Towards cost-effective and robust AI microservice deployment in edge computing environments

Chunrong Wu, Qinglan Peng, Yunni Xia, Yong Jin, Zhentao Hu

PII: S0167-739X(22)00331-4
DOI: https://doi.org/10.1016/j.future.2022.10.015
Reference: FUTURE 6622

To appear in: *Future Generation Computer Systems*

Received date : 8 July 2022
Revised date : 2 September 2022
Accepted date : 17 October 2022

Towards Cost-Effective and Robust AI Microservice Deployment in Edge Computing Environments

Chunrong Wu[a], Qinglan Peng[a,*], Yunni Xia[b], Yong Jin[a] and Zhentao Hu[a]

[a]*School of Artificial Intelligence, Henan University, Zhengzhou, 450046, China.*
[b]*Software Theory and Technology Chongqing Key Lab, Chongqing University, Chongqing, China*

ARTICLE INFO

Keywords:
Multi-access Edge Computing
AI Edge Service
Service Deployment
Service Placement
Microservice Mixed Orchestration
Cost-effective Scheduling

ABSTRACT

As a newly emerged promising computing paradigm, Multi-access Edge Computing (MEC) is capable of energizing massive Internet-of-Things (IoT) devices around us and novel mobile applications, especially the computing-intensive and latency-sensitive ones. Meanwhile, featured by the rapid development of cloud-native technologies in recent years, delivering Artificial-Intelligence (AI) capabilities in a microservice way in the MEC environments comes true nowadays. However, currently MEC systems are still restricted by the limited computing resources and highly dynamic network topology, which leads to high service deployment/maintenance cost. Therefore, how to cost-effectively and robustly deploy edge AI microservices in failure-prone MEC environments has become a hot issue. In this study, we consider an edge AI microservice that can be implemented by composing multiple Deep Neural Networks (DNN) models, in this way, features of different DNN models are aggregated and the deployment cost can be further reduced while fulfilling the Quality-of-Service (QoS) constraint. We propose a Three-Dimension-Dynamic-Programming-based algorithm (TDDP) to yield cost-effective multi-DNN orchestration and load allocation plans. For the robust deployment of the yield orchestration plan, we also develop a robust microservice instance placement algorithm (TLLB) by considering the three levels of load balance including applications, servers, and DNN models. Experiments based on real-world edge environments have demonstrated that the proposed orchestration and placement methods can achieve lower deployment costs and less QoS loss when faced with edge node failures than traditional approaches.

List of Abbreviations

DNN	Deep Neural Network
EAP	Edge Application Provider
EIP	Edge Infrastructure Provider
ESB	Enterprise Service Bus
IoT	Internet of things
MEC	Multi-access Edge Computing
SOA	Service-Oriented Architecture
QoS	Quality of Service
QPS	Query Count Per Second
TDDP	Proposed microservice orchestration algorithm
TLLB	Proposed microservice placement algorithm

