Joint DNN Model Deployment, Selection, and Configuration for Heterogeneous Inference Services Toward Edge Intelligence

Hebin Huang, Junbin Liang, and Geyong Min, *Member, IEEE*

***Abstract*—Edge intelligence is an emerging paradigm in edge computing that deploys Deep Neural Network (DNN) models on edge servers with limited storage and computation capacities to provide inference services for high mobility and real-time applications, such as autonomous driving or smart surveillance, with varying accuracy and delay requirements. Adapting application configurations (e.g., image resolution or video frame rate) while selecting different DNN models and deployment locations can provide high-accuracy, low-delay inference services that meet user requirements. However, the configurations and DNN models of various inference services are highly heterogeneous. As balancing inference accuracy, resource cost, and delay is a multi-objective programming problem, it is a great challenge to obtain the optimal solution. To address this challenge, we propose a novel online framework to jointly optimize the configuration adaption, DNN model selection, and deployment for heterogeneous inference services. Specifically, we first formulate this joint optimization problem as an integer linear programming problem and prove it is NP-hard. Then, we further model the problem as a Partial Observable Markov Decision Process (POMDP) and solve it by developing a Heterogeneous-Agent Reinforcement Learning (HARL) based algorithm, named Heterogeneous Inference Service ProvidER (HISPER). It allows agents to have different action spaces corresponding to different types of configurations and DNN models. Finally, extensive experiments demonstrate that the proposed algorithm outperforms other state-of-the-art counterparts.**

***Index Terms*—Edge intelligence, DNN inference, mobile edge computing, service deployment, reinforcement learning.**

I. Introduction

RECENTLY, Deep Neural Network (DNN) has been widely applied in extensive Artificial Intelligence (AI) applications, such as autonomous driving [1], [2], smart city, and digital twin [3]. Users can submit their data to an Inference Service (IS) via a nearby Access Point (AP), which then processes it through a DNN model and generates results, namely DNN inference. Performing DNN inference tasks involves a large number of multiplication and accumulation operations, which highly demand computational resources [4]. Besides, some ISs also require real-time responses. High computing delay can negatively impact user experience or even lead to service faults.

To support compute-intensive DNN inference tasks and provide low-delay ISs, edge intelligence has emerged as an enabling paradigm to pave the last mile of AI [5]. It combines AI, network function virtualization, and Mobile Edge Computing (MEC) to push various resources from cloud to edge servers, deploying DNN models on edge servers to provide ISs closer to users [6]. In this manner, edge intelligence enables mobile devices with high mobility to perform complex DNN inference in real time. For example, in vehicle networking, Generative Adversarial Networks (GANs) have been used to predict the movement of vehicles [7]. Although such deployment can reduce computation delay, some DNN models (e.g., GPT [8] and ViT [9]) have billions of weights, which will consume tremendous storage resources. Applying model compression methods, such as model weight quantization and pruning, to DNN models can effectively reduce the number of weights, thereby generating model variants of different sizes to accommodate varying resource constraints. It should be emphasized, however, that these compressed model variants typically exhibit reduced inference accuracy due to the parameter reduction. Consequently, an IS must provide multiple DNN models of varying scales to offer flexible resource-accuracy profiles [10]. This allows for the dynamic selection of an appropriate model variant for each inference request, enhancing the overall inference accuracy of the edge network without exceeding resource constraints [11]. Furthermore, some applications like video analytics have configurations (e.g., frame rate, resolution) that affect inference accuracy and delay. These configurations should also be dynamically adapted through co-optimization with model selection to balance accuracy, resource consumption, and delay.

As shown in Fig. 1, when an edge server receives multiple inference requests with different ISs, it first adjusts configurations and selects DNN models for each request, then deploys the selected DNN models to suitable edge servers. However, since edge servers are heterogeneous and geographically distributed, their computing performance and network bandwidth vary across locations. Consequently, identical inference requests may experience varying processing times on different edge servers. To meet service delay requirements, both

Received 24 September 2024; revised 19 May 2025; accepted 4 July 2025. Date of publication 10 July 2025; date of current version 3 October 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62362005, and in part by the Key Research and Development Program of Guangxi under Grant AD25069071. Recommended for acceptance by E. Ngai. *(Corresponding author: Junbin Liang.)*

Hebin Huang and Junbin Liang are with the School of Electrical Engineering, Guangxi Key Laboratory of Multimedia Communications and Network Technology, School of Computer and Electronic Information, Guangxi University, Nanning 530004, China (e-mail: liangjb@gxu.edu.cn).

Geyong Min is with the Department of Computer Science, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: g.min@exeter.ac.uk).

Digital Object Identifier 10.1109/TMC.2025.3586793

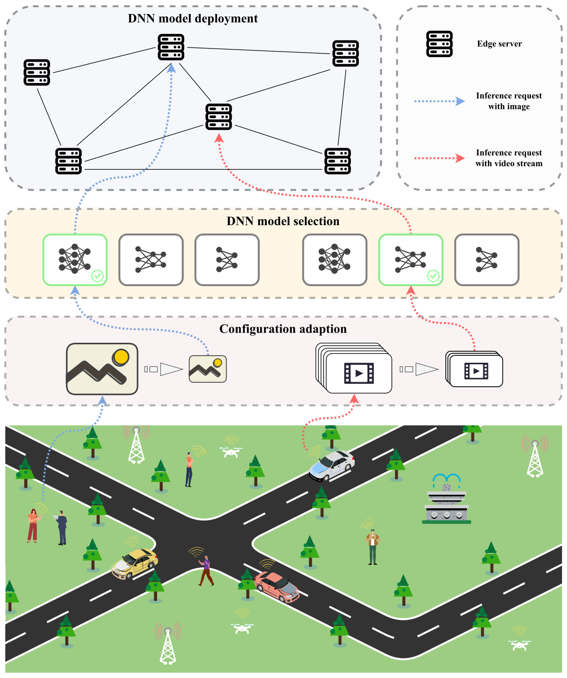


Fig. 1. Configuration adaption, DNN model selection, and deployment.

configuration adaption and DNN model selection must account for the deployment location of the selected DNN model. Conversely, different combinations of configurations and DNN models exhibit variations in inference accuracy and delay, which in turn demand different computing resources. Therefore, deploying the selected DNN model on an appropriate edge server requires considering computation delay, data transmission delay, and resource quota under the specific configuration and DNN model. Given these interdependencies, configuration adaption, DNN model selection, and deployment are tightly coupled, and highly valuable for jointly optimizing.

However, the joint optimization of configuration adaptation, model selection, and deployment faces two key challenges. First, the heterogeneity in configurations and DNN models across different ISs, combined with differing edge server computing capabilities, leads to a large solution space for this joint optimization problem. It is difficult to find a feasible solution to this problem in a short time. Second, designing an efficient heuristic algorithm for heterogeneous IS provisioning requires a deep understanding of the specific resource demands of different configurations and DNN models, which necessitates extensive experience with these ISs and limits the algorithm's scalability.

To this end, in this paper, we consider a multiple ISs scenario consisting of multiple edge servers and propose a novel online optimization algorithm named Heterogeneous Inference Service ProvidER (HISPER) for providing flexible heterogeneous ISs in edge networks. The HISPER can learn to make joint online decisions for multiple inference requests and perform suitable trade-offs between inference accuracy, cost, and delay. Our optimization problem aims to maximize inference accuracy while minimizing resource cost and inference delay. Considering the unpredictable distribution of inference requests and dynamic resource fluctuations in edge networks, the system states are partially observable. Each edge server only maintains local request information and lacks knowledge of other servers' request states. This condition naturally fits the Partially Observable Markov Decision Process (POMDP), which effectively manages partial observability and uncertainty. Therefore, we model our joint optimization problem as a POMDP, incorporating a continuous state space, and develop a specialized maximum entropy Heterogeneous-Agent Reinforcement Learning (HARL) algorithm to solve the problem efficiently.

To evaluate the performance of our algorithm, we first collect the performance data of different DNN models in two inference services under different configurations. Based on the results, we conduct simulation experiments to evaluate the performance of HISPER. The superiority of our algorithm is demonstrated in comparison with four other state-of-the-art algorithms.

Our main contributions are summarized as follows:

- We formulate the joint configuration adaption, DNN model selection, and deployment problem as an integer linear programming problem for providing heterogeneous inference services, and model it as a POMDP to capture the system dynamics in the edge networks.
- We solve the POMDP by proposing a novel maximum entropy heterogeneous agent reinforcement learning algorithm, where the action space of all agents is heterogeneous, named HISPER. Specifically, we apply Hybrid Reward Architecture (HRA) and auto-regressive policy in the maximum entropy HARL framework to help agents quickly learn effective joint decisions and perform good trade-offs between inference accuracy, resource cost, and inference delay.
- We measure inference accuracy and data size for two real-world inference services, each evaluated with multiple DNN models and configurations. The results reveal the impact of various model variants and configurations on inference accuracy, demonstrating significant performance differences.
- Simulation experiments are conducted to evaluate the performance of the HISPER by using the performance data in a real-world edge network topology. Compared to the state-of-the-art counterpart, our algorithm attains the highest accuracy with low cost and delay.

The rest of the paper is organized as follows. The related work is discussed in Section II. Section III describes the system model and problem formulation. Then, we model the problem as a POMDP and propose the HISPER algorithm to solve in Sections IV and V, respectively. In Section VI, simulation experiments are performed. Finally, Section VII concludes the paper.

II. Related Work

Supporting ISs in edge networks has attracted the attention of many scholars, with the aim of providing high accuracy and low delay or resource-efficient DNN inference tasks in edge networks. The related literature can be summarized in two categories: configuration adaption-based IS provisioning and DNN model selection-based IS provisioning.

A. Configuration Adaption-Based IS Provisioning

By adapting application configurations, the request data can be flexibly resized to balance resource usage and inference accuracy. Against this backdrop, researchers have explored optimization strategies from multiple perspectives.

Some studies have focused on constructing comprehensive multi-factor optimization frameworks. For instance, Ran et al. [12] proposed a deep learning framework to maximize the number and the accuracy of each frame by adjusting five degrees: resolution, DNN model variants, computation locations, video bitrate, and frame rate. Similarly, Wang et al. [13] jointly optimized configuration (frame rate and resolution) adaption and bandwidth allocation for multiple video streams, aiming to maximize accuracy and minimize energy consumption under latency constraint. Their algorithm can work online without requiring future information, but it may lead to suboptimal performance when handling highly dynamic video content. Fan et al. [14] proposed a model incorporating DNN deployment, data size control, task offloading, and resource allocation to minimize total costs while ensuring queue stability and accuracy, using a tunable penalty mechanism to limit deployment adjustments. However, the parameters of this mechanism must be carefully tuned to maintain system flexibility. Furthermore, Zhao et al. [15] designed *EdgeAdaptor* to jointly optimize configuration adaption, model selection, and resource provisioning in large-scale edge networks, to minimize long-term resource costs and accumulated accuracy degradation under real-time latency constraints.

When adapting configurations for an inference request, searching for the best configuration could increase resource overhead. To address this, Jiang et al. [16] developed *Chameleon* to reduce the configuration search space from exponential to linear through three key observations: temporal correlation, cross-camera correlation, and independence of configurations. However, these observations struggle to handle abrupt video content changes and depend on inter-camera similarity, which may lead to suboptimal decisions. Yang et al. [17] demonstrated in their simulation that computational demand grows exponentially with video quality and proposed a gradient estimation method that integrates video quality control and resource allocation to gradually approximate optimal configurations. However, their method only considers single-camera scenarios, which limits its applicability to multi-camera environments. Xiao et al. [18] pointed out that a high compression ratio in edge vision systems reduces data transmission and bandwidth usage, but it may lose critical visual information, lowering inference accuracy. They employed a spatial attention mechanism to compress key visual regions differentially, optimizing communication efficiency and inference accuracy through adaptive compression offloading and resource allocation.

