







ExpertDRL: Request Dispatching and Instance Configuration for Serverless Edge Inference With Foundation Models

Yue Zeng , Junlong Zhou , *Member, IEEE*, Baoliu Ye , *Member, IEEE*, Zhihao Qu , *Member, IEEE*, Song Guo , *Fellow, IEEE*, Tianjian Gong, and Pan Li 

Abstract—The prevalence of the pre-training & fine-tuning paradigm enables machine learning models to quickly adapt to various downstream tasks by fine-tuning pre-trained foundation models (FMs), greatly facilitating various IoT applications that rely on model inference in dynamic edge serverless environments. Efficiently dispatching inference requests and configuring instances to batch inference requests can significantly enhance resource efficiency. However, existing serverless inference solutions are tailored for traditional models, make coarse-grained request dispatching and instance configuration decisions, fail to exploit the shared model backbone characteristics of the FM and capture delayed rewards in dynamic environments, and ignore communication latency between edge sites, resulting in high costs and constraint violations. In this paper, we leverage our insight that fine-grained batch inference requests can effectively exploit the shared model backbone feature of FM to save monetary costs. We propose an algorithm that incorporates deep reinforcement learning (DRL) and expert intervention for fine-grained request dispatching and instance configuration, where the DRL component outputs fractional solutions as guidance, while the expert intervention module integrates our insights—batching reduces monetary costs at the expense of increased inference latency, whereas higher configurations shorten inference latency. This module rounds fractional solutions and adjusts instance configurations to search for optimal solutions while satisfying constraints, with theoretical guarantees rigorously

proved. Finally, we conducted our experiments on an OpenFaas-based platform and simulator, and extensive trace-driven evaluation results show that ExpertDRL can save costs by up to 85.14% and improve request acceptance ratio by up to 26.93% , compared to the state-of-the-art solution.

Index Terms—Foundation model, serverless computing, edge computing, deep reinforcement learning.

I. INTRODUCTION

THE pretraining & fine-tuning paradigm are thriving, which differ from traditional machine learning (ML) training models from scratch, but instead fine-tunes the pre-trained FMs to adapt to various downstream tasks, enabling faster convergence and higher accuracy [1]. This advancement has greatly contributed to the proliferation of diverse IoT applications that rely on model inference, such as object detection, vehicle recognition, and machine translation [2]. For instance, Facebook processes hundreds of trillions of inference requests daily [3], with these applications typically demanding millisecond-level latency. Edge computing, which brings computational resources closer to users, facilitates these low-latency inference services. Meanwhile, serverless computing has gained popularity for ML services due to its autonomous resource scaling, ease of use, and pay-as-you-go cost model [4].

In serverless edge environments, inference services operate as follows. First, each distributed edge site (ES) pulls the well-trained model from the cloud server and stores it locally. These models are then loaded into containers and instantiated as function instances with specific resource configurations (e.g., memory, GPU) to handle each request generated by IoT devices. However, handling each inference request individually in serverless environments incurs frequent invocation costs and underutilizes the device's parallel computing potential [5]. An ideal solution is to batch process inference requests, which cache inference requests dispatched to a specific functional instance in a batch processing queue. Once the queue is full or the deadline for inference requests is approaching, these inference requests are fed into the functional instance to perform batched inference, which processes multiple requests in parallel, enables fewer instance invocation costs, and provides more opportunities to reduce memory and computational overhead with optimization techniques (e.g., memory reuse and graph optimization, etc.) [6].

Received 13 December 2024; revised 25 February 2025; accepted 17 March 2025. Date of publication 20 March 2025; date of current version 6 August 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62402226, Grant 62172224, and Grant 62441225, in part by the Natural Science Foundation of Jiangsu Province under Grant BK20241453 and Grant BK20220138, in part by Central Universities under Grant 30924010817 and Grant 30922010318, and in part by the Open Project Program of the National Key Laboratory for Novel Software Technology under Grant KFKT2024B17. Recommended for acceptance by C. Lin. (*Corresponding author: Junlong Zhou.*)

Yue Zeng, Junlong Zhou, and Tianjian Gong are with the Department of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: zengyue@njust.edu.cn; jlzhou@njust.edu.cn; gongtianjian19@163.com).

Baoliu Ye is with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China (e-mail: yeb@nju.edu.cn).

Zhihao Qu is with the Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing 211100, China (e-mail: quzhihao@hhu.edu.cn).

Song Guo is with the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (e-mail: songguo@cse.ust.hk).

Pan Li is with the College of Electronic and Information Engineering, Southwest University, Chongqing 400715, China (e-mail: 942907022@qq.com).

Digital Object Identifier 10.1109/TMC.2025.3553201

However, cost-effectively dispatching inference requests and configuring function instances with FMs is non-trivial. First, each inference request may have distinct latency and accuracy requirements, while each model-equipped function instance exhibits varying accuracy performance, and ESs differ in capacity, with varying latency to each source IoT device. Besides, under different configurations, each function instance incurs different cost overheads and inference latencies, and in dynamic serverless edge environments, current decisions can influence future ones, complicating the optimization of long-term rewards.

What's more, unlike traditional models, the FMs may share the model backbone, as shown in Fig. 1. In the traditional ML paradigm, as shown in Fig. 1(a), the neural network model is randomly initialized and all parameters are trained from scratch, resulting in models trained on different tasks having completely different model parameters. In contrast, in the pre-training & fine-tuning paradigm, a well-pretrained model on extensive datasets (called FM) is used to initialize the model, which is then fine-tuned to adapt to downstream tasks. As shown in Fig. 1(b), the FM is fine-tuned with prompt tuning [7] (a widely adopted fine-tuning method), where the model backbone is frozen, the model head is randomly initialized, and a few parameters (called prompt) are inserted, where only the model head and prompt are fine-tuned (less than 0.01% of the parameters [7]). Obviously, the FMs derived from the same FM and fine-tuned to different downstream tasks have exactly the same model backbone since it is always frozen; this is called model backbone sharing. Batch processing on shared model backbones provides new opportunities to reduce invocation overhead and memory consumption (detailed in Section III-B), while introducing higher complexity. All the above-mentioned factors make it challenging to determine how to dispatch incoming requests to the appropriate function instances on ESs and configure the corresponding resources and batch sizes in dynamic serverless edge environments. Although several cost-effective schemes [2], [5], [6] have been proposed for dispatching inference requests and configuring function instances, they are tailored for traditional models and fail to leverage the backbone-sharing characteristics of FMs. They only coarse-grainedly batch inference requests dispatched to the exact same model, neglecting to batch requests for similar FMs with identical model backbones, resulting in higher monetary costs (detailed in Section III-B). Moreover, these schemes [2], [5], [6] are customized for cloud environments without considering communication delays between ESs, and make one-shot decisions, which may lead to latency constraint violations and suboptimal solutions due to uncaptured delayed rewards¹ in dynamic environments (detailed in Section III-B).

In this paper, we investigate how to dispatch inference requests and configure function instances with FMs in dynamic edge scenarios, taking into account critical intrinsic factors such as batching, limited physical resources, model diversity, accuracy, and latency requirements. To capture all these characteristics, we establish mathematical connections between them and

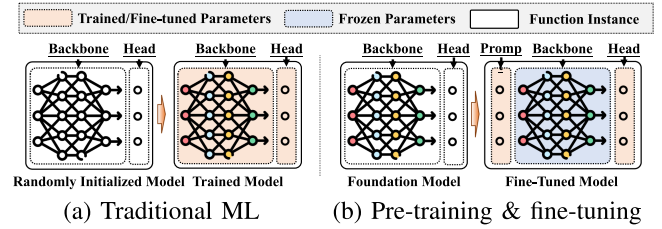


Fig. 1. Traditional ML paradigm versus pre-training & fine-tuning paradigm.

characterize it as an integer nonlinear program, and prove its NP-hardness. To tackle this problem, we propose a DRL-based solution tailored for the formalized problem while capturing delayed rewards in dynamic environments. Moreover, to avoid constraint violations and falling into bad local optima caused by stochastic exploration in DRL in large-scale solution spaces, we utilize DRL output fractional solutions as a guideline. By incorporating our key insights—larger batches reduce costs but increase latency, and configuration upgrades lower latency at the expense of higher costs—to round and adjust fractional solutions for constraint satisfaction and high-quality outcomes. Finally, we conducted our experiments on an OpenFaas-based platform and simulator, and extensive trace-driven evaluation results validate the optimality, superiority, practicability, and robustness of our algorithm in terms of cost, acceptance ratio, and execution time.

The main contributions of this paper are summarized as follows.

- To the best of our knowledge, we are the first to study how to dispatch inference requests and configure function instances with FMs in dynamic edge scenarios, taking into account critical intrinsic factors such as batching, limited physical resources, model diversity, accuracy, and latency requirements. We establish mathematical correlations between the factors with respect to our in-depth insights and analysis, formalized this problem as integer nonlinear programming, and proved its NP-hardness.
- We craft a fine-grained request dispatching and functional instance configuration scheme based on DRL and our insights, where DRL is responsible for outputting fractional solutions as guidelines, while an expert intervention algorithm based on our insights is responsible for rounding and adjusting the fractional solutions to avoid constraint violations and trapping in bad local optima. The approximation performance of this algorithm is rigorously proven.
- Extensive evaluation results driven by Alibaba trace show that, compared with the state-of-the-art solution, ExpertDRL can save costs by up to 85.14% and increase the request acceptance ratio by up to 26.93%.

The rest of the paper is organized as follows. Section II briefly introduces related work, Section III clarifies our motivation, Section IV formalizes the studied problem and analyzes its complexity. ExpertDRL is proposed and analyzed in Section V, and evaluated in Section VI. Section VII concludes this paper.

II. RELATED WORK

This section summarizes related work into three categories and discusses their limitations and differences relative to our work.

¹ *Delayed reward.* In dynamic environments, the current decision determines both the immediate reward and the next state of the environment [8]. The delayed reward is a metric used to measure the value of the next state, indicating the impact of the current decision on the future and helping to achieve higher long-term returns [9].

Inference Service Provision: A fundamental issue in enabling AI on IoT devices is how cost-effectively deliver low-latency, high-accuracy inference services for massive IoT devices [13]. Since different model structures and versions present different accuracy, resource cost, and latency performance, many works [2], [5], [6], [10], [11], [12], [13] have investigated how to efficiently select models to cost-effectively provide high-performance inference services. As pioneers, Crankshaw et al. [10] proposed an inference system with a two-tier architecture and identified that model selection is crucial for achieving low latency, high throughput, and high accuracy. On this basis, MArk [11], Jellyfish [12], EdgeAdaptor [13] were proposed to adaptively configure resources and select models to trade-off between resource cost, inference accuracy and latency. However, these schemes only comply with SLOs (e.g., accuracy and latency) in a best-effort manner and struggle to effectively cope with applications with differentiated performance requirements. As this concern, BATCH [5], INFaaS [2], INFless [6] are proposed to select appropriate model versions and configure computational resources to meet user-specified latency and accuracy requirements. The above studies greatly simplify the inference service provisioning, enabling higher resource efficiency and throughput. However, they only focus on traditional models, rather than FMs, missing the opportunities offered by FMs, such as fine-grained batching on shared model backbones.

