Scheduling Inference Workloads in the Computing Continuum with Reinforcement Learning

Scheduling Inference Workloads in the Computing Continuum with Reinforcement Learning

Gabriele Castellano[§], Juan-José Nieto [‡], Antonino Angi[*], Francisco Àlvarez Terribas[‡], Jordi Luque[‡], Ferrán Diego Andilla[‡], Alessio Sacco[*], Flavio Esposito[†], Fulvio Risso[*]
[§] *Huawei Technologies*, France
[‡] *Telefónica Research*, Spain
[*]Computer and Control Engineering, *Politecnico di Torino*, Italy
[†]Computer Science, *Saint Louis University*, USA

***Abstract*—As many recent real-time applications (*e.g.,* Augmented/Virtual Reality, cognitive assistance) rely on Deep Neural Networks (DNNs) for inference tasks, edge computing has appeared as a key enabler to deploy such applications as closest to the data sources, helping meet stringent latency and throughput demands. However, the limited resources typically available at the edge create significant challenges for efficiently managing inference workloads. Thus, a trade-off between network and processing time should be considered when it comes to end-to-end delay requirements. In this paper, we focus on the problem of scheduling inference jobs of DNN models in such edge-cloud continuum at short timescales (*i.e.,* a few milliseconds). Through simulations, we analyze several policies in the realistic network settings and workloads of a large ISP, highlighting the need for a dynamic scheduling policy that can adapt to varying network conditions and workload demands. To this end, we propose ASET, a Reinforcement Learning (RL)-based scheduling algorithm able to dynamically adapt its decisions according to the system conditions. Our results show that ASET effectively provides the best performance compared to a set of static policies when scheduling over a distributed pool of edge-cloud resources.**

***Index Terms*—job scheduling, reinforcement learning, edge cloud continuum**

I. Introduction

The popularity of Deep Neural Networks (DNNs) has increased in recent years, particularly in applications like Augmented/Virtual Reality (AR/VR), cognitive assistance, and video surveillance. DNN model training is typically done offline in centralized data-centers or distributed. However, DNN inference task is usually performed *online* with constraints in terms of accuracy, throughput, and latency, which may significantly differ across applications. Providing an inference service requires addressing several challenges, such as selecting the appropriate model variant, processing unit, and nodes and resources [1], [2]. This requires management at different timescales, with schedulers selecting computing instances for new requests and orchestrators optimizing model placement across nodes.

Edge computing is considered a key enabler to deploy DNN-based applications with stringent delay or bandwidth requirements. Due to the less availability of resources at edge, multiple inference models of different capacities should be considered, and end-to-end delay requirements may lead to considering a trade-off between network delay and processing time. For these reasons, the optimal selection of inference models while scheduling real-time requests at the edge is still a challenging task. Recent work [3], [4] combines edge computing and deep learning, but none analyzes inference workload optimization considering different application constraints in realistic edge network settings.

In this paper, we focus on the problem of *scheduling* DNN inference requests in both edge and cloud infrastructures, while taking into account not only accuracy (*i.e.,* model selection) but also throughput and latency constraints under realistic edge deployment settings. First, we model our edge-cloud continuum inference system and provide a definition of the scheduling problem, and propose several static scheduling policies both original and from literature. From evaluating static policies, we observe that different applications may benefit differently from each scheduling policy. Based on the insights derived by this analysis we propose ASET (Adaptive Scheduling of Edge Tasks), an adaptive scheduling algorithm based on Reinforcement Learning, which dynamically monitors system conditions and apps requirements optimizing its decisions accordingly. We evaluate ASET simulating three topologies based on the realistic network of a large ISP and a real GPU-equipped dc-cloud, showing that ASET outperforms static policies when resources are distributed across the edge network.

II. Related Work

The provisioning of on-demand inference services has been investigated in several recent works.

Cloud inference workload. Most solutions address the scheduling of inference queries over the resources of a Data Center by proposing algorithms and strategies to improve the performance. The most complete solutions is provided by INFaaS [2], which focuses on ease of use, providing transparent scheduling of incoming queries over available model variants, and autoscaling of deployed models based on load thresholds. However, all the previous works address the scheduling problem only from the boundaries of a data center, considering neither *(i)* network latency, thus becoming no suitable in scenarios with real-time constraints, nor *(ii)* resource constrained clusters.

