7-2024

Reinforcement learning based online request scheduling framework for workload-adaptive edge deep learning inference

Xinrui TAN

Hongjia LI

Xiaofei XIE
Singapore Management University, xfxie@smu.edu.sg

Lu GUO

Nirwan ANSARI

See next page for additional authors

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2024.3429571

Reinforcement Learning Based Online Request Scheduling Framework for Workload-Adaptive Edge Deep Learning Inference

Xinrui Tan, Hongjia Li, *Member, IEEE*, Xiaofei Xie, Lu Guo, Nirwan Ansari, *Fellow, IEEE*, Xueqing Huang, *Member, IEEE*, Liming Wang, Zhen Xu, and Yang Liu, *Senior Member, IEEE*

Abstract—The recent advances of deep learning in various mobile and Internet-of-Things applications, coupled with the emergence of edge computing, have led to a strong trend of performing deep learning inference on the edge servers located physically close to the end devices. This trend presents the challenge of how to meet the quality-of-service requirements of inference tasks at the resource-constrained network edge, especially under variable or even bursty inference workloads. Solutions to this challenge have not yet been reported in the related literature. In the present paper, we tackle this challenge by means of workload-adaptive inference request scheduling: in different workload states, via adaptive inference request scheduling policies, different models with diverse model sizes can play different roles to maintain high-quality inference services. To implement this idea, we propose a request scheduling framework for general-purpose edge inference serving systems. Theoretically, we prove that, in our framework, the problem of optimizing the inference request scheduling policies can be formulated as a Markov decision process (MDP). To tackle such an MDP, we use reinforcement learning and propose a policy optimization approach. Through extensive experiments, we empirically demonstrate the effectiveness of our framework in the challenging practical case where the MDP is partially observable.

Index Terms—Edge computing, deep learning inference serving systems, efficient deep learning inference, reinforcement learning.

1 Introduction

Edge computing has been highlighted as a promising solution to support the fast-growing latency-sensitive applications for the widespread mobile and Internet-of-Things (IoT) devices [1], which are expected to generate more than half of the world's data by 2025 [2]. Given the inherent latency and bandwidth limitations between end devices and the remote cloud, it is predicted that more than 75% of the edge-generated data will be analyzed locally at the network edge [3], [4]. Consequently, to support a myriad of edge applications, ranging from those with ultra-reliability and low-latency requirements (*e.g.*, cooperative autonomous driving [5]) to those intending to maintain massive connections of IoT devices (*e.g.*, intelligent manufacturing [6] and smart cities [7]), it is urgent to push the deep learning inference services, such as object detection and speech recognition, from the large-scale cloud data centers to the edge servers.

As compared with cloud-based inference serving systems, the limited edge resources and highly fluctuating workload bring the following great challenges to the provision of edge inference services with guaranteed quality-of-service (QoS).

- Since it is common for massive mobile/IoT devices to initiate simultaneous communications and generate bursty traffic [8], the arrival rate of inference requests can be time-varying and the workload of an edge inference serving system can be highly bursty (see Section 6.1 for our discussion on the workload attributes). For the cloud-based inference serving systems, how to process the time-varying workload has long been one of its main concerns. Both the state-of-the-art [9]–[12] and the state-of-the-practice [13], [14] heavily rely on the resource scaling mechanisms, where the computational resources are scaled in or out according to the arrival rate of inference requests and the QoS requirements, *e.g.*, the system response latency. However, these mechanisms implicitly assume that there are always sufficient cloud computing resources for fulfilling the scalabil-

- *X. Tan (E-mail: tanxinrui@ict.ac.cn) is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China, with the School of Cyber Security, University of the Chinese Academy of Sciences, Beijing 100049, China, and also with the CAS Key Laboratory of AI Safety, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.*
- *H. Li (corresponding author, E-mail: lihongjia@iie.ac.cn), L. Wang (E-mail: wangliming@iie.ac.cn) and Z. Xu (E-mail: xuzhen@iie.ac.cn) are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China.*
- *X. Xie (E-mail: xfxie@smu.edu.sg) is with the School of Computing and Information Systems, Singapore Management University, Singapore 188065.*
- *L. Guo (E-mail: guolu@travelsky.com.cn is with the Research and Development Center, TravelSky Technology Limited, Beijing 101318, China.*
- *N. Ansari (E-mail: nirwan.ansari@njit.edu) is with the Advanced Networking Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA.*
- *X. Huang (E-mail: xhuang25@nyit.edu) is with the Department of Computer Science, School of Engineering and Computing Sciences, New York Institute of Technology, New York, NY 10023, USA.*
- *Y. Liu (E-mail: yangliu@ntu.edu.sg) is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798.*

This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1005200, and in part by the Climbing Program of Institute of Information Engineering, Chinese Academy of Sciences under Grant E3Z0031. X. Tan acknowledges the financial support of this research by the China Scholarship Council.

ity. Unfortunately, such assumption often fails at the network edge, because the edge servers, dominated by various physical constraints, are well-known to be highly resource-constrained [15], [16]. As a result, the resource scaling mechanisms can hardly obtain sufficient resource elasticity at the network edge.

- Meanwhile, to facilitate efficient deep learning inference on resource-constrained devices, most existing studies focus on the design and training of lightweight deep learning models [17]–[21], but these models inevitably sacrifices accuracy for small model size and corresponding shorter inference latency. Taking the results of differentiable soft quantization [22] over the ResNet-18 model on the ILSVRC-2012 ImageNet dataset [23] for example, as compared to the full-precision model, the 4-bit, 3-bit and 2-bit quantization cause 0.34%, 1.24% and 5.19% accuracy degradation, respectively. Owing to the fundamental trade-off between accuracy and latency, the resource-constrained edge inference serving system cannot rely on a single lightweight model to consistently perform well in different states of workload: for example, when the workload is in the heavy state, to guarantee the inference service quality, the lightweight deep learning model is preferable to boost the overall service rate of the system; when the workload is in the light state, i.e., the request arrival rate is far lower than the overall service rate, the lightweight model can be "over-compressed" in the sense of sacrificing too much accuracy.

To address the above challenges, in contrast to the conventional inference serving systems, we consider to predeploy deep learning models with diverse sizes (*i.e.*, different trade-off between inference accuracy and compute latency) in virtual machines (VMs) or containers of the edge inference serving system; and then, to boost the system goodput and QoS perceived by users, we explore the diversity of model sizes, and adaptively schedule deep learning inference requests with the time-varying arrival rate to VMs or containers that host deep learning models with different sizes. Despite the straightforward logic of our idea, moving from idea to implementation is technically challenging. In the following, we will outline the specific technical challenges and our technical contributions, respectively.

1.1 Technical Challenges

To implement our idea of workload-adaptive inference request scheduling, we must address the following technical challenges.

- At the resource-constrained network edge, it is impossible to use complicated methods to decide how to schedule each individual arriving request, since such methods will incur significant computational overhead when bursts occur. Therefore, the underlying method to schedule each request must be of very low complexity to avoid the scheduling procedure itself becoming a bottleneck.
- In real practice, there is no prior knowledge about the workload. In addition, the service-time distributions of deployed model instances may be hard to be known exactly in advance. These raise the problem of how to achieve good inference service quality without the knowledge of the workload and models.
- To adaptively schedule inference requests, it would be ideal to always know exactly the current workload state. In practice, however, we cannot directly observe whether the workload is in a bursty or normal state. Particularly for certain bursty workloads, the current workload state is not directly observable but sometimes also difficult to estimate; and the future workload states are inherently unpredictable.

1.2 Our Technical Contributions

In view of the above challenges, we propose a reinforcement learning based online request scheduling framework to empower edge inference serving systems with the workload-adaptiveness. The technical contributions are detailed below.

- To ensure the scheduling efficiency and effectiveness, a probabilistic scheduling framework is proposed to schedule the inference requests with the minimum computational effort, where the probability distribution that schedules each inference request to be served by different models is periodically adjusted to match the workload-state variation.
- To deal with the sequential decision-making problem on how to periodically find the optimal probability distributions for request scheduling, we first theoretically prove that such a problem can be formulated as a Markov decision process (MDP), where the arrival of inference requests is assumed to follow a Markovian arrival process (MAP), which has been extensively used in the literature to model bursty traffic [24], and the underlying MAP phases representing the workload states are observable. Then, it makes perfect sense to use reinforcement learning to tackle the MDP. Specifically, based on the proximal policy optimization (PPO) algorithm [25], we propose a policy optimization approach to learn good request scheduling policies that decide the probability distributions for request scheduling. Note that such policies are learned from the past experience of inference request scheduling, so our policy optimization approach does not require any prior knowledge of the workload, nor of the models.
- To handle the partially observable Markov decision process (POMDP) induced by the unobservability of workload states in practice, we show that, by simply observing the states of the deployed model instances, memoryless policies can still be learned to achieve good performance. The intuition behind this is that when the workload state changes, the model instance states will change accordingly. Thus, the model instance states can convey the workload-state information that is useful for the workload-adaptive inference request scheduling.

We employ a highly bursty workload to empirically evaluate our request scheduling framework on an object

recognition task, where the validation set of ILSVRC-2012 ImageNet dataset is used as the test data. The experimental results not only demonstrate that the proposed framework can effectively handle the latency requirements while achieving good quality of inference services, but also verify that our main idea of workload-adaptive inference request scheduling can be realized: when the workload is in its burst state, our framework can significantly reduce the violation ratio of the latency requirement as compared to a baseline that only uses heavyweight models; while when the workload is in its normal state, our framework can significantly improve the goodput as compared to a baseline that only uses lightweight models. Also, we find that our framework performs well when the arrival rate of inference requests changes periodically, showing the generality of our framework to general variable workloads. Moreover, our experiments show that our framework can be easily extended by integrating adaptive batching to further boost performance. Finally, while we believe that our framework is generally applicable to any efficiency metrics that can be measured from the device, in the interest of space, we focus on achieving good accuracy-latency trade-offs in this paper.

1.3 Organization

The rest of this paper is organized as follows: the related work is reviewed in Section 2; the system model and the request scheduling framework are presented in Section 3; the policy optimization approach based request scheduling policies are described in Section 4, followed by experiments in Section 5; the discussion is presented in Section 6; finally, our work is concluded in Section 7.

2 Related Work

In this section, we outline related work in two main areas: workload-adaptive inference serving systems and efficient deep learning inference at the edge.

2.1 Workload-Adaptive Inference Serving Systems

Even since the rise of deep learning, deep learning inference serving systems have long been developed and practically applied to support various deep learning applications [26], where for one given inference task, multiple instances of the same deep learning model trained for this task are deployed in the cloud and then serve the corresponding incoming inference requests. Recently, with the success of deep learning in many latency-sensitive and massive-connectivity tasks, most studies have focused on the design of inference serving systems capable of meeting latency requirements under variable workloads, leading to several state-of-the-art systems from the literature [9]–[12], [27] and some state-of-the-practice systems that are already in use [13], [14].

For instance, SageMaker [13] employs a feedback control method to dynamically scale the computational resources, where some customized event-condition-action rules for resource scaling can be followed by monitoring the system operating conditions and adjusting the resources accordingly; MArk [9], instead, employs a predictive based method that predicts the upcoming workload states and proactively scales up or down the resources; Clockwork [10] is designed to simultaneously serve the inference requests of many different inference tasks, where the resources can be effectively shared between different tasks to meet the latency goal of each task respectively; More recently, Romero *et al.* [28] also proposed a system that can reduce the cost of resource scaling through coordinated use of multiple models, but their system still mainly uses resource scaling to handle the variability of workloads, and implicitly requires great resource elasticity of the computing infrastructure. These systems have devoted considerable effort towards exploiting the resource elasticity of the underlying computing infrastructure. However, unlike the public cloud that can immediately offer thousands of graphic processing units (GPUs) and other high-performance hardware accelerators to MArk to handle bursts, the network edge is constrained in terms of almost all types of hardware. Thus, although these systems have been shown to work well in the cloud, they cannot effectively handle bursty workloads at the network edge.