Foundation Model: The FM refers to a model that has been pre-trained on a wide range of data and which can be adapted to various downstream tasks through fine-tuning [14]. In contrast to traditional models that are randomly initialized and trained from scratch, the FM can be quickly adapted to downstream tasks by fine-tuning a small subset of parameters and achieving higher accuracy with fewer training rounds [1], [15]. As a result, many fine-tuning schemes have been proposed, including head tuning [16], adapter tuning [17], prompt tuning [7], etc. In these schemes, the model backbone is frozen and only a small number of parameters are fine-tuned. This provides us with an opportunity to reuse the frozen model backbone, which has the potential to significantly reduce the resource footprint. Although these works have greatly exploited the advantages brought by the FM during model training, its impact on model inference has been ignored. To fill this gap, we investigate how to efficiently select models and configure resources for the FM specific to its properties.

DRL Solution: To cope with complex dynamic environments, DRL is applied to various applications, including games [18], [19], traffic optimization [20], network planning [21], network management [8], [9], [22], etc. In these mechanisms, DRL agents interact with the environment to learn the rules behind the complex environment, while capturing rewards in the dynamic environment, greatly improving the performance of these applications. However, random exploration by DRL agents can lead to frequent constraint violations, resulting in a significant loss of turnover. To prevent constraints from being violated, RuleDRL [9] incorporates heuristics to avoid violating constraints while providing approximate performance guarantees for service provisioning. The above advanced DRL schemes inspire us to design a well-designed DRL scheme for inference

service provision with FMs to cope with its high-dimensional constraints and complex dynamic environment.

Differing from existing work, this paper investigates a novel inference service provisioning problem with FMs, aiming to design a novel solution based on DRL, integrating our insights to adhere to constraints and converge faster to resource-cost efficient solutions. The differences between existing work and this paper are summarized in Table I.

III. PRIMER AND MOTIVATION

This section first details the principles and benefits of FMs and batch processing, then illustrates our motivation with examples.

A. Primer

Foundation Model: In the traditional ML paradigm, models are randomly initialized and trained from scratch on large-scale datasets for specific downstream tasks, resulting in substantial training overhead. In contrast, the pre-training & fine-tuning paradigm initializes the model with pre-trained models and fine-tunes it for various downstream tasks, which is warm-started rather than trained from scratch [15]. These models that are pre-trained on extensive datasets and can be adapted to diverse downstream tasks are called foundation models (FMs) [14]. Since FMs have been well pre-trained, they already has strong feature extraction capabilities and generalization, and can be adapt to various downstream tasks by fine-tuning only a small number of parameters, requiring few training epochs to achieve higher accuracy [1].

Batching: Batch processing is a unique feature of ML tasks, especially important for model inference. In traditional inference service systems, inference requests arrive individually and are immediately dispatched to corresponding function instances for one-by-one processing. However, processing inference requests one by one may result in low CPU/GPU utilization, which in turn leads to low throughput and high cost [5]. As an ideal solution, batch processing has been proposed and widely adopted, which caches inference requests that arrive one by one into a buffer queue, where the queue length is the batch size. Once the buffer is full or the inference requests in the queue approach the deadline, the inference requests in the buffer are fed into the corresponding model for batch processing [6].

In this way, multiple inference requests can be processed in batches, thereby reducing costs for the following two reasons. 1) Fewer invocations. In serverless computing, more invocations incur higher costs [26]. Batch processing handles multiple inference requests simultaneously, enabling fewer invocations and thereby lowering invocation expenses. 2) Less memory consumption. Existing ML frameworks integrate various optimization techniques (e.g., memory reuse and graph optimization, etc.), and batch processing provides more opportunities for these optimizations to reduce memory consumption. For example, in the ResNet-v2 model function instance, the minimum memory needed for a batch size of 1 is 1280 MB, while for a batch size of 20, it is only 1664MB [5]. This means that handling $20\times$ more requests raises memory consumption by merely 30% .

TABLE I
SUMMARY OF RELATED RESEARCH

Related Work	Inference Service Provision	Foundation Model	Delayed Reward	Accuracy	Latency
[10]–[13]	✓	✗	✗	✗	✗
[2], [5], [6]	✓	✗	✗	✓	✓
[1], [7], [14]–[17]	✗	✓	✗	✗	✗
[8], [9], [18]–[25]	✗	✗	✓	✗	✗
This paper	✓	✓	✓	✓	✓

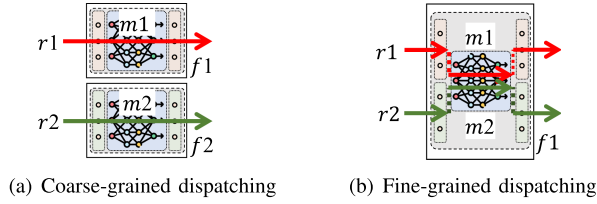


Fig. 2. An example used to demonstrate the advantages brought by FMs.

B. Motivation

In the following, we illustrate our two observations regarding the characteristics of FMs and dynamic serverless edge environments, which inspire our motivation.

Observation 1: The FM offers the opportunity for fine-grained batch processing, leading to fewer invocations and memory consumption.

Assuming that there are two fine-tuned FMs ($m1$, $m2$) that are derived from the same FM and have been fine-tuned on two different downstream tasks. This implies that both FMs share the same model backbone. Besides, each downstream task is associated with an inference request, denoted as $r1$ and $r2$. In traditional request dispatching schemes (e.g., INFless [6]), requests are coarse-grained batched, where only requests dispatched to the exact same model can be batched. In this case, as shown in Fig. 2(a), requests $r1$ and $r2$ are respectively dispatched to function instances $f1$ and $f2$ with FM $m1$ and FM $m2$, and fed into the corresponding models for forward propagation for inference. In this case, the coarse-grained request dispatch ignores the fact that FM $m1$ and FM $m2$ share the same model backbone and fails to take advantage of this new feature brought by FM.

Fig. 2(b) shows a fine-grained request dispatching scheme, where one function instance loads both FMs $m1$ and $m2$ simultaneously, sharing the model backbone.² When requests $r1$ and $r2$ are fed into the instance, the two requests are fed into the prompts of FMs $m1$ and $m2$, respectively, and are batched into the model backbone and finally fed into their respective model heads. This enables the benefits of batching for FMs, such as fewer instance innovations and less memory consumption

² Note that loading FMs $m1$ and $m2$ in one instance add only negligible overhead compared to loading either model $m1$ or $m2$, since only a small number of fine-tuned parameters need to be loaded additionally, it accounts for less than 0.01% of the total parameters [7].

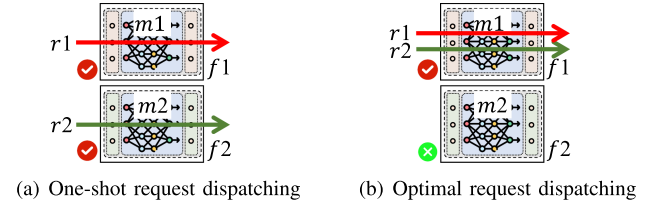


Fig. 3. An example used to demonstrate the shortcomings of one-shot solutions.

overhead, as mentioned above. Since both requests can be fed into one function instance simultaneously, this involves only one invocation (instead of two for each request), and batch processing on the model backbone enables less memory consumption.

Observation 2: The existing one-shot decision fails to capture long-term rewards, leading to suboptimal dispatch decisions.

In dynamic edge environments, inference requests arrive sequentially, and current dispatching decisions can impact future ones due to potential batch processing. However, existing request dispatching schemes make one-shot decisions, which fail to capture delayed rewards in dynamic environments, leading to suboptimal solutions.

For example, as shown in Fig. 3, there are two inference requests, $r1$ and $r2$, that arrive successively, with accuracy requirements of 70% and 75%. There are also two FMs, $m1$ and $m2$, loaded in function instances $f1$ and $f2$, which have accuracies of 72% and 76%, respectively. With the same resource configuration, their latencies are 2.75 ms and 2.87 ms for a batch size of 1, and 2.85 ms and 2.96 ms for a batch size of 2.³ In the one-shot dispatching scheme, as shown in Fig. 3(a), request $r1$ arrives and is dispatched to function instance $f1$, which results in less cost due to shorter inference latency. When request $r2$ arrives, it is dispatched to function instance $f2$ due to its accuracy requirement. This results in a 2-unit invocation cost and a total inference latency of 5.62 ms (2.75 ms on function instance $f1$ and 2.87 ms on function instance $f2$). In the optimal request dispatching scheme, as shown in Fig. 3(b), both requests $r1$ and $r2$ are dispatched to function instance $f1$, which results in a one-unit invocation cost and a total inference latency of 2.96 ms

³ The parameter settings are based on reported results [27] and our experimental results deployed on OpenFaas [28].

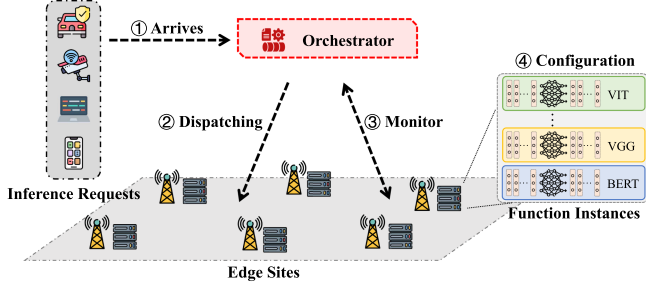


Fig. 4. System framework.

in the same configuration, which clearly consumes less monetary cost due to fewer invocations and shorter function durations.

These two observations inspired us to design an inference request dispatching and instance configuration scheme tailored for the FM, enabling fine-grained batch processing while capturing delayed rewards in dynamic edge environments.

IV. FORMULATION

This section first introduces the system model, problem definition, then formalizes the studied problem, and finally analyzes its hardness and challenges.

A. System Model

As shown in Fig. 4, our inference service system consists of four components, which can be modeled as follows. The notations to be used are listed in Table II.

Inference Request: Inference requests are generated by massive IoT devices, and these requests arrive one by one. Let i denote the i -th inference request, and \mathcal{I} denote the set of all inference requests. Each request i demands specific SLO performance requirements, such as the latency not exceeding \mathcal{L}^i and the inference accuracy being no lower than \mathcal{A}^i . Besides, there is a set of downstream tasks \mathcal{D} , let D_d^i denote whether inference request i belongs to the downstream task $d \in \mathcal{D}$.

Service Orchestrator: The service orchestrator is owned by the inference service provider and is responsible for orchestrating arriving requests and resources on the infrastructure. Once an inference request arrives, it determines which ES to dispatch the request to, and decides on servicing the request using which FM function instance $f_{m,o}^e$, along with the corresponding configuration.