Edge inference workload. Fewer and more recent are the trends that combine DNN with edge computing [3], with the

	Dynamic Continuum	Multi-App	Real-world testing	Latency-aware	Cloud Continuum	Multi-model Variant
Muri [5]	✘	✘	✔	✘	✘	✔
DRL+DQN [6]	✘	✘	✘	✔	✔	✘
TapFinger [7]	✘	✔	✔	✔	✘	✔
Oakestra [8]	✔	✔	✔	✔	✔	✘
Meta-RL [9]	✘	✔	✔	✘	✘	✔
ASET	✔	✔	✔	✔	✔	✔

TABLE I: Comparative analysis against the most recent works on scheduling inference workloads at the edge.

aim of overcoming scalability and latency limitations of cloud computing. In [10], authors propose an approach to schedule tasks across multiple edge servers, seeking minimization of end-to-end latency; whereas VideoEdge [4] study the problem of processing data streams from scattered devices, exploiting the geographically distributed edge/cloud clusters. However, neither processing nor network latencies are taken as constraints, thus making this approach not suitable for interactive or critical scenarios.

Tasks scheduling. With the rapid and increasing number of real-time applications that require powerful resources and fast processing time, *e.g.*, augmented reality, smart healthcare, and autonomous driving, the proliferation of workloads scheduling techniques for edge computing has emerged [11]. In this context, Muri [5] proposes an adaptation of the Blossom algorithm to group DL training (over GPUs) jobs by exploiting multi-resource interleaving of these tasks to achieve high resource utilization and reduce job completion time; whereas Oakestra [8], a hierarchical orchestration framework designed for edge computing aimed at supporting applications's SLAs at the dynamic edge variations.

Deep Reinforcement Learning (DRL) has risen as a powerful approach for real-time decision-making, allowing adaptive and efficient optimization even for online task scheduling and resource allocation [12], [13]. Recently, TapFinger [7] trains a multi-agent reinforcement learning (MARL) algorithm in combination with heterogeneous attention network graphs (HAN) to support each agent's optimal decision by providing all the necessary input features (*e.g.,* resource requirements, edge components); whereas Zheng et al. [6]introduce a model employing a DRL with a Deep Q-Network (DQN) approach, utilizing the servers' state, networks, and tasks as inputs for the scheduling process. To avoid the requirement of a training dataset, a meta-RL solution has been proposed in [9], where the RL algorithm does not use any initial training set but explores possible combination paths where the model's accuracy is low in order to gain more experience.

As summarized in Table I, to the best of our knowledge, none of the existing works addresses the problem simultaneously considering *(i)* end-to-end latency, accuracy, and throughput constraints, *(ii)* edge-cloud computing and multi-cluster deployment, *(iii)* real-time job dispatching, *(iv)* optimization on model variant selection for ML inference workloads [14].

III. Scheduling in edge-cloud infrastructure

A. System modeling

Applications and data-streaming sources. We consider a set of sources running a variety of applications, each relying on one or more DNN inference tasks. Every application generates *queries* to be processed, i.e., each query represents the request to perform a specific inference task $j \in J$, where J is the set of inference tasks supported by the system. Since applications often require more than one query to be processed, we treat sequential queries as streams. Therefore, each query q belongs to a stream $i \in I$, being I the entire set of streams currently served by the system. Every stream has a set of requirements such as a maximum end-to-end delay D^i, and a minimum required accuracy A^i.

DNN Models and Variants. Every inference task j can be served using a *Deep Neural Network (DNN) model* m among the set of M^j models that are trained for task j. Therefore, the system provides a total of $N_m = \sum_{j \in J} |M^j|$ DNN models. Using object detection as an example application, a model m represents a particular pre-trained Neural Network architecture (e.g., yolo-v3, ssd-mobilenet-v1) with an accuracy A_m (mean average precision - mAP). A model m can be deployed and run through different setups and underlying hardware (e.g., SSD Mobilenet v1 on *(i)* Tensorflow-GPU with batch size 8, or on *(ii)* Opencv-CPU batch size 1 and 2 replicas, and more), thus obtaining a set V^m of different *model variants*. A model variant v features a given *processing delay* D_v, throughput *capacity* C_v (i.e., the maximum number of queries it can process per second), and *resource usage* $\boldsymbol{r}_v \in \mathbb{R}_+^k$.