In general, our request scheduling framework differs from these existing systems in three aspects: first, although these existing systems also use other mechanisms, such as adaptive batching, almost all of them rely on resource scaling to play a key role in handling variable workloads, while our framework is designed for the network edge where resource scaling is ineffective; second, unlike many state-of-the-art systems which require the future workload states to be predictable [9], [12], [27], our framework does not have such a requirement and thus can perform well when the workload-state changes are unpredictable; third, unlike many state-of-the-art systems [9], [10], our framework does not assume that the service-time distributions of model instances are all deterministic, as this assumption can be far from reality, for example, when recurrent models are used to process inference requests with different lengths, or when early-exit models are used to perform adaptive inference [21]. Moreover, in contrast to some systems that control the accuracy-latency trade-off by heuristics or empirical rules, our framework, by applying reinforcement learning, can automatically learn to achieve a good trade-off between accuracy and latency. Note that although MArk also employs a countermeasure for the failures of predictive-based method against unpredictable workloads, this countermeasure is still a simple and relatively trivial feedback control method.

2.2 Efficient Deep Learning Inference at the Edge

Among recent studies that can facilitate efficient deep learning inference at the network edge, we find that the majority still concentrates on model compression to find good lightweight models performing well on general and specific resource-constrained devices, where the conventional model compression techniques include quantization [17], pruning [18], knowledge distillation [19], and lightweight architecture design and search [20]. Besides these studies on model compression, there also exist some preliminary yet promising lines of research. For instance, Guo *et al.* [29] suggested caching some inference requests as well as their corresponding inference results at the network edge, so that when an arriving inference request is similar to one of the cached requests, the previous inference result can be reused

to avoid repetitive computation; Li *et al.* [30] proposed an end-edge collaborative inference method, which adaptively partitions deep learning models between end devices and edge servers to accelerate deep learning inference; Marco *et al.* [31] presented a method that can accelerate deep learning inference by adaptively selecting appropriate lightweight models for inference requests with low inference difficulty.

Despite these research efforts, such studies are rarely conducted from the perspective of online inference serving, as they generally ignore the variability of workloads; we remark that there is a lack of studies investigating how deep learning inference can be efficiently performed under variable workloads at the network edge. A notable exception is the study by Jin *et al.* [34], where the idea of adaptively using different models for different workload states is also implicitly introduced; however, this study simplifies the edge inference serving system too much and imposes some strong assumptions so as to formulate a nonlinear integer programming problem; also, this study cannot provide any QoS guarantees. More recently, some studies such as Jellyfish [35] and ModelIO [36] have been proposed to improve end-to-end QoS of the edge deep learning inference services, where they also, in some sense, adopt the idea of workload-adaptive inference request scheduling. However, their simple and highly task-specific system models restrict them from provisioning general-purpose deep learning inference services. By saying that our framework is "general-purpose", we mean that our framework is general in its applicability: it can not only handle general workloads, but also theoretically incorporate deep learning models with general service time distributions.

While in the literature there are many task-scheduling approaches designed for the network edge, and some of them are even reinforcement learning based [37]–[41], we find that the differences between the system models of our framework and these task scheduling approaches are so large that these task scheduling approaches are neither applicable to the practical edge inference serving systems nor straightforwardly comparable to our framework. One of the key differences is that these task scheduling approaches require the service times of tasks to be all deterministic and exactly predictable, while our framework allows the service times of the inference requests served by a model instance to follow any general and unknown statistical distribution. Also, these task scheduling approaches only consider simple workloads such as Poisson processes, where the arrival rate is constant over time. This reflects the fact that these task scheduling approaches do not aim to adaptively schedule the tasks to match the time-varying workload states. In a nutshell, compared with these task-scheduling approaches, our technical novelty is mainly three-fold:

- To efficiently schedule the inference requests when bursts occur, we employ the probabilistic routing method, which allows our framework to choose the model instances for hundreds of inference requests by executing a relatively computationally intensive decision-making process only once;
- To provide general-purpose deep learning inference services, instead of the common practice of heuristically hand-crafting decision processes, we establish theoretical results showing how to derive the true Markov decision process for our system model specifically designed for the general-purpose but resource-constrained deep learning inference systems, noting that these theoretical results are the key to enable our framework to handle general workloads and incorporate deep learning models with general service time distributions.
- To automatically learn good request scheduling policies with almost no prior knowledge of the workloads and deep learning models, we derive the tractable stochastic policy for our specific action set, note that without this tractable policy, the stable PPO method cannot be used.

3 System Model and Problem Formulation

In this section, we first detail how our request scheduling framework can enable an edge inference serving system to adaptively serve inference requests, and then formulate the problem of finding the optimal request scheduling policy. For the sake of convenience, we summarize the frequently used notations in Table 1 (see Appendix A in the supplementary material for a full table of notations). Also, we provide a detailed illustration of our framework in Appendix B.

3.1 System Model

3.1.1 System Overview

With reference to the example shown in Fig. 1, we consider an edge inference serving system serving an inference task to meet a typical tail latency requirement, which requires that a certain percentage of the inference requests should be completed within a required time. In this system, we have a set $N = \{1, 2, \ldots, n\}$ of deep learning model instances trained for the same inference task, where these model instances can use different models with different model sizes. Following the common practice, we let these model instances run within their own VMs or containers, so that their deployment can be quite flexible at the network edge. In the inference serving system, our inference request scheduling framework introduces a virtual network function, dubbed *scheduler*, to receive the inference requests from end devices and schedule them on the n deep learning model instances. Note that for simplicity, the scheduler only determines a "intra-system path", as illustrated in Fig. 1, for each arriving request to deliver the request to one of the model instances; once a request reaches a model instance, it will be queued for processing. However, as will be shown later, our framework can be extended to allow the scheduler to support more complex control. In the following, we reasonably assume that the latency of delivering an inference request from the scheduler to a model instance is negligible, as the scheduler is physically positioned close to the model instances, and the bandwidth between the scheduler and each model instance can be sufficiently large. Also, we assume that the communication between the scheduler and the edge devices is handled by the network infrastructure, meaning that the latency incurred when an end device sends a request to the scheduler is not a consideration of the system.

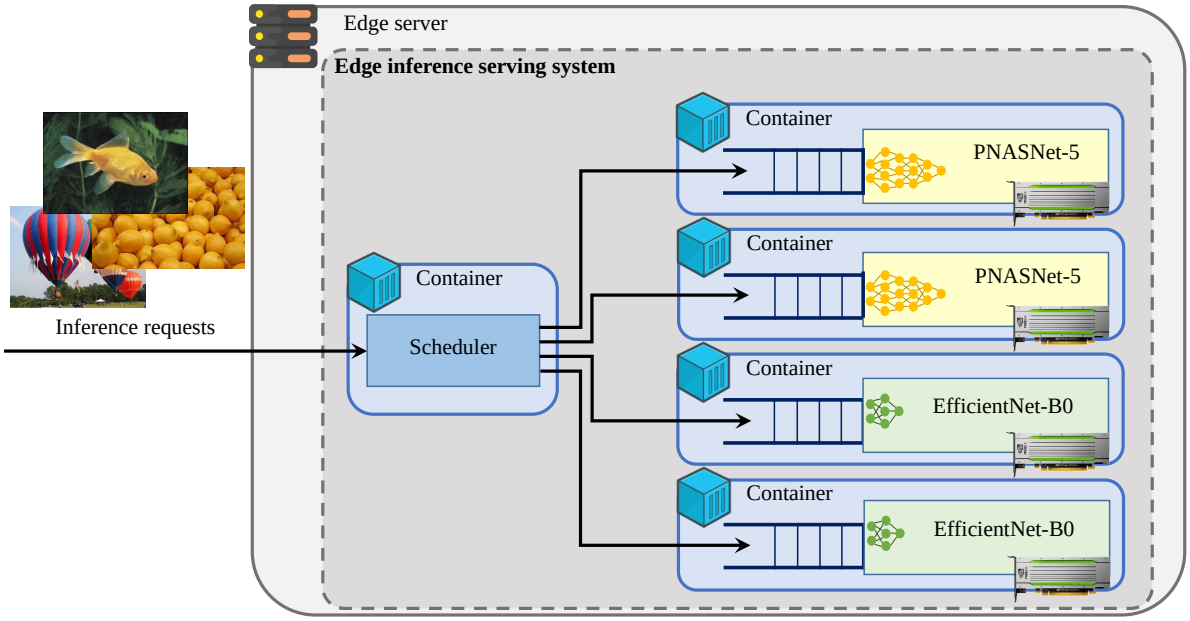


Fig. 1: An example of our edge inference serving system for an object recognition task (see Appendix B in the supplementary material for a more detailed illustration), where two model instances of a lightweight EfficientNet-B0 model [32] and two model instances of a heavyweight PNASNet-5-Large model [33] are deployed. According to our main idea, the two EfficientNet model instances should serve most of the arriving inference requests when the workload is in a bursty state, and the two PNASNet model instances should serve most of the arriving inference requests when the workload is in a normal state.

3.1.2 System Workload

To model the arrival of the inference requests, we assume that the requests arrive at the scheduler in a one-by-one fashion. We further assume that the request arrival stream aggregated from all the end devices follows a Markovian arrival process (MAP) introduced by Lucantoni [24], which has not only been extensively used to model the bursty arrival processes commonly arise in communication applications, but recently also been used to model various traffic streams at the network edge [42]–[44]. The MAP generalizes the Poisson process by allowing the arrival rate to vary according to a continuous-time Markov chain (CTMC) with a finite state space $M = \{1, 2, \ldots, m\}$, where m is an unknown positive integer; it is characterized with an unknown matrix representation (D_0, D_1), where D_0 is an $m \times m$ matrix with negative diagonal elements and non-negative off-diagonal elements that govern the state transitions without arrivals, and D_1 is a non-negative $m \times m$ matrix whose elements represent state transitions with an arrival. As usual, the states of the underlying CTMC are referred to as the phases of the MAP. Note that for the workloads following a MAP, the workload states are exactly the MAP phases. Moreover, the underlying CTMC is assumed to be irreducible and positive recurrent. Since, in general, the underlying CTMC is not a predictable process, the MAP phase changes can be very unpredictable. More details concerning MAP can be found in [45].

3.1.3 Inference Request Serving

For each deep learning model instance, the service process can be naturally modeled as a finite-capacity queue, where it is assumed that inference is carried out with a batch size of 1, which is customary when performing deep learning inference with real-time considerations [46]. To be specific, the inference requests are assumed to be served one-by-one at each model instance on a first-come-first-served basis, where each model instance's service times (*i.e.*, the inference latency times) are generally distributed and mutually independent, which allows the service times of different model instances follow different unknown distributions that depend jointly on the model architectures and the hardware resources available to the model instances. Moreover, for notational simplicity, without loss of generality, we assume that all model instances have the same buffer capacity c, which is the maximum number of inference requests that can be stored in the buffers of their VMs or containers (including the one that is in progress, if any); we remark that our request scheduling framework in this paper can be easily extended to the case where the model instances have different queue capacities. When the buffer of a model instance is full, any new inference request scheduled to the model instance will be rejected and considered lost. Now, we can define the state of each model instance as $(l, \omega, \varphi_1, \varphi_2, \ldots, \varphi_l)$ where l denotes the total number of inference requests in service and waiting for service in the buffer of the model instance, ω denotes the elapsed service time of the request in service, and φ_i denotes the age of the i^{th} oldest requests for $i \in \{1, 2, \ldots, l\}$. Note that: the age of an inference request is the time elapsed since the request's arrival; for notational simplicity, we shall let $\omega = 0$ when the model instance is idle. Therefore, we further define a set U which contains all the possible model instance states as elements, namely,

$$U := \bigcup_{i=0}^{c} V_i. \tag{1}$$

TABLE 1: Notations frequently used in Section 3

Symbol	Description
n	The number of deep learning model instances
N	The set of model instances in the system
m	The number of MAP phases
M	The set of MAP phases
c	The buffer capacity of model instances
l	The number of inference requests in a model instance
ω	The elapsed service time of a request being served
φ_i	The age of the i^{th} oldest requests in a model instance
U	The set containing all the possible model instance states
τ	The duration of each time slot
$s^{(t)}$	The system state at the beginning of time slot t
S	The set of all possible system states
$o^{(t)}$	The observation of the system state $s^{(t)}$
O	The set of all possible observations
$h^{(t)}$	The MAP phase at the beginning of time slot t
$u_i^{(t)}$	The state of model instance i when time slot t begins
$a^{(t)}$	The action for request scheduling during time slot t
Δ	The probability simplex over the set of model instances
$r^{(t)}$	The reward of the system for time slot t
$\upsilon^{(t)}$	The total number of requests delayed during time slot t
$b^{(t)}$	The total number of requests rejected during time slot t
g_i	The expected accuracy of model instance i

Here, $V_0 = \{0\}$ and, any element in V_i for $i \in \{1, 2, \dots, c\}$ is of the form $(i, \omega, \varphi_1, \varphi_2, \dots, \varphi_i)$, where $\omega, \varphi_1, \varphi_2, \dots, \varphi_i$ are non-negative real numbers satisfying $\omega \leq \varphi_1 \geq \varphi_2 \geq \cdots \geq \varphi_i$.