Infrastructure: In the infrastructure, there is a set of ESs \mathcal{E} , and each ES e has memory capacity R^e and GPU resource capacity G^e . The communication time from the source of each inference request i to ES e is denoted as $t_{comm}^{i,e}$. In commercial serverless platforms, each function instance needs to be configured with a specific memory size (e.g., ranging from 1024 MB to 10240 MB), and the CPU quota is allocated in proportion to the configured memory size [6]. Each function instance can also be configured with additional GPU resources to accelerate computation. The monetary costs per unit of memory resources and GPU resources are denoted as κ and ζ , respectively. Besides, each function instance invocation incurs a σ -unit monetary cost.

TABLE II
SUMMARY OF NOTATIONS

Inputs	Descriptions
\mathcal{I}	Set of inference requests
\mathcal{E}	Set of ESs
\mathcal{M}	Set of FMs
\mathcal{O}	Set of optimization methods for models
\mathcal{D}	Set of downstream tasks
\mathcal{B}	Set of batch size settings
\mathcal{R}	Set of memory configurations
\mathcal{G}	Set of GPU configurations
\mathcal{K}	Total monetary cost generated by all inference requests
$\mathcal{K}_{m,o}^e$	Monetary cost generated by function instance $f_{m,o}^e$
\mathcal{A}^i	Accuracy requirement of inference request i
\mathcal{L}^i	Latency requirement of inference request i
D_d^i	Binary constant indicates whether inference request i belongs to downstream task d
R^e	Memory capacity on ES e
G^e	GPU capacity on ES e
$M_{m,o,d}$	Fine-tuned FM with model structure m and optimization method o fine-tuned on downstream task d
$\overline{M}_{m,o}$	Backbone network with model structure m and optimization method o
$a_{m,o,d}$	Inference accuracy of fine-tuned FM $M_{m,o,d}$
$f_{m,o}^e$	Function instance with backbone model $\overline{M}_{m,o}$ on ES e
$\Gamma_{m,o}^e$	Duration of function instance $f_{m,o}^e$
t_{start}^i	Start time when request i is generated by IoT device
t_{dead}^i	Deadline for request i
t_{arrive}^i	Time when request i arrives at the corresponding dispatched function instance
$t_{comm}^{i,e}$	Communication time from the source of inference request i to ES e
$\hat{t}_{m,o}(\bullet)$	Computing time consumed by model m with optimization o and configuration \bullet to execute inference tasks
κ	Monetary cost per unit of memory per unit of time
ζ	Monetary cost per unit of GPU per unit of time
σ	Monetary cost per invocation
Variables	Descriptions
$x_{m,o}^{i,e}$	Binary variable indicating whether to dispatch inference request i to function instance $f_{m,o}^e$
$y_{m,o}^{i,e}$	Binary variable indicating whether function instance $f_{m,o}^e$ is invoked
$b_{m,o}^e$	Variable indicating the batch size configured for function instance $f_{m,o}^e$
$r_{m,o}^e$	Variable indicating the memory resources configured for function instance $f_{m,o}^e$
$g_{m,o}^e$	Variable indicating the GPU resources configured for function instance $f_{m,o}^e$

Fine-tuned FM: In ML, various models are designed to serve different tasks and exhibit different performances, and the set of these models is denoted as \mathcal{M} . Besides, there is a set \mathcal{O} of optimization methods (e.g., layer fusion and quantization, etc. [2]) that can be used to optimize these models to reduce their sizes significantly, with only a minor trade-off in accuracy. These optimized models are pre-trained on cloud data centers and are fine-tuned for a set \mathcal{D} of downstream tasks.

The fine-tuned FMs are cached on ESs to provide low-latency inference services to IoT users. Each fine-tuned FM, which is based on a model structure $m \in \mathcal{M}$, an optimization method $o \in \mathcal{O}$, and fine-tuned for a downstream task $d \in \mathcal{D}$, can be denoted as $M_{m,o,d}$. This model yields accuracy $a_{m,o,d}$ on the downstream task d . Notably, fine-tuned FMs that share the same model structure m and optimization method o differ only slightly in a small number of fine-tuned parameters, and they have a common model backbone $\overline{M}_{m,o}$. Each function instance $f_{m,o}^e$ on ES e loads all fine-tuned FMs that share the same model

backbone $\overline{M}_{m,o}$.⁴ When function instance $f_{m,o}^e$ is equipped with configurations $(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e)$, it results in an inference delay $\hat{t}_{m,o}^e(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e)$, where $b_{m,o}^e$, $r_{m,o}^e$, and $g_{m,o}^e$ denote its configured memory size, GPU size, and batch size, respectively. Finally, the set of batch size configurations is denoted as \mathcal{B} , and the set of allowed memory configurations and GPU configurations are denoted as \mathcal{R} and \mathcal{G} , respectively.

B. Problem Definition

In our research, we investigate how to dispatch inference requests and configure function instances in resource-constrained edge serverless environments to meet their accuracy and latency requirements, while minimizing monetary costs. This problem is called the Inference Service Dispatching and Configuration problem with Foundation Model (ISDC-FM), and it is formally defined as follows.

Definition 1: Given a set \mathcal{E} of ESs and a set of inference service requests \mathcal{I} from IoT devices, where each ES e has memory capacity R^e and GPU capacity G^e . Besides, each inference request i demands specific SLO performance, such as the experienced latency not exceeding \mathcal{L}^i and the inference accuracy not falling below \mathcal{A}^i . Moreover, there is a set of fine-tuned FMs; each fine-tuned FM $M_{m,o,d}$ with an inference accuracy denoted by $a_{m,o,d}$. There is also a set of function instances, where each function instance $f_{m,o}^e$ loads all fine-tuned FMs with model structure m and optimization method o , and they share the same model backbone $\overline{M}_{m,o}$. The functional instance $f_{m,o}^e$, equipped with configuration $(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e)$, has an inference delay of $\hat{t}_{m,o}^e(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e)$. ISDC-FM problem aims to determine which function instance each inference request is dispatched to and the configuration of that function instance to meet the latency and accuracy requirements of that inference request under limited resources while minimizing resource cost.

C. Problem Formulation

ISDC-FM problem aims to minimize monetary costs while being subject to limited capacity, latency, and accuracy requirements. Next, we analyze and formalize cost, accuracy, latency, and capacity models.

Cost Model: In serverless computing environments, the monetary cost consumed by each function instance $f_{m,o}^e$ depends on its duration, configured memory, the monetary cost of each function invocation, and the monetary cost per resource per time unit [26].⁵ Then, the monetary cost of function instance $f_{m,o}^e$ can be expressed as

$$\mathcal{K}_{m,o}^e = \kappa \Gamma_{m,o}^e r_{m,o}^e + \sigma, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, \quad (1)$$

⁴ Since loading FMs with a shared model backbone only loads the fine-tuned parameters additionally (less than 0.01% of the total parameters [7]), which introduces negligible overhead.

⁵ Although storing inference requests in the wait queue may introduce additional storage monetary costs, it is negligible. This is because the size of the inference request is small and the cost of storage resources is significantly lower (less than 0.25%) than the cost of memory resources required to perform inference tasks [26].

where $y_{m,o}^e$ is a binary variable indicating whether function instance $f_{m,o}^e$ is invoked. $\Gamma_{m,o}^e$ denotes the duration of the functional instance $f_{m,o}^e$, and $r_{m,o}^e$ denotes the memory size configured for it. Similarly, further considering the monetary cost incurred for configuring GPU resources for function instances, the monetary cost of function instance $f_{m,o}^e$ can be expressed as

$$\mathcal{K}_{m,o}^e = \kappa \Gamma_{m,o}^e r_{m,o}^e + \zeta \Gamma_{m,o}^e g_{m,o}^e + \sigma y_{m,o}^e, \quad \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, \quad (2)$$

where $g_{m,o}^e$ denotes the GPU resources configured for function instance $f_{m,o}^e$. Then, the total cost generated by all function instances on all ESs can be expressed as

$$\mathcal{K} = \sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} \sum_{e \in \mathcal{E}} \mathcal{K}_{m,o}^e. \quad (3)$$

Accuracy Constraints: Once an inference request i is dispatched to function instance $f_{m,o}^e$, it will be served by the corresponding fine-tuned FM $M_{m,o,d}$. The inference accuracy of this fine-tuned FM should not be lower than the accuracy requirement specified by this request. This can be expressed as

$$\sum_{e \in \mathcal{E}} \sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} \sum_{d \in \mathcal{D}} a_{m,o,d} D_d^i x_{m,o}^{i,e} \geq \mathcal{A}^i, \forall i \in \mathcal{I}, \quad (4)$$

where $x_{m,o}^{i,e}$ is a binary variable indicating whether to dispatch request i to function instance $f_{m,o}^e$.

Latency Constraints: After an inference request i is generated by the IoT device at starting time t_{start}^i , the request takes communication time $t_{comm}^{i,e}$ to arrive at the dispatched function instance $f_{m,o}^e$. Then, the time for the inference request i to arrive at the dispatched function instance can be expressed as

$$t_{arrive}^i = t_{start}^i + \sum_{e \in \mathcal{E}} \sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} x_{m,o}^{i,e} t_{comm}^{i,e}, \forall i \in \mathcal{I}. \quad (5)$$

Besides, the deadline for a request i depends on its start time and the maximum latency \mathcal{L}^i it tolerates. Then, the deadline for request i can be expressed as

$$t_{dead}^i = t_{start}^i + \mathcal{L}^i, \forall i \in \mathcal{I}. \quad (6)$$

Moreover, after all the requests dispatched to the function instance $f_{m,o}^e$ arrive, these requests can be batched and fed into the function instance for processing. The time for each request to complete processing should not exceed the deadline, which can be expressed as

$$\max\{t_{arrive}^{i'} x_{m,o}^{i',e}, \forall i' \in \mathcal{I}\} + \Gamma_{m,o}^e \leq t_{dead}^i, \forall i \in \mathcal{I}. \quad (7)$$

Also, the duration of function instance $f_{m,o}^e$ is dominated by the computation time consumed for executing the inference tasks,⁶ which can be expressed as

$$\Gamma_{m,o}^e = \hat{t}_{m,o}^e(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e), \forall m \in \mathcal{M}, o \in \mathcal{O}. \quad (8)$$

⁶ The duration of a functional instance is composed of startup time and computation time, with startup time being negligible. As it has been reported [29] that cold starts occur in less than 1% of workloads in production AWS environments, and the latency for warm-start latency is negligible.