Network topology and computing clusters. We consider a geographically distributed cloud-edge infrastructure composed by N_ν computing *clusters* (e.g., a centralized data center, a telco regional cloud, an eNodeB) typically organized in a hierarchical topology. Each cluster potentially provides different resources. We denote $\boldsymbol{c}_n \in \mathbb{R}_+^k$ the overall capacity of cluster n, with c_{nk} representing the amount of resource $k \in \mathbb{N}$ available on cluster n. On a long timescale, an *orchestrator* selects the appropriate set of model variants to deploy, optimizes their placement across the clusters, and allocates the appropriate resources.

B. Scheduling problem definition

We assume a scheduler is located at the nearest compute cluster available to existing stream sources, i.e., antenna/eNodeB or the home gateway/central office. It follows every stream source is served by a *scheduler* s among N_s different

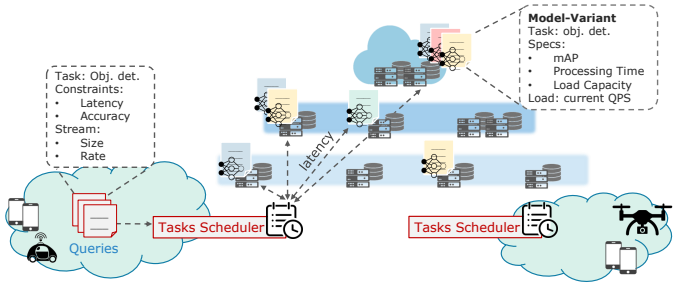


Fig. 1: The scheduler dispatches streams of queries on available model variants based on their constraints and geographical position of clusters.

ones (one per each lower layer cluster). Each scheduler s has a given average network delay d_n^s towards each cluster n; we also model the associated delay deviation as σ_n^s. We denote δ_i the additional access delay that affects stream i. Every scheduler is aware of each model variant v currently available on each cluster n, each with its current load $L_{vn}(t)$ (measured in terms of incoming queries per second). Based on the current conditions, for every stream i it serves, a scheduler s decides which model variant v on which cluster n should be used to process stream i based on application requirements. Specifically, it considers the stream data size ζ_i, its data rate ρ_i, its bit rate b_i, the maximum tolerated end-to-end delay D^i and the minimum required accuracy A^i, satisfying the following constraints:

(1) the selected model variant v is a valid implementation of task j required by i, $v \in V^m \wedge m \in M^j$;

(2) the load capacity of the chosen model variant is not exceeded, $L_{vn}(t) + \eta_v^i \rho_i \leq C_v$, where η_v^i the fractional load of stream i for model variant v;

(3) the sum of expected network delay and processing time does not exceed the maximum tolerated delay, $2(\delta_i + d_n^s + 2\sigma_n^s) + b_i\zeta_i + D_v(\zeta_i) \leq D^i$, where the addendum are round-trip propagation time, transmission delay for one query and the time needed to process the query,respectively;

(4) the selected model provides an adequate accuracy.

A graphical representation of the scheduling problem is depicted in Figure 1 and be formally defined as follows:

Definition 1. *(scheduling policy). Let us consider a stream i to be processed through a task j on an edge-cloud infrastructure that features a set of V^m compatible model variants over N_ν clusters ($|N| = N_\nu$). A scheduling policy is any function*

$$\beta \colon I \rightarrow V^m, N \tag{1}$$

that binds stream i to a feasible model variant $v \in V^m$ deployed on cluster $n \in N$, so that Constraints (1), (2), (3), and (4) are satisfied.

Based on the scheduling decisions, in a given time instant t the stream i will feature a *reject ratio* $q_i^R(t) \in [0, 1]$, i.e., the fraction of queries from stream i that have not been processed by the system because of resource unavailability, and a *failure ratio* $q_i^F(t) \in [0, 1]$, i.e. the fraction of queries that have been served violating one or more application requirements (i.e., delivered out of maximum tolerated delay). Therefore, the scheduler aims to maximize the fraction of queries that are served successfully, i.e., to minimize the sum of reject ratio and failure ratio.