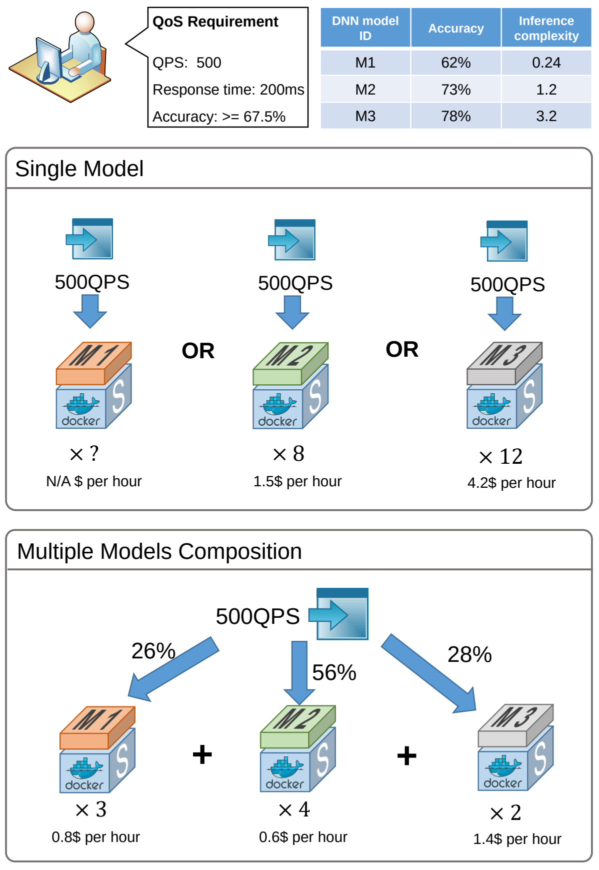
1. Introduction

Recent years have witnessed the prosperity of cloud-native techniques such as Docker, Swarm, and Spring Native, which promoted the emergence of microservices architecture [6, 31]. Different from traditional Service-Oriented Architecture (SOA) relays Enterprise service bus (ESB), it splits the whole system into a massive lightweight, single purpose, and re-deployable applications, which enhances the scalability and reliability of the system and realize DevOps[40]. Meanwhile, the significant breakthroughs in AI (especially in deep neural networks and its applications) techniques produce numerous novel applications (e.g., augmented reality, metaverse[36], and intelligent transportation systems[47, 48]), which greatly changed daily lives. The advanced AI and microservices techniques promote the prosperity of mobile/Internet of things (IoT) devices and applications, according to the prediction of Cisco, there will be 29.3 billing active mobile and IoT devices connected, and more than 75% web actions will happen on these end-devices[4]. However, due to the restriction of computing capabilities and battery life, applications on mobile and IoT devices usually choose to invoke AI services deployed on the remote cloud to fulfill their functional requirements.

However, with mobile application providers and end-users increasing demands on high responsiveness and smooth real-time experience, the traditional cloud computing paradigm can not fulfill the low end-to-end service invoking delay requirement[26]. With the enhancement of advanced 5G communication technologies, Multi-access Edge Computing (MEC) emerged to tackle this problem. The MEC paradigm aims to deploy computing resources to the edge of networks (which could be base stations, network sinks, or CDN nodes)

Chunrong Wu, Qinglan Peng, Yong jin and Zhentao Hu are with the School of Artificial Intelligence, Henan University, Zhengzhou, 450046, China. Yunni Xia is with the School of Computers, Chongqing University, Chongqing 400044, China. The corresponding author of this work is Qinglan Peng (qinglan.peng@hotmail.com).

ORCID(s): 0000-0002-2691-093X (C. Wu); 0000-0002-8908-5201 (Q. Peng); 0000-0001-9024-732X (Y. Xia)



DNN model ID	Accuracy	Inference complexity
M1	62%	0.24
M2	73%	1.2
M3	78%	3.2

Figure 1: An example of AI microservices mixed deployment.

to provide pervasive computing capabilities[32]. In this way, application providers can deploy their mobile applications to the edge servers which are close to end-users reduce latency, and end-users are capable of offloading computing-intensive tasks to nearby edge servers in a service-invoking way to overcome the restriction of limited hardware resources. As a promising computing paradigm in the post-cloud computing era[50], the advantages of MEC have been widely concerned and multiple cloud providers are exploring its best practices.

Though the MEC paradigm has many advantages, its distributed and multiple-access natures also bring lots of challenges, e.g., the limited computing power of edge servers and the uncertain, error-prone network connectivity. For edge application providers (EAPs), there are also problems with deploying AI microservice applications on the MEC environments[39]. For example, the AI microservices are usually implemented by Deep Neural Networks (DNN) models and there are multiple DNN models available for the same services, different DNN models have different performances (e.g., inference rates, computing amount, and accuracy or precision). Employing a single DNN model to implement an AI microservice suffer from high deployment cost and may lead to a bad user experience resulting from the preference of certain inputs. As shown in Figure 1, composing multiple DNN models to implement a single AI microservice and allocate proper request loads to them helps to reduce deployment costs when fulfilling the Quality-of-Service (QoS) requirements. And this is the cost-effective multi-DNN Model orchestration problem that we are interested in this study.

While for the edge infrastructure providers (EIPs), who maintain an edge resource pool and get revenue from providing edge computing resources to EAPs, they could serve multi-tenant (e.g., support the deployment plans from multiple EAPs) at the same time[42]. However, different from the centralized cloud computing data centers, the edge resources pool is constructed by distributed edge servers with heterogeneous configurations, which are prone to resource failure and unavailable. Thus for EIPs, how to place the edge AI microservices instances requested by multi-tenant to proper edge nodes to reduce the impact of edge resource failures is the robust edge AI microservice instances placement problem, which is also to be investigated in this study.