Capacity Constraints: The resources consumed by all function instances on each ES $e \in \mathcal{E}$ should not exceed its capacity [6]. This can be expressed as

$$\sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} r_{m,o}^e \leq R^e, \forall e \in \mathcal{E}, \quad (9)$$

$$\sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} g_{m,o}^e \leq G^e, \forall e \in \mathcal{E}. \quad (10)$$

Besides, the inference requests dispatched to each function instance should not exceed the batch size configured for it, which can be expressed as

$$\sum_{i \in \mathcal{I}} x_{m,o}^{i,e} \leq b_{m,o}^e, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}. \quad (11)$$

Moreover, each inference request i must be dispatched to exactly one function instance $f_{m,o}^e$ for inference, which can be expressed as

$$\sum_{e \in \mathcal{E}} \sum_{m \in \mathcal{M}} \sum_{o \in \mathcal{O}} x_{m,o}^{i,e} = 1, \forall i \in \mathcal{I}. \quad (12)$$

Instance Invocation Constraints: A function instance needs to be invoked to handle inference requests whenever any inference request is dispatched to that instance. This can be represented as

$$y_{m,o}^e = \begin{cases} 1, & \text{if } \sum_{i \in \mathcal{I}} x_{m,o}^{i,e} > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (13)$$

Based on the above analysis, ISDC-FM problem can be formalized as

$$\begin{aligned} & \min_{x_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e} \mathcal{K} \\ \text{s.t. } & (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12), (13), \end{aligned}$$

$$\begin{aligned} & x_{m,o}^{i,e} \in \{0, 1\}, b_{m,o}^e \in \mathcal{B}, r_{m,o}^e \in \mathcal{R}, g_{m,o}^e \in \mathcal{G}, \\ & \forall i \in \mathcal{I}, e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}. \end{aligned}$$

D. Problem Complexity

1) Hardness Result: Theorem 1: ISDC-FM problem is NP-hard.

Proof: This theorem can be proved by showing that a special case of the ISDC-FM problem is equivalent to a classical NP-hard problem, i.e., the 0-1 knapsack problem [30]. We assume $\kappa = 1$, $\zeta = 0$, $\sigma = 0$, $|\mathcal{E}| = 1$, $|\mathcal{O}| = 1$, $|\mathcal{D}| = 1$, $a_{m,o,d} = 1$, $t_{start}^i = 0$, $t_{comm}^{i,e} = 0$, $\mathcal{L}^i = \infty$, $G^e = 0$. In this case, our problem is equivalent to the 0-1 knapsack problem [30]. For brevity, the formal proof is omitted. ■

The above theorem excludes the polynomial time algorithm for ISDC-FM problem, even when all inference requests are accepted, unless $P=NP$.

2) Challenges: Despite the existence of rule-based approximation algorithms [31] for efficiently addressing the 0-1 knapsack problem, these algorithms fail to effectively handle our ISDC-FM problem due to the following reasons.

1) High-dimensional Constraints: The principle behind the rule-based approach is to leverage experts' insights to explore

high-quality solutions under constraints, which can effectively handle simple cases with few constraints. However, ISDC-FM problem involves high-dimensional constraints, including latency constraints, accuracy constraints, instance invocation constraints, and capacity constraints. Dealing with such a complex problem makes it challenging to gain effective insights to guide the exploration of high-quality solutions, even for domain experts.

2) Delayed Reward: These rule-based schemes are one-shot solutions that assume that all requests are known or make near-optimal or optimal decisions only for the current state. However, in real-world environments, inference requests arrive one by one and cannot be accurately predicted. Moreover, one-shot decisions made only for the current state may be sub-optimal due to the fact that current decisions may affect future decisions, bringing delayed rewards, and these solutions fail to capture delayed rewards.

DRL is crafted to tackle multi-stage decision-making problems that can effectively handle sequentially arriving inference requests in dynamic environments, capture the delayed rewards therein, and extract the hidden complex rules behind them via neural networks. Next, we devise a DRL-based solution for ISDC-FM problem.

V. ALGORITHM DESIGN

This section first discusses challenges in applying DRL to address ISDC-FM problem, then designs a DRL-based solution that incorporates expert insights, and finally analyzes its theoretical properties.

A. Algorithm Framework

1) Motivation for Algorithm Framework: Although DRL excels at dealing with multi-stage decision problems and can capture its delayed rewards to optimize long-term rewards, it faces the following challenges to cope with the ISDC-FM problem.

The first challenge is how to deal with feasibility. DRL explores the optimal solution in the solution space by stochastic exploration; however, this may lead to frequent constraint violations and rejections of requests. This issue is particularly pronounced in the ISDC-FM problem, which is subject to high-dimensional constraints, including accuracy, latency, capacity, and instance invocation constraints, which results in the solution space filled with numerous infeasible solutions. Although punishing actions that violate constraints can alleviate constraint violations, frequent constraint violations still occur (as verified in the results of Fig. 11). To address this challenge, we designed an algorithm that combines DRL with a rounding scheme, where the DRL is responsible for generating fractional solutions to guide the decision-making, and the rounding scheme rounds the fractional solutions while checking all the constraints to prevent constraint violations.

The second challenge is how to deal with optimality. DRL explores the optimal solution in the solution space of ISDC-FM problem through random exploration and gradient descent. However, the ISDC-FM problem is an integer nonlinear programming problem with high-dimensional constraints, resulting

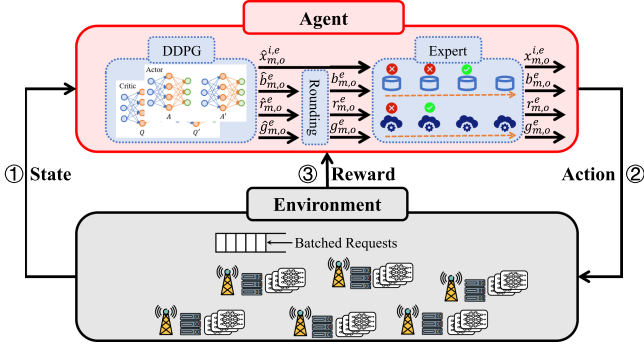


Fig. 5. Algorithm framework.

in numerous local optima within the solution space. Besides, the infeasible solutions in the solution space further complicate the exploration of the optimal solution. To address this challenge, we design a rule-based scheme to explore the optimal solution, which incorporates our insights derived from experimental results.

2) *Workflow in Algorithm Framework*: The algorithmic framework we adopted, depicted in Fig. 5, is DRL-based and consists of two main entities: the DRL agent and the environment, where the environment involves all infrastructure, and the agent is responsible for collecting states from the environment, takes actions, receiving corresponding rewards. The algorithm we designed is implemented in the DRL agent called Expert-DRL, which consists of a DDPG-based DRL algorithm and an expert insight-based rounding algorithm. The DDPG algorithm takes actions based on states discovered from the environment and outputs fractional solutions. Then, these fractional solutions are fed into an expert insight-based rounding algorithm, which rounds the fractional solution or further improves the rounded decision and generates the final action to be executed in the environment.

B. Algorithm Design

1) *RL Model*: RL models can be denoted as a triple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, where \mathcal{S} , \mathcal{A} , and \mathcal{R} denote state, action, and reward, respectively. To handle ISDC-FM problem, they can be devised as follows.

State: As shown in the Fig. 5, the state records information in the environment, including detailed information about the incoming inference request, inference requests that have been dispatched, model instances and ESs. Then, the state can be modeled as $\mathcal{S} = \langle \mathcal{E}, \mathcal{M}, \mathcal{O}, \mathcal{D}, \mathcal{B}, \mathcal{R}, \mathcal{G}, \mathcal{R}^e, \mathcal{G}^e, a_{m,o,d}, t_{comm}^{i,e}, t_{start}^i, \mathcal{A}^i, \mathcal{L}^i, \mathcal{D}_d^i, \kappa, \zeta, \sigma, x_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e, t_{comm}^{i,e}, t_{start}^i, \mathcal{L}_i, \hat{t}_{m,o}(b', r', g'), \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, d \in \mathcal{D}, b' \in \mathcal{B}, r' \in \mathcal{R}, g' \in \mathcal{G}, i' \in \mathcal{I}' \rangle$, where \mathcal{I}' denotes the set of existing inference requests. Obviously, the state size is $|\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}| + |\mathcal{I}'||\mathcal{E}||\mathcal{M}||\mathcal{O}| + 2|\mathcal{E}||\mathcal{M}||\mathcal{O}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}| + |\mathcal{I}'||\mathcal{E}| + 3|\mathcal{E}| + 2|\mathcal{D}| + |\mathcal{M}| + |\mathcal{O}| + |\mathcal{B}| + |\mathcal{R}| + |\mathcal{G}| + 2|\mathcal{I}'| + 4$.

In this case, the size of the state increases with the number of arriving requests (e.g., $|\mathcal{I}'|$), which conflicts with the fact

that the number of neurons in the input layer is fixed in the DRL. Next, we re-model the state as follows. As shown in (7), and (11), the dispatch and configuration update of the currently arriving request is only affected by the number of requests dispatched on each instance and the latest arrival time of the existing requests, while other details are irrelevant. Therefore, the state can be remodeled as $\mathcal{S} = \langle \mathcal{E}, \mathcal{M}, \mathcal{O}, \mathcal{D}, \mathcal{B}, \mathcal{R}, \mathcal{G}, \mathcal{R}^e, \mathcal{G}^e, a_{m,o,d}, t_{comm}^{i,e}, t_{start}^i, \mathcal{A}^i, \mathcal{L}^i, \mathcal{D}_d^i, \kappa, \zeta, \sigma, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e, \hat{t}_{m,o}(b', r', g'), \mathcal{T}_{m,o}^e, \mathcal{N}_{m,o}^e, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, d \in \mathcal{D}, b' \in \mathcal{B}, r' \in \mathcal{R}, g' \in \mathcal{G} \rangle$, where $\mathcal{T}_{m,o}^e$ and $\mathcal{N}_{m,o}^e$ represent the latest arrival time and the number of requests dispatched to instance $f_{m,o}^e$, respectively, at the current time, defined as $\mathcal{T}_{m,o}^e = \max\{t_{arrive}^i x_{m,o}^{i,e}, \forall i \in \mathcal{I}'\}$, $\mathcal{N}_{m,o}^e = \sum_{i \in \mathcal{I}'} x_{m,o}^{i,e}$. Consequently, the state size fixed to $|\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}| + 5|\mathcal{E}||\mathcal{M}||\mathcal{O}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}| + 4|\mathcal{E}| + 2|\mathcal{D}| + |\mathcal{M}| + |\mathcal{O}| + |\mathcal{B}| + |\mathcal{R}| + |\mathcal{G}| + 6$.

Action: The action involves decisions on dispatching inference requests and configuring resources (e.g., $x_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e$). Then, the action can be modeled as $\mathcal{A} = \langle x_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O} \rangle$. Clearly, the action size is $4|\mathcal{E}||\mathcal{M}||\mathcal{O}|$.

Reward: The reward is feedback from the environment, indicating the quality of the action. In ISDC-FM problem, the objective is to minimize monetary cost while adhering to all constraints. Naturally, the higher the monetary cost, the smaller the reward, while violating constraints leads to penalties. Thus, the reward can be modeled as $\mathcal{R} = -(\mathcal{K} - \mathcal{K}') - \theta \rho^i$, where \mathcal{K}' denotes the total monetary cost before request i arrives, and ρ^i is a binary variable indicating whether any constraints are violated ($\rho^i = 1$ if constraints are violated, and 0 otherwise). Besides, θ is the penalty factor; the higher its value, the more constraint violation is more intolerable.