C. Static scheduling policies

The following original and state-of-the-art static scheduling policies are considered:

1) closest: bind stream i to any feasible model variant v^* located on the cluster n^* that features the lower network latency to serving scheduler s, i.e., $n^* = \arg\min_{n \in N}(d_n^s + 2\sigma_n^s)$. This policy may lead to the early saturation of smaller clusters at the very edge, as they are always preferred [15].

2) load balancing: bind the input stream to model variant v^* on cluster n^* such that $(v^*, n^*) = \arg\min_{v,n \in V^m \times N} L_{vn}(t)$. This policy may lead to unfair allocation when latency-sensitive applications are in the minority.

3) farthest: bind stream i to any feasible model variant v^* located on the cluster n^* with the highest (still feasible) network latency, i.e. $n^* = \arg\max_{v \in N}(d_n^s + 2\sigma_n^s)$. This policy preserves smaller clusters at the very edge for those apps that really need them [16].

4) cheaper: bind stream i to model variant v^* on cluster n^* such that the expected end-to-end delay is maximized, i.e., $(v^*, n^*) = \arg\max_{v,n \in V^m \times N}(2(d_n^s + 2\sigma_n^s) + D_v(\zeta_i))$. We designed this policy as an improvement over *farthest*, as it additionally tries to preserve the most performing model variants.

5) random-proportional latency: bind stream i to model variant v on cluster n with probability $1/(2(d_n^s + 2\sigma_n^s) + D_v(\zeta_i))$. This guarantees that, on a large enough number of streams, bindings are proportionate to end-to-end delays [17].

6) random-proportional load: bind stream i to model variant v on cluster n with probability $C_v/L_{vn}(t)$. This guarantees that, on a large enough number of streams, bindings are proportional to the capacity of each model variant.

7) least impedance: bind stream i to model variant v^* on cluster n^* such that end-to-end latency to s is minimized, i.e., $(v^*, n^*) = \arg\min_{v,n \in V^m \times N}(2(d_n^s + 2\sigma_n^s) + D_v(\zeta_i))$ [17]. This greedy policy leads to the best performance when the overall load is low, but may suffer from a high rejection rate once the closest and fastest model variants are saturated.

IV. ASET Scheduling Algorithm

Our adaptive scheduling approach aims to learn the optimal policy depending on current system conditions, e.g, current applications, network topology, and stream arrivals that vary over time. Due to the lack of labeled data, the optimal policy learning is formulated as a Reinforcement Learning (RL) problem; hence, an intelligent agent tries to learn the optimal policy selection strategy according to the observed state of the environment and the maximization of a reward (typically maximizing the fraction of queries that are served successfully), as shown in Figure 2.

The proposed adaptive scheduling aims to optimize static network scheduling policies by maximizing the percentage of successfully dispatched streams. The *agent* interacts with the

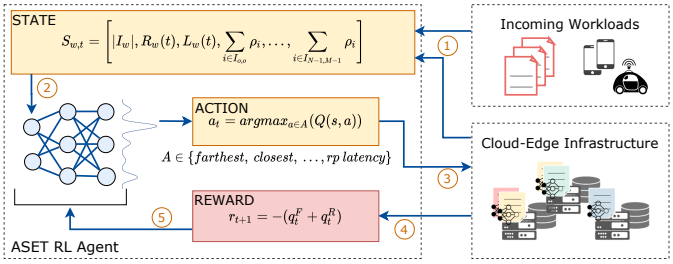
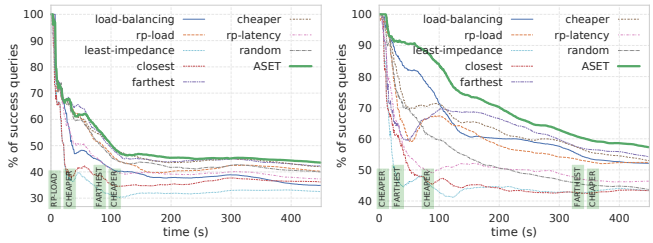


Fig. 2: Algorithm overview. State S_t, sampled from the environment, is forwarded through the agent DNN, which outputs action A_t; performing A_t on the environment contributes to reward r_{t+1} obtained at the next step.