3.1.4 Inference Request Scheduling

For the scheduler responsible for request scheduling, a classic probabilistic routing method [47]–[52] is adopted to schedule inference requests among deep learning models. This simple but efficient method requires a probability distribution over the model instance set N to be specified, so that each model instance is associated with a probability. Upon an arrival, the scheduler randomly selects one of the model instances according to the specified probability distribution, and then schedules the request to the selected model. In other words, the scheduler randomly splits the arrival process into n sub-processes, one for each model instance. Owing to its simplicity, this probabilistic routing method can be implemented very efficiently, allowing each request to be scheduled immediately upon its arrival. More importantly, even under a high arrival rate of requests, the scheduler can still work well without falling into a congestion collapse. Note that, by leveraging *e.g.,* Alias method [53], the probabilistic routing can be scalable to the number of model instances (n), even if n is very large.

3.2 Problem Formulation

Recall that the edge inference serving system needs to satisfy a tail latency requirement, such as requiring 99% of the inference requests to be responded within 1 second. To achieve our goal of adaptively scheduling requests so as to meet this tail latency requirement and achieve high inference service quality, our purpose is to periodically adjust the probability distribution for request scheduling, which results in a sequential decision-making problem that can be cast in the reinforcement learning framework. Specifically, we assume that time is slotted into discrete-time intervals of equal duration τ, such that the time interval $[t\tau, (t+1)\tau)$ is referred to as time slot t, where $t \in \{0, 1, 2, \dots\}$. In the reinforcement learning framework, for any given time slot t, at the beginning of the time slot, the probability distribution over N is adjusted as follows:

1) The scheduler (in reinforcement learning terminology referred to as an agent) perceives an observation $o^{(t)}$ of the current system state $s^{(t)} = (h^{(t)}, u_1^{(t)}, u_2^{(t)}, \dots, u_n^{(t)}) \in S$, where $h^{(t)} \in M$ is the current phase of the MAP; for $i \in N$, $u_i^{(t)} \in U$ is the current state of model instance i, including the current number and ages of the inference requests waiting for service from model instance i and being served by model instance i, together with the elapsed service time of the inference request currently being served by model instance i; $S = M \times U^n$ is the set of all system states, noting that U is the model-instance-state set formally defined in Section 3.1.3.
2) On the basis of $o^{(t)}$, the scheduler samples an action $a^{(t)} = (y_1^{(t)}, y_2^{(t)}, \dots, y_n^{(t)}) \in \Delta$ from a policy $\pi(\cdot|o^{(t)})$ to specify the probability distribution used to schedule the inference requests arriving during this time slot, where Δ is the $(n-1)$-dimensional probability simplex; for $i \in N$, $y_i^{(t)}$ gives the probability that an inference request is scheduled to model instance i during this time slot; π is a probability density function over the action set Δ, conditioned on the observation.

Then, during the time slot t, the scheduler uses the action $a^{(t)}$ to schedule the arrival requests. At the end of the time slot t, the system updates itself as follows:

1) In part as a consequence of $a^{(t)}$, the scheduler receives a stochastic real-valued reward $r^{(t)} \sim R(\cdot|s^{(t)}, a^{(t)})$, where $R(\cdot|\cdot,\cdot) : \mathbb{R} \times S \times \Delta \to [0, \infty]$ is the conditional density function characterizing the reward dynamics so that $R(r^{(t)}|s^{(t)}, a^{(t)})$ gives the probability density for the reward of a time slot being $r^{(t)}$ when the system state at the beginning of the time slot is $s^{(t)}$ and the action used in the time slot is $a^{(t)}$.
2) The system transitions into a new system state $s^{(t+1)} \sim P(\cdot|s^{(t)}, a^{(t)})$, where $P(\cdot|\cdot,\cdot) : S \times S \times \Delta \to [0, \infty]$ is the conditional density function characterizing the system transition dynamics so that $P(s^{(t+1)}|s^{(t)}, a^{(t)})$ gives the probability density for the system state at the end of a time slot being $s^{(t+1)}$ when the system state at the beginning of the time slot is $s^{(t)}$ and the action used in the time slot is $a^{(t)}$.

To attain our goal of achieving good inference service quality while fulfilling the tail latency requirement, we define the multi-objective reward $r^{(t)}$ as the following, for all $t \in \{0, 1, 2, \dots\}$,

$$r^{(t)} = -\upsilon^{(t)} - \alpha b^{(t)} + \beta \textstyle\sum_{i=1}^{n} g_i \nu_i^{(t)}, \tag{2}$$

where $\upsilon^{(t)} \geq 0$ denotes the number of delayed inference requests which have completed service during time slot t with overall system response latencies (*i.e.,* the sojourn

times) longer than the required time of the tail latency requirement; $b^{(t)} \in \{0, 1, 2, \dots\}$ denotes the number of inference requests rejected in time slot t; for $i \in N$, $g_i \in [0, 1]$ denotes the expected accuracy of model instance i, and $\nu_i^{(t)} \in \{0, 1, 2, \dots\}$ denotes the number of requests that have completed service without violating the latency requirement at model instance i during time slot t; $\alpha > 0$ and $\beta > 0$ are two predetermined weights that trades off the different terms to meet the tail latency requirement. By maximizing the expected cumulative reward over time, we propose to increase the average $\sum_{i=1}^{n} g_i \nu_i^{(t)}$ over time to improve the system goodput, and meanwhile decrease the average $\upsilon^{(t)} + b^{(t)}$ over time to ensure the tail latency requirement. Note that since $g_1, g_2, \dots, g_n$ are data-dependent, they are usually intractable in practice. However, a dataset independent of the models' training data can serve as the validation data to evaluate the models unbiasedly, and then for all $i \in N$, the validation accuracy of model instance i on the validation data can be used as an approximation of g_i. Eventually, we arrive at the following result (see Appendix C in the supplementary material for the formal statement), which justifies the use of reinforcement learning approaches to tackle the above sequential decision-making problem.

Proposition 1 (Informal statement)**.** *If the system states are fully observable, that is, $\forall t \in \{0, 1, 2, \dots\}, o^{(t)} = s^{(t)}$, then the above sequential decision-making problem is a Markov decision process (MDP), where the future of the process depends on the history of the process only through the current state of the system.*

Notice that, to ensure the decision process is Markovian, the scheduler has to monitor the model instance states, as well as perceive the phases of the underlying MAP. Unfortunately, although the model instance states are easily accessible, it is practically impossible for the scheduler to access the MAP phases. As a result, in practice, we have to deal with a partially observable Markov decision process (POMDP). In this POMDP, the underlying MAP phases that directly expose the workload-state information are unobservable, and so the scheduler needs to decide how to schedule the requests by observing only the model instance states. Nevertheless, these model instance states can still reveal some indirect workload-state information, because the transitions of MAP phases have direct impacts on the model instance states, especially on the queue lengths (*i.e.*, the numbers of requests waiting for service). Hence, hereafter, we let $o^{(t)} = (u_1^{(t)}, u_2^{(t)}, \dots, u_n^{(t)})$ for all $t \in \{0, 1, 2, \dots\}$, and let $O = U^n$ denote the set of observations. Remark that this observation definition is quite general for many arrival processes that are non-Markovian or more general than the MAP, implying that the MAP assumption does not restrict the generality of our framework. Furthermore, the two conditional density functions characterizing the system transition dynamics and the reward dynamics can be alternatively denoted as $\tilde{P}(\cdot, \cdot|\cdot, \cdot, \cdot) : M \times O \times M \times O \times \Delta \to [0, \infty]$ and $\tilde{R}(\cdot|\cdot, \cdot, \cdot) : \mathbb{R} \times M \times O \times \Delta \to [0, \infty]$, respectively.

Despite the partial observability of the practical decision process, we still consider a parameterized memoryless policy $\pi_\theta(\cdot|\cdot) : \Delta \times O \to [0, \infty]$, with parameters $\theta \in \Theta$, where Θ is an appropriate compact Euclidean parameter set. Then, given a discount factor $\gamma \in (0, 1)$ and the density $\mathcal{I}(\cdot, \cdot) : M \times O \to \mathbb{R}$ of an initial system state distribution, our objective is to find the optimal policy parameters that maximize the time-discounted expected cumulative reward, *i.e.*,

$$\max_{\theta \in \Theta} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r^{(t)} \Big| (h^{(0)}, o^{(0)}) \sim \mathcal{I}(\cdot, \cdot), a^{(t)} \sim \pi_\theta(\cdot|o^{(t)}), (h^{(t+1)}, o^{(t+1)}) \sim \tilde{P}(\cdot, \cdot|h^{(t)}, o^{(t)}, a^{(t)}), r^{(t)} \sim \tilde{R}(\cdot|h^{(t)}, o^{(t)}, a^{(t)})\Big]. \quad (3)$$

Note that the system can have quite complex dynamics: in general, neither the system transition dynamics nor the reward dynamics can be expressed in closed-form. This fact, together with the uncountability of both the state and action sets, suggests that classic dynamic programming methods are inapplicable to our problem, even if the MAP phases are observable, the MAP parameters are known, and the service-time distributions are also all known. Moreover, since U is a "combinatorial" set where each element is structured, it is hard to construct a system simulation model that can output valid system states, thus preventing us from using model-based methods. Therefore, we need a model-free policy gradient approach to tackle our problem.

4 Policy Optimization Approach

In this section, we present the model-free policy gradient approach that we use to find good request scheduling policies. At first glance, to tackle our policy optimization problem, it seems that deep deterministic policy gradient (DDPG) algorithm [54] is a convenient choice among the most widely used policy gradient algorithms. This is because DDPG is based on the deterministic policy gradient theorem [55], which allows the elimination of the action density in the policy gradient computation. As will be shown later, it is not so straightforward to derive the closed-form of policies over our action set Δ. Despite the convenience, we empirically find that DDPG fails to learn good policies for the scheduling-policy-optimization-problem instances in our experiments. Therefore, we instead use proximal policy optimization (PPO) algorithm [25], which is empirically much more stable for our problem.