2) *DRL Algorithm*: As modeled above, in the RL model, the state is large-scale continuous, while the action is large-scale discrete, which is exactly what DDPG [32] excels at. Therefore, DDPG is adopted in the RL agent. After the state, action, and reward are specified, the DDPG algorithm is determined, and its details are omitted; please refer to [32] for details.

However, DDPG may fall into poor local optima due to the abundance of local optima in the ISDC-FM problem with large solution spaces and multiple constraints. Inspired by the idea of relaxed rounding schemes [33], it first solves the relaxed problem to obtain a fractional solution with good optimality and uses this fractional solution to guide the final decision, aiming for a near-optimal solution that satisfies the constraints. We let DDPG output the fractional solution and round the fractional resource configuration decision while retaining the fractional dispatch decision to guide subsequent decisions. The reason for keeping the fractional dispatch decision is due to its complexity—it may influence future decisions (as illustrated in Section III-B), and it is retained to guide the following dispatch decisions. The detailed DRL pseudocode is shown in Algorithm 1.

3) *Expert Intervention Algorithm*: As mentioned above, the solutions output by DRL may violate the constraints and fall into poor local optima. Therefore, next, we guide decisions based on

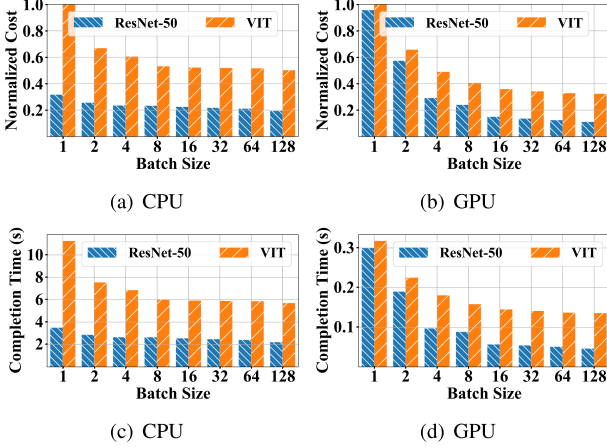


Fig. 6. Batch size versus completion time & normalized cost.

Algorithm 1: DRL Algorithm.**Input:** Parameters in ISDC-FM.**Output:** $\hat{x}_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e$: decision variables.

- 1: $\hat{x}_{m,o}^{i,e}, \hat{b}_{m,o}^e, \hat{r}_{m,o}^e, \hat{g}_{m,o}^e \leftarrow$ call DDPG;
- 2: $b_{m,o}^e, r_{m,o}^e, g_{m,o}^e \leftarrow$ round $\hat{b}_{m,o}^e, \hat{r}_{m,o}^e$, and $\hat{g}_{m,o}^e$ to the nearest integer;
- 3: **return** $\hat{x}_{m,o}^{i,e}, b_{m,o}^e, r_{m,o}^e, g_{m,o}^e$

our insights, avoiding poor suboptimal solutions in the process of approaching the optimal solution, and adjusting constraint-violating solutions to obtain feasible ones.

For model inference tasks, function instances with different configurations (e.g., batch size, memory, GPU) may yield different performance, such as inference latency, monetary cost, etc. [5]. To explore their relationships, we conducted experiments in an OpenFaas-based serverless computing environment and gained the following two insights.

Insight 1: Batch processing helps to save money cost in serverless environments either executed on CPU or GPU.

Fig. 6 shows the normalized cost, completion time and inference time for the VIT and ResNet-50 models under different batch size settings. As shown in Fig. 6(a) and (b), as the batch size increases, the normalized cost of executing all 128 inference tasks on both models gradually decreases, whether on the CPU or GPU. This is because larger batch sizes provide more opportunities for ML optimization techniques (e.g., memory reuse, graph optimization, etc.) to reduce computational overhead while improving resource utilization, thereby shortening the time to complete all inference tasks, as shown in Fig. 6(c) and (d), thereby saving costs. Besides, larger batch sizes result in fewer function instance invocations, which further reduces the cost.

The above insights inspire us to prioritize dispatching requests to instances that have already been dispatched with requests, which can batch process these requests instead of launching new instances to handle them. Besides, the fractional solution output by DDPG about request dispatch is a rating score for

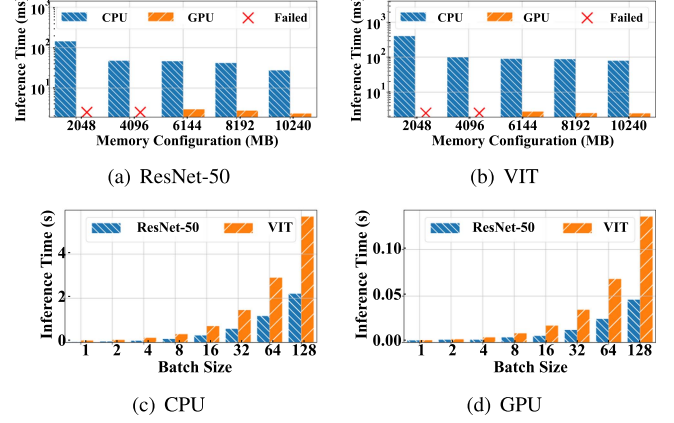


Fig. 7. Memory size & batch size versus inference time. × means out-of-memory error.

each dispatch decision, where a higher score indicates a higher preference for dispatching requests to that instance. Therefore, we define the dispatch priority of each function instance as

$$\mathcal{P}_{m,o}^e = \alpha \left[\frac{\sum_{i \in \mathcal{I}'} x_{m,o}^{i,e}}{|\mathcal{I}'|} \right] + \tilde{x}_{m,o}^{i,e}, \quad (14)$$

where \mathcal{I}' denotes the set of dispatched requests, $\lceil \frac{\sum_{i \in \mathcal{I}'} x_{m,o}^{i,e}}{|\mathcal{I}'|} \rceil = 1$, when function instance $f_{m,o}^e$ is dispatched with any request; otherwise, $\lceil \frac{\sum_{i \in \mathcal{I}'} x_{m,o}^{i,e}}{|\mathcal{I}'|} \rceil = 0$. Besides, α is a constant parameter, and $\alpha \gg 1$ (note that $\tilde{x}_{m,o}^{i,e} \leq 1$).

The motivation behind the above formula is as follows. The first term is used to ensure that instances that have already been dispatched with requests are prioritized for dispatching inference requests, which enables more batch processing. Besides, the second term is used to ensure that instances with higher scores in DDPG are prioritized for dispatching requests, which may result in higher immediate and delayed rewards.

Insight 2: Larger memory configurations or equipping GPU can shorten model inference time, while larger batch sizes extend inference time.

Fig. 7 shows the inference time of the VIT and ResNet-50 models under different memory size and batch size settings. As shown in Fig. 7(a) and (b), inference time gradually decreases as memory configuration increases. This reduction occurs because CPU resources are allocated in proportion to memory size, meaning that larger memory configurations bring higher CPU allocations and enhanced computational power, resulting in shorter inference times. Besides, equipped with GPU can shorten the inference time by orders of magnitude due to its powerful parallel processing capabilities. The fork in the figure indicates that the function instance failed to be launched due to insufficient memory.⁷ Moreover, larger batch sizes may result in longer inference times, as shown in Fig. 7(c) and (d). This is because larger batch sizes lead to higher computational and memory

⁷ This occurs mainly because enabling GPU in serverless environments involves loading additional drivers and dependencies, such as CUDA and cuDNN, which incur substantial memory overhead.

Algorithm 2: Expert Intervention Algorithm.**Input:** Parameters in RLSP.**Output:** $\mathbb{A}, x_{m,o}^{i,e}, y_{m,o}^e, r_{m,o}^e, g_{m,o}^e$: decision variables.

```

1:  $\mathbb{A} \leftarrow \text{False}$ ;
2:  $\mathbb{L} \leftarrow \{(e, m, o), \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}\}$ ;
3: Calculate  $\mathcal{P}_{m,o}^e$  based on (14);
4: Sort  $\mathbb{L}$  in descending order according to  $\mathcal{P}_{m,o}^e$ ;
5: Sort  $\mathcal{R}, \mathcal{G}$  in descending order;
6: for  $(e, m, o) \in \mathbb{L}$  do
7:    $b', r', g' \leftarrow b_{m,o}^e, r_{m,o}^e, g_{m,o}^e$ ;
8:    $x_{m,o}^{i,e} \leftarrow 1$ ;
9:    $b_{m,o}^e \leftarrow \min\{b'' \in \mathcal{B} | b'' \geq \sum_{e' \in \mathcal{E}, m' \in \mathcal{M}, o' \in \mathcal{O}} x_{m',o'}^{i,e'}\}$ ;
10:   $\mathcal{R}' \leftarrow \{r \in \mathcal{R} | r \geq r_{m,o}^e\}$ ;
11:   $\mathcal{G}' \leftarrow \{g \in \mathcal{G} | g \geq g_{m,o}^e\}$ ;
12:  for  $r \in \mathcal{R}'$  do
13:    for  $g \in \mathcal{G}'$  do
14:       $r_{m,o}^e \leftarrow r$ ;
15:       $g_{m,o}^e \leftarrow g$ ;
16:      if (4), (7), (9), (10) is satisfied then
17:         $\mathbb{A} \leftarrow \text{True}$ ;
18:        return  $x_{m,o}^{i,e}, y_{m,o}^e, r_{m,o}^e, g_{m,o}^e$ ;
19:      end if
20:    end for
21:  end for
22:  /*Restore to the original configuration.*/
23:   $x_{m,o}^{i,e} \leftarrow 0$ ;
24:   $b_{m,o}^e, r_{m,o}^e, g_{m,o}^e \leftarrow b', r', g'$ ;
25: end for
26: return  $\mathbb{A}, x_{m,o}^{i,e}, y_{m,o}^e, r_{m,o}^e, g_{m,o}^e$ 

```

loads, which may exceed the parallel processing capabilities of the hardware and induce data transfer and I/O bottlenecks.

This inspires us to correct decisions that violate latency constraints by upgrading memory or GPU configurations, while avoiding unnecessary larger batch size configurations due to higher inference latency.