In particular domains, parameter-accuracy relationships were also investigated. Wu et al. [4] showed that the inference accuracy grows sublinearly with the sampling rate in fault diagnosis. They developed a deep reinforcement learning method to jointly optimize sampling, offloading, and resource allocation, minimizing service delay while ensuring accuracy requirements.

B. DNN Model Selection-Based IS Provisioning

An IS has DNN models with different performances, selecting a suitable one can reduce inference delay while achieving higher accuracy. To improve user experience in inference service provisioning, Lu et al. [19] employed a multi-armed bandit algorithm to select compressed DNN models automatically. They leveraged a machine learning model to approximate the user's Quality of Experience (QoE) and trained the model based on users' QoE feedback. Xie et al. [11] pointed out that DNN model selection is performed in a best-effort manner, which causes unfairness between AI applications. They modeled the problem as a Nash bargaining game with cooperative game theory to ensure efficient and fair DNN model selection. Li et al. [20] addressed DNN model deployment in 6 G networks, where edge nodes can execute inference tasks locally or offload them to the cloud or other nodes. They aimed to minimize computation time and energy consumption while maximizing inference accuracy. Wang et al. [21] proposed a deployment method that considered DNN model variants and allowed for the coexistence of multiple inference paradigms, such as model routing, model cascading, and model splitting, to achieve a better trade-off between system accuracy, service scale, and deployment cost. Xiao et al. [22] explored optimal DNN model deployment between cloud and edge, focused on accuracy, latency, and energy consumption.

C. Difference Between This Paper and Existing Work

In the studies on adapting configuration, they focused on how to allocate resources for DNN models and rarely considered where DNN models are deployed. Conversely, in the papers on DNN model selection, although some works deployed DNN models after selection, they did not adjust configurations to better balance accuracy and resource cost. Besides, these works mainly used heuristic algorithms to adapt configurations and select DNN models. However, manually designing an evaluation function is required to evaluate the impact of different configurations and DNN models on inference accuracy, resource cost, and delay. Additionally, the heuristic algorithm's operating time is proportional to the number of configurations and DNN models, which results in a higher runtime in multi-service and multi-model scenarios. Adopting learning-based approaches to address multi-objective optimization problems in edge networks can improve Quality of Service (QoS) [23]. However, in the works that employed reinforcement learning for their algorithm, they only considered the configurations of IS to be the same, which is unsuitable for heterogeneous ISs. Therefore, we further consider the heterogeneity of ISs and propose a heterogeneous agent reinforcement learning-based method to enhance the quality of inference services for users.

The work most related to this paper is [24], in which Zhang et al. proposed a scheme named *Novas* for video analytics. *Novas* can adapt the configuration automatically when video content changes and schedule video analysis requests to edge servers. Our work is different from [24] in at least the following two aspects. First, we consider a scenario where different ISs have distinct configurations. By jointly adapting these configurations, we can select more suitable ones for each IS while mitigating

TABLE I
MAIN NOTATIONS

Notation	Definition
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	The MEC network with the set of edge servers $\mathcal{V}$ and the set of links $\mathcal{E}$
$e_{k,k'}$, $w_{k,k'}$	The link between edge server v_k and $v_{k'}$ with bandwidth $w_{k,k'}$
A_k, B_k	The computing and memory resource of edge server v_k
$\mathcal{S}$	The set of all ISs
$\mathcal{C}_s$, $\mathcal{M}_s$	The set of configurations and DNN models for IS s
h_s	The number of configurations for IS s
c_i^s	The tuple of configuration combination with length h_s for IS s
m_j^s	The DNN model for IS s
A_j^s, B_j^s	The computing and memory requirements of DNN model m_j^s for IS s
$\Phi(c_i^s, m_j^s)$	The function to present the accuracy of DNN model m_j^s after adapting to the configuration c_i^s for IS s
$\mathcal{D}_u(t)$	The data size of inference request $\gamma_u(t)$ after adapting to configuration c_i^s in time slot t
$\Gamma(t)$	The set of all inference requests in time slot t
$\gamma_u(t)$	The inference request in time slot t
$d_u^{\gamma}(t)$, $data_u^{\gamma}(t)$	The delay requirement and data size of inference request $\gamma_u(t)$ in time slot t
$d_u^{acce}(t)$, $d_u^{tran}(t)$, $d_u^{comp}(t)$	The access delay, transmission delay, and computation delay of inference request $\gamma_u(t)$ in time slot t
$\Psi_u^{rou}(t)$, $\Psi_u^{comp}(t)$	The routing cost and computing cost of inference request $\gamma_u(t)$ in time slot t
$\Psi^{mem}(t)$	The memory cost of all deployed DNN models in time slot t
$Acc(t)$, $d^{total}(t)$, $\Psi(t)$	The total inference accuracy, delay, and cost of all inference requests in time slot t
$\alpha_{u,i}^s(t)$	Whether adapt to configuration c_i^s for inference request $\gamma_u(t)$ in time slot t
$\beta_{u,j}^s(t)$	Whether select DNN model m_j^s for inference request $\gamma_u(t)$ in time slot t
$\zeta_{u,k}^{s,j}(t)$	Whether the selected DNN model m_j^s is deployed on edge server v_k for inference request $\gamma_u(t)$ in time slot t

inter-IS resource competition. This approach enhances the quality of service for inference requests. Second, we employ a HARL algorithm to cope with the heterogeneity of ISs and the dynamics of edge networks, without designing a configuration adaption algorithm for each IS. Meanwhile, compared to the heuristic algorithm, our algorithm can make decisions faster.

III. System Model and Problem Description

In this section, we first present the system model and notations, then formulate the joint application configuration adaption, DNN model selection, and deployment problem as an integer linear programming problem, named Joint Adaption, Selection, and Deployment Problem (JASDP). Table I lists the main notations in our paper.

A. Network Model

We model a resource-constrained edge intelligence network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_k | 1 \leq k \leq |\mathcal{V}|\}$ represents the set of heterogeneous edge servers with the number of $|V|$, and $\mathcal{E}$ denotes the set of physical links between edge servers. Each edge server has distinct computation and memory capabilities, denoted by A_k Tera FLoating-point Operations Per Second (TFLOPs/S) and B_k MB memory resources, respectively. To provide high-accuracy ISs with low delay and resource cost to users, the heterogeneous resources of edge servers must be carefully considered to determine the optimal configurations, DNN model variants, and deployment locations, increasing the dimensionality and complexity of the solution space. Each edge server is equipped with an Access Point (AP), enabling users to connect via their nearest AP. Adjacent edge servers are connected via high-speed links, where the link between $v_k, v_{k'} \in \mathcal{V}$ is denoted as $e_{k,k'}$ with bandwidth $w_{k,k'}$ Mbps

The hardware resources of each edge server have been virtualized into a set of containers or virtual machines to run multiple DNN models to provide ISs for users. We also consider a time-slotted model, where edge servers receive inference requests at the beginning of each time slot. In time slot $t\,(1 \leq t \leq T)$, the edge servers that receive inference requests will adapt configurations, select DNN models, and deploy the selected DNN models.

B. Inference Service Model

We use $\mathcal{S} = \{s | 1 \leq s \leq |\mathcal{S}|\}$ to denote all types of ISs. When an edge server receives one or more inference requests sent by edge applications, it first adjusts application configuration to resize the request data for a lower data transmission delay or higher inference accuracy. Each type of IS has different configurations, for service $s \in \mathcal{S}$, let h_s denote the number of configurations, and we use a set $\mathcal{C}_s = \{c_i^s | 1 \leq i \leq |\mathcal{C}_s|\}$ to represent different combinations of configuration. Each c_i^s is a tuple of length h_s. For example, suppose there are two configurations ($h_s = 2$) in video analytics service which include frame rate (e.g., from 10 fps to 30 fps) and resolution (e.g., from 320 p to 1080 p), then $\mathcal{C}_s = \{(10\text{ fps},320\text{ p}), (10\text{ fps},480\text{ p}), \ldots, (30\text{ fps},1080\text{ p})\}$. This definition allows us to model different IS configurations consistently.

In addition, we use another set $\mathcal{M}_s = \{m_j^s | 1 \leq j \leq |\mathcal{M}_s|\}$ to present DNN models for IS $s \in \mathcal{S}$, which contains DNN model variants with different sizes. For instance, VGG [25] and ResNet [26] are popular DNN models in image recognition services. The VGG is implemented by convolutional neural networks, and VGG variants include VGG-16, VGG-19, etc. While ResNet is implemented by residual neural networks with variants including ResNet-18, ResNet-34, etc. For each DNN model m_j^s, we denote the computing requirements of m_j^s by A_j^s Tera FLoating-point Operations Per seconds (TFLOPs) and its memory requirements by B_j^s MB. We also assume that each DNN model m_j^s is running in a lightweight container on an edge server, named the DNN model instance.

An inference request includes the required IS and corresponding delay requirements. Let $\Gamma(t) = \{\gamma_u(t) | 1 \leq u \leq |\Gamma(t)|\}$ be a collection of all inference requests in time slot t, each $\gamma_u(t)$ is a 4-tuple $(v_u^\gamma(t), s_u^\gamma(t), d_u^\gamma(t), data_u^\gamma(t))$. Specifically, $v_u^\gamma(t) \in \mathcal{V}$ represents an edge server that the user connects to, defined as the user's local server, and $s_u^\gamma(t) \in \mathcal{S}$ is an IS type required by the user. Notations $d_u^\gamma(t)$ and $data_u^\gamma(t)$ are the user's requirements on inference delay and the amount of requested data, respectively. We further assume that the user device will connect to the closest AP when a user is in an overlapping area covered by multiple APs.

C. Decision Variables

In JASDP, we define three decision variables to determine: which configuration should be adapted to, which DNN model should be selected, and where to deploy the selected DNN model for an inference request, respectively.

1) Configuration Adaption: This decision variable decides which configuration should be adapted to. For inference request $\gamma_u(t)$ which requires service $s \in \mathcal{S}$ in time slot t, we use a binary decision variable $\alpha_{u,i}^s(t) \in \{0, 1\}$ to indicate whether it will adapt to configuration c_i^s. If c_i^s is selected to adapt for $\gamma_u(t)$, we let $\alpha_{u,i}^s(t) = 1$; otherwise, $\alpha_{u,i}^s(t) = 0$.

2) DNN Model Selection: This decision variable decides which DNN model should be selected. For inference request $\gamma_u(t)$ which requires service $s \in \mathcal{S}$ in time slot t, we use a binary decision variable $\beta_{u,j}^s(t) \in \{0, 1\}$ to indicate whether DNN model m_j^s is selected. If m_j^s is selected for $\gamma_u(t)$, we let $\beta_{u,j}^s(t) = 1$; otherwise, $\beta_{u,j}^s(t) = 0$.

3) DNN Deployment Decision: This decision variable decides the deployment location of the selected DNN model. Let binary decision variable $\zeta_{u,k}^{s,j}(t)$ indicate whether edge server v_k deploys the selected DNN model m_j^s for inference request $\gamma_u(t)$ in time slot t. If v_k is assigned to deploy DNN model, we let $\zeta_{u,k}^{s,j}(t) = 1$; otherwise, $\zeta_{u,k}^{s,j}(t) = 0$. In addition, we allow different inference requests to share the same DNN model in an edge server and stipulate that only one DNN model of the same instance can be deployed on the same edge server. Therefore, for a specific DNN model $m_j^s \in \mathcal{M}_s$ and an edge server $v_k \in \mathcal{V}$, we have $\sum_{u\in|\Gamma(t)|} \beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t) \leq 1$.