Next, we detail our policy optimization approach based on PPO. Since both the standard state-action value function and state value function are useless when memoryless policies are considered for our POMDP, we start by defining the observation-action value function $Q_\theta : O \times \Delta \to \mathbb{R}$ of a policy π_θ, for all $o \in O$ and $a \in \Delta$, as follows:

$$Q_\theta(o, a) \coloneqq \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r^{(t)} \Big| o^{(0)} = o, a^{(0)} = a, h^{(0)} \sim d_\theta(\cdot|o), r^{(t)} \sim \tilde{R}(\cdot|h^{(t)}, o^{(t)}, a^{(t)}), a^{(t)} \sim \pi_\theta(\cdot|o^{(t)}), (h^{(t+1)}, o^{(t+1)}) \sim \tilde{P}(\cdot, \cdot|h^{(t)}, o^{(t)}, a^{(t)})\Big], \quad (4)$$

where $d_\theta(\cdot|\cdot) : M \times O \to [0, 1]$ is the conditional visitation distribution of the hidden MAP phases given the observations; for completeness, the definition of d_θ is provided in Appendix D in the supplementary material. Also, we define

the observation value function $V_\theta : O \to \mathbb{R}$ of a policy π_θ by the following, for all $o \in O$,

$$V_\theta(o) \coloneqq \mathbb{E}\Big[\sum_{t=0}^{\infty}\gamma^t r^{(t)} \,\Big|\, o^{(0)}=o, h^{(0)}\sim d_\theta(\cdot|o), a^{(t)}\sim\pi_\theta(\cdot|o^{(t)}), r^{(t)}\sim\tilde{R}(\cdot|h^{(t)},o^{(t)},a^{(t)}), (h^{(t+1)},o^{(t+1)})\sim\tilde{P}(\cdot,\cdot|h^{(t)},o^{(t)},a^{(t)})\Big], \tag{5}$$

Then, the advantage function $A_\theta : O \times \Delta \to \mathbb{R}$ of a policy π_θ is defined for all $o \in O$ and $a \in \Delta$ as follows:

$$A_\theta(o,a) \coloneqq Q_\theta(o,a) - V_\theta(o). \tag{6}$$

By using PPO, given an old policy $\pi_{\tilde{\theta}}$ with its corresponding advantage function $A_{\tilde{\theta}}$, and a clipping threshold $\epsilon \in [0,1)$, we aim to find a new policy that maximizes the following surrogate objective function with respect to θ,

$$\mathbb{E}\Big[\min\Big(\frac{\pi_\theta(a|o)}{\pi_{\tilde{\theta}}(a|o)}A_{\tilde{\theta}}(o,a), C_\epsilon\Big(\frac{\pi_\theta(a|o)}{\pi_{\tilde{\theta}}(a|o)}\Big)A_{\tilde{\theta}}(o,a)\Big)\Big|o\sim\rho_{\tilde{\theta}}(\cdot), a\sim\pi_{\tilde{\theta}}(\cdot|o)\Big], \tag{7}$$

where the term $\pi_\theta(a|o)/\pi_{\tilde{\theta}}(a|o)$ is well-known as the importance weight [56], $\rho_{\tilde{\theta}}(\cdot)$ is the observation visitation density of $\pi_{\tilde{\theta}}$ (see Appendix D in the supplementary material for the definition), and C_ϵ is a cut function defined for all $x \in \mathbb{R}$ as: if $x < 1-\epsilon$, then $C_\epsilon(x) = 1-\epsilon$; else if $x > 1+\epsilon$, then $C_\epsilon(x) = 1+\epsilon$; otherwise, $C_\epsilon(x) = x$. The main rationale behind this surrogate objective is to approximately impose a soft constraint on the distance between the old and new policies in the parameter space, such that stable policy improvement can be achieved by avoiding destructively large policy updates. If, for every sampled observation-action pair, the importance weight of a policy is always above $1+\epsilon$ when the advantage function is positive, or below $1-\epsilon$ when the advantage function is negative, then the gradient of this surrogate objective is zero, *i.e.*, the policy can no longer be updated further based on the gradient.

Observe that, in order to use first-order methods to optimize the surrogate objective, π_θ must be differentiable with respect to θ. To obtain such a π_θ over the probability simplex Δ, a very intuitive idea for action sampling is to first draw a random sample from an n-variate Gaussian with parameters given by a θ-parameterized neural network, and then use the softmax function to map the random sample from $\mathbb{R}^n$ to Δ. Specifically, suppose that there is a neural network defining a deterministic function $f(\cdot,\cdot) : O \times \Theta \to \mathbb{R}^n$, given an observation o and policy parameters θ, we first feed o into the neural network parameterized by θ to compute $f(o,\theta)$; then, we sample an action $a = (y_1, y_2, \ldots, y_n)$ by generating each y_i as follows:

$$y_i = \frac{\exp\big(f_i(o,\theta)+\xi_i\big)}{\sum_{k=1}^{n}\exp\big(f_k(o,\theta)+\xi_k\big)}, \tag{8}$$

where, for all $i \in N$, ξ_i is an independent sample drawn from $\mathcal{N}(0,\sigma^2)$ with $\sigma > 0$, and $f_i(o,\theta)$ denotes the i^{th} element in $f(o,\theta)$. Although the softmax is a non-invertible function, in the following result, we derive the closed-form expression of the policy π_θ corresponding to our action sampling routine, so that the derivatives of this tractable π_θ with respect to θ can be computed.

Proposition 2. *Let $(z_1, z_2, \ldots, z_n) \sim \mathcal{N}(f(o,\theta), \Sigma)$ where $o \in O$, $\theta \in \Theta$, $\sigma > 0$, and Σ is the $n \times n$ diagonal matrix with all diagonal elements equal to σ^2. If a random variable $a = (y_1, y_2, \ldots, y_n)$ is given by $y_i = \exp(z_i)/\sum_{k=1}^{n}\exp(z_k)$ for all $i \in N$, then by letting $q_i = -f_i(o,\theta) + \log y_i$ for all $i \in N$, the probability density function of a is given by*

$$\pi_\theta(a|o) = \frac{\exp\big((2n\sigma^2)^{-1}\big(\sum_{k=1}^{n}q_k\big)^2 - (2\sigma^2)^{-1}\sum_{i=1}^{n}q_i^2\big)}{\sqrt{n}(\sigma\sqrt{2\pi})^{n-1}\prod_{j=1}^{n}y_j}. \tag{9}$$

Proof. See Appendix E in the supplementary material. □

With the aid of this tractable π_θ, we present a practical policy optimization algorithm in Algorithm 1. As is shown, the scheduler in Algorithm 1 keeps performing the update of its policy periodically. For each policy update period, the latest updated policy $\pi_{\tilde{\theta}}$, also known as the actor, is used to schedule inference requests for T time slots (T is a sufficiently large positive integer) to collect a trajectory of $\pi_{\tilde{\theta}}$. This trajectory is formed by the observation $o^{(t)}$, the action $a^{(t)}$ and the reward $r^{(t)}$ of time slot t in the period for all $t \in \{0,1,\ldots,T-1\}$, as well as the final observation $o^{(T)}$. Then, as an approximation to the intractable surrogate objective, we train the actor with the following objective:

$$\max_{\theta\in\Theta}\sum_{t=0}^{T-1}\min\Big(\frac{\pi_\theta(a^{(t)}|o^{(t)})}{\pi_{\tilde{\theta}}(a^{(t)}|o^{(t)})}\tilde{A}^{(t)}, C_\epsilon\Big(\frac{\pi_\theta(a^{(t)}|o^{(t)})}{\pi_{\tilde{\theta}}(a^{(t)}|o^{(t)})}\Big)\tilde{A}^{(t)}\Big), \tag{10}$$

where $\tilde{A}^{(t)}$ is an estimation of the unknown advantage $A_{\tilde{\theta}}(o^{(t)},a^{(t)})$ for all $t \in \{0,1,\ldots,T-1\}$. By using the generalized advantage estimation (GAE) method [57], we compute the advantage estimation $\tilde{A}^{(t)}$ for all $t \in \{0,1,\ldots,T-1\}$ as follows:

$$\tilde{A}^{(t)} = \sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\big(r^{(k)} + \tilde{V}_{\phi^*}(o^{(k+1)}) - \tilde{V}_{\phi^*}(o^{(k)})\big), \tag{11}$$

where $\lambda \in [0,1]$ is a predefined parameter that adjusts the estimation's bias-variance trade-off; $\tilde{V}_{\phi^*}$ is a parametric function (with parameters ϕ^*) learned to appropriate the observation value function V_θ, which is not known in practice. This approximation of the observation value function is known as the critic, and is also tuned periodically with a mean-squared error loss.

Note that because we use memoryless policies for our POMDP, either the observation-action value function defined in Eq. (4) or the observation value function defined in Eq. (5) generally cannot be written in an exact recursive form. As a result, compared with the situation if our approach is applied directly to the underlying MDP of our POMDP, the advantage estimators may be more biased, and the approximate problem given in Eq. (10) may have more deviation from the exact problem given in Eq. (3). Still, our empirical investigation shows that memoryless policies can perform quite well. Alternatively, the estimation method introduced by Shmyrin [58] allows us to achieve the optimal estimation of the MAP phases, where this method requires knowing the arrival times of the inference requests and the parameters (D_0 and D_1) of the MAP. However, even though the MAP-phase estimation is optimal, it can still be very inaccurate for certain MAPs. Moreover, in

Algorithm 1 PPO-based request scheduling policy optimization

Input: initial policy parameters $\tilde{\theta}$, initial approximate observation value function parameters ϕ^*

loop
  for $t = 0, 1, \ldots, T-1$ **do**
    Run the edge inference serving system for a time slot by following $\pi_{\tilde{\theta}}$ to collect $o^{(t)}, a^{(t)}, r^{(t)}$ and $o^{(t+1)}$
  end for
  Compute the advantage estimators $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots, \tilde{A}^{(T-1)}$ according to Eq. (11)
  Train the actor to find a good parameter setting $\theta \in \Theta$ to the optimization problem in Eq. (10)
  Train the critic for a good parameter setting ϕ to minimize $\frac{1}{T}\sum_{t=0}^{T-1}\big(\tilde{V}_{\phi}(o^{(t)}) - \tilde{V}_{\phi^*}(o^{(t)}) - \tilde{A}^{(t)}\big)^2$
  $\tilde{\theta} \leftarrow \theta$ and $\phi^* \leftarrow \phi$
end loop

practice, we can not know the parameters of the MAP. Therefore, to estimate the MAP phases, we have to first estimate the number of MAP phases and then estimate the MAP parameters. Since the MAP parameter estimation is inexact, the empirical MAP-phase estimation can even be far from optimal. Thus, we argue that it is meaningless to perform MAP-phase estimation in our framework. For general POMDPs, although recurrent neural networks (RNNs) are widely used to provide memories of past observations to estimate the current state [59]–[61], and have shown to be effective for some challenging POMDPs [62], we empirically find that RNNs are ineffective for the scheduling-policy-optimization-problem instances in our experiments. From a theoretical perspective, they are also unlikely to be effective. This is due to the fact that since the MAP phase is the only observable component of a system state, by using an RNN to estimate the underlying system states, the RNN is essentially served as an estimator of the MAP phases, and the implicit MAP-phase estimation performed by the RNN can never be better than the optimal MAP-phase estimation. Moreover, the information available to the RNN for MAP-phase estimation is very incomplete because the observations do not include the exact arrival times of the inference requests; and the MAP-phase information is also not explicitly provided to guide the RNN training. These two factors naturally degenerate the accuracy of MAP-phase estimation.

5 Experiments

In this section, we conduct extensive experiments to evaluate our request scheduling framework. In the following, we first describe our experimental setup, including the experimental settings, the baseline approaches for comparison, and the evaluation metrics to measure the performance of inference request serving. Then, we provide the experimental results.

5.1 Experimental Setup

5.1.1 Simulation Scenario

We evaluate the effectiveness of our framework using a realistic simulation scenario like Fig. 1, where an inference task of object recognition is processed by an edge server equipped with an Intel Xeon Platinum 8259CL processor and four Nvidia Tesla T4 GPUs. Just as illustrated in Fig. 1, for the target inference task, we employ a pretrained PNASNet-5-Large model [33] and a pretrained EfficientNet-B0 model [32] from the Pytorch image models library [63] to, respectively, serve as a heavyweight model and a lightweight model, where on the validation set of ILSVRC-2012 ImageNet dataset [23] the former model achieves 82.78% top-1 classification accuracy, while the accuracy of the latter model is 77.69%. For both the heavyweight and lightweight models, we deploy two model instances with buffer capacity $c = 15$ on the edge server, and assign one GPU to each model instance, so that there are a total of four model instances and the computational resources available for each model instance are almost the same. For the configuration of schedulers, we set the duration τ of each time slot to 1 second. Note that, we use the validation set of the ImageNet dataset as the test data for performance evaluation, meaning that during evaluation the inference requests are drawn randomly and independently from the validation set of the ImageNet dataset; while for the training of the schedulers, we use the "matched frequency" version of the ImageNet-V2 dataset [64] as the validation data to provide the validation accuracy of each model instance in the reward computation of Eq. (2). Following the standard preprocessing, we resize the ImageNet images to 256×256 pixels, and center crop the resized images to 224×224 pixels.