Based on the above insights, we designed the expert intervention algorithm, detailed in Algorithm 2. This algorithm begins by initializing variables, including the decision on whether to accept a request and a set of candidate dispatch instances. Then, we assign a priority to each candidate dispatch instance according to (14) and sort the memory and GPU resource configuration in descending order. Subsequently, we check whether each instance can meet the request requirements one by one in descending order of priority. We start by initializing the instance configuration based on the rounded DDPG output, which provides suggested values for parameters such as batch size, memory, and GPU configuration. The current instance is temporarily designated as the dispatch instance, after which the batch size is adjusted to match the request's requirements precisely, avoiding excess batch sizes that could extend the inference time, resulting in higher costs and latency constraint violations. Finally, we verify whether the latency constraint is satisfied; if not, configurations are greedily upgraded until the constraint is met. Once the request requirements are met, it is accepted; otherwise, it is

Algorithm 3: ExpertDRL Algorithm.**Input:** Parameters in RLSP.**Output:** \mathbb{A} : binary variable indicates whether to accept the request; $x_{m,o}^{i,e}, b_{m,o}^e, b_{m,o}^e, g_{m,o}^e$: decision variables.

```

1:  $\hat{x}_{m,o}^{i,e}, b_{m,o}^e, b_{m,o}^e, g_{m,o}^e \leftarrow \text{Algorithm 1}$ ;
2:  $x_{m,o}^{i,e}, b_{m,o}^e, b_{m,o}^e, g_{m,o}^e \leftarrow \text{Algorithm 2}$ ;
3: return  $\mathbb{A}, x_{m,o}^{i,e}, b_{m,o}^e, b_{m,o}^e, g_{m,o}^e$ 

```

rejected, and the configuration of the corresponding instance is reverted to its previous state.

4) *ExpertDRL Algorithm*: The ExpertDRL algorithm, outlined in Algorithm 3, begins by invoking the DRL algorithm (Algorithm 1) to generate a fractional request dispatching decision and a recommended instance configuration, including parameters like batch size, memory, and GPU settings. Then, Algorithm 2 is called to make the final request dispatch decision based on the fractional request dispatch decision and our two insights and adjust the instance configuration to save costs and meet constraints.

C. Algorithm Analysis

Next, we analyze the theoretical properties of our algorithm, including the algorithm complexity and its approximation ratio.

1) *Algorithm Complexity*: In the ExpertDRL algorithm, the computational complexity of the offline training process depends on both the volume of training data and the length of the training period. Once training is completed, the trained model can be utilized for online inference, enabling function instance configuration and request dispatching decisions for newly arrived requests. As the training phase is conducted offline, our primary concern is the computational complexity during the online running process [8].

Theorem 2: In ExpertDRL algorithm, the online running process runs in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|\log(|\mathcal{E}||\mathcal{M}||\mathcal{O}|) + |\mathcal{R}|\log|\mathcal{R}| + |\mathcal{G}|\log|\mathcal{G}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}|(|\mathcal{R}||\mathcal{G}|(|\mathcal{D}| + |\mathcal{I}| + |\mathcal{M}||\mathcal{O}|)) + \mathbb{N}\mathbb{M}^2 + |\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}||\mathbb{M}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}||\mathbb{M}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}||\mathbb{M}|)$.

Proof: As shown in Algorithm 3, the ExpertDRL algorithm consists of two algorithms, such as the DRL algorithm and the expert intervention algorithm. First, we analyze the complexity of the DRL algorithm as follows. Assume that the neural network in the DRL algorithm has \mathbb{N} hidden layers and each layer contains \mathbb{M} neurons. Besides, in the RL model, the state size is $|\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}| + 5|\mathcal{E}||\mathcal{M}||\mathcal{O}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}| + 4|\mathcal{E}| + 2|\mathcal{D}| + |\mathcal{M}| + |\mathcal{O}| + |\mathcal{B}| + |\mathcal{R}| + |\mathcal{G}| + 6$ and the action size is $4|\mathcal{E}||\mathcal{M}||\mathcal{O}|$. This means that the input and output layers of the DRL neural network contain $|\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}| + 9|\mathcal{E}||\mathcal{M}||\mathcal{O}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}| + 4|\mathcal{E}| + 2|\mathcal{D}| + |\mathcal{M}| + |\mathcal{O}| + |\mathcal{B}| + |\mathcal{R}| + |\mathcal{G}| + 6$ neurons. Thus, the DDPG runs in $O(\mathbb{N}\mathbb{M}^2 + |\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}||\mathbb{M}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}||\mathbb{M}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}||\mathbb{M}|)$. Then, the fractional configuration decision is rounded in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|)$. Thus, the DRL algorithm runs in $O(\mathbb{N}\mathbb{M}^2 + |\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}||\mathbb{M}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}||\mathbb{M}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}||\mathbb{M}|)$.

Second, we analyze the complexity of the expert intervention algorithm as follows. In this algorithm, we first initialize and sort the candidate configurations in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|\log(|\mathcal{E}||\mathcal{M}||\mathcal{O}|) + |\mathcal{R}|\log|\mathcal{R}| + |\mathcal{G}|\log|\mathcal{G}|)$. Then, the algorithm rounds the request dispatching decisions and upgrades and adjusts the configurations running in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|(|\mathcal{R}||\mathcal{G}|(|\mathcal{D}| + |\mathcal{I}| + |\mathcal{M}||\mathcal{O}|)))$. Thus, the expert intervention algorithm runs in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|\log(|\mathcal{E}||\mathcal{M}||\mathcal{O}|) + |\mathcal{R}|\log|\mathcal{R}| + |\mathcal{G}|\log|\mathcal{G}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}|(|\mathcal{R}||\mathcal{G}|(|\mathcal{D}| + |\mathcal{I}| + |\mathcal{M}||\mathcal{O}|)))$. Based on the above analysis, as in Algorithm 3, the ExpertDRL algorithm runs in $O(|\mathcal{E}||\mathcal{M}||\mathcal{O}|\log(|\mathcal{E}||\mathcal{M}||\mathcal{O}|) + |\mathcal{R}|\log|\mathcal{R}| + |\mathcal{G}|\log|\mathcal{G}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}|(|\mathcal{R}||\mathcal{G}|(|\mathcal{D}| + |\mathcal{I}| + |\mathcal{M}||\mathcal{O}|)) + \text{NM}^2 + |\mathcal{M}||\mathcal{O}||\mathcal{B}||\mathcal{R}||\mathcal{G}||\mathcal{M}| + |\mathcal{E}||\mathcal{M}||\mathcal{O}||\mathcal{M}| + |\mathcal{M}||\mathcal{O}||\mathcal{D}||\mathcal{M}|)$. ■

Obviously, our algorithm runs in polynomial time, and its execution time is verified to be acceptable in Section VI-B3.

2) *Algorithm Approximation*: In edge serverless environments, although the capacity of each ES is limited, the abundance of ESs provides sufficient resources for model inference [8]. Therefore, without loss of generality, we assume that the ESs can accommodate all inference requests, and that all requests are accepted.

Theorem 3: When all inference requests are accepted, ExpertDRL algorithm approximates the optimal solution by factor $\frac{\mathbb{E}^{\max}}{\mathbb{E}^{\min}}$, where $\mathbb{E}^{\min} = \min \left\{ \frac{(\kappa r_{m,o}^e + \zeta g_{m,o}^e) \cdot \hat{t}_{m,o}(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e) + \sigma}{b_{m,o}^e}, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, b_{m,o}^e \in \mathcal{B}, r_{m,o}^e \in \mathcal{R}, g_{m,o}^e \in \mathcal{G} \right\}$ and $\mathbb{E}^{\max} = \max \left\{ \frac{(\kappa r_{m,o}^e + \zeta g_{m,o}^e) \cdot \hat{t}_{m,o}(b_{m,o}^e, r_{m,o}^e, g_{m,o}^e) + \sigma}{\max\{b_{m,o}^e, |b_{m,o}^e < b_{m,o}^e, \forall b_{m,o}^e \in \mathcal{B}\} + 1}}, \forall e \in \mathcal{E}, m \in \mathcal{M}, o \in \mathcal{O}, b_{m,o}^e \in \mathcal{B}, r_{m,o}^e \in \mathcal{R}, g_{m,o}^e \in \mathcal{G} \right\}$. ■

Proof: See Appendix A, available online. ■

It should be noted that our ExpertDRL can approach the optimal solution well and have good optimality by exploring the solution space limited by expert insights and optimizing the neural network, which is verified by the evaluation results in Section VI-B1.

VI. EVALUATION

This section first introduces the evaluation settings and then analyzes and discusses the evaluation results.

A. Settings

Evaluation environment: We conduct our evaluation in both experimental and simulated environments. The experimental setup is deployed on OpenFaaS [28], a widely used commercial serverless framework [6], [34], [35]. The simulation environment is built using a Python simulator, and the ML framework is implemented with PyTorch [36]. All evaluations are performed on a computer equipped with a 24-core Intel(R) Core(TM) i9-13900 K CPU @ 3.0 GHz, 16 GB RAM, and an NVIDIA GeForce RTX 4090 GPU.

Infrastructure: There are 20 ESs in the infrastructure, and their GPU capacity and memory capacity are derived from the Alibaba *cluster-trace-gpu-v2023* trace data [37]. The

number of inference requests is 100. The memory price is \$1.66667e-5 per GB per second, the GPU price is \$8.1492e-5 per GPU per second [38], the function instance invocation price is \$2e-7 invocation, which is referenced in the pricing model of the commercial serverless platform AWS Lambda [26]. Moreover, the communication delay from each IoT device to each ES is distributed in [1, 10] milliseconds [8].

Inference Request: For realistic inference workloads, the start time for each inference request is derived from Twitter trace [39], which collects time information of stream requests in the absence of publicly available inference service traces. Besides, the latency requirements for inference requests are distributed between 50 and 200 milliseconds [40], and the downstream tasks to which they belong are random. Moreover, the required accuracy of each inference request is distributed from 52% to 83% [40]. The accuracy required by inference requests does not exceed the highest accuracy supported by the corresponding models.

Foundation Model: In our evaluation, there are 5 typical FMs [40], including ViT [41], VGG-19 [42], ResNet-50 [43], RegNet [44], and MobileNet [45]. Each model is equipped with two optimizations, and their weights are recorded as 16-bit or 32-bit floating-point types. Besides, each model is fine-tuned to two types of downstream tasks. The accuracy of each model on downstream tasks is derived from the reported results [27]. The inference latency under different configurations of each functional instance is obtained through our experiments on our OpenFaaS-based platform. Finally, the batch sizes are set to [1, 2, 4, 8, 16, 32] [2], GPU configuration is either 0 or 1, and memory configuration spans from 1024MB to 10240MB [26].

DRL: Each neural network has three layers, and each hidden layer contains 1024 neurons [8]. Meanwhile, in DDPG, the learning rates for Actor and Critic are set to 0.001 and 0.003 [9], respectively, with a discount factor of 0.99. Besides, the penalty parameter ρ is set to 0.01, determined based on the monetary cost. Inference requests arrive one by one, and future requests are unknown. Unless otherwise specified, the above parameters are adopted as default settings. All data points are collected from 20 runs.

Metrics: In our evaluation, the evaluated metrics include normalized total cost, normalized average cost, acceptance ratio, and execution time. These metrics refer to the cost incurred by all requests and the average cost of accepted requests, the ratio of accepted requests, and the execution time of the algorithm, which are used to verify the optimality, superiority, and practicality of the algorithm in different aspects. *Compared Solutions*. We evaluate ExpertDRL with the following algorithms.