To address the aforementioned challenges, we propose two approaches (shorts for TDDP and TLLB) to solve the cost-effective and robust edge AI microservices deployment problems from the EAP and EIP perspectives respectively. Where the TDDP approach is developed for EAPs, it can be integrated into microservices governance frameworks such as Apache Dubbo and Spring Boot. While the TLLB approach is designed for EIPs, it can be integrated into edge resources governance systems such as KubeEdge and K3S. To the best of our knowledge, this is the first work considering multi-DNN models orchestration for the edge AI microservice deployment problem and solving it from both EAP's and EIPs' perspectives. The main contributions of this work are as follows: 1) We propose the cost-effective and robust edge AI microservice deployment problem, where edge AI microservices are allowed to be implemented by multi-DNN models, in this way, the advantages of different DNN models are composed to achieve cost-effectively and QoS qualified edge AI microservice deployment. We present two approaches to solve it from the perspectives of both EAPs and EIPs; 2) For EAPs, we develop a dynamic programming-based algorithm (TDDP) to solve the cost-effective multi-DNN orchestration problem. It takes the performance metrics of different DNN models and the required QoS constraint of the edge AI microservice as inputs and employs a three-dimensional dynamic programming mechanism to find the optimal orchestration plan; 3) For EIPs, we develop a three-layer-load-balance-aware algorithm to solve the robust edge AI microservice instance placement problem. It considers three different levels of load balance (e.g., tenant level, edge server level, and DNN model level) to minimize the impact of edge resource failures on the deployed microservice instances belonging to multi-tenant as much as possible.

The remainder of this paper is organized as follows. Section II uses two motivating examples to illustrate the cost-effective and robust AI microservice deployment problems. Section II introduces our system model and presents the problem formulation of the proposed problems. The solutions proposed from the EAP and EIP perspectives respectively are described in Section IV. Section V presents our experimental results and analysis. Section VI reviews the related studies. Section VII concludes this study and indicates our future concerns.

2. Motivating Examples

In this section, we use two real-world examples to make the proposed problems clearer and illustrate the practical significance of solving them.

2.1. Scenario A

Edge AI microservice selection and composition are the key challenges to achieving high response and scaleable service provision[30]. For example, for an edge application provider (EAP), suppose there is a flower recognition business (like PictureThis or PlantNet APPs in google play) in its mobile application. Considering that deploying the image classification service in a remote cloud may lead to long response time and heavy data transmission, with the help of a novel edge computing paradigm and microservice architecture, this EAP plan to deploy the flower recognition function in a containerized microservice way to the edge servers where close to the end-users to improve the user experience.

Suppose the non-functional requirements of the flower recognition business are defined as follows: 1) Accuracy requirement: the average accuracy of recognition result should be higher than 70%; 2) Response time requirement: the response time per service invocation should be less than 800 milliseconds; 3) Throughput requirement: the flower recognition business can support 200 requests per second (i.e., 200 QPS). For the image classification function to implement flower recognition business, there could be multiple DNN models that can choose (e.g., AlexNet, ResNet, DenseNet, ..., etc.), and each model has a different QoS (e.g., recognition accuracy and inference computation amount). Meanwhile, edge infrastructure providers (EIPs) provision edge computing resources in a Container-as-a-Service (CaaS) way, and there are multiple container types with different configurations available to choose from.

As shown in Figure 2, under this circumstance, as an EAP, the proposed cost-effective multi-DNN models mixed orchestration for edge AI microservice aims

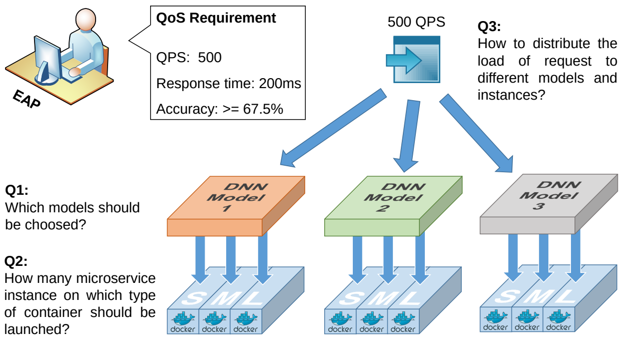

EAP
QoS Requirement
QPS: 500
Response time: 200ms
Accuracy: >= 67.5%
500 QPS
Q3:
How to distribute the load of request to different models and instances?
Q1:
Which models should be choosed?
Q2:
How many microservice instance on which type of container should be launched?
DNN Model 1
DNN Model 2
DNN Model 3
S M L
docker