5.1.2 Workloads

In the main experiments, we adopt a two-phase MAP with the following parameter matrices D_0 and D_1 as the workload,

$$D_0 = \begin{bmatrix} -180.036 & 0.036 \\ 0.0002 & -10.002 \end{bmatrix}, \quad D_1 = \begin{bmatrix} 180 & 0 \\ 0 & 10 \end{bmatrix}, \quad (12)$$

where this MAP is also known as a switched Poisson process (SPP). Roughly speaking, this arrival process is alternately in its two phases during exponentially distributed times. If the MAP is in the first phase, then independently of the phase process, inference requests arrive according to a Poisson process with intensity 180; otherwise, inference requests arrive according to another Poisson process with intensity 10, independently of the phase process as well. Hereafter, we refer to the first and second states of the MAP, respectively, as the bursty and normal states of the workload. We remark that this MAP has several characteristics. First, since the underlying CTMC of the MAP is not a predictable process, the workload-state switching between the bursty and normal states is unpredictable. Second, this MAP can be highly bursty as the arrival rate in the bursty

state is 18 times higher than that in the normal state, making it challenging for an edge inference serving system with constrained and constant resources to maintain good QoS in both states. Third, the mean holding time of the normal state is 18 times longer than that of the bursty state, implying that, after a sufficiently long time, the inference requests will be a fifty-fifty mixture of the arrivals in the two states, such, that the two states can have equal importance in evaluation. Fourth since approximately 95% of a long trajectory is in the normal state, the training of schedulers may suffer from this "state imbalance". More specifically, due to the "state imbalance", the rewards yielded in the bursty state are very sparse, maybe leading to a sparse reward problem. Since in the literature, it seems difficult for DDPG to tackle sparse reward problems [65]–[67], the sparsity of rewards may be a reason for the failures of DDPG. Throughout our experiments, the initial phase of this MAP is sampled according to the stationary distribution. Hereafter, this MAP is also referred to as our main workload. Moreover, as detailed later, we also use a non-Markovian arrival process to evaluate the generality of our framework.

5.1.3 Latency Requirement

With regards to the above experimental settings and the unpredictability of our main workload, we adopt a moderate 98^{th} percentile (P98) latency of 300 milliseconds (ms) as the benchmarking latency requirement, noting that similar P98 latency requirements have also been adopted in previous work [9]. This latency requirement dictates that 98% of the inference requests must be completed within 300 ms.

5.1.4 Algorithm Settings

Here we give the detailed algorithm settings, including the neural network architecture settings, the parameter settings for Algorithm 1, and other parameter settings of the MDP. Specifically, in our main experiments, we find that by setting the two weights α and β to 1, the schedulers can be trained to meet the latency requirement. For more implementation details of our policy optimization approach, we let the policy network of actor and the value network of critic use the almost same architecture of a two-hidden-layer fully-connected tanh network with 90 hidden units in the first hidden layer and 60 hidden units in the second hidden layer, where there is no parameter sharing between these two networks; we set the discount factor as $\gamma = 0.99$, the clipping threshold $\epsilon = 0.2$ and the GAE parameter $\lambda = 0.95$. Note that before feeding the observations into the neural networks, we convert the queue lengths to binary vectors through one-hot encoding; we normalize the elapsed service times between 0 and 1; we apply zero-padding to convert the ages into vectors of length 16, and then normalize them into the range $[0, 1]$, respectively.

5.1.5 Training Settings

For training details, we develop a simulator to facilitate faster training, where the model instances are assumed to have deterministic service times; we set $T = 5 \times 10^4$, $\sigma = 0.698$ and train a single actor for 250 million time slots, meaning that the workload traces used for training are instances of our main workload with duration of 250 million time slots; we optimize the training objectives in each policy update period using the Adam optimizer [68] for 5 epochs with an initial learning rate of 2.5×10^{-4} and a mini-batch size of 10^4.

5.1.6 Evalution Settings

For evaluation settings, we consider long runs over 0.6 million time slots, meaning that the workload traces used for evaluation are instances of our main workload with the duration of 0.6 million time slots. Note that all the instances used for training and evaluation are generated independently. In the evaluation, if a policy optimized by our approach is evaluated, the actor will act deterministically with $\sigma = 0$. Since, in practice, all of our model instances exhibit negligible service-time variability, we find that the additional assumptions on service times do not introduce discrepancies between simulation and reality.

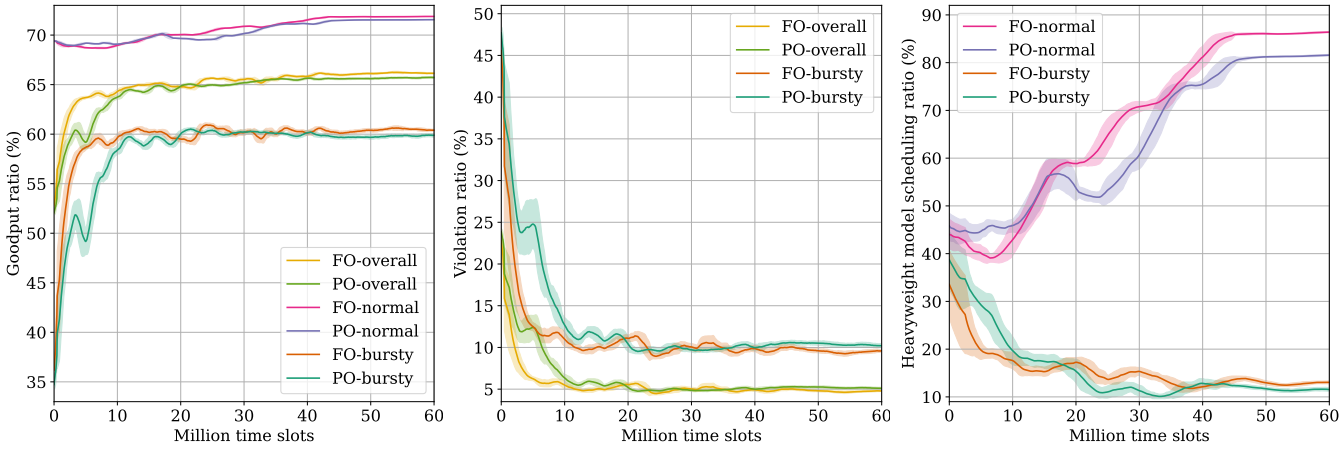
5.1.7 Baseline Approaches

Based on our framework, we also develop a variant with adaptive batching, as will be detailed later. Under the practical partial observability, we respectively refer to our framework and its adaptive-batching variant as RL-PO and RL-PO-B for short. We compare them against the following baseline approaches.

- The conventional approach with lightweight models (LW for short) only uses the model instances of the lightweight model. That is, the four model instances in the edge server are all the model instances of EfficientNet, and the inference requests are scheduled with fairness among the model instances.
- The conventional approach with heavyweight models (HW for short) only uses the model instance of the heavyweight model, where the inference requests are served by four model instances of PNASNet. Following the previous work [10], we also develop a variant (referred to as HW-E) that can proactively reject the buffered inference requests that will inevitably miss the latency deadline.
- Our proposed approach with the full observability (RL-FO for short) and its adaptive-batching variant (RL-FO-B for short) assume that the states of the workload can be observed, such that the policies for the underlying MDP can be directly learned.
- The single queue approach (SQ for short) serves the inference requests using a standard single queue model with the same hybrid model instance setting as our framework. To improve the utilization of heavyweight models, if there are two idle model instances, respectively, of PNASNet and EfficientNet, then the upcoming arrival will be served by the model instance of PNASNet.
- The request scheduling approach of InFaaS [28] also uses the same hybrid model instance setting as our framework. We refer to this baseline approach as InFaaS for short. Briefly, for this baseline approach, time is also divided into fixed-duration time slots; the goal is to achieve "load balancing" between PNASNet and EfficientNet while avoiding any model instance being continuously overloaded. Note that,

	Approach	Overall			Bursty state					Normal state				
		Ratio (%)	Rate (RPS)	VR (%)	Ratio (%)	VR (%)	Reject (%)	Delay (%)	HWSR (%)	Ratio (%)	VR (%)	Reject (%)	Delay (%)	HWSR (%)
MAP (SPP)	LW	77.69	14.72	0	77.69	0	0	0	—	77.69	0	0	0	—
	HW	41.43	7.851	49.95	0.089	99.89	48.11	51.78	—	82.78	0.007	0	0.007	—
	HW-E	69.05	13.08	16.61	55.31	33.19	33.19	0	—	82.78	0	0	0	—
	RL-PO	79.24	15.01	1.447	75.85	2.891	1.499	1.393	12.78	82.62	0.003	0	0.003	96.25
	RL-FO	80.14	15.18	0.366	77.64	0.732	0.293	0.439	10.66	82.62	0	0	0	97.25
	RL-PO-B	79.49	15.05	0.884	76.56	1.594	1.499	1.393	3.664	82.43	0.174	0	0.174	93.14
	RL-FO-B	80.18	15.19	0.079	77.82	0.161	0.064	0.096	5.099	82.54	0	0	0	93.69
	SQ	79.59	15.08	0	78.86	0	0	0	22.85	80.37	0	0	0	52.65
	SA	80.39	15.23	0.499	78.01	0.998	0.102	0.895	19.99	82.78	0	0	0	99.99
	InFaaS	78.22	14.82	1.695	76.12	3.389	0.001	3.388	24.22	80.35	0	0	0	52.18
Non-Markovian	LW	77.69	14.72	0	77.69	0	0	0	—	77.69	0	0	0	—
	HW	41.45	7.854	49.93	0.125	99.85	48.14	51.71	—	82.77	0.011	0	0.011	—
	HW-E	69.05	13.08	16.59	55.32	33.17	33.17	0	—	82.78	0	0	0	—
	RL-PO	79.23	15.01	1.553	75.75	3.103	2.246	0.857	12.11	82.72	0.003	0	0.003	98.78
	RL-FO	80.27	15.21	0.329	77.76	0.651	0.563	0.008	11.63	82.77	0.008	0	0.008	99.93

TABLE 2: Results for the P98 latency requirement of 300 ms. "Ratio", "VR", and "HWSR" stand for the goodput, violation, and heavyweight model scheduling ratios, respectively. "Rate" stands for the goodput rate, which measures the density of correctly and timely classified inference requests in the number of requests per second (RPS). "Reject" stands for the request rejection ratio, which is the proportion of rejected inference requests. "Delay" stands for the request delay ratio, which is the proportion of delayed inference requests. The "Overall" columns show the overall results over entire workloads. The "Bursty state" and "Normal state" columns show the results in the bursty and normal states, respectively. The "MAP (SPP)" rows show the results on our main Markovian workload. The "Non-Markovian" rows show the results of our non-Markovian workload. The gray background is used to highlight the results of our framework.