D. Inference Accuracy and Delay Model

Adjusting configurations can affect data quality and size, further influencing inference accuracy and delay. To model this influence, we define a function $\Phi(c_i^s, m_j^s)$ to present the accuracy of DNN model m_j^s after adjusting to configuration c_i^s for IS s. The inference accuracy for an inference request $\gamma_u(t)$ is $Acc_u(t) = \Phi(c_i^s, m_j^s) \cdot \alpha_{u,i}^s(t) \cdot \beta_{u,j}^s(t)$. The total accuracy of all inference requests in time slot t can be calculated as

$$Acc(t) = \sum_{u\in|\Gamma(t)|} Acc_u(t) \tag{1}$$

After applying configuration c_i^s, the data size of inference request $\gamma_u(t)$ becomes

$$\mathcal{D}_u(t) = \sum_{s\in|\mathcal{S}|} \sum_{i\in|\mathcal{C}_s|} \eta_i^s \cdot data_u^\gamma(t) \cdot \alpha_{u,i}^s(t), \tag{2}$$

where $0 < \eta_i^s \leq 1$ is a scaling factor used to adjust the size of request data, determined by different configurations

To calculate inference delay, we consider three delays: access delay $d_u^{acce}(t)$, data transmission delay $d_u^{tran}(t)$, and computation delay $d_u^{comp}(t)$.

The access delay is incurred when a user makes an inference request from their local server. At time slot t, the access delay to edge server $v_u^\gamma(t)$ can be expressed as

$$d_u^{acce}(t) = \frac{\mathcal{D}_u(t)}{\rho_u(t)} \tag{3}$$

where $\rho_u(t)$ is wireless uplink transmission rate from local server $v_u^\gamma(t)$. When the selected DNN model has not been deployed in the user's local server, the request data must be transmitted to the edge server $v_k \in \mathcal{V}$ where the DNN model is deployed. The transmission delay of $\gamma_u(t)$ under configuration c_i^s can be calculated as

$$d_u^{tran}(t) = \begin{cases} 0, & v_u^\gamma(t) = v_k, \\ \sum_{e\in P_{u,k}} \frac{\mathcal{D}_u(t)}{w_e}, & v_u^\gamma(t) \neq v_k. \end{cases} \tag{4}$$

where $P_{u,k}$ is a set of the shortest paths from $v_u^\gamma(t)$ to v_k and w_e represents the bandwidth of path e.

We assume that each inference request arrives at the edge server at the start of each time slot and can be finished within the same slot. In an edge server, all the received inference requests in the current time slot need to be processed sequentially and the requests are handled in the order they were received. Thus, the server workload of each inference request is different since they need to wait for the completion of previous requests. Let $h_{u,k}^{req}(t)$ be the total computation resources needed by previous requests, and $h_k^{sys}(t)$ be the system workload in v_k, the total server workload of v_k in time slot t is $H_{u,k}(t) = h_{u,k}^{req}(t) + h_k^{sys}(t)$. After finishing all inference requests in a time slot, all the deployed DNN model instances will be released.

Some ISs, such as object tracking, whose data type is video streaming, their inference process is performed frame-by-frame. Adjusting the frame rate can influence the number of inferences in these ISs. We use I_i^s to denote the number of inferences under configuration c_i^s. The computation delay for inference request $\gamma_u(t)$ is calculated as

$$d_u^{comp}(t) = \sum_{s\in|\mathcal{S}|} \sum_{i\in|\mathcal{C}_s|} \sum_{j\in|\mathcal{M}_s|} \sum_{k\in|\mathcal{V}|} \big(A_j^s I_i^s \alpha_{u,i}^s(t) + H_{u,k}(t)\big) \frac{\beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t)}{A_k}, \tag{5}$$

where A_j^s is the computing resource required for DNN model m_j^s, and A_k is the computing capacity of edge server v_k.

The inference delay of inference request $\gamma_u(t)$ in time slot t can be calculated as

$$d_u^{total}(t) = d_u^{acce} + d_u^{tran}(t) + d_u^{comp}(t) \tag{6}$$

And the total inference delay $d_u^{total}(t)$ of all inference requests is

$$d^{total}(t) = \sum_{u\in|\Gamma(t)|} d_u^{total}(t). \tag{7}$$

E. Cost Model

In resource-limited edge networks, routing and deployment costs are the major factors affecting IS provisioning [6]. In our problem, the cost model $\Psi(t)$ comprises routing cost $\Psi_u^{rou}(t)$ for data transmission and deployment cost, where the latter accounts for the memory cost $\Psi^{mem}(t)$ for DNN deployment and computing costs $\Psi_u^{comp}(t)$ for processing inference requests.

The routing cost $\Psi_u^{rou}(t)$ is defined as the cost of routing request data under configuration c_i^s from the user's local server $v_u^\gamma(t)$ to the edge server v_k where the selected DNN model is deployed in time slot t. Let ψ_e^{rou} be the unit routing cost per Mbps in link e. The routing cost is calculated as:

$$\Psi_u^{rou}(t) = \sum_{e\in P_{u,k}} \psi_e^{rou}\mathcal{D}_u(t) \tag{8}$$

It increases with both the distance between $v_u^\gamma(t)$ and v_k, and the data size of inference requests.

Deploying a DNN model on an edge server consumes computing and memory resources. Since each DNN model is deployed on the server at most once in a time slot, we first calculate the memory cost of all the deployed DNN models:

$$\Psi^{mem}(t) = \sum_{s\in|\mathcal{S}|}\sum_{j\in|\mathcal{M}_s|}\sum_{k\in|\mathcal{V}|}\min\left(1, \sum_{u\in|\Gamma(t)|}\beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t)\right)\psi^{mem}B_j^s \tag{9}$$

where ψ^{mem} is the unit memory cost per MB. Moreover, the computing cost of an inference request $\gamma_u(t)$ is calculated by the duration of processing inference requests and the amount of requested data. Let ψ^{comp} be the unit computing cost per second, the computing cost in server v_k during time slot t is

$$\Psi_u^{comp}(t) = \sum_{s\in|\mathcal{S}|}\sum_{j\in|\mathcal{M}_s|}\sum_{k\in|\mathcal{V}|}\psi^{comp}\frac{A_j^s}{A_k}\beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t) + \psi^{mem}\mathcal{D}_u(t) \tag{10}$$

The total cost of all inference requests in time slot t is

$$\Psi(t) = \sum_{u\in|\Gamma(t)|}\left(\Psi_u^{rou}(t) + \Psi_u^{comp}(t)\right) + \Psi^{mem}(t) \tag{11}$$

In practice, these costs are tightly coupled. For instance, deploying DNN models closer to users reduces both transmission delay $d_u^{tran}(t)$ and routing cost $\Psi_u^{rou}(t)$, but increases memory cost Ψ^{mem}, since it fails to leverage other deployed DNN models efficiently. Additionally, selecting lightweight DNN models or adapting to lower configurations to provide ISs reduces Ψ^{mem} and $\Psi_u^{comp}(t)$, but this sacrifices inference accuracy, potentially requiring retransmissions of misclassified data and indirectly increasing $\Psi_u^{rou}(t)$. Considering a vehicle recognition service, adjusting the resolution or frame rate of video streams directly impacts the data size of an inference request, thereby affecting both $\Psi_u^{rou}(t)$ and $\Psi_u^{comp}(t)$. Meanwhile, multiple DNN variants are available for vehicle recognition, each with different memory and computing requirements, influencing Ψ^{mem} and $\Psi_u^{comp}(t)$. Additionally, the deployment location of the DNN model directly determines the $\Psi_u^{rou}(t)$. When processing inference requests, it is crucial to carefully consider the coupling relationships among these three types of costs.

F. Problem Formulation

In this section, we formulate the JASDP as an integer linear programming (ILP) problem, whose optimization objective is to

$$\max \sum_{t=1}^{T}\left(Acc(t) - d^{total}(t) - \Psi(t)\right) \tag{12}$$

$$\text{s.t.} \sum_{u\in|\Gamma(t)|}\beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t) \le 1, \forall t, \forall s, \forall j, \forall k \tag{13}$$

$$\sum_{s\in|\mathcal{S}|}\sum_{j\in|\mathcal{M}_s|}\min\left(1, \sum_{u\in|\Gamma(t)|}\beta_{u,j}^s(t)\zeta_{u,k}^{s,j}(t)\right)B_j^s + \sum_{u\in|\Gamma(t)|}\mathcal{D}_u(t) \le B_k, \forall t, \forall k \tag{14}$$

$$d_u^{total} \le d_u^\gamma, \forall u \tag{15}$$

$$\alpha_{u,i}^s(t) \in \{0,1\}, \forall t, \forall u, \forall s, \forall i \tag{16}$$

$$\beta_{u,j}^s(t) \in \{0,1\}, \forall t, \forall u, \forall s, \forall j \tag{17}$$

$$\zeta_{u,k}^{s,j}(t) \in \{0,1\}, \forall t, \forall u, \forall s, \forall k \tag{18}$$

Formula (12) is the maximization objective of the joint optimization problem. Constraint (13) indicates that the same type of DNN model can only be deployed once in an edge server. Constraint (14) ensures that the memory occupied by inference request and DNN model in an edge server does not exceed the server's memory capacity. Constraint (15) ensures that user requirements on inference delay must be satisfied.

G. Problem Analysis

Theorem 1: The JASDP in a resource-constrained edge network is NP-hard.

Proof: The NP-hardness of the JASDP is shown by a reduction from a classic NP-hard problem – the minimum-cost Generalized Assignment Problem (GAP) as follows.

Given a set of items $Item = \{item_1, item_2, \ldots, item_n\}$ and a group of bins $Bin = \{bin_1, bin_2, \ldots, bin_m\}$. Each item $item_i, 1 \le i \le n$ has a cost $cost_i$ with size $size_i$, each bin $bin_i, 1 \le j \le m$ has capacity cap_j. The minimum-cost GAP is to assign as many items as possible to the m bins such that the total cost of assigned items is minimized, subject to the capacity cap_j of each bin.

A special case of JASDP is considered for reduction, where the configuration and DNN model of each inference request are given, and inference delay can always be satisfied. The special case includes an MEC network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a set of inference requests Γ that arrive at a single time slot. Each edge server (bin) k in $\mathcal{G}$ has a memory resource B_k, if a DNN m_j^s (item) of an inference request $\gamma_u \in \Gamma$ is deployed in an edge server, it consumes memory resource B_j^s and incurs a cost including data routing cost, DNN computing and memory cost. Since the inference accuracy of each request is determined by the configuration and the DNN model, the goal of this special case becomes minimizing the total cost, subject to each edge server's memory capacity.

It can be seen that this special JASDP is equivalent to the minimum-cost GAP. Since the minimum-cost GAP is NP-hard, and any instance of GAP can be polynomially reduced to this special JASDP case, solving JASDP would inherently solve GAP. Consequently, JASDP is at least as hard as GAP. Given that GAP is NP-hard and the reduction is polynomial-time computable, JASDP must also be NP-hard. The theorem then follows. □

Though a heuristic algorithm can address the GAP, applying this algorithm to this problem is time-consuming and makes it hard to cope with the dynamic changes of edge networks. Deep Reinforcement Learning (DRL) based algorithm can learn from history to handle complex and dynamic environments with lower execution time, which suits our problem. Since configurations and DNN models in each IS are different, the action space of DRL agents of each IS also varies, namely heterogeneous agents. In this paper, we design a Heterogeneous Agent Reinforcement Learning (HARL) based algorithm, named Heterogeneous Inference Service ProvidER (HISPER), to tackle JASDP with heterogeneous inference services.