(a) Cloud-based topology. (b) Edge-based topology.

Fig. 3: The ASET RL agent infers the optimal policy sequence based on the system conditions, seeking an optimal binding between workloads and model variants that maximizes the percentage of success queries. Plots show two runs on a cloud-based topology and on an edge-based one (see Section V).

environment at discrete time steps t, collecting observations from edge-cloud infrastructure to build up the current state $S_t \in S$ of the environment, being S the set of possible *states*. The agent evaluates a set of actions and chooses an action $A_t \in A$ from available network scheduling policies β. Every time the agent takes an action, the environment state is observed, and a reward score is used as feedback to improve policy selection. Rewards r_{t+1} are defined as a linear combination of failed queries and rejected queries due to resource constraints. The particular policy β_t, selected by the agent at time t, is used to dispatch all incoming streams during the subsequent time window $[t, t+T]$. Therefore, given the corresponding states sequence $\mathbf{S} = [S_0, S_T, S_{2T}, ..., S_{kT}]$ with $k \in \mathbb{N}$, the resulting overall scheduling policy $\beta(\mathbf{S}) = [\beta_0, \beta_T, \beta_{2T}, ..., \beta_{kT}]$ dynamically maps, with the corresponding baseline policies β_t, a stream i to a model variant v and its deployment on cluster n. The intuition of this behavior is provided in Figure 3.

A. Deep Q-Learning policy optimization

Our RL agent has to cope with a discrete set of actions, with $A \subset \mathbb{N}$ and modeled as a stochastic process with no memory, which is a Markov Decision Process [18] (MDP). In this work, our MDP defined by tuples $(S, A, \mathcal{T}, \mathcal{R}, \gamma)$ represents states comprised of partial observations from the system. Nonetheless, the model parameters of such MDP are unknown, i.e., the transition probabilities $\mathcal{T}(s'|a,s)$ and the rewards $\mathcal{R}(s'|a,s)$ of taking the action $A_t = a$ and moving from state $S_t = s$ to state $S_{t+1} = s'$. In Q-Learning, the optimal pair values (s,a), i.e., those yielding to the sequence of optimal actions, are generally called Quality-Values (Q-Values) and noted as $Q^*(s,a)$ [19]. They correspond to the sum of weighted rewards that the RL agent can expect on average after performing action a on state s. It is also known as the *expected return of actions*,

$$Q(s,a) = \mathbb{E}_{t^* \sim \pi_\phi} \{G_t | S_t = s, A_t = a\}. \quad (2)$$

where $G(t)$ is the expected return of rewards over time.

Bellman [18] showed that if an agent's trajectory follows the highest Q-Values, then its policy is optimal and leads to the highest $G(t)$ as well. In fact, Q-Learning is an adaptation of Bellman's value iteration algorithm, where a policy is implicitly, or off-line, learned by following trajectories yielding to the highest Q-Values [19]. In order to scale for large MDPs with a large number of states, a solution is to approximate the optimal $Q^*(s,a)$ using a Deep Neural Network, named Deep Q-Network (DQN) [20], to get an estimate $Q(s,a;\phi) \approx Q^*(s,a)$, where ϕ stands for the parameters of the DQN model.

B. State Encoding

We model the state in a continuous fashion, representing the environment in a given time t as a set of some particular features sampled from the system and averaged along a time window of size T. Features are evaluated separately for each available worker $w \in W$,and are as follows: *(i)* the number $|I_w|$ of streams currently served by worker w, being $I_w = \{i \in I |\ \beta(i) = (v,n)\}$; *(ii)* the current throughput $R_w(t)$ of worker w, in terms of responses delivered at the time instant t; *(iii)* the current load $L_w(t)$, measured in terms queries per second normalized on input size (as defined in Section III-B); *(iv)* number of incoming instant queries grouped by stream characteristics, e.g., queries of all streams that require end-to-end delay within a given range $[\delta^1, \delta^2[$ and features a data rate in the interval $[\rho^4, +\infty[$, i.e., $\sum_{i \in I_{1,4}} \rho_i$, where $I_{1,4} = \{i \in I |\ D^i \in [\delta^1, \delta^2[\wedge \rho_i \in [\rho^4, +\infty[\}$.