- INFless [6]: This algorithm dispatches inference requests to the function instances that meet the accuracy requirements and employs resource-efficient configurations for the selected instances.
- DDPG [32]: This algorithm is a simplified version of our solution, in which DDPG [32] directly produces integer solutions for request dispatching and instance configuration decisions.

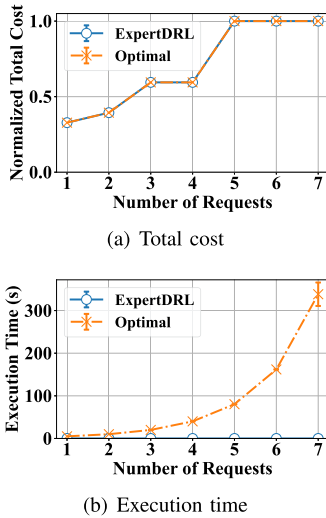


Fig. 8. Compared with the optimal solution in normalized total cost and execution time.

- EI: The expert intervention (EI) algorithm is a simplified version of ExpertDRL, which makes request dispatching and instance configuration decisions that satisfy all constraints based on our insights, without the assistance of DDPG.
- Optimal: This algorithm explores the solution space for the optimal solution using the branch-and-bound method [46].

B. Result Evaluation

1) *Optimality*: We evaluate the optimality of our scheme by comparing ExpertDRL with the optimal solution in normalized total cost and execution time under different request numbers. Due to the exponential growth of the solution space, searching for the optimal solution is time-consuming, and the evaluation is limited to small-scale settings to obtain results within a reasonable timeframe.⁸ The result is shown in Fig. 8. As shown in Fig. 8(a), the normalized total cost of our algorithm and the optimal solution gradually increases as the number of inference requests rises. This is because additional function instances or resources are needed to handle the growing number of inference requests. Besides, the normalized total cost does not grow proportionally, as batching allows for better utilization of the hardware's parallel processing capability, improving resource efficiency and avoiding redundant computations. Moreover, our algorithm always hits the optimal solution, demonstrating its good optimality. In addition, the normalized total cost may not always increase with the number of requests. For instance, when the number of requests is 4, the normalized total cost may remain the same as when the number of requests is 3. This occurs because newly arriving requests can be batched

⁸ In the small-scale setting, we set $\mathcal{E} = 1$, $\mathcal{M} = 2$, $\mathcal{O} = 1$, $\mathcal{D} = 1$, $\mathcal{R} = 2$, $\mathcal{B} = 5$, $\mathcal{G} = 2$. The reason for this setting is to keep the execution time within a suitable time, as it exhibits exponential growth with the increase of these parameters. For example, when increasing the number of ESs and setting $\mathcal{E} = 2$, $\mathcal{I} = 5$, the optimal solution was not obtained after running for 12 hours, let alone running 20 times for averaging.

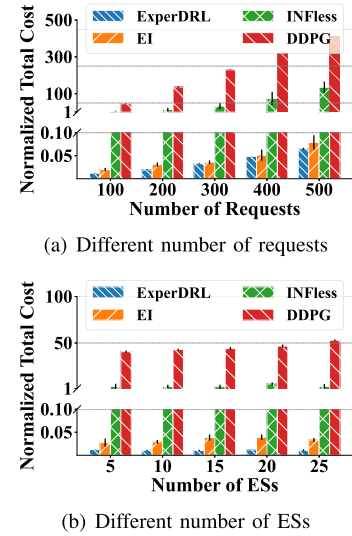


Fig. 9. Compared with other solutions in normalized total cost.

under the same configuration without requiring an upgrade (e.g., $b_{m,o}^e = 4$). As shown in Fig. 8(b), the execution time of our algorithm is significantly shorter than that of the optimal solution, where the execution time for the optimal solution grows exponentially, while the execution time of our algorithm remains below 2 ms. For example, when the number of requests is 7, the optimal solution takes 338.3 seconds to make decisions for these requests, while our algorithm only takes 1.91 milliseconds, which means our algorithm is 177120 times faster. This is because as the number of requests increases, the solution space grows exponentially, making it time-consuming to search for the optimal solution, even with the branch-and-bound method. In summary, our algorithm closely approximates the optimal solution, exhibits strong optimality, and offers significantly shorter execution times compared to the optimal solution.

2) *Superiority*: We evaluate the superiority of our algorithm by comparing ExpertDRL with three benchmark algorithms in normalized total cost, normalized average cost, and request acceptance ratio under different numbers of requests and ESs. Among them, the normalized total cost includes the monetary cost of accepted requests and the penalty cost caused by rejected requests, while the normalized average cost indicates the average cost of all accepted requests. The results are shown in Figs. 9, 10, and 11.

As shown in Fig. 9(a), our algorithm consistently outperforms the other three schemes and always has the lowest normalized total cost. For example, when the number of requests is 500, the total normalized cost of our algorithm is 0.065, while the total normalized cost of EI, INFless, and DDPG is 0.079, 134.667, and 310.41, respectively. This means that our scheme significantly outperforms all the other three schemes. This is because compared to INFless and DDPG, our algorithm can avoid the high penalty cost caused by frequent request rejections due to constraint violations (detailed in the discussion of Fig. 11). Compared to EI, our algorithm can better evaluate the value of ESs and corresponding function instances for inference requests

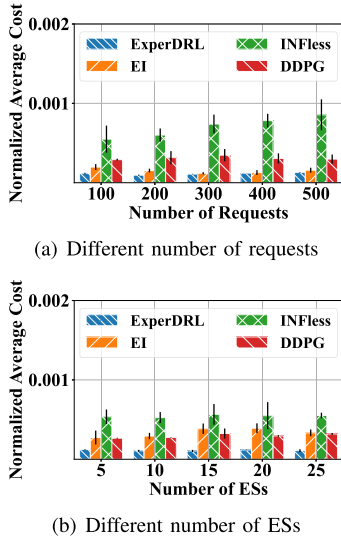


Fig. 10. Compared with other solutions in normalized average cost.

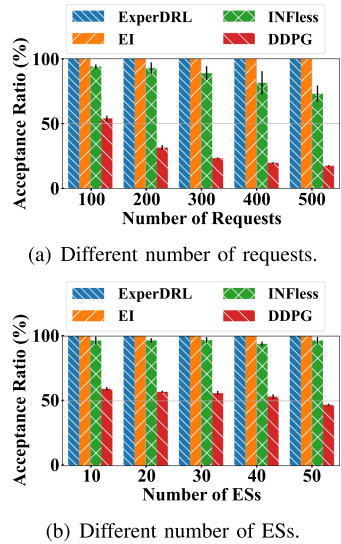


Fig. 11. Compared with other solutions in resource cost.

through DRL. Besides, the cost of all algorithms gradually increases with the number of requests, since more function instances and higher resource configurations are required to handle more inference requests, and even more requests being rejected. Similar results can be found as shown in Fig. 9(b), where our algorithm still has the lowest normalized total cost. In addition, as the number of ESs increases, the cost of DDPG gradually increases slightly, because more ESs lead to a larger solution space, which makes it easier to fall into poor local optima.

As shown in Fig. 10(a), our algorithm always has the lowest normalized average cost. For example, when the number of requests is 300, the normalized average cost of our algorithm is 0.00011, while the normalized average costs of EI, INFless, and DDPG are 0.00012, 0.00074, and 0.00035. This indicates that

our algorithm achieves a cost reduction of 85.14% compared to the state-of-the-art solution (INFless). This is because INFless is designed for traditional models, while our solution is tailored to the characteristics of the FM, enabling inference requests to be batched in a fine-grained manner for higher resource utilization, thus reducing resource costs. Additionally, our algorithm incorporates DRL to capture long-term rewards, further enhancing resource efficiency and cost savings. Similar results can be observed in Fig. 10(b), where our algorithm still outperforms the other three algorithms across varying numbers of ESs.

As shown in Fig. 11(a), our algorithm and EI can always accept all requests, while INFless and DDPG face frequent request rejections. For example, when the number of requests is 500, the request acceptance ratio of our algorithm and EI are both 100%, while the request acceptance ratio of INFless and DDPG are 73.07% and 17.47%, respectively. This means that our algorithm can improve the acceptance ratio by 26.93% compared to the state-of-the-art solution (INFless). This is because INFless is tailored for cloud data centers rather than distributed edge environments, where the differentiated communication delays from data sources to distributed ESs fall outside of its design considerations, leading to frequent latency constraint violations. In addition, DDPG randomly explores the solution space to search for the optimal solution, which is filled with numerous infeasible solutions due to high-dimensional constraints, leading to frequent constraint violations. In contrast, our algorithm and EI quickly locate high-quality solutions based on our insights, checking constraints to avoid violations. Moreover, as the number of requests increases, the request acceptance ratio of DDPG and INFless gradually declines. This is because more requests lead to more batch processing opportunities, which can result in longer processing delays and more inclined to suffer from latency constraint violations. Similar results can be observed in Fig. 11(b), where our algorithm consistently achieves a higher request acceptance ratio across different numbers of ESs. Moreover, as the number of ESs increases, the request acceptance ratio of DDPG gradually decreases, as the expanded solution space causes its neural network to converge on poorer solutions.

3) *Practicality*: We evaluate the practicality of our algorithm by comparing it with the benchmark algorithms in terms of convergence and execution time. The results are shown in Fig. 12.

As shown in Fig. 12(a), as the training progresses, the DDPG algorithm gradually converges, but it still faces high costs due to frequent constraint violations. The normalized total cost of the other three algorithms is significantly lower than that of DDPG, due to fewer penalty costs caused by constraint violations. Besides, the normalized total costs of INFless and EI are fixed at 3.33 and 0.172. Fig. 12(b) shows an enlarged Fig. 12(a) with low data values ($y \in [0, 0.5]$), which only contains two algorithms, such as ExpertDRL and EI. Compared to DDPG, our algorithm starts with a better local optimal solution under the guidance of EI, and converges to a lower cost as training progresses, outperforming all other schemes. Moreover, Fig. 12(c) illustrates the request acceptance ratio of the DDPG algorithm during the training process, where it is clear that the acceptance ratio gradually increases and converges. Fig. 12(d) shows the execution time of all four algorithms, where our algorithm's

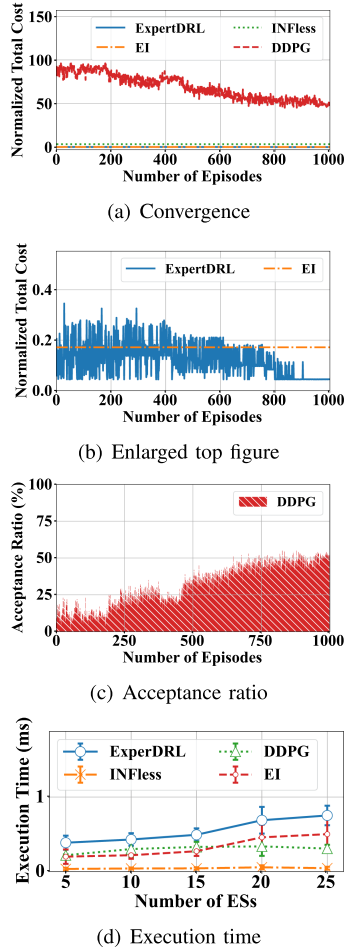


Fig. 12. Compared with other solutions in convergence and execution time.

execution time is slightly higher than the others due to the need for both DDPG assistance and EI guidance. Nonetheless, our algorithm's execution time always does not exceed 1 ms, which is negligible and worthwhile for typical inference requests with 20-100 milliseconds. Overall, the above results confirm that our algorithm has good practicality in both execution time and convergence.