Figure 2: Three questions to be solved under edge AI microservices orchestration scenario.

to solve: 1) how to choose suitable DNN models with different QoS to implement the image classification function to fulfill the accuracy requirement; 2) how many microservice instances should be launched on what kind of container type to minimize the deployment cost while fulfilling the throughput requirement; 3) how to distribute the request load to the lunched instances to fulfill the response time requirement. Compared with traditional edge AI deployment approaches that only consider a single DNN model, the proposed multi-model mixed orchestration scheme can fully leverage the QoS features of different DNN models, draw on each other's strengths, and finally reduce the deployment cost by composing them together. Besides, different DNN models may prefer different kinds of inputs, and implementing an edge AI microservice with multiple DNN models can improve the user-perceived QoS to some degree. Therefore, solving the proposed cost-effective multi-DNN-model mixed orchestration problem is of great practical significance.

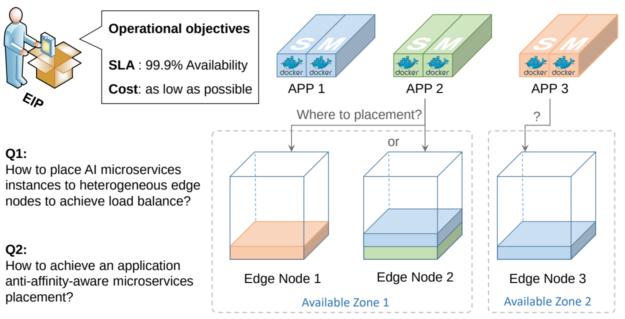


Figure 3: Two challenges need to be tackled in edge AI microservices placement scenario.

2.2. Scenario B

After get the orchestration plans of their edge AI microservices, EAPs begins to submit the microservice instance lunch requests (usually described by ymal files that specify the container type, application image, and library dependencies) to EIPs to finalize the microservice deployment.

Generally, an EIP usually manages an edge resource network that contains a bunch of edge nodes deployed in a certain city, it receives multiple mixed orchestration plans submitted by different EAPs and provides edge computing resources in a CaaS way[41]. As shown in Figure 3, for this EIP, how to properly schedule the submitted orchestration plans to minimize the impact of failures is the proposed robust microservice instance placement problem, and it aims to solve: 1) how to place the edge AI microservice instances with different resource requirements to heterogeneous edge nodes to achieve a load balance between them; 2) how to place the edge AI microservice instances from different applications fairly to edge nodes to achieve a load balance between different applications. The first target of the proposed placement problem is to balance the resource utilization among edge servers to reduce the probability of server performance deterioration and even crash. The second target holds the idea of "Don't Put All your Eggs in One Basket", and once failures happen, it guarantees the impact of failures on a certain application is limited, i.e., all applications share the risks of edge resources failure fairly.

Note that it is well-recognized that edge resources are more prone to encounter failures (e.g., network breakdowns[17] and server crashes[46]) than cloud resources. Therefore, in error-prone MEC environments, the proposed robust microservice instance placement problem is also of great practical significance.

3. System Model and Problem Formulation

In this section, we present our system model and give the definitions of cost-effective multiple DNN model mixed orchestration problems for mobile application vendors, and the robust multiple AI microservice instance placement problem for edge infrastructure providers.

3.1. Cost-Effective Multiple DNN Model Mixed Orchestration

In MEC environments, edge infrastructure and resources providers deploy edge servers in the 5G core network or User Plane Function (UPF)[23], and employ edge resource management platforms (e.g., KuberEdge, K3S, Akri, ..., etc) to manage them and provider computing services to the mobile application vendors in a serverless way[14]. Meanwhile, mobile application vendors are allowed to deploy their AI business or functions at the edge end in a microservice way to realize dynamic scaling, elastic pricing, and continuous integration/delivery. Edge infrastructure providers usually provide various kinds of container instances with different resource configurations for mobile application providers to deploy their AI microservices, and we use o to denote the count of available container instance types in this study.