C. Training

The proposed RL scheduling agent is trained over a series of episodes that resemble various scenarios. Each episode corresponds to a different workload execution with given parameters, e.g. requirements from tasks, number of clients per minute (λ) or the seed value (ζ) for random number generation (RNG), and is concluded when the percentage of success queries, q_t^S, falls below a given threshold θ or when a timeout H is reached. This allows us to speed up the training by terminating unsuccessful or steady episodes quickly. At every time step t, a reward r_t scores the rate of successful queries normalized by corresponding time window. Additionally, we employ an ϵ-greedy exploration policy with parameter ϵ dynamically updated. The architecture of our DQN consists of a stacking of convolutional layers that extracts temporal correlations from the state tensor and many outputs as different static policies β.

V. Performance Evaluation

A. Evaluation settings

System Prototype demonstrates edge inference system functionalities, with a Master deploying workers, a Docker con-

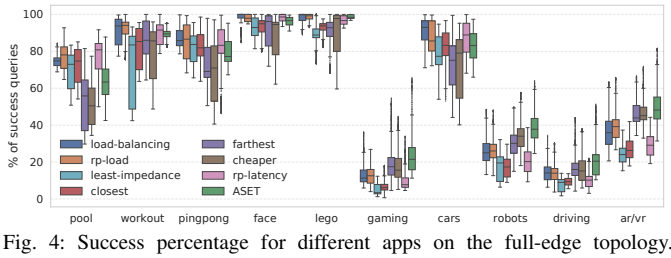


Fig. 4: Success percentage for different apps on the full-edge topology. Average values for episodes with dynamic clients rates.

tainer running worker pipelines, and a Monitoring agent collecting stats on model variant usage and performance. This prototype is used to profile pre-trained inference models' resource usage and performance.

Simulation Setup uses a simulated environment to evaluate a large-scale approach. Workers simulate inference tasks based on model variant profiling information, generating empty responses for each batch of queries after simulating processing and network delays. Other system components are deployed using their prototype implementation, ensuring a realistic timescale operation.

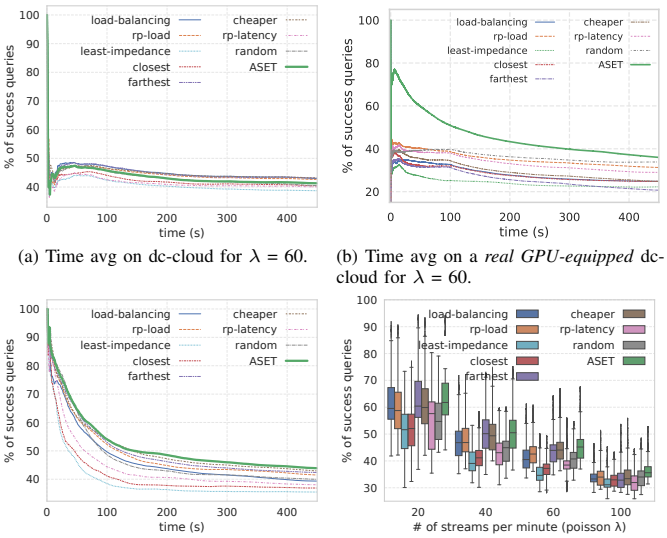
Network topology assesses scheduling performance in a large ISP network topology using a cloud-to-edge topology with clusters of different sizes deployed hierarchically. The topology provides computation capabilities at different layers, including network access, central offices, operator data centers, and remote cloud. The simulations focus on three scenarios: dc-cloud, co-dc-cloud, and full-edge topology. The evaluation assumes 5G radio access technology with network/transmission delays ranging from a few milliseconds to ten milliseconds.

Requests workload are generated using a Poisson distribution, with each generator running on average clients per time. Each client requests a stream with randomized characteristics, including frame rate, latency, model accuracy, and frame sizes. Modeling metrics capture realistic queries characteristics, with a generator causing almost 1000 queries per second on the antenna.