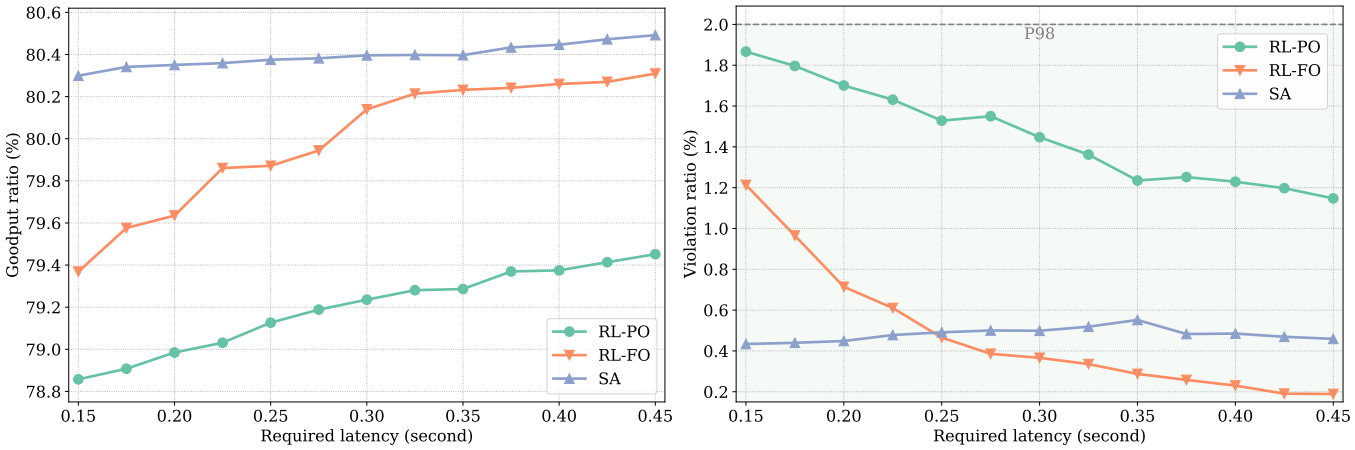
(a) Goodput ratio. (b) Violation ratio. (c) PNASNet scheduling ratio.

Fig. 2: Learning curves of our framework on our main workload. "PO" and "FO" stand for the partial and full observability, respectively. The curves labeled "overall" present the overall results. The curves labeled "normal" and "bursty" present the results in the normal and bursty states, respectively. The shaded regions represent the standard deviation over trials.

to ensure that the performance of this baseline approach is comparable, the duration of each time slot is set to 0.1 second, which is an order of magnitude smaller than the default setting of InFaaS.

- The stationary approximation approach (SA for short) not only assumes that the workload states are observable, but also assumes that the holding times in the bursty and normal states are always sufficiently long, such that if a fixed probability distribution is used for request scheduling in each state, the inference serving system can almost always be in its stationary states. Under these assumptions, if the MAP parameters and all the information about the model instances are also known, then using Erlang's waiting time distribution of the $M/D/1$ queue [69], a solvable approximation problem of maximizing the heavyweight model utilization with respect to the tail latency requirement can be formulated, where two probability distributions are found, respectively, for the bursty and normal states. During the evaluation, these two distributions are respectively used in their corresponding states. More details about this baseline approach can be found in Appendix F.

Notably, our comparisons are fair in the sense that, during



(a) Goodput ratio.

(b) Violation ratio.

Fig. 3: Results for varying required latencies.

evaluation, the computational resources available for all the approaches are the same. However, due to the additional strong assumptions, the fully-observable versions of our framework and the stationary approximation approach are most likely intractable in practice.

5.1.8 Evaluation Metrics

For evaluation metrics, we introduce the goodput ratio, which is the proportion of inference requests that are correctly classified within the required latency among the total number of inference requests arriving. Also, to examine whether the tail latency requirement is met, we introduce the violation ratio, which is the proportion of delayed or rejected inference requests among the total number of inference requests arriving. Furthermore, to gain insight into how the inference requests are scheduled, we introduce the heavyweight model scheduling ratio, which is the proportion of inference requests that are scheduled to the model instances of PNASNet among the total number of inference requests arriving.

5.2 Experimental Results

5.2.1 Main Results on the MAP

We summarize our results in Table 2. For the experiments on our main Markovian workload, we need to first investigate two primary questions: can our framework adaptively serve inference requests as expected, and can our framework outperform the conventional approaches? From the results in Table 2, we answer both questions positively. Specifically, even under the partial observability, our framework exhibits two distinct behaviors in the bursty and normal states: in the normal state, more than 96% of the inference requests are scheduled to the heavyweight PNASNet model, achieving 82.62% goodput ratio with almost no rejected and delayed requests; by contrast, in the bursty state, only 12.78% of the inference requests are scheduled to the heavyweight PNASNet model, such that the burstiness of the workload is properly handled with less than 3% violation ratio. Since these behaviors appear to be consistent with our idea of workload-adaptive inference request scheduling, we demonstrate that this idea can be implemented by our framework. For the comparisons between our proposed and conventional approaches, as shown in Table 2, under the partial observability and without violating the latency requirement, our framework achieves 79.24% overall goodput ratio, which is more than 1.5% higher than that of the best performing conventional approach using only EfficientNet. Here, we remark that this advantage of our framework is compelling since the goodput does not include any correctly classified but delayed requests. Looking into the results, we observe that the conventional approaches suffer from either "over-compressing" or "under-compressing" of the models. More specifically, for the conventional approach using only EfficientNet, there are no delayed and rejected requests no matter what state the workload is in; however, when the workload is in its normal state, this approach keeps its model instances idle most of the time, thus leading to the waste of the underlying computational resources. As a result, even though this approach, due to its zero violation ratio, achieves higher goodput ratio than our framework in the bursty state, it yields significantly lower goodput ratio than our framework in the normal state (77.69% versus 82.62%) and performs worse overall than our framework. For the conventional approach using only PNASNet and without proactive rejection of delayed requests, the edge inference serving system is tricked into a denial-of-service status when bursts occur, as almost all of the inference requests are either delayed or rejected in the bursty state, while our framework can always maintain the availability of the deep learning inference service.

By comparing the results of our framework under the partial and full observability, we can evaluate how the unobservability of workload states impacts the performance of our proposed method, and empirically evaluate the performance of our policy optimization approach in a similar fashion to the "ablation study". Specifically, from Table 2, we find that, under the full observability, our framework achieves 80.14% overall goodput ratio with less than 0.4% overall violation ratio, showing that if the workload states are observable, then the performance of our framework can be substantially improved. Since our framework gives simi-

lar results in the normal state under different observability, this improvement under the full observability is attributed to the good inference request scheduling in the bursty state, where the significant increase of the goodput ratio and the decrease of the violation ratio show that the learned policy can respond rapidly and effectively to the explicit burstiness. Also, in Fig. 2, we present the learning curves of our framework under the partial and full observability to illustrate how the evaluation metrics change over the course of training, where we find that with the knowledge of workload states, both the performance and the sample efficiency of our policy optimization approach can be improved. In summary, besides the inevitable negative impact of the workload-state unobservability on the performance of our framework, the above findings indicate that our policy optimization approach can effectively leverage useful information to learn better policies. Moreover, we vary the required latency of the P98 latency requirement from 150 ms to 450 ms in steps of 25 ms and compare our framework under different observability with the stationary approximation approach. The results, plotted in Fig. 3b, show that, even under the partial observability, our framework can always meet the P98 latency requirements with the required latencies in our experimental range. Meanwhile, the results in Fig. 3a show that, under the same algorithm settings of our policy optimization approach, our framework achieves a goodput ratio that decreases as we decrease the required latency; however, even for the lowest required latency of 150 ms and under the partial observability, our framework still outperforms the conventional approaches. As we can see in Fig. 3b, due to the approximation errors of various types, the stationary approximation approach maintains a violation ratio of about 0.5%, which is smaller than the target violation ratio of 2%. Nevertheless, in the bursty state, such an approach often schedules more inference requests to the heavyweight PNASNet model than our framework to achieve higher goodput ratio. In our experiments, we find that our policy optimization approach tends to find "conservative" policies avoiding scheduling more than 15% of the inference requests to the heavyweight PNASNet model in the bursty state, thereby limiting its performance.

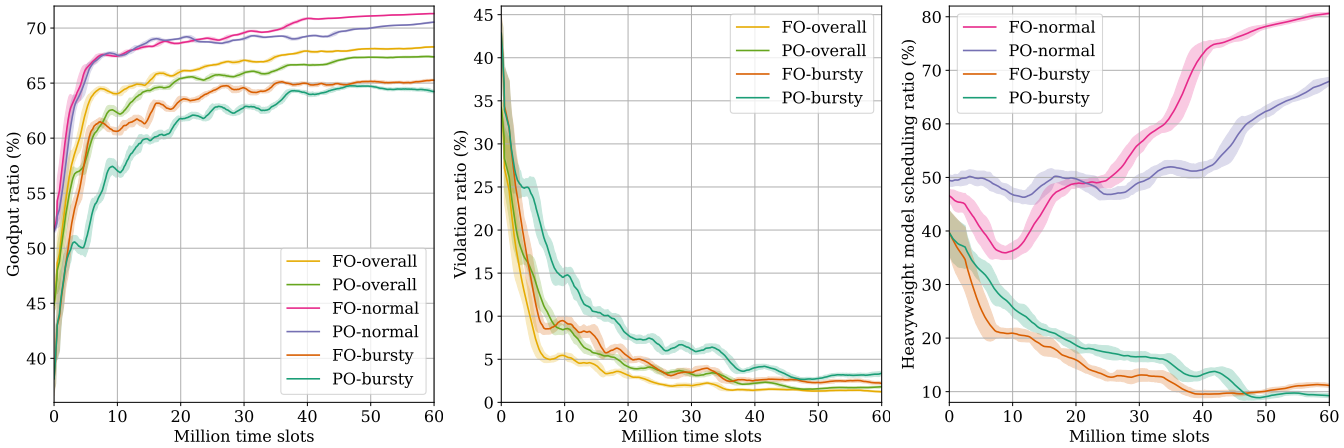
From Table 2, we can see that our framework outperforms the request scheduling approach of InFaaS: even under the partial observability, our framework enjoys about 1.1% accuracy gap with this baseline approach while yielding lower violation ratio. Besides, the results show that this baseline approach suffers from two major drawbacks. First, in the normal state, it performs poorly due to the heavyweight model scheduling ratio of about 50%. Second, in the bursty state, its performance is highly dependent on the time slot duration. This is because by reducing the time slot duration, there will be fewer inference requests being scheduled to the overloaded heavyweight model instances, and thus the violation ratio can be reduced. However, from an engineering perspective, it may be difficult to further reduce the time slot duration.

Finally, we find that the single queue approach can also, to some extent, realize our idea of workload-adaptive inference request scheduling. As we can see from the results in Table 2, in the bursty state, this approach achieves superior goodput ratio, where the high heavyweight model scheduling ratio of 22.85% does not cause any delay and rejection of the inference requests; however, in the normal state, this approach schedules only slightly more than half of the inference requests to the heavyweight model and fails to take full advantage of the heavyweight model, thereby yielding lower goodput ratio than our framework. Overall, this approach has the same level of performance as the adaptive-batching variant of our framework under the partial observability, where our framework is advantageous in the normal state but worse in the bursty state. Compared to our framework, the main advantages of the single queue approach are that it is simple, easy to implement, and does not need any training. However, these advantages come at a cost: this approach is agnostic of the latency requirements, meaning that the quality of inference services is hard to be controlled according to the requirements. Moreover, as will be discussed later, we believe that our framework has much better extensibility and greater potential than the single queue approach.