IV. Partial Observable Markov Decision Process of Our Problem

We assign multiple agents to each IS to process the corresponding requests. As multiple inference requests arrive simultaneously in a time slot, each agent lacks information about the specific requirements of the requests handled by other agents. Therefore, each agent needs to perform an action (adapt configuration, select and deploy DNN model) to process the received inference request according to their observation in a time slot. When all agents have made their actions in the current time slot, we calculate the reward of the current environment state for their actions and obtain the next observations for all agents. In this section, we introduce the Partial Observable Markov Decision Process (POMDP) of JASDP based on the formulated problem.

A. Agents

As each IS has multiple agents to process inference requests, we assume that the number of agents for each IS is the same, represented by N, then the set of agents of all ISs can be defined as $\Lambda = \{agent_n | 1 \leq n \leq N|\mathcal{S}|\}$.

B. Observation

In a HARL environment, each agent can only observe the requirements of the inference request that it processes. To enable agents to perform high-reward actions despite partial observability, each agent can also observe the state of all edge servers, which provides critical information for their decision-making. Specifically, for $agent_n$ that needs to handle inference request $\gamma_u(t)$ of IS s, its observation o_t^n in time slot t includes: (1) The requirement of inference request $(v_u^\gamma(t), d_u^\gamma(t), data_u^\gamma(t))$; (2) The state of all edge servers $(H^{sys}(t), H^{mem}(t), \mathcal{W}(t))$, where $H^{sys}(t)$ and $H^{mem}(t)$ are two vectors include the system workload and memory usage of all edge servers, respectively, and $\mathcal{W}(t)$ is a symmetric matrix with weights representing the bandwidth of all links. $H^{sys}(t)$ and $H^{mem}(t)$ enable agents to select suitable DNN models and deployment locations to prevent server overloads, while $\mathcal{W}(t)$ guides configuration adaptions to minimize end-to-end latency during inference. For instance, if $H^{sys}(t)$ indicates high workload on one server, the agent may deploy a lightweight DNN model elsewhere to avoid high delays.

C. State

The global state in POMDP includes all inference requests and the state of all edge servers, in time slot t, the state defined as $\mathfrak{s}_t = (\Gamma(t), H^{sys}(t), H^{mem}(t), \mathcal{W}(t))$.

D. Action

Each agent has three actions, corresponding to configuration adaption, DNN model selection, and deployment. In time slot t, $agent_n$ takes independent action $a_t^n = (a_t^{n,1}, a_t^{n,2}, a_t^{n,3})$ where $a_t^{n,1} \in \mathcal{C}_s, a_t^{n,2} \in \mathcal{M}_s, a_t^{n,3} \in \mathcal{V}$, and $s = n \bmod N$, representing the IS s corresponding to $agent_n$.

Let $\mathcal{A}_t = (a_t^1, \ldots, a_t^{|\Lambda|})$ be the joint action of all agents, and $\pi^n(\cdot|\mathfrak{s}_t)$ be the policy of $agent_n$, we define the joint policy of all agents as $\boldsymbol{\pi}(\cdot|\mathfrak{s}_t) = \prod_{n=1}^{|\Lambda|} \pi^n(\cdot|\mathfrak{s}_t)$.

E. Reward

The reward function in HARL is used to evaluate all agents' actions, and the objective of all agents is to maximize the reward. In our problem, we aim to maximize inference accuracy and minimize the inference delay and cost, these three objectives are therefore included in our reward function.

Considering that the accuracy metrics of each IS are different, we first standardize the accuracy of all DNN models. For DNN model $m_j^s(t)$ selected by $agent_n$ under configuration $c_i^s(t)$ in IS $s = n \bmod N$, the standardized accuracy is calculated as:

$$Acc_n^{std}(t) = \frac{\Phi(c_i^s(t), m_j^s(t)) - Acc_s^{\min}}{Acc_s^{\max} - Acc_s^{\min}} \tag{19}$$

where $Acc_s^{\max}$ and $Acc_s^{\min}$ are the maximum and minimum accuracy provided in s, respectively.

Besides, the workload of all edge servers changes dynamically, and some agents' actions may violate the delay requirement of inference requests or exceed the memory capacity of edge servers. In this case, some inference requests that fail to meet the requirements cannot be accepted for processing, which significantly impacts the user experience. We set the accuracy obtained by those agents to a penalty $R_{penalty} < 0$. The accuracy of $agent_n$ in time slot t becomes:

$$Acc_n(t) = \begin{cases} Acc_n^{std}(t), & \text{if accepted} \\ R_{penalty}, & \text{if not accepted} \end{cases} \tag{20}$$

Then, the reward function of all agents in time slot t is defined as

$$r(\mathfrak{s}_t, \mathcal{A}_t) = \sum_{n=1}^{\mathbb{N}} Acc_n(t) - d^{total}(t) - \Psi(t) \tag{21}$$

V. Design of Heterogeneous Inference Service Provider (HISPER)

Some on-policy DRL algorithms, such as Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO), have low data efficiency, meaning that each collected data from the environment contributes relatively little to the learning process. In contrast, off-policy algorithms like Deep Deterministic Policy Gradient (DDPG) generally make better use of data. However, these off-policy methods often suffer from highly unstable training and poor convergence. Furthermore, they are sensitive to hyperparameters and difficult to adapt to different complex environments [27].

To address this problem, maximum entropy reinforcement learning introduces entropy to enhance the policy's robustness. Based on this, we proposed our HISPER algorithm, which adopts an auto-regressive policy and hybrid reward architecture to address multiple actions and rewards and train heterogeneous agents in a Centralized Training with Decentralized Execution (CTDE) way.

A. Preliminary

In maximum entropy Multi-Agent Deep Reinforcement Learning (MADRL), the entropy $\mathcal{H}$ measures the randomness of $agent_n$'s policy π^n in state $\mathfrak{s}_t$, which is expressed as:

$$\mathcal{H}(\pi^n(\cdot|\mathfrak{s}_t)) = \mathbb{E}_{a_t^n \sim \pi^n(\cdot|\mathfrak{s}_t)}[-\log \pi^n(a_t^n|\mathfrak{s}_t)] \tag{22}$$

Then, the maximum entropy objective of HISPER is:

$$\boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=1}^{T} r(\mathfrak{s}_t, \mathcal{A}_t) + \alpha \sum_{n=1}^{|\Lambda|} \mathcal{H}(\pi^n(\cdot|\mathfrak{s}_t)) \right] \tag{23}$$

where α is a temperature constant that trade-off between action rewards and maximum entropy, the higher α, the more exploratory the algorithm becomes. When $\alpha = 0$, the objective reduces to standard MADRL.

B. Auto-Regressive Policy

The action of $agent_n$ contains three sub-actions that are interrelated. For example, to achieve a higher reward, selecting a DNN model with higher accuracy requires a higher-performing edge server. Consequently, adapting to a lower configuration is necessary to minimize inference delay. Therefore, the policy $\pi^n(a^n(t)|\mathfrak{s}_t)$ is a joint distribution of each sub-action, which can be represented in an auto-regressive manner:

$$\pi^n(a_t^n|\mathfrak{s}_t) = \prod_{l=1}^{3} \pi^{n,l}\left(a_t^{n,l}|\mathfrak{s}_t, a_t^{n,<l}\right) \tag{24}$$

Note that this auto-regressive process is defined based on the sequence of operations that occur after an edge server receives an inference request, i.e., adapt configuration first, then select and deploy the DNN model. This representation allows the agent to make actions in a sequential decision-making manner instead of choosing a full action represented in a vector [28]. With this representation, the entropy of $agent_n$ becomes:

$$\mathcal{H}(\pi^n(\cdot|\mathfrak{s}_t)) = \sum_{l=1}^{3} \mathcal{H}\left(\pi^{n,l}(a_t^{n,l}|\mathfrak{s}_t, a_t^{n,<l})\right) \tag{25}$$

The conditional probability distribution in (25) can be easily obtained by using an auto-regressive model, such as Long Short-Term Memory (LSTM) or Recurrent Neural Network (RNN).

C. Hybrid Reward Architecture

The objective of our JASDP is to balance inference accuracy, resource cost, and delay, which results in the rewards of reinforcement learning being controlled by different factors in the reinforcement learning environment. Simply using a single reward to calculate the Q-value fails to capture how an action balances multiple objectives and may cause the algorithm to fall into sub-optimal. The Hybrid Reward Architecture (HRA) was first proposed by the paper [29]. They decomposed rewards into different reward functions and learned a separate value function for each component reward function. The HRA enables agents to tap into a deep well of domain knowledge, which has been shown to facilitate faster learning and enhanced performance in various domains, including video game playing, mimicry learning, etc [30]. In our algorithm, we decompose the rewards of all agents into three parts: $r^1(\mathfrak{s}_t, \mathcal{A}_t) = \sum_{n=1}^{|\Lambda|} Acc_n(t)$, $r^2(\mathfrak{s}_t, \mathcal{A}_t) = -d^{total}(t)$, and $r^3(\mathfrak{s}_t, \mathcal{A}_t) = -\Psi(t)$.

D. Heterogeneous Agent Soft Policy Iteration

To maximize the maximum entropy objective in (23), we need to iteratively and alternately update the soft Q-function and the joint policy.

We first define soft Q-function of rewards $r^i, \forall i \in \{1, 2, 3\}$ as:

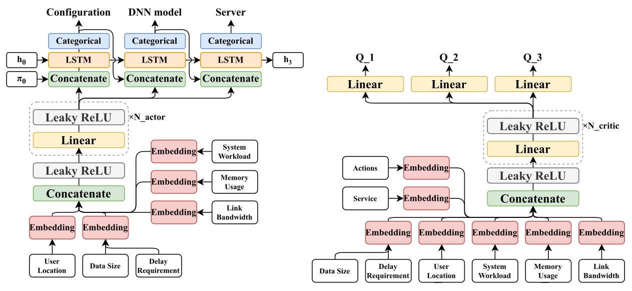
$$Q^i(\mathfrak{s}_t, \mathcal{A}_t) = r^i(\mathfrak{s}_t, \mathcal{A}_t) + \varphi \mathbb{E}_{\mathfrak{s}_{t+1}}[V^i(\mathfrak{s}_{t+1})] \tag{26}$$

where φ represents discount factor, and the soft value function $V^i(\mathfrak{s}_t), \forall i \in \{1, 2, 3\}$ is calculated as follows:

$$V^i(\mathfrak{s}_t) = \mathbb{E}_{\mathcal{A} \sim \boldsymbol{\pi}} \left[Q^i(\mathfrak{s}_t, \mathcal{A}_t) + \alpha \sum_{n=1}^{|\Lambda|} \mathcal{H}(\pi^n(\cdot|\mathfrak{s}_t)) \right] \tag{27}$$

To update the joint policy, [31] pointed out that a maximum entropy MADRL problem can be elegantly decomposed into a sum of n distinct maximum entropy MADRL sub-problems, where each agent sequentially optimizes individual Kullback-Leibler (KL) divergence, leading to the optimization of joint soft policy. For each state, we update the joint soft policy according to

$$\boldsymbol{\pi}_{\text{new}} = \arg\min_{\boldsymbol{\pi} \in \Pi} \mathrm{D}_{\mathrm{KL}} \left(\boldsymbol{\pi} || \frac{\exp(\frac{1}{\alpha} \sum_{i=1}^{3} Q^i_{\boldsymbol{\pi}_{\text{old}}}(\mathfrak{s}, \cdot))}{Z_{\boldsymbol{\pi}_{\text{old}}}(\mathfrak{s}, \cdot)} \right) \tag{28}$$



(a) actor (b) critic

Fig. 2. The actor and critic network.

where Π is joint policy space, $\mathrm{D}_{\mathrm{KL}}(a||b)$ represents the KL-divergence between distribution a and b, $Z_{\boldsymbol{\pi}_{\text{old}}}(\mathfrak{s},\cdot)$ is a partition function which normalizes the distribution.