Computing clusters and model variant assumes a hardware distribution across clusters, with computing capabilities increasing from the access network to the cloud. It focuses on DNN models for object detection, a challenging and computation-intensive inference service. The prototype profiles MobileNet-SSD, Yolo-v3, and Tinyyolo-v2 models with CPU and GPU variants, batch sizes, and resource allocation. Simulations are run on top of these topologies, scaling workers up to resource saturation.

B. Experimental Results

We compare the performance of the baseline policies distinguishing results for different applications [21], [22]. As a performance metric we consider the percentage of queries that are successfully processed by the system satisfying the application QoS requirements. Figure 4 shows results of multiple runs with dynamic client rate. Results suggest that there is no one-size-fits-all policy, as various applications may benefit



(a) Time avg on dc-cloud for $\lambda = 60$.

(b) Time avg on a *real GPU-equipped* dc-cloud for $\lambda = 60$.

(c) Time avg on co-dc-cloud for $\lambda = 60$.

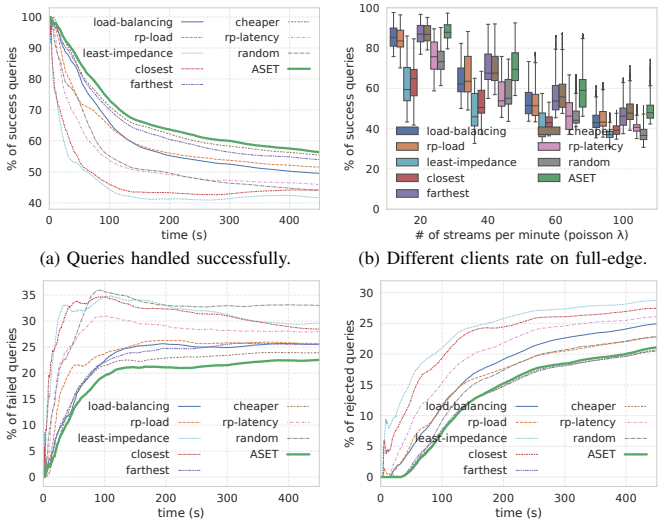
(d) Different clients rate on co-dc-cloud.

Fig. 5: Performance of ASET compared to static policies for the dc-cloud topology on (a) simulated and (b) real-deployment scenario, and (cd) the co-dc-cloud topology.

differently from each policy. In the following, we compare the performance of the ASET RL scheduling approach with the performance of static policies, evaluating the benefits it can introduce in the various scenarios.

Cloud deployment. When all the available resources are located in a few centralized clusters, the various static policies have small differences in performance and a dynamic approach has little room for improvement. Results for the dc-cloud topology are shown in Figures 5ab. In particular, Figure 5a plots, for every moment of the simulation (time axis), the percentage of queries that are handled successfully, averaging multiple runs with different workloads. The graph shows that, for this topology, ASET does not improve over static policies, and it even performs worse for higher lambdas. We then replicated the dc-cloud experiments on a real deployment scenario configured with the RTX-2070-SUPER GPUs, reporting the percentage of successful queries over time in Figure 5b. As visible from the figure, we can observe that *(i)* ASET performs better than the static policies and that *(ii)* the obtained pattern is consistent with the simulated results (Figure 5a). This result is crucial and validates the strong correlation between the emulated and real environment, thus validating the accuracy of the simulation and subsequent observations. Figures 5cd shows that moving some resources to Central Offices (co-dc-cloud topology) makes a huge difference: in general, all the policies achieve a higher success ratio on this configuration (Figure 5c), as they can exploit the additional lower latency spots, and the higher level of distribution gives to ASET a certain margin of improvement.

Edge deployment. The results so far suggest that a good distribution of computing resources is a key factor to improve against static scheduling policies. As shown in Figure 6, the benefits of using a dynamic scheduling approach become more concrete in a full-edge topology, where resources are better



(a) Queries handled successfully. (b) Different clients rate on full-edge.

(c) Queries delivered with QoS violations. (d) Queries rejected for lack of resources.