5.2.2 Results of the Adaptive-batching Variant

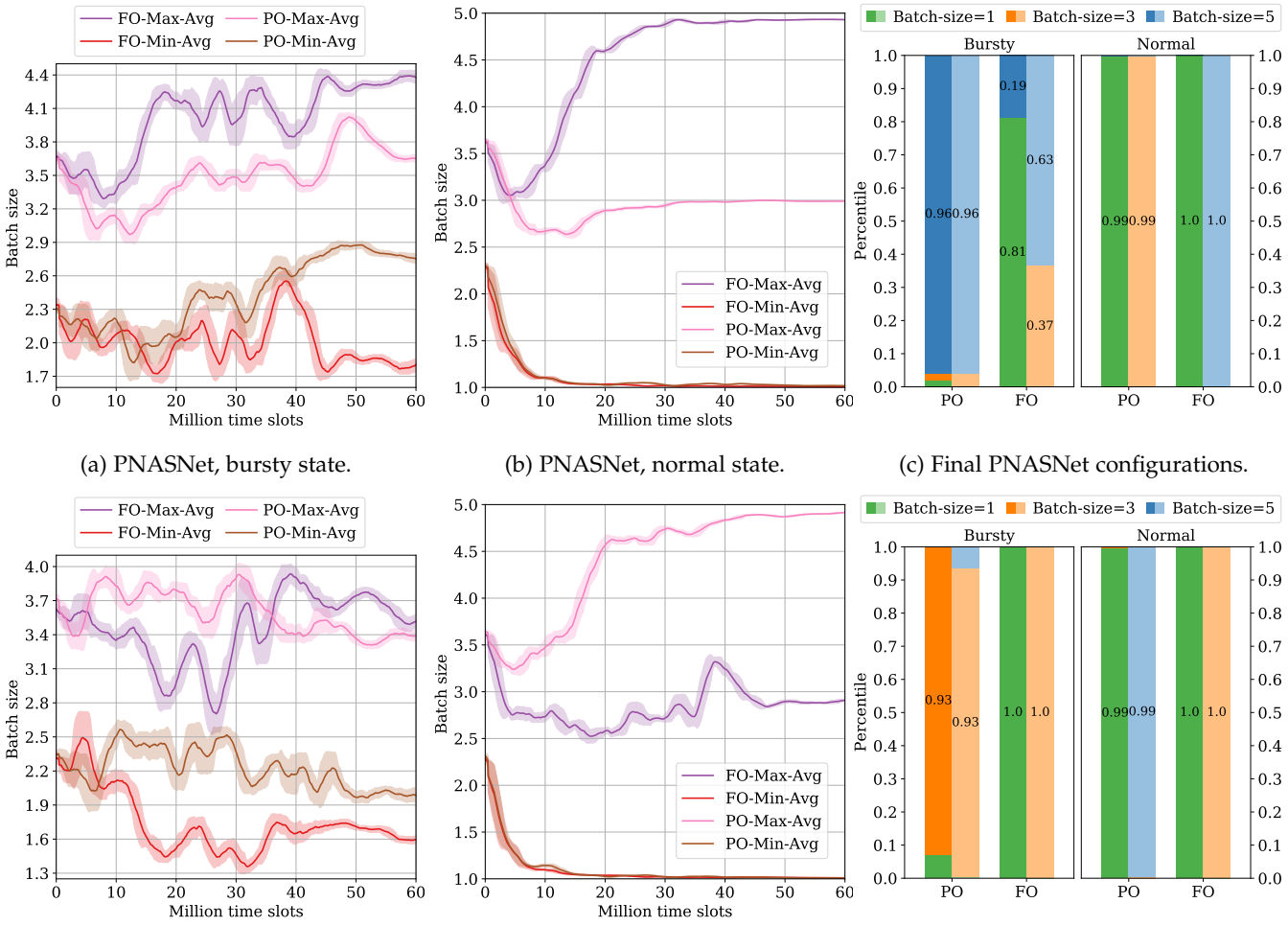
Here, we develop an adaptive-batching variant as an example to show the extensibility of our framework. With the help of adaptive batching, our framework can not only determine how to schedule the inference requests, but also determine how the inference requests are served by the model instances, thereby allowing more fine-grained control over the quality of inference services. Note that a detailed discussion is in Section 6.2 to fully illustrate the potential extensibility of our framework.

To integrate adaptive-batching in our framework, we slightly modify the general bulk service rule introduced by Neuts [70], and consider that the model instances follow our modified rule to serve the inference requests in batches. Specifically, at the beginning of each time slot, the scheduler not only determines the probability distribution for request scheduling, but also determines the batch configurations by selecting a minimum batch size and a maximum batch size for each model deployed in the framework, where the maximum batch size is no less than the minimum batch size. Then, under our modified rule, if the number of inference requests waiting for service from a model instance is less than the corresponding minimum batch size when the model instance finishes serving a batch, the model instance must wait until the number of inference requests in the buffer reaches the minimum batch size, whereupon all the inference requests in the buffer are served together in a batch; if the number of inference requests waiting in the buffer is no less than the corresponding minimum batch size, but less than the corresponding maximum batch size, a batch of inference requests is served, where the batch size is the maximum of the elements that are in a predetermined set and no more than the number of waiting requests; if the number of waiting requests is no less than the corresponding maximum batch size, the model instance serves a batch whose size is the corresponding maximum batch size. In our implementation, we use $\{1, 3, 5\}$ as the predetermined set of all the batch sizes that are available for both the PNASNet and EfficientNet models, so that the minimum and maximum batch sizes should also be selected from this set. Of course, the system state of this adaptive-batching



(a) Goodput ratio. (b) Violation ratio. (c) PNASNet scheduling ratio.

Fig. 4: Learning curves of the adaptive-batching variant on our main workload.



(a) PNASNet, bursty state. (b) PNASNet, normal state. (c) Final PNASNet configurations.

(d) EfficientNet, bursty state. (e) EfficientNet, normal state. (f) Final EfficientNet configurations.

Fig. 5: Additional results of the adaptive-batching variant. (a), (b), (d), and (e) plot the average minimum and maximum batch sizes of the two models against the number of time slots during training. (c) and (f) plot the final batch configurations learned for the PNASNet and EfficientNet models, respectively, where the minimum and maximum batch size configurations are distinguished by darker versus lighter shades.

variant should include the batch sizes of all the batches in service.

In our experiments, we evaluate the adaptive-batching variant on our main workload under different observability

and provide the empirical performance results in Table 2. As shown by the results, the main advantage of the adaptive-batching variant is that it effectively reduces the violation ratio of our framework: under the partial observability, the overall violation ratio is reduced from almost 1.45% to less than 0.9%; under the full observability, the overall violation ratio is reduced from almost 0.37% to less than 0.1%. Because of this advantage, under the partial observability, the adaptive-batching variant improves the overall goodput ratio to 79.49%, yielding an improvement of 1.8% over the conventional approach using only EfficientNet. However, keeping most of the algorithm settings (including the two weights α and β) unchanged, the policies learned for the adaptive-batching variant appear to be more "conservative": in the bursty state, whatever the observability is full or partial, the heavyweight model scheduling ratio drops to less than 6%; even in the normal state, there is a smaller proportion of inference requests scheduled to the heavyweight PNASNet model. This may explain why, under the full observability, the goodput ratio improvement achieved by the adaptive-batching variant is so slight. This "conservative" performance may be caused by the fact that the policies are learned in a naive manner similar to H-PPO [71], where the dependence between discrete and continuous components of actions is neglected.

In Fig. 4, we present the learning curves of the adaptive-batching variant. As can be seen by comparing Fig. 4c with Fig. 2c, the adaptive-batching variant requires more training time to achieve the same level of heavyweight model scheduling ratio. Moreover, to better understand how adaptive batching works, in Fig. 5, we illustrate the final batch configurations learned for the two models and the evolution of the batch configurations over the course of training. As we can see, most of the curves in Fig. 5b and Fig. 5e rise or fall monotonically and become flatter as the training time increases, implying the importance of the batch configurations in the normal state. In particular, to prevent a high proportion of inference requests from being delayed, for any of the two models, the minimum batch size should always be set to small values in the normal state. Meanwhile, in Fig. 5a and Fig. 5d, the curves appear to be more variable, possibly because sometimes the performances of some different batch configurations in the bursty state are similar. Despite the improvement, we notice from Fig. 5c that, under the partial observability, the adaptive-batching variant finally uses the problematic batch configurations where, in the bursty state, the heavyweight PNASNet model mostly only serves batches of size 5, since such batch configurations require high heavyweight model scheduling ratio to perform well in the bursty state.

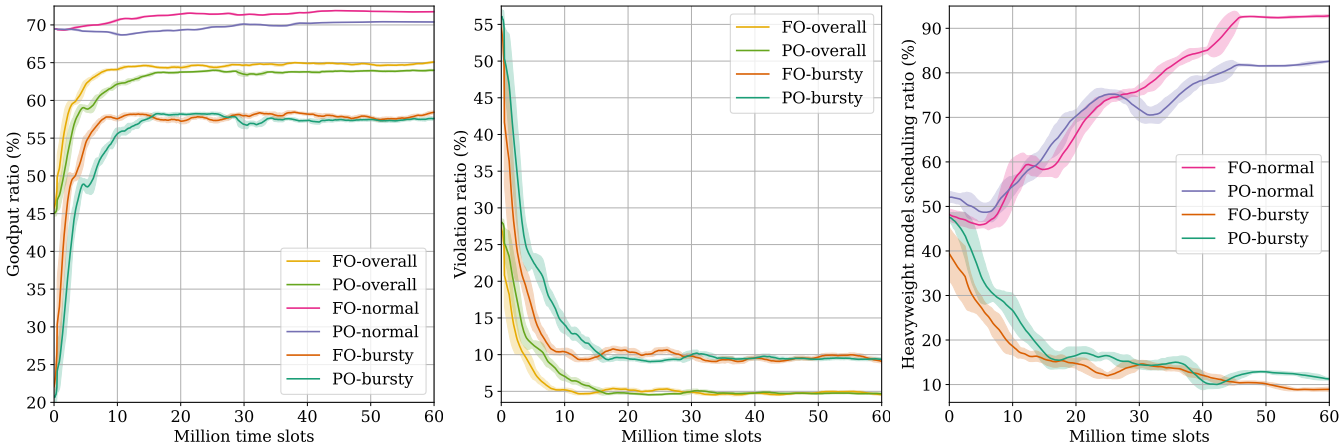
5.2.3 Results on the Non-Markovian Workload

Here, we adopt another non-Markovian arrival process as the workload to demonstrate the generality of our framework to handle more general workloads. Through the experiments on this non-Markovian workload, we aim to show that the practical use of our framework is not restricted by the assumption that the workload is a MAP. Essentially, this non-Markovian workload is a time-varying Poisson process with the arrival rate periodically switching between a low and a high value, where in each period of the arrival rate process, the arrival rate first stays at the low value for a fixed time duration, and then stays at the high value for the rest of the period. For convenience, if the arrival rate is at the high value, we say that the workload is in the bursty state; otherwise, we say the workload is in the normal state. For the purpose of comparison, we set the low and high values of the arrival rate to 10 and 180, respectively. Also, the time durations of the normal and bursty states in each period are set to the mean sojourn times of our main workload's normal and bursty states, respectively. Of course, the system state under this non-Markovian workload should include the directly unobservable time elapsed since the last arrival rate switching. Except for the modifications with respect to the new system state, we use the same algorithm settings as on our main workload. Note that the key distinction between our main workload and the non-Markovian workload is that the state switching of our main workload is unpredictable and governed by a CTMC; while the state switching of the non-Markovian workload is deterministic and periodic.

In the lower part of Table 2, we provide the experimental results on the non-Markovian workload. As can be seen, under the partial observability, our framework on the non-Markovian workload yields almost the same performance as that on our main workload, and consistently outperforms all the conventional approaches. Under the full observability, our framework performs even slightly better on the non-Markovian workload than on our main workload, achieving 80.27% goodput ratio on the non-Markovian workload compared to 80.14% on our main workload. We believe this superior performance of our framework on the non-Markovian workload under the full observability is non-trivial: in both the bursty and normal states, higher heavyweight model scheduling ratio is achieved with similar or even lower violation ratio, implying that the learned policy may be able to predict the workload states in the upcoming time slot and use the prediction to guide the request scheduling. Also, the arrival rate process is a deterministic process dependent only on time and thus is predictable, implying that it is possible to accurately estimate the current workload state and the elapsed time since the last arrival rate switching. Then, our framework also can work under the full observability in practice. Moreover, we present the learning curves of our framework on the non-Markovian workload in Fig. 6, where we see that the learning curves on the two workloads are similar. Consequently, these empirical results verify the effectiveness of our framework on the non-Markovian workload.

