pricing and optimal resource allocation in serverless edge computing," *IEEE Transactions on Networking*, vol. 34, pp. 4333–4344, 2026.

[19] I. Paschalidis and Y. Liu, "Pricing in Multiservice Loss Networks: Static Pricing, Asymptotic Optimality and Demand Substitution Effects," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, pp. 425–438, 2002.

[20] A. Krishnan K. S., C. Singh, S. T. Maguluri, and P. Parag, "Optimal Pricing in a Single Server System," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 8, no. 4, 2023.

[21] A. K. K.S., C. Singh, S. T. Maguluri, and P. Parag, "Optimal Pricing in Multi-server Systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 154, p. 102282, 2022.

[22] D. H. Lee, "Optimal Pricing Strategies and Customers' Equilibrium Behavior in an Unobservable M/M/1 Queueing System with Negative Customers and Repair," *Mathematical Problems in Engineering*, vol. 2017, no. 1, p. 8910819, 2017.

[23] A. N. Elmachtoub and J. Shi, "The Power of Static Pricing for Reusable Resources," *Proc. of ACM Conference on Economics and Computation*, 2023.

[24] Z. Zhang, Z. Li, and C. Wu, "Optimal Posted Prices for Online Cloud Resource Allocation," *Proc. of ACM POMACS*, vol. 1, June 2017.

[25] 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*, pp. 668–679, 2024.

[26] A. Javeed, V. Fodor, and G. Dán", "NeuRO: Inference-time Profiling and Orchestration of ML Applications at the Edge," in *Proc. of IEEE INFOCOM*, 2026.

[27] 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.

[28] 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*, p. 78–93, 2022.

[29] J. Roberts, U. Mocci, J. Virtamo, and A. Andersen, *Broadband Network Teletraffic: Final Report of Action COST 242*. Springer Verlag, 1996.

[30] J. Aein, "A multi-user-class, blocked-calls-cleared, demand access model," *IEEE Transactions on Communications*, vol. 26, no. 3, pp. 378–385, 1978.

[31] J. Kaufman, "Blocking in a shared resource environment," *IEEE Transactions on Communications*, vol. 29, no. 10, pp. 1474–1481, 2003.

[32] J. Roberts, "A service system with heterogeneous user requirements: Application to multi-service telecommunications systems," *Proc. of Performance of Data Communications Systems and their Applications*, pp. 423–431, 1981.

[33] L. Lakatos, L. Szeidl, and M. Telek, *Introduction to Queueing Systems with Telecommunication Applications*, vol. 388. Springer, 2013.

[34] 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, 2022.

[35] M. K. Kasi, S. Abu Ghazalah, R. N. Akram, and D. Sauveron, "Secure Mobile Edge Server Placement Using Multi-Agent Reinforcement Learning," *MDPI Electronics*, vol. 10, no. 17, 2021.

[36] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2021.

[37] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[38] 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, IEEE, 2024.