E. HISPER Algorithm

Repeatedly applying (26) and (28) can increase the soft Q-function monotonically, leading to the convergence of the policies [31]. Since the soft policy iteration process needs to be applied to the entire action space and state space, it is difficult to apply it to a continuous state space. In practice, we can express the soft policy iteration as a stochastic optimization problem and solve it by stochastic gradient descent [32].

In HISPER, there are two critic networks Q_{ω_1}, Q_{ω_2}, two target networks $Q_{\overline{\omega}_1}, Q_{\overline{\omega}_2}$, and actor networks $\pi^n_{\theta^n}(a^n_t|\mathfrak{s}_t)$ of all $agent_n \in \Lambda$, parameterized respectively by $\omega_j, \overline{\omega_j}, \forall j \in \{1,2\}$, and $\theta^n, \forall n \in |\Lambda|$. The critic networks and actor networks can be considered as function approximators of the soft Q-function and the policy of $agent_n$. The target networks are used to slowly track the actual critic networks to improve stability [27]. Both critic networks and target networks would output three values corresponding to the Q-value of different rewards, denoted as $Q^i_{\omega_j}, Q^i_{\overline{\omega}_j}, \forall i \in \{1,2,3\}, \forall j \in \{1,2\}$. We also define a replay buffer $\mathcal{B}$ to store data generated during each time slot, which not only addresses the cold start problem in HISPER training but also enables efficient reuse of historical data. The data in a time slot is denoted by b_t, which contains observation o^n_t, joint actions $\mathcal{A}_t$, reward r_t, and the observation of the next time slot o^n_{t+1} of all $agent_n \in \Lambda$. In addition, The global state $\mathfrak{s}_t$ can be obtained by $\mathfrak{s}_t = \bigcup o^n_t, \forall n \in \Lambda$.

Fig. 2 shows the architecture of the actor and critic networks, each actor network takes the observation o^n_t of $agent_n$ as input, while the critic network uses the global state $\mathfrak{s}_t$ and the actions of all agents. Since the global state contains all inference requests' information, the critic network also needs a token (i.e., the *Service* in Fig. 2(b)) to indicate the IS that each inference request corresponds to. The input data for both networks will be processed through an embedding layer to extract features and then concatenated. In the actor network, those features will pass through *N_actor* fully connected layers. The output layer of the actor network comprises three LSTM units and three categorical units. The LSTM unit receives two inputs: hidden state h_i and the concatenation of the output from the last fully connected layer and the probability distribution π_i of the previous action. Initially, h_0 and π_0 are two zero vectors. The categorical unit converts the LSTM outputs into an action probability distribution and selects an action. In the critic network, the extracted features are processed by *N_critic* fully connected layers, and it outputs three Q-values: *Q_1*, *Q_2*, and *Q_3*, corresponding to three different sub-rewards.

The parameters of critic networks can be trained by minimizing the Bellman residual:

$$J_{Q^i}(\omega_j) = \mathbb{E}_{b_t\sim\mathcal{B}}\left[\frac{1}{2}(Q^i_{\omega_j}(\mathfrak{s}_t,\mathcal{A}_t) - (r^i_t + \varphi V^i_{\overline{\omega}_j}(\mathfrak{s}_{t+1})))^2\right] \tag{29}$$

In our algorithm, the actor networks are updated sequentially. During the sequential update, each actor network has a unique optimization objective that takes into account all previous networks' updates. Let $\mathrm{Sym}(n)$ to denote the set of permutations of integers $\{1,\ldots,n\}$, and use $n_{1:q} = \{1,\ldots,q\}$ to represent a permutation from 1 to q. In the update process of actor networks, we draw a permutation $n_{1:|\Lambda|} \in \mathrm{Sym}(|\Lambda|)$ and sequentially update the actor network of each agent n_p according to (28). Since the partition function $Z_{\text{old}}(\mathfrak{s},\cdot)$ does not contribute to the gradient, it can be disregarded in the loss function of actor network [27]. Thus, the policy parameters can be trained by minimizing the expected KL-divergence:

$$\begin{aligned} J_{\pi^{n_p}}(\theta^{n_p}) = {} & \mathbb{E}_{b_t\sim\mathcal{B},\mathcal{A}^{n_{1:p-1}}_t\sim\boldsymbol{\pi}^{n_{1:p-1}}_{\theta^{n_{1:p-1}}_{\text{new}}},\mathcal{A}^{n_p}_t\sim\boldsymbol{\pi}^{n_p}_{\theta^{n_p}}} \\ & \times\Big[\alpha\log\pi^{n_p}_{\theta^{n_p}}(\mathcal{A}^{n_p}_t|\mathfrak{s}_t) \\ & - Q_{\pi^{n_{1:p}}_{\text{old};\theta}}(\mathfrak{s}_t,\mathcal{A}^{n_{1:p-1}}_t,\mathcal{A}^{n_p}_t)\Big] \end{aligned} \tag{30}$$

where $\mathcal{A}^{n_{1:q}}_t = \{a^i_t|\forall i\in n_{1:q}\}$ and $\pi^{n_{1:q}}_{\theta^{n_{1:q}}_{\text{new}}} = \prod_{i=1}^{|n_{1:q}|}\pi^i_{\theta^i}$ are respectively the joint action and joint policy of agents $\{agent_i|i\in n_{1:q}\}$, and n_p represents the pth element in permutation.

The HISPER algorithm is presented in Algorithm 1, which is composed of two parts: data collecting (Lines 5 to 12) and network training (Lines 13 to 20). The input of the algorithm consists of the parameters of critic networks ω_1,ω_2 and actor networks $\{\theta^n\}_{n\in|\Lambda|}$, and some parameters needed by the training process, including total time slots T, training steps TS, batch size bs, and training interval $interval$. After finishing the algorithm, it outputs well-trained actor networks for all agents. The training process is illustrated in Fig. 3.

At the beginning of the algorithm, the replay buffer $\mathcal{B}$ and the parameters of target networks $\overline{\omega}_1,\overline{\omega}_2$ are first initialized. In the data collecting part, an action is first picked from each agent's actor network π^n_θ based on their observations. Then execute the joint actions in the environment and obtain observations $o^n_{t+1},\forall n\in|\Lambda|$ for the next time slot of all agents and reward r_t. Finally, push the collected data in this time slot to the replay buffer.

We perform training every $interval$ training steps. In the network training part, the target networks' outputs estimate the Q-values of input states, therefore, we use their outputs and the rewards sampled from the replay buffer to compute

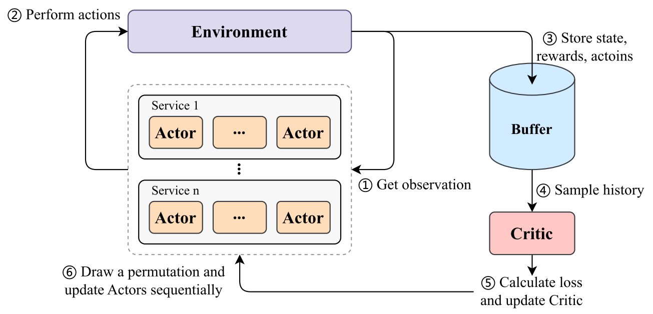


Fig. 3. The training process of HISPER.

Algorithm 1: HISPER.

```
Input: parameter of two critic networks $\omega_1$, $\omega_2$,
       parameter of actor network of all agents
       $\theta^n, \forall n \in |\Lambda|$, total time slots $T$, training steps
       $TS$, batch size $bs$, training interval $interval$
Output: Well-trained actor networks $\theta^n, \forall n \in |\Lambda|$ of
        all agents
1  begin
2      Initialize replay buffer $\mathcal{B}$;
3      Set the parameter for the target network
       $\overline{\omega}_1 \leftarrow \omega_1, \overline{\omega}_2 \leftarrow \omega_2$;
4      for $ts \leftarrow 1, 2, \ldots, TS$ do
5          for $t \leftarrow 1, 2, \ldots, T$ do
6              Get observation $o_t^n, \forall n \in |\Lambda|$;
7              Select action $a_t^n \sim \pi_\theta^n(\cdot|o_t^n), \forall n \in |\Lambda|$;
8              $\mathcal{A}_t \leftarrow (a_t^1, \ldots, a_t^{|\Lambda|})$;
9              Execute $\mathcal{A}_t$ in the environment;
10             Get next observation $o_{t+1}^n$, reward $r_t$;
11             $b_t \leftarrow \{(o_t^n, \mathcal{A}_t, r_t, o_{t+1}^n)|\forall n \in |\Lambda|\}$;
12             $\mathcal{B} \leftarrow \mathcal{B} \cup b_t$;
13         if $ts$ mod $interval$ = 0 then
14             Sample $\{b_t|t \in \{1, \cdots, bs\}\}$ from $\mathcal{B}$;
15             Calculate target Q-value $y_t^i, \forall i \in \{1, 2, 3\}$
               by Eq. (31);
16             Update parameters of critic networks by
               one step of gradient descent using Eq.
               (32);
17             Draw a permutation of agents
               $n_{1:|\Lambda|} \in \mathrm{Sym}(|\Lambda|)$ at random;
18             for $n_p = n_1, \ldots, n_{|\Lambda|}$ do
19                 Update parameter of actor network
                   $\theta^{n_p}$ by solving Eq. (33) with policy
                   gradient descent;
20         Update target networks by Eq. (34);
```

target Q-values y_i^t for training the critic networks in the network training part. However, estimating Q-values with target networks may introduce overestimation bias, potentially leading to suboptimal policies [33]. Selecting the minimum Q-value between two target networks for computing target value suppresses high-variance Q-value estimations and reduces target value variance, which improves training stability. The $y_t^i, \forall i \in \{1, 2, 3\}$ is expressed as

$$y_t^i = r_t^i + \varphi\Bigg[\min_{j=1,2} Q_{\overline{\omega}_j}^i(\mathfrak{s}_{t+1}, \mathcal{A}_{t+1}) - \alpha \sum_{n=1}^{|\Lambda|} \sum_{l=1}^{3} \log \pi_\theta^{n,l}(a_{t+1}^{n,l}|\mathfrak{s}_{t+1}, a_{t+1}^{n,<l})\Bigg] \quad (31)$$

We use the Mean Square Error (MSE) to calculate the loss of critic networks $Q_{\omega_j}, \forall j \in \{1, 2\}$, and the parameters of critic networks can be trained by gradient descent on the following loss function:

$$L_Q(\omega_j) = \frac{1}{bs} \sum_{t=1}^{bs} \sum_{i=1}^{3} (y_t^i - Q_{\omega_j}^i(\mathfrak{s}_t, \mathcal{A}_t))^2 \quad (32)$$

The actor networks of all agents are updated sequentially by using updated critic networks. As shown in Line 17, the update sequence of the actor network is determined by a random permutation $n_{1:|\Lambda|} \in \mathrm{Sym}(|\Lambda|)$. Since the outputs of actor networks are discrete actions, it is not possible to calculate the gradients of the output layers. Therefore, we use Gumble-Softmax trick [34] to parameterize the actor network $\pi_{\theta^{n_p}}^{n_p}$, denoted by $\pi_{\hat{\theta}^{n_p}}^{n_p}$. The parameters of the actor network can be trained by gradient descent on the loss function for $\pi_{\theta^{n_p}}^{n_p}$:

$$L_{\pi^{n_p}}(\theta^{n_p}) = \frac{1}{bs} \sum_{t=1}^{bs} \Bigg[\alpha \log \pi_{\hat{\theta}^{n_p}}^{n_p}(a_{\hat{\theta}^{n_p}}^{n_p}|o_t^{n_p}) - \sum_{i=1}^{3} \min_{j=1,2} Q_{\omega_j}^i\left(\mathfrak{s}_t, \mathcal{A}_{\theta_{\text{new}}^{n_{1:p-1}}}^{n_{1:p-1}}, a_{\hat{\theta}^{n_p}}^{n_p}, \mathcal{A}_{\theta_{\text{old}}}^{n_{p+1:q}}(o_t^{n_{p+1:q}})\right)\Bigg] \quad (33)$$

At the end of the network training part, we smoothly update the parameters of target networks $\overline{\omega}_j, \forall j \in \{1, 2\}$ by the following

$$\overline{\omega}_j \leftarrow \tau\overline{\omega}_j + (1 - \tau)\omega_j \quad (34)$$

where $\tau \in [0, 1]$ is a smoothing constant that controls the update rate.