5.2.4 Results on the Efficiency of Probabilistic Routing

In our implementation, the probabilistic routing of arriving inference requests is run as an individual thread. Empirically, we found that our framework allows an inference request to be scheduled to a model instance within a few nanoseconds upon its arrival. Since the average overhead of probabilistic routing is several orders of magnitude smaller than the average interarrival time when our main workload is in its burst state, it is demonstrated that the request scheduling overhead is indeed negligible.



(a) Goodput ratio. (b) Violation ratio. (c) PNASNet scheduling ratio.

Fig. 6: Learning curves of our framework on the non-Markovian workload.

6 Discussion

In this section, we discuss why burstiness is one of the most important edge inference workload attributes; we also discuss the further extensibility and fault tolerance of our framework.

6.1 Bursts in Edge Inference Workloads

Before we move on to discuss the edge deep learning inference workloads, it is important to know the bursts are generally caused by what. In a nutshell, the phenomenon of sudden workload bursts is also known by names such as "flash crowd", "flash event" or "spike", and is well-known to be triggered by unpredictable external events [72], [73]. This external-event-triggered nature, coupled with the sustainable Internet-of-things (IoT) growth in the network edge, makes burstiness one of the most important edge workload attributes. To see this, consider a typical edge service scenario where a few edge servers serve hundreds of thousands of event-driven IoT devices together. If an unexpected event triggers a number of IoT devices to simultaneously generate workload, then a "flash crowd" will appear. In related analyses, Eismann *et al.* [74] study the workload attributes of serverless applications and find that 81% of the analyzed use cases exhibit bursty workloads; Pekar *et al.* [75] study the workload attributes of IoT applications and find that workload bursts are likely to occur in most of the analyzed use cases. These analyses show that bursts normally exist in modern IoT and serviceless computing services. In the case of edge deep learning inference, we provide the following examples to further illustrate how bursty workloads can occur:

- For the private chatbot system providing internal tools for the employees of an organization, where the domain-specific language models are deployed in the local private cloud of the organization, if an unexpected event happens and prompts many employees to simultaneously and intensively query the models, then a system workload burst will occur;
- for the vehicular edge computing system [76] in the vehicle-to-everything scenario, where the deep learning models are deployed in the roadside units to enable various applications, such as autonomous driving and road safety, if traffic congestion is caused by accidents, then there will be service workload bursts.

6.2 Extension of Our Framework

Beyond the adaptive-batching variant, we believe that our framework can be further extended in the following ways.

- Our framework can be further extended to enable more combinatorial and complex actions, *e.g.*, adaptively ensembling models, adaptively changing the model instances. If the computing infrastructure has sufficient resource elasticity, our framework can even be extended to support adaptive resource scaling. However, there are many open issues that must still be addressed, such as how to tackle the highly-structured system states with time-varying model instance settings.
- Mainly for system-model simplicity, in our framework, the update of the probability distribution for request scheduling is time-triggered. However, since the decision process is directly derived from the system's underlying continuous-time Markov process, our framework can be naturally extended to allow the update of the probability distribution for request scheduling to be triggered by events such as the rejection of an inference request.
- Although our framework can only handle one inference task, it can be extended to allow adaptive resource provisioning for multiple inference tasks. From a technical perspective, this extension of our framework is roughly equivalent to extending our framework to enable adaptive resource scaling for a single inference task.

6.3 Fault Tolerance of Our Framework

For simplicity and clarity, we restrict the focus of this paper to optimizing the service quality of the inference serving

systems whose model instance settings are expected to be able to invariant over time. To handle the unexpected faults leading to changes in model instance settings, a straightforward approach sufficient for small-scale inference serving systems is per-taining multiple request scheduling policies for different pre-designed model instance settings, and then accordingly switching the model instance setting and request setting when faults are located.

7 Conclusion

In this paper, we considered the question: how can a deep learning inference service be adaptive to different workload states of a bursty workload at the resource-constrained network edge? To address this question, we introduced the idea of workload-adaptive inference request scheduling, and proposed a request scheduling framework for general-purpose edge inference serving systems to implement this idea, where the inference requests are scheduled to be served by different types of models according to a policy learned by reinforcement learning. Empirically, we not only demonstrated the effectiveness of our framework in practical settings, but also showed that our framework provides the generality to handle general workloads. In the future, we believe that our framework can be extended in various ways as discussed earlier, and there will be new optimization methods to improve the request scheduling performance of our framework.

References

[1] N. Ansari and X. Sun, "Mobile edge computing empowers internet of things," *IEICE Transactions on Communications*, vol. 101, no. 3, pp. 604–619, 2018.

[2] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world from edge to core," International Data Corporation, IDC white paper, 2018.

[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] R. van der Meulen, "What edge computing means for infrastructure and operations leaders," Gartner, Techical report, 2018. [Online]. Available: https://tinyurl.com/4s98xc5p

[5] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[6] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[7] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial informatics*, vol. 13, no. 5, pp. 2140–2150, 2017.

[8] A. Alnoman, S. K. Sharma, W. Ejaz, and A. Anpalagan, "Emerging edge computing technologies for distributed iot systems," *IEEE Network*, vol. 33, no. 6, pp. 140–147, 2019.

[9] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 1049–1062.

[10] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, 2020, pp. 443–462.

[11] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 613–627.

[12] C. Zhang, M. Yu, F. Yan *et al.*, "Enabling cost-effective, slo-aware machine learning inference serving on public cloud," *IEEE Transactions on Cloud Computing*, 2020.

[13] Amazon Web Services. Use amazon sagemaker elastic inference. [Online]. Available: https://tinyurl.com/55yww9h6

[14] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 109–120.

[15] M. Enguehard, Y. Desmouceaux, and G. Carofiglio, "Efficient latency control in fog deployments via hardware-accelerated popularity estimation," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 3, pp. 1–23, 2020.

[16] R. Olaniyan, O. Fadahunsi, M. Maheswaran, and M. F. Zhani, "Opportunistic edge computing: concepts, opportunities and research challenges," *Future Generation Computer Systems*, vol. 89, pp. 633–645, 2018.

[17] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[18] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[20] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[21] X. Tan, H. Li, L. Wang, X. Huang, and Z. Xu, "Empowering adaptive early-exit inference with latency awareness," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 9825–9833.

[22] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4852–4861.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[24] D. M. Lucantoni, "New results on the single server queue with a batch markovian arrival process," *Communications in Statistics. Stochastic Models*, vol. 7, no. 1, pp. 1–46, 1991.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving," *arXiv preprint arXiv:1712.06139*, 2017.

[27] H. Qin, S. Zawad, Y. Zhou, S. Padhi, L. Yang, and F. Yan, "Reinforcement-learning-empowered mlaas scheduling for serving intelligent internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6325–6337, 2020.

[28] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *2021 {USENIX} Annual Technical Conference ({USENIX}{ATC} 21)*, 2021, pp. 397–411.

[29] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 19–34.

[30] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[31] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 1, pp. 1–28, 2020.

[32] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[33] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural

architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[34] Y. Jin, L. Jiao, Z. Qian, S. Zhang, N. Chen, S. Lu, and X. Wang, "Provisioning edge inference as a service via online learning," in *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2020, pp. 1–9.

[35] V. Nigade, P. Bauszat, H. Bal, and L. Wang, "Jellyfish: timely inference serving for dynamic edge networks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 277–290.

[36] X. Wang, G. Gao, X. Wu, Y. Lyu, and W. Wu, "Dynamic dnn model selection and inference off loading for video analytics with edge-cloud collaboration," in *Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2022, pp. 64–70.

[37] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5449–5465, 2020.

[38] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[39] C. Zhou, W. Wu, H. He, P. Yang, F. Lyu, N. Cheng, and X. Shen, "Deep reinforcement learning for delay-oriented iot task scheduling in sagin," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 911–925, 2020.

[40] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A drl agent for jointly optimizing computation offloading and resource allocation in mec," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 508–17 524, 2021.

[41] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in iot edge computing," *Sensors*, vol. 21, no. 5, p. 1666, 2021.

[42] Z. Zhuang, J. Wang, Q. Qi, J. Liao, and Z. Han, "Adaptive and robust routing with lyapunov-based deep rl in mec networks enabled by blockchains," *IEEE Internet of Things Journal*, 2020.

[43] Y. Zhang, B. Feng, W. Quan, G. Li, H. Zhou, and H. Zhang, "Theoretical analysis on edge computation offloading policies for iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4228–4241, 2018.

[44] P. Borylo, A. Lason, J. Rzasa, A. Szymanski, and A. Jajszczyk, "Energy-aware fog and cloud interplay supported by wide area software defined networking," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.

[45] D. M. Lucantoni, "The bmap/g/1 queue: a tutorial," *Performance Evaluation of Computer and Communication Systems*, pp. 330–358, 1993.

[46] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[47] M. Combé and O. J. Boxma, "Optimization of static traffic allocation policies," *Theoretical Computer Science*, vol. 125, no. 1, pp. 17–43, 1994.

[48] X. Guo, Y. Lu, and M. S. Squillante, "Optimal probabilistic routing in distributed parallel queues," *Performance Evaluation Review*, vol. 32, no. 2, pp. 53–54, 2004.

[49] S. C. Borst, "Optimal probabilistic allocation of customer types to servers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 23, no. 1, pp. 116–125, 1995.

[50] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 248–260, 1975.

[51] J. Anselmi and G. Casale, "Heavy-traffic revenue maximization in parallel multiclass queues," *Performance Evaluation*, vol. 70, no. 10, pp. 806–821, 2013.

[52] J. Sethuraman and M. S. Squillante, "Optimal stochastic scheduling in multiclass parallel queues," in *Proceedings of the international conference on Measurement and modeling of computer systems*, 1999, pp. 93–102.

[53] R. A. Kronmal and A. V. Peterson Jr, "On the alias method for generating random variables from a discrete distribution," *The American Statistician*, vol. 33, no. 4, pp. 214–218, 1979.

[54] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *Proceedings of the International Conference on Learning Representations*, 2016.

[55] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the International Conference on Machine Learning*, 2014.

[56] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method*. John Wiley & Sons, 2016, vol. 10.

[57] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[58] I. Shmyrin, "States of a map flow of events: Optimal estimation by the maximal a posteriori state probability criterion," *Automation and Remote Control*, vol. 65, no. 9, pp. 1444–1451, 2004.

[59] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of the IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[60] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *arXiv preprint arXiv:1507.06527*, 2015.

[61] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[62] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[63] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[64] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" in *International Conference on Machine Learning*. PMLR, 2019, pp. 5389–5400.

[65] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms," in *International conference on machine learning*. PMLR, 2018, pp. 1039–1048.

[66] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments," *arXiv preprint arXiv:1910.04281*, 2019.

[67] G. Matheron, N. Perrin, and O. Sigaud, "The problem with ddpg: understanding failures in deterministic environments with sparse rewards," *arXiv preprint arXiv:1911.11679*, 2019.

[68] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[69] G. J. Franx, "A simple solution for the m/d/c waiting time distribution," *Operations Research Letters*, vol. 29, no. 5, pp. 221–229, 2001.

[70] M. F. Neuts, "A general class of bulk queues with poisson input," *The Annals of Mathematical Statistics*, vol. 38, no. 3, pp. 759–770, 1967.

[71] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," *arXiv preprint arXiv:1903.01344*, 2019.

[72] X. Zhou, Z. Zhao, R. Li, Y. Zhou, T. Chen, Z. Niu, and H. Zhang, "Toward 5g: When explosive bursts meet soft cloud," *IEEE network*, vol. 28, no. 6, pp. 12–17, 2014.

[73] M. Curiel and A. Pont, "Workload generators for web-based systems: Characteristics, current status, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1526–1546, 2018.

[74] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "A review of serverless use cases and their characteristics," *arXiv preprint arXiv:2008.11110*, 2020.

[75] A. Pekar, J. Mocnej, W. K. Seah, and I. Zolotova, "Application domain-based overview of iot network traffic characteristics," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–33, 2020.

[76] Y.-J. Ku, P.-H. Chiang, and S. Dey, "Real-time qos optimization for vehicular edge computing with off-grid roadside units," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 11 975–11 991, 2020.