VII. CONCLUSION

In this paper, we design a cost-effective inference request dispatching and instance configuration scheme, called ExpertDRL, which is tailored for popular FMs in serverless edge computing environments. The scheme leverages our insight that fine-grained request dispatching can fully exploit the shared backbone characteristics of FMs, enabling more granular batch processing to enhance resource efficiency and reduce costs. Specifically, we first establish the mathematical correlation between fine-grained request dispatching and instance configuration with respect to monetary cost, inference latency, and inference accuracy based on the property that FMs share a model backbone. We then formalize this problem as an integer nonlinear programming and analyze its complexity. To

address the challenges posed by delayed rewards and high-dimensional constraints, ExpertDRL integrates DRL with expert intervention, where DRL captures delayed rewards in dynamic environments and provides fractional solution guidance, while expert intervention incorporates our insights (larger batches bring lower monetary costs while introducing higher latency, and upgrading configurations can shorten latency at the expense of monetary costs) and fractional solution guidance to make request dispatch decisions and adjust instance configuration to meet constraints and avoid poor local optima. This algorithm is rigorously proven to have theoretical guarantees. Finally, extensive experimental and simulation results validate that ExpertDRL significantly outperforms existing ones in terms of cost and request acceptance ratio.

Machine learning-driven resource management solutions have become a popular trend, and ExpertDRL incorporates our insights and DRL to effectively explore optimal solutions while avoiding constraint violations, but may face noteworthy offline training overheads. Our future work plans incorporate more innovative machine learning techniques, such as transfer learning, attention mechanisms, contrastive learning, etc., to achieve faster convergence and higher efficiency.

REFERENCES

- [1] J. Nguyen, K. Malik, M. Sanjabi, and M. Rabbat, "Where to begin? On the impact of pre-training and initialization in federated learning," 2022, *arXiv:2206.15387*.
- [2] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *Proc. USENIX Annu. Techn. Conf.*, 2021, pp. 397–411.
- [3] L. Kevin, R. Vijay, and A. William, "Accelerating facebook's infrastructure with application-specific hardware," 2024. [Online]. Available: <https://engineering.fb.com/2019/03/14/data-center-engineering/accelerating-infrastructure/>
- [4] A. Ali, R. Pincirol, F. Yan, and E. Smirni, "Optimizing inference serving on serverless platforms," in *Proc. VLDB Endowment*, vol. 15, no. 10, 2022, pp. 2071–2084.
- [5] A. Ali, R. Pincirol, F. Yan, and E. Smirni, "BATCH: Machine learning inference serving on serverless platforms with adaptive batching," in *Proc. IEEE/ACM Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–15.
- [6] Y. Yang et al., "INFless: A native serverless system for low-latency, high-throughput inference," in *Proc. ACM Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2022, pp. 768–781.
- [7] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2021, pp. 3045–3059.
- [8] Y. Zeng et al., "SafeDRL: Dynamic microservice provisioning with reliability and latency guarantees in edge environments," *IEEE Trans. Comput.*, vol. 73, no. 1, pp. 235–248, Jan. 2024.
- [9] Y. Zeng et al., "RuleDRL: Reliability-aware SFC provisioning with bounded approximations in dynamic environments," *IEEE Trans. Serv. Comput.*, vol. 16, no. 5, pp. 3651–3664, Sep./Oct. 2023.
- [10] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *Proc. Symp. Netw. Syst. Des. Implementation*, 2017, pp. 613–627.
- [11] C. Zhang, M. Yu, W. Wang, and F. Yan, "MARK: Exploiting cloud services for cost-effective, SLO-Aware machine learning inference serving," in *Proc. USENIX Annu. Techn. Conf.*, 2019, pp. 1049–1062.
- [12] V. Nigade, P. Bauszat, H. Bal, and L. Wang, "Jellyfish: Timely inference serving for dynamic edge networks," in *Proc. Real-Time Syst. Symp.*, 2022, pp. 277–290.
- [13] 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.

- [14] R. Bommasani et al., "On the opportunities and risks of foundation models," 2021, *arXiv:2108.07258*.
- [15] H.-Y. Chen, C.-H. Tu, Z. Li, H. W. Shen, and W.-L. Chao, "On the importance and applicability of pre-training for federated learning," 2022, *arXiv:2206.11488*.
- [16] M. E. Peters et al., "Deep contextualized word representations," in *Proc. Conf. North Amer. Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2018, pp. 2227–2237.
- [17] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2790–2799.
- [18] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [19] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [20] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM Special Int. Group Data Commun.*, 2018, pp. 191–205.
- [21] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2021, pp. 258–271.
- [22] H. Yu, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Accelerating serverless computing by harvesting idle resources," in *Proc. ACM Int. Conf. World Wide Web*, 2022, pp. 1741–1751.
- [23] N. He et al., "Leveraging deep reinforcement learning with attention mechanism for virtual network function placement and routing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1186–1201, Apr. 2023.
- [24] Y. Rao, J. Lu, and J. Zhou, "Attention-aware deep reinforcement learning for video face recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 3931–3940.
- [25] W. Dong, Z. Zhang, and T. Tan, "Attention-aware sampling via deep reinforcement learning for action recognition," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 8247–8254.
- [26] "AWSlambda pricing," 2024. [Online]. Available: <https://aws.amazon.com/lambda/pricing/>
- [27] "Pre-trained model accuracy," 2024. [Online]. Available: <https://pytorch.org/vision/stable/models.html>
- [28] "OpenFaaS," 2024. [Online]. Available: <https://www.openfaas.com/>
- [29] "Operating lambda: Performance optimization—part 1," 2021. [Online]. Available: <https://aws.amazon.com/cn/blogs/compute/operating-lambda-performance-optimization-part-1/>
- [30] "Knapsack problem," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Knapsack_problem#0-1_knapsack_problem
- [31] D. Pisinger and P. Toth, "Knapsack problems," in *Handbook of Combinatorial Optimization*. Berlin, Germany: Springer, 1998, pp. 299–428.
- [32] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2016.
- [33] H. Q. Ngo, Approximation algorithms based on LP relaxation. SUNY at Buffalo, Department of Computer Science and Engineering, 2005. [Online]. Available: <https://cse.buffalo.edu/hungngo/classes/2005/594/notes/relaxation-rounding.pdf>
- [34] Q. Zeng, K. Hou, X. Leng, and Y. Chen, "DirectFaaS: A clean-slate network architecture for efficient serverless chain communications," in *Proc. ACM Int. Conf. World Wide Web*, 2024, pp. 2759–2767.
- [35] Y. Li, L. Zhao, Y. Yang, and W. Qu, "Rethinking deployment for serverless functions: A performance-first perspective," in *Proc. IEEE/ACM Int. Conf/High Perform. Comput., Netw., Storage Anal.*, 2023, pp. 1–14.
- [36] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.
- [37] Q. Weng et al., "Beware of fragmentation: Scheduling GPU-Sharing workloads with fragmentation gradient descent," in *Proc. USENIX Annu. Techn. Conf.*, 2023, pp. 995–1008.
- [38] "Price autoDL," 2024. [Online]. Available: <https://www.autodl.com/home>
- [39] "Twitter streaming traces," 2018. [Online]. Available: <https://archive.org/details/archiveteam-twitter-stream-2018=04>
- [40] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Architecture*, 2021, pp. 446–459.
- [41] A. Dosovitskiy et al., "An image is worth 16 x 16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.
- [42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [43] 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.
- [44] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10428–10436.
- [45] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.
- [46] S. Boyd and J. Matingley, *Branch and Bound Methods*. Princeton, NJ, USA: Citeseer, 2007.



Yue Zeng received the PhD degree from Nanjing University. He is currently an associate professor with the Nanjing University of Science and Technology. He has published more than a dozen papers in top journals and conferences, including *IEEE Transactions on Computers*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Communications*, *IEEE Transactions on Cloud Computing* and *IEEE CVPR*. His research interests include edge intelligence, deep reinforcement learning, machine learning training and inference, federated learning, and serverless computing.



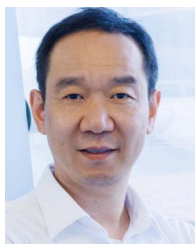
Junlong Zhou (Member, IEEE) is an associate professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, China. His research interests include edge computing, cloud computing, and embedded systems, where he has published 120 papers, including more than 40 in premier *IEEE/ACM Transactions*. He has been a subject area editor for *Journal of Systems Architecture* and an associate editor for *Journal of Circuits, Systems, and Computers* and *IET Cyber-Physical Systems: Theory & Applications*.



Baoliu Ye (Member, IEEE) received the PhD degree in computer science from Nanjing University, China, in 2004. He is a full professor with the Department of Computer Science and Technology, Nanjing University. He served as a visiting researcher with the University of Aizu, Japan from March 2005 to July 2006, and the dean of School of Computer and Information, Hohai University since January 2018. His current research interests mainly include distributed systems, cloud computing, wireless networks with more than 70 papers published in major conferences and journals. He served as the TPC co-chair of HotPOST12, Hot-POST11, P2PNet10. He is the regent of CCF, the secretary-general of CCF Technical Committee of Distributed Computing and Systems.



Zhihao Qu (Member, IEEE) received the BS and PhD degrees in computer science from Nanjing University, Nanjing, China, in 2009, and 2018, respectively. He is currently an associate professor with the College of Computer and Information, Hohai University. His research interests are mainly in the areas of wireless networks, edge computing, and distributed machine learning.



Song Guo (Fellow, IEEE) is currently a full professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. He is also the Changjiang chair professor awarded by the Ministry of Education of China. His research interests include edge AI, mobile computing, and distributed systems. He has been recognized as a highly cited researcher (Web of Science) and was the recipient of more than 14 best paper awards from IEEE/ACM conferences, journals, and technical committees. He is the editor-in-chief of the *IEEE Open*

Journal of the Computer Society. He was on the IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board of a number of prestigious international journals, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, and *IEEE Internet of Things Journal*.



Pan Li received the MS degree from the Department of Electronic Information Engineering, Southwest University, Chongqing, China, in 2019. She is currently working toward the PhD degree in computer science and technology with Southwest University. Her research interests include network functions virtualization, software defined networking, and edge computing.



Tianjian Gong received the BS degree in information science and engineering from Hohai University, Changzhou, China, in 2024. He is currently working toward the PhD degree with the School of Computer Science and Engineering, Nanjing University of Science and Technology. His research interests include serverless computing and edge computing.