Based on the design of the HISPER, it can be easily applied to existing AI inference frameworks, such as a task-oriented integrated sensing-communication-computation split AI inference framework [35]. This framework jointly optimized multi-device sensing power, communication time allocation, and quantization bits allocation to maximize the discriminant gain (a metric for inference accuracy) by addressing resource allocation challenges. Specifically, HISPER can tackle the device heterogeneity issues inherent in the framework, while its auto-regressive policy can easily extend the agent's action to a joint decision of sensing power, communication time allocation, and quantization bits allocation.

VI. Experimental Results and Analyses

In this section, we comprehensively evaluate the HISPER algorithm for JASDP. We first introduce how the performance data of DNN models under different configurations are obtained. Then we present the experiment settings based on a real-world

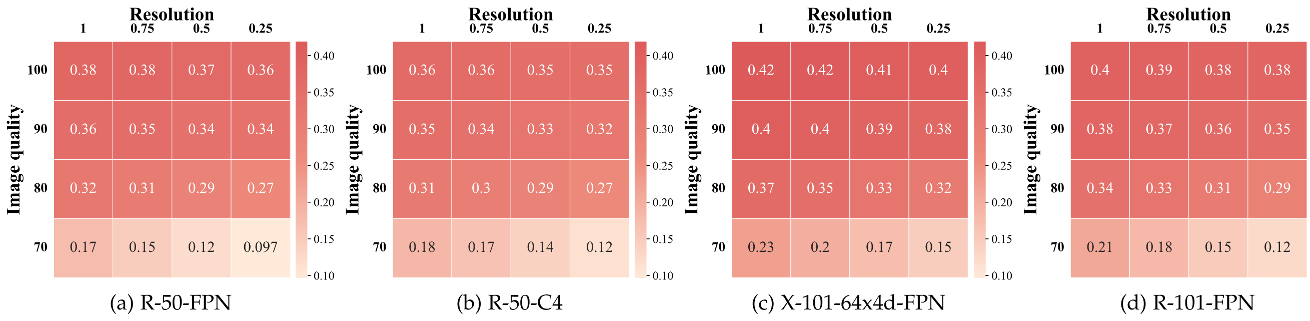


Fig. 4. Accuracy of different object detection models under varying configurations.

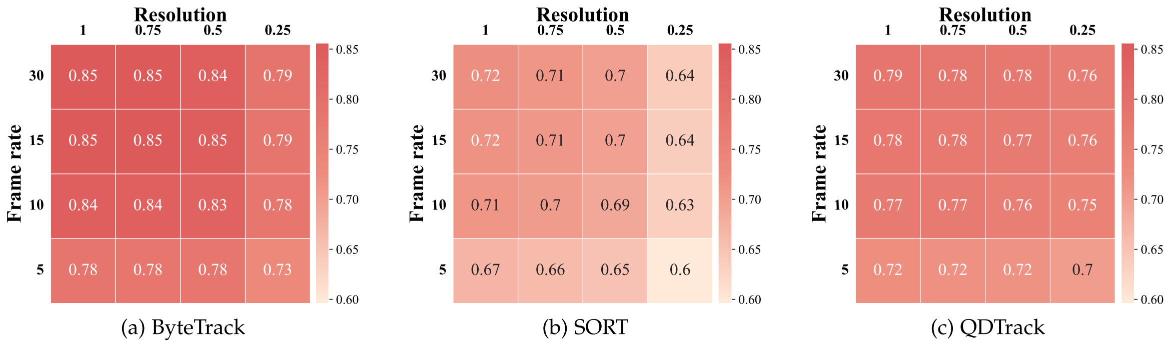


Fig. 5. Accuracy of different multiple object tracking models under varying configurations.

MEC environment. Four state-of-the-art algorithms are finally described to compare the performance of the HISPER.

A. Performance Data of DNN Models

To obtain the performance of different DNN models under different configurations, we benchmark DNN models of two real-world ISs: object detection and multiple object tracking.

The object detection service is used to recognize multiple objects in an image. We select four Faster R-CNN [36] models with different backbone networks and evaluate them on the COCO dataset [37]. Except for image resolution, the compression quality of an image can also influence both inference accuracy and data size, we consider resolution and image quality as the configurations of this service. In the COCO dataset, there are various resolutions for the images, we use four scaling factors $[1, 0.75, 0.5, 0.25]$ as the values for the resolution configuration. Since the images in the COCO dataset are all in JPG format, the quality of an image can be represented by JPG quality values, which are $[100, 90, 80, 70]$. In object detection, DNN model performance is evaluated by Mean Average Precision (mAP), and we use it as the accuracy value of this service. Fig. 4 shows the accuracy of four object detection models under different configurations. The accuracy of this service ranges from 0.42 to 0.097, with darker colors indicating higher accuracy. It can be observed that with higher image quality, the resolution has less effect on accuracy. Meanwhile, at a fixed resolution, image quality significantly influences accuracy.

The multiple object tracking service tracks the movement of multiple objects within a video stream. We select three models with different tracking methods and detectors and evaluate them on the MOT17 dataset [41]. For video stream, resolution and frame rate can be the configuration of this service. Like the object detection service, the resolution configuration has four scaling factors, i.e., $[1, 0.75, 0.5, 0.25]$, and the frame rate is $[30, 15, 10, 5]$. We use Multiple Object Tracking Accuracy (MOTA) as the accuracy value in this service. Fig. 5 shows three multiple object tracking models' accuracy under different configurations. The accuracy of this service ranges from 0.85 to 0.6, with darker colors indicating higher accuracy. As the original video resolution in the MOT17 dataset is 1080p, even though the resolution is reduced to half, all DNN models still achieve accuracy comparable to the original resolution. However, the accuracy of DNN models still decreases at the same resolution.

We also evaluate the impact of different configurations on data size in two ISs. As shown in Fig. 6, we use the data size of the highest configuration as the reference point. The figures show scaling factors η_i^s relative to the reference point. As the configuration level decreases, the data size diminishes rapidly. Moreover, for each IS, there is only one configuration where the data size is approximately equal to that of the reference point.

The memory and computational resources required by DNN models for these two ISs to perform a single inference (such as detecting an image or a video frame) are fixed. Tables II and III illustrate resource requirements for the two ISs' DNN models.

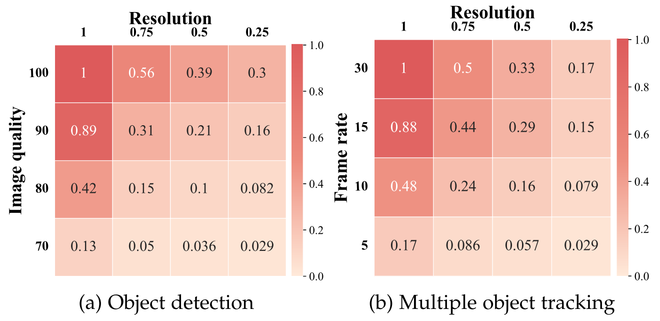


Fig. 6. Different data size under varying configurations in two services.

TABLE II
THE DNN MODELS OF OBJECT DETECTION

Backbone	Memory (MB)	TFLOPs
R-50-FPN (FP16)	159.48	0.1747
R-50-C4	129.31	0.8193
X-101-64x4d-FPN	380.91	0.3688
R-101-FPN	232.13	0.2380

TABLE III
THE DNN MODELS OF MULTIPLE OBJECT TRACKING

Method	Detector	Memory (MB)	TFLOPs
ByteTrack [38]	YOLOX-X	378.02	0.3959
SORT [39]	R50-FPN	157.93	0.1448
QDTrack [40]	R50-FPN	216.95	0.1416

B. Experiment Settings

We evaluate the HISPER with real-world locations of edge servers from the EUA dataset [42]. Specifically, we extract 16 locations from the dataset, i.e., $|\mathcal{V}| = 16$. If the Euclidean distance between any two edge servers is less than 0.5 km, they are considered to be linked. The bandwidths of the links between all edge servers are randomly chosen from the set $[50, 100, 200, 300, 400]$ Mbps. This setting aligns with existing studies [43], [44], both of which adopted real-world edge server link bandwidth in their experiment. Based on 5G uplink measurements under non-ideal conditions from [45] and the upload speed ranges in [44], the wireless uplink transmission rate of all edge servers is randomly selected from [36,48,64,72] Mbps. Considering the heterogeneity of edge servers, in our experiments, the computing capacity and storage capacity of all edge servers are randomly selected from $[0.5, 1.26, 1.33]$ TFLOPs/s and $[1, 2, 4]$ GB, respectively. These values are derived from widely adopted edge computing devices, including the NVIDIA Jetson TX2 series [46] and the Raspberry Pi 4B [47]. Notably, the experimental settings are not limited to the EUA dataset but can be generalized to other network topologies.

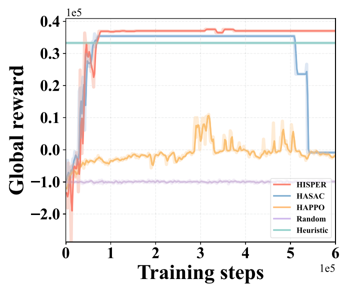
The total time slot of our experiment is set to 100. At each time slot, two ISs have the same number of requests, which arrive randomly. We conduct two experiments with four and five inference requests per service, corresponding to eight and ten agents, respectively. The data sizes and delay requirements of the two ISs are different. For the object detection service, we set the data size range from 1 MB to 10 MB, and the delay requirement is picked from $[0.5, 1, 1.5, 2]$ s. For the multiple object tracking service, we set the data size range from 5 MB to 20 MB, and the delay requirement is picked from $[1, 5, 10, 15, 20, 25]$ s. The values of the delay requirement are calculated based on the specifications of the edge servers and the requirements of all DNN models, ensuring that any DNN model can be selected to provide service. Besides, the system workload and memory usage of all edge servers before receiving inference requests in each time slot range from 0.1 to 0.3 TFLOPs and 100MB to 900MB, respectively. By using the hourly rate (price) of a machine learning optimized ml.g6.xlarge Amazon SageMaker real-time inference instance as a reference, the unit computing cost ψ^{comp} is set at \$0.0003 per second. The unit memory cost ψ^{mem} is set at \$0.016 per GB, based on the pricing in Amazon SageMaker data processing. The unit routing cost ψ_e^{rou} is set to \$0.002 per Mbps [48]. If some inference requests can not be satisfied, the accuracy of the corresponding agents will be set to a penalty value $R_{panalty} = -100$.

The embedding layer in HISPER's actor and critic networks is a fully connected layer with 32 dimensions. The actor network has two hidden layers with dimensions of 256 and 256, while the critic has five hidden layers with dimensions of 512, 512, 256, 256, and 128. The learning rates for the actor and critic are respectively set to 2×10^{-4} and 5×10^{-4}. We prefill the experience replay buffer using 9×10^4 samples to address the cold start problem in the training phase of HISPER. For all experiments, we use 20 threads with 20 different random seeds for sampling to train networks, with each round of sampling across all threads counting as 20 training steps. All networks are evaluated using another 10 random seeds. The total training step is set to 6×10^5, training and evaluating all networks every 2,000 steps (which means the 20 threads are sampled 100 times). The experiment is conducted with an NVIDIA GeForce RTX 4070 Ti GPU and an Intel(R) Core(TM) i7-13700KF processor with 32 GB of RAM.