Fig. 6: Performance of ASET compared with static policies for the full-edge topology. (a) (c) and (d) show averages of multiple runs with $\lambda = 60$.

distributed on multiple smaller clusters in different locations. In fact, Figure 6a shows that the dynamic approach of ASET is able to achieve a constant improvement over any static policy, with a higher success ratio over time. In particular, Figures 6cd show that, while maintaining the same rejection rate as the best static-policy, ASET effectively reduces the number of queries that are handled violating one or more QoS requirements. Moreover, Figure 6b shows that an ASET agent trained only for $\lambda = 60$ can also generalize on different requests rate, even supporting a load of more than 1600 queries per second ($\lambda = 100$) on a single antenna.

VI. Conclusions

The paper introduces ASET, an adaptive algorithm based on Reinforcement Learning, for scheduling inference workloads at the network edge. It solves the problem of exploiting scattered clusters of resources to serve inference queries from multiple edge applications. ASET optimizes the binding between inference stream requests and available DNN models, maximizing throughput and satisfying inference accuracy and end-to-end delay requirements. The approach was evaluated over a large ISP network topology and heterogeneous edge applications, showing that ASET improves performance compared to static policies when resources are deployed across the entire edge-cloud infrastructure.

Acknowledgment

We thank Carlos Segura, Diego Perino and Aravindh Raman for useful discussions along meetings. This work has been partially funded by the EU Horizon Europe research and innovation programme under Grant Agreement No. 101070516 (NebulOuS) and No. 101070473 (FLUIDOS). This text reflects only the authors' view, and the Commission is not responsible for any use that may be made of the information it contains.

References

[1] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *ACM Special Interest Group on Data Communication*, 2019.

[2] 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.

[3] J. Chen and X. Ran, "Deep learning with edge computing: A review," *IEEE*, 2019.

[4] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE Symposium on Edge Computing*, 2018.

[5] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 428–440.

[6] T. Zheng, J. Wan, J. Zhang, and C. Jiang, "Deep reinforcement learning-based workload scheduling for edge computing," *Journal of Cloud Computing*, vol. 11, no. 1, p. 3, 2022.

[7] Y. Li, X. Zhang, T. Zeng, J. Duan, C. Wu, D. Wu, and X. Chen, "Task placement and resource allocation for edge machine learning: a gnn-based multi-agent reinforcement learning paradigm," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[8] G. Bartolomeo, M. Yosofie, S. Bäurle, O. Haluszczynski, N. Mohan, and J. Ott, "Oakestra: A lightweight hierarchical orchestration framework for edge computing," in *USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 215–231.

[9] J. Yang, L. Bao, W. Liu, R. Yang, and C. Q. Wu, "On a meta learning-based scheduler for deep learning clusters," *IEEE Transactions on Cloud Computing*, 2023.

[10] S. Khare, H. Sun, J. Gascon-Samson, K. Zhang, A. Gokhale, Y. Barve, A. Bhattacharjee, and X. Koutsoukos, "Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs," in *IEEE Symposium on Edge Computing*, 2019.

[11] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[12] M. Cheong, H. Lee, I. Yeom, and H. Woo, "Scarl: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153 432–153 444, 2019.

[13] H. Lee, J. Lee, I. Yeom, and H. Woo, "Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling," *IEEE Access*, vol. 8, pp. 185 570–185 583, 2020.

[14] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.

[15] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, 2015.

[16] J. Long, M. Dong, K. Ota, and A. Liu, "A green tdma scheduling algorithm for prolonging lifetime in wireless sensor networks," *IEEE Systems Journal*, 2015.

[17] C. Cicconetti, M. Conti, and A. Passarella, "An architectural framework for serverless edge computing: design and emulation tools," in *IEEE International Conference on Cloud Computing Technology*, 2018.

[18] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, 1957.

[19] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv:2005.01643*, 2020.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, 2013.

[21] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017.

[22] A. Cartas, M. Kocour, A. Raman, I. Leontiadis, J. Luque, N. Sastry, J. Nuñez-Martinez, D. Perino, and C. Segura, "A reality check on inference at mobile networks edge," in *2nd International Workshop on Edge Systems, Analytics and Networking*, 2019.