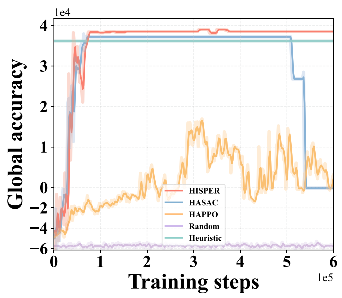
C. Comparison Algorithms

We compare the performance of the HISPER to that of four comparison algorithms. We use the same hidden layer neural networks as HISPER for HASAC and HAPPO for fair comparison.

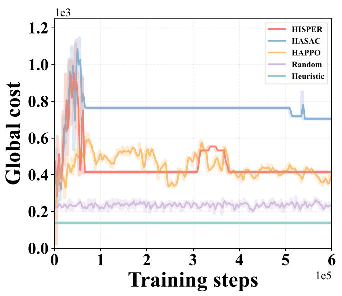
- *Heterogeneous-Agent Soft Actor-Critic (HASAC) [31]:* This algorithm is a maximum entropy HARL-based algorithm without HRA and the auto-regressive policy. It uses the total reward to calculate the Q-value and outputs the action simultaneously in its actor network. This algorithm is used to examine the effect of the HRA and auto-regressive policy structure in HISPER.
- *Heterogeneous-Agent Proximal Policy Optimization (HAPPO) [49]:* This algorithm is an on-policy HARL algorithm based on Proximal Policy Optimization (PPO). It uses the potential decomposition theorem to guarantee the monotonic improvement of agents and updates agents with a random permutation during the training phase. This algorithm is used to compare the monolithic improvement effect of heterogeneous agents.



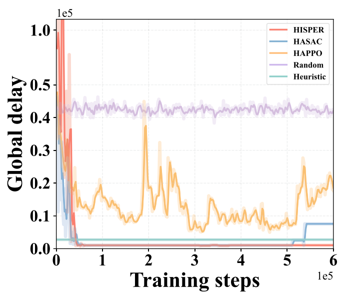
(a) Global reward with 8 agents



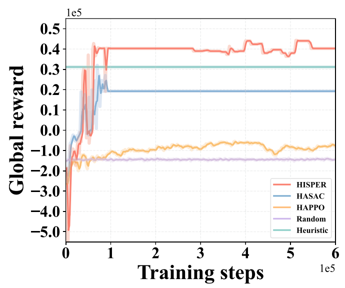
(b) Global accuracy with 8 agents



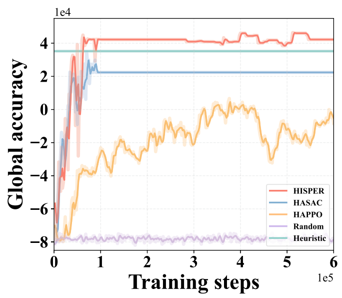
(c) Global cost with 8 agents



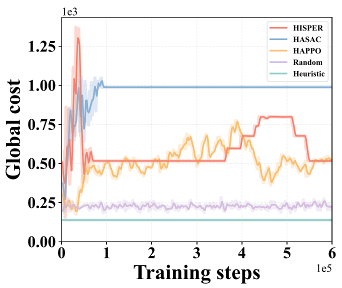
(d) Global delay with 8 agents



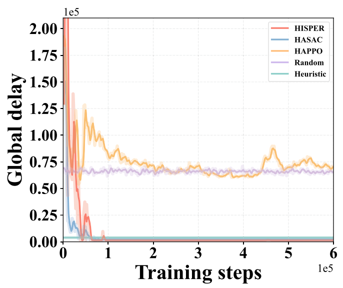
(e) Global reward with 10 agents



(f) Global accuracy with 10 agents

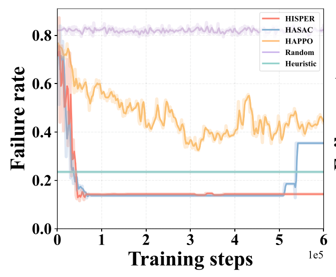


(g) Global cost with 10 agents

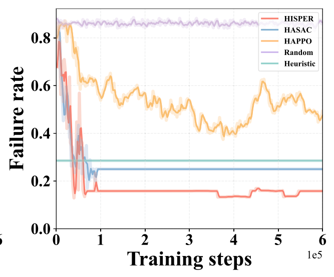


(h) Global delay with 10 agents

Fig. 7. Global rewards of all algorithms under two environments.



(a) Failure rate with 8 agent



(b) Failure rate with 10 agent

Fig. 8. Failure rate under 8 and 10 agents.

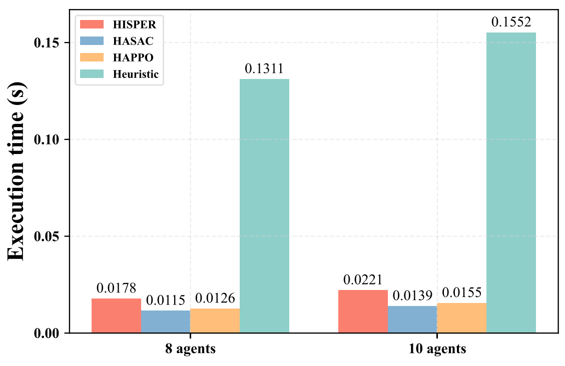


Fig. 9. Execution time of four algorithms under two environments.

- *Heuristic:* We design a heuristic algorithm where each agent independently selects the action with the highest reward for themselves. Since the action space in this problem is large, this algorithm is used to compare the execution efficiency with our algorithm.
- *Random:* Each agent's action is randomly selected, representing the naive performance of the environment without any optimization.

D. Evaluation of HISPER and Comparison Algorithms

We show the performances of four algorithms in two experimental configurations: the environment with 8 inference requests and the environment with 10 inference requests per time slot, thus the number of agents for these two environments is 8 and 10. We set the value of alpha α to 2 in the 8-agent environment, and 2.5 in the 10-agent environment since it is more complex.

We first present the training time of HISPER. In the 8-agent environment, the reward stabilized at around 1.08×10^5 training steps, taking about 5.84 minutes in total. The average time for each training step in this environment was about 0.0255 seconds, while every 2,000 training steps, including neural network training, took about 6.64 seconds on average. With 10 agents, the increased number of agents led to longer times. The reward stabilized after roughly 9.8×10^4 training steps, taking 6.72 minutes. Each training step averaged about 0.0321 seconds, and every 2,000 steps with neural network training averaged around 8.38 seconds.

Fig. 7 shows the evaluation results of all algorithms under 10 different random seeds. The y-axis values in the graph represent the sum of values obtained in 100 time slots. To compare the different algorithms more clearly, we scale the y-axis of Fig. 7(a), (b), (d), and (e). The total reward and the three sub-rewards are plotted in the figure, providing insight into each algorithm's performance with different numbers of agents.

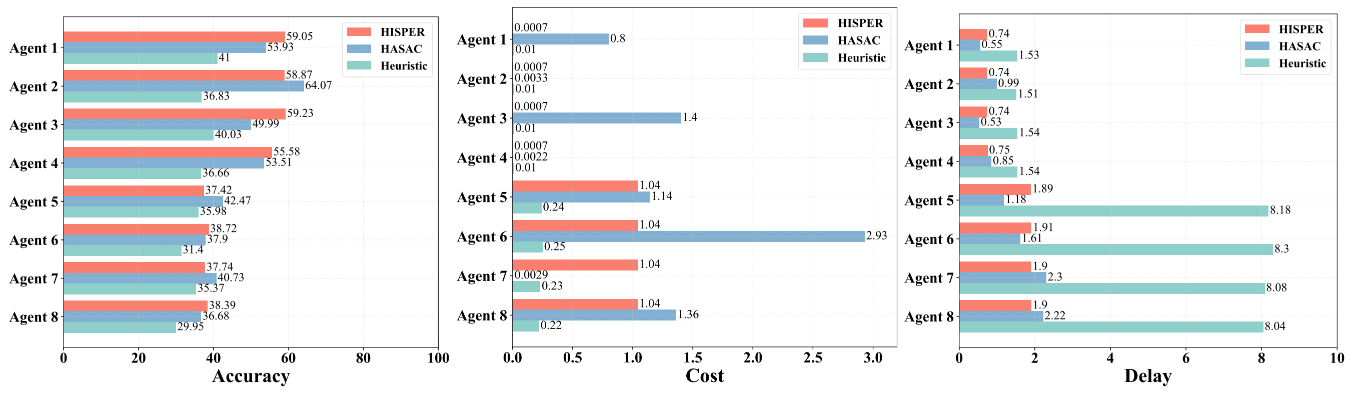


Fig. 10. Performance of each agent under 8-agent environment.

In terms of global reward (Fig. 7(a) and (e)), HISPER consistently achieves the highest and most stable reward across both 8 and 10 agents. It suggests that HISPER optimizes its policy effectively, balancing the trade-offs between accuracy, cost, and delay. Although the convergence speed of HASAC is similar to that of HISPER, the performance deteriorates as the number of agents increases. The capacity of the buffer fills up at 2.8×10^5 training steps, after which HISPER's performance fluctuates slightly, indicating that HISPER is still exploring other possible solutions. In Fig. 7(a), HASAC's global reward shows a significant drop at about 5.1×10^5 training steps, it failed to explore the action with a higher reward and make three-way trade-offs. In the environment with 8 agents, HAPPO's average reward is close to zero and has great fluctuation, while in the environment with 10 agents, HAPPO performs only slightly better than the randomized algorithm. Unlike HASAC and HISPER, HAPPO does not have a training buffer, which means that historical samples are important for algorithmic improvement in a huge search space. Without any optimization, the global reward of the random algorithm is always the lowest one, while the global reward of the heuristic algorithm remains a fixed value. The heuristic's reward is worse than HISPER and HASAC in the 8-agent environment, and only worse than HISPER in the 10-agent environment. It suggests that obtaining a higher global reward requires trade-offs between different inference requests, rather than choosing the action with the highest reward for each request.

Accuracy is a key factor of the global reward, as shown in Fig. 7(b) and (f), the global accuracy trends of all algorithms are similar to their global reward trends. In these two figures, HAPPO archives the lowest global accuracy except for random, it performs nearly negative accuracy under the 10-agent environment, which implies it fails to serve as many inference requests as possible. While for the global cost (Fig. 7(c) and (g)), the heuristic algorithm performs the lowest cost, followed by the random algorithm. The costs of HISPER and HAPPO are similar, but HISPER's performance is more stable than HAPPO's. Though the global accuracy of HASAC under the 8-agent environment is close to HISPER's, its cost is about two times higher than the cost of HISPER. It suggests that HISPER is able to use a lower cost to obtain a higher accuracy than HAPPO and HASAC. In both environments, HISPER's cost fluctuates between 3×10^5 to 4×10^5 training steps in the 8-agent environment and between 3.6×10^5 to 5.44×10^5 training steps in the 10-agent environment. Meanwhile, the accuracy also changes, indicating that HISPER is exploring the balance between accuracy and cost.

As shown in Fig. 7(d) and (h), the heuristic algorithm, HISPER, and HASAC show nearly identical delay performance, while the random algorithm and HAPPO perform significantly worse. At the beginning of training, HISPER exhibits low accuracy, high cost, and high delay. As training proceeds, HISPER gradually learns to balance these three factors, finally achieving the highest reward.

Fig. 8 shows the failure rate under two environments, representing the proportion of requests that were not successfully satisfied out of the total number of requests. The HISPER always archives the lowest failure rate in providing ISs, followed by HASAC and the heuristic algorithm. Before 5×10^5 training steps, HASAC shows similar performance to HISPER in the 8-agent environment. However, its performance declines when scaled to the 10-agent environment, indicating that HISPER is more robust in handling multiple inference requests.

Fig. 9 shows the execution time of four algorithms within a time slot. As the HISPER adopts LSTM to output three actions, its execution time is slightly higher than HASAC and HAPPO. It is evident that the execution time of the heuristic algorithm is higher than the other three DRL-based algorithms, suggesting that they are more efficient. Although the execution time of the heuristic algorithm is sufficient for some inference requests, its execution time increases rapidly as the number of requests expands.

E. Performance of Each Agent in Comparison Algorithms

In this subsection, we show the performance of each agent in three comparison algorithms, i.e., HISPER, HASAC, and the heuristic algorithm. Fig. 10 presents the three average sub-rewards obtained by the three algorithms over 100 time slots for 8 agents. In the 8-agent environment, agents 1 to 4 handle object detection requests, and agents 5 to 8 focus on multiple object tracking requests. As shown in Fig. 10, three algorithms perform similar average inference accuracy, among these eight agents, five agents in HISPER achieve the highest accuracy, while three agents in HASAC. In these two ISs, the accuracy of agents 1 to 4 is always higher than that of agents 5 to 8. In HISPER, the accuracy obtained by agents of the same IS is more balanced, which indicates that the algorithm can provide a fairer IS. Although the accuracy of HASAC is higher or comparable to that of HISPER in some agents, it consumes a higher cost than HISPER, and its cost is unbalanced across different types of ISs. The HISPER archives the lowest cost in the object detection service, while higher than the heuristic algorithm in the multiple object tracking service. For the performance of inference delay, the heuristic algorithm performs the highest inference delay in all agents, which suggests it failed to balance cost and delay. The HASAC and HISPER exhibit similar inference delay. However, HASAC incurs a higher cost compared to HISPER, which achieves the same delay at a lower cost. This suggests that HISPER offers the best trade-offs.

VII. Conclusion and Future Work

In this paper, we have studied heterogeneous ISs provisioning in edge networks, where different ISs have various configurations and DNN models. Specifically, we have proposed an online algorithm HISPER to jointly optimize the application configuration adaption, DNN model selection, and deployment to trade-offs between inference accuracy, resource cost, and inference delay. Simulation experimental results show that our algorithm can achieve the highest reward compared to other state-of-the-art algorithms in different environments and achieves good three-way trade-offs among inference accuracy, cost, and delay.

In our implementation, HISPER assigns a dedicated agent to each inference request. As the number of inference requests increases, so does the number of agents. This leads to higher computational demands and training complexity. To address these scalability challenges, we will improve HISPER in the future: (1) Assigning similar inference requests (based on service type) to a single agent, reducing the total number of agents needed. (2) Introducing a two-tier structure where high-level agents coordinate request allocation while low-level agents handle specific decisions, distributing computational load more efficiently.

References

[1] Y. Qiu et al., “Spotlighter: Backup age-guaranteed immersive virtual vehicle service provisioning in edge-enabled vehicular metaverse,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 13375–13391, Dec. 2024, doi: 10.1109/TMC.2024.3425896.

[2] J. Liang, S. Huang, Y. Qiu, L. Liu, F. Aziz, and M. Chen, “Sustainable virtual network function placement and traffic routing for green mobile edge networks,” *IEEE Trans. Green Commun. Netw.*, vol. 8, no. 4, pp. 1450–1465, Dec. 2024, doi: 10.1109/TGCN.2024.3392813.

[3] X. Liang, W. Liang, Z. Xu, Y. Zhang, and X. Jia, “Multiple service model refreshments in digital twin-empowered edge computing,” *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2672–2686, Sep./Oct. 2024, doi: 10.1109/TSC.2023.3341988.

[4] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, “Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning,” *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4988–4998, Jul. 2021.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,” in *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[6] Y. Qiu, J. Liang, V. C. Leung, and M. Chen, “Online security-aware and reliability-guaranteed ai service chains provisioning in edge intelligence cloud,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5933–5948, May 2024.

[7] L. Zhao, Y. Liu, A. Y. Al-Dubai, A. Y. Zomaya, G. Min, and A. Hawbani, “A novel generation-adversarial-network-based vehicle trajectory prediction method for intelligent vehicular networks,” *IEEE Internet Things J.*, vol. 8, no. 3, pp. 2066–2077, Feb. 2021.

[8] T. Brown et al., “Language models are few-shot learners,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.

[9] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” in *Proc. 9th Int. Conf. Learn. Representations*, Austria, 2021, pp. 1–22.

[10] Y. Qiu et al., “Reliable or green? Continual individualized inference provisioning in fabric metaverse via multi-exit acceleration,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11449–11465, Dec. 2024.

[11] J. Xie, Z. Zhou, T. Ouyang, X. Zhang, and X. Chen, “Fair DNN model selection in edge AI via a cooperative game approach,” in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst.*, 2023, pp. 383–394.

[12] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, “DeepDecision: A mobile deep learning framework for edge video analytics,” in *Proc. 2018 IEEE Conf. Comput. Commun.*, 2018, pp. 1421–1429.

[13] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, “Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics,” in *Proc. 2020 IEEE Conf. Comput. Commun.*, 2020, pp. 257–266.

[14] W. Fan et al., “DNN deployment, task offloading, and resource allocation for joint task inference in IIoT,” *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1634–1646, Feb. 2023.

[15] K. Zhao et al., “EdgeAdaptor: Online configuration adaption, model selection and resource provisioning for edge DNN inference serving at scale,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5870–5886, Oct. 2023.

[16] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, “Chameleon: Scalable adaptation of video analytics,” in *Proc. 2018 Conf. ACM Special Int. Group Data Commun.*, 2018, pp. 253–266.

[17] P. Yang, F. Lyu, W. Wu, N. Zhang, L. Yu, and X. S. Shen, “Edge coordinated query configuration for low-latency and accurate video analytics,” *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4855–4864, Jul. 2020.

[18] W. Xiao, Y. Hao, J. Liang, L. Hu, S. A. Alqahtani, and M. Chen, “Adaptive compression offloading and resource allocation for edge vision computing,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 10, no. 6, pp. 2357–2369, Dec. 2024, doi: 10.1109/TCCN.2024.3400820.

[19] B. Lu, J. Yang, J. Xu, and S. Ren, “Improving QoE of deep neural network inference on edge devices: A bandit approach,” *IEEE Internet Things J.*, vol. 9, no. 21, pp. 21409–21420, Nov. 2022.

[20] J. Li, F. Lin, L. Yang, and D. Huang, “AI service placement for multi-access edge intelligence systems in 6G,” *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1405–1416, May/Jun. 2023.

[21] L. Wang, X. Ren, C. Zhao, F. Zhao, and S. Yang, “MPDM: A multi-paradigm deployment model for large-scale edge-cloud intelligence,” *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8773–8785, May 2023.

[22] W. Xiao, Y. Miao, G. Fortino, D. Wu, M. Chen, and K. Hwang, “Collaborative cloud-edge service cognition framework for DNN configuration toward smart IIoT,” *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 7038–7047, Oct. 2022.

[23] X.-L. Huang, X. Ma, and F. Hu, “Machine learning and intelligent communications,” *Mobile Netw. Appl.*, vol. 23, pp. 68–70, 2018.

[24] L. Zhang et al., “Novas: Tackling online dynamic video analytics with service adaptation at mobile edge servers,” *IEEE Trans. Comput.*, vol. 73, no. 9, pp. 2220–2232, Sep. 2024, doi: 10.1109/TC.2024.3416675.

[25] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, 2015, pp. 1–14.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[28] O. Vinyals et al., “StarCraft II: A new challenge for reinforcement learning,” 2017, *arXiv: 1708.04782*.

[29] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5398–5408.
[30] W. Yu, T. J. Chua, and J. Zhao, "User-centric heterogeneous-action deep reinforcement learning for virtual reality in the metaverse over wireless networks," *IEEE Trans. Wireless Commun.*, early access, pp. 1–14, May 23, 2023, doi: 10.1109/TWC.2023.3277226.
[31] J. Liu et al., "Maximum entropy heterogeneous-agent reinforcement learning," in *Proc. 12th Int. Conf. Learn. Representations*, Vienna, Austria, 2024, pp. 1–42.
[32] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1352–1361.
[33] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 1587–1596.
[34] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. 5th Int. Conf. Learn. Representations*, Toulon, France, 2016, pp. 1–13.
[35] D. Wen et al., "Task-oriented sensing, computation, and communication integration for multi-device edge AI," *IEEE Trans. Wireless Commun.*, vol. 23, no. 3, pp. 2486–2502, Mar. 2024.
[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[37] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. 13th Eur. Conf. Comput. Vis.*, Zurich, Switzerland, 2014, pp. 740–755.
[38] Y. Zhang et al., "ByteTrack: Multi-object tracking by associating every detection box," in *Proc. 17th Eur. Conf. Comput. Vis.*, Tel Aviv, Israel, 2022, pp. 1–21.
[39] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc. 2016 IEEE Int. Conf. Image Process.*, 2016, pp. 3464–3468.
[40] T. Fischer et al., "QDTrack: Quasi-dense similarity learning for appearance-only multiple object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 12, pp. 15380–15393, Dec. 2023.
[41] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A benchmark for multi-object tracking," 2016, *arXiv:1603.00831*.
[42] P. Lai et al., "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. 16th Int. Conf. Service-Oriented Comput.*, Hangzhou, China, 2018, pp. 230–245.
[43] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 2020–2033, Sep. 2019.
[44] J. Wang, J. Hu, G. Min, Q. Ni, and T. El-Ghazawi, "Online service migration in mobile edge with incomplete system information: A deep recurrent actor-critic learning approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 11, pp. 6663–6675, Nov. 2023.
[45] D. Xu et al., "Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption," in *Proc. Annu. Conf. ACM Special Int. Group Data Commun. Appl. Technol. Architectures Protoc. Comput. Commun.*, 2020, pp. 479–494.
[46] Nvidia, "Jetson modules, support, ecosystem, and lineup," Accessed: Jun. 24, 2024, 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules
[47] R. P. Ltd., "Raspberry pi 4 model b specifications–raspberry pi," Accessed: Apr. 08, 2025. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
[48] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, and A. Galis, "Efficient NFV-enabled multicasting in SDNs," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2052–2070, Mar. 2019.
[49] J. Kuba et al., "Trust region policy optimisation in multi-agent reinforcement learning," in *Proc. 10th Int. Conf. Learn. Representations*, 2022, pp. 1–27.

Hebin Huang received the BE degree from Beijing Information Science and Technology University, Beijing, China, in 2021. He is currently working toward the MS degree in electronic and information engineering with Guangxi University, Nanning, China, since 2022. His research interests include mobile edge cloud, network function virtualization, and deep learning.

Junbin Liang received the BE and MS degrees from Guangxi University, Nanning, China, and the PhD degree from Central South University, Changsha, China, in 2000, 2005, and 2010, respectively. He was a visiting professor with the University of British Columbia from 2019 to 2020. He is currently a professor with Guangxi University, Nanning, China. His research interests include sensor-cloud systems, fog computing, and distributed computing.

Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the PhD degree in computing science from the University of Glasgow, United Kingdom, in 2003. He is a professor of High Performance Computing and Networking with the Department of Computer Science, University of Exeter, United Kingdom. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, modelling and performance engineering.