An AI microservice can be implemented by multiple DNN models. For example, models like ResNet, AlexNet, LeNet, ..., etc, can be employed to implement an image classification service. In multiple DNN model mixed orchestration schema, an AI microservice can be implemented by various microservice instances that are deployed in containers with different configurations, these instances have the same functionality but might employ different DNN models and thus have different QoS. Suppose a mobile application vendor has an AI services S (e.g., image classification, text split, signature identify, etc) to be deployed at the edge platform. The request strength (i.e., query per second, QPS) of AI service S is λ, and there are n kinds of DNN models $\boldsymbol{M} = \{M_1, M_2, ..., M_n\}$ available to choice for implementing S. Different DNN models have different qualities (e.g., the accuracy of classification, the precision of text classification, the error rate of audio recognition) and inferential computation, we use q_i to denote the quality of model M_i, μ_{ij} to denote the inference speed (i.e., how many requests it can handle per second) of model M_i on j-th type of container instances. Let S_{ij} denotes the set of microservice instances that employ model M_i and chose j-th type of container configuration, w_{ij} denotes the ratio of total request strength that allocated to the instances in S_{ij} ($\sum_{i=1}^{n}\sum_{j=1}^{o} w_{ij} = 1$), $\mathcal{W}$ denotes the set constructed by all w_{ij}. Due to the microservice instances in the same set S_{ij} have the same container type, the request strength of each instance can be calculated as:

$$\lambda_{ij} = \frac{\lambda \cdot w_{ij}}{b_{ij}}, \tag{1}$$

In this way, the request processing of each microservice instance can be modeled as an M/M/1 queue[33, 46, 44], and according to Little's law, the expected inference time can be estimated as:

$$I_{ij} = \frac{1}{\mu_{ij} - \lambda_{ij}}. \tag{2}$$

where μ_{ij} is the inference rate of the microservice instances that employ model M_i and j-th container configuration, and $\mu_{ij} - \lambda_{ij} > 0$.

Let b_{ij} denote the number of microservice instances in set S_{ij}, $\mathcal{B}$ denotes the set constructed by all b_{ij}, p_j denotes the price of j-th container configuration provided by edge infrastructure providers (we consider on-demand pricing model in this study, and the billing intervals are accurate to the seconds). A multiple DNN model mixed orchestration plan of AI microservice S can thus be denoted as a 2-tuple $P = (\mathcal{B}, \mathcal{W})$, and its deployment cost can be calculated as:

$$C(\mathcal{B}) = \sum_{i=1}^{n} \sum_{j=1}^{o} b_{ij} \cdot p_j, \tag{3}$$

and the QoS of S can evaluated as:

$$Q(\mathcal{W}) = \sum_{i=1}^{n} q_i \cdot \Big(\sum_{j=1}^{o} w_{ij} \Big), \tag{4}$$

In this paper, the response time of invoking edge AI microservices consists of data transmission time and model inference time [43, 24]. We use d to denote the average input data size of the requests to AI microservice S, and β to denote the average data transmission ratios between end-users and edge servers (these data can be obtained from historical data), the response time of requests that dispatched to microservice instances in S_{ij} can be calculated as:

$$R_{ij}(\mathcal{B}, \mathcal{W}) = I_{ij} + \frac{d}{\beta}. \tag{5}$$

Therefore, given the QoS requirement ϕ and the response time constraint τ of an AI microservice S, its cost-effective multiple DNN model mixed orchestration problems can be formulated as:

$$\begin{aligned}
\mathbf{P1} \min_{(\mathcal{B},\mathcal{W})} &: C(\mathcal{B}), && && (6)\\
\text{s.t.} &: Q(\mathcal{W}) \geq \phi, && && (7)\\
& R_{ij}(\mathcal{B}, \mathcal{W}) \leq \tau, && \forall i, j && (8)\\
& \textstyle\sum_{i=1}^{n} \sum_{j=1}^{o} w_{ij} = 1, && && (9)\\
& 1 \geq w_{ij} \geq 0, && \forall w_{ij} \in \mathcal{W} && (10)\\
& b_{ij} \in \{0, 1, 2 \ldots\}. && \forall b_{ij} \in \mathcal{B} && (11)
\end{aligned}$$

where the target of problem **P1** is to minimize the edge AI microservice deployment cost as shown in (6); constraint (7) indicates that the total weighted QoS of all microservice instances meet the mobile application vendor-defined QoS requirement; constraint (8) states that the response time of all requests should not exceed the defined constraint; constraint (9) guarantees that all request loads have been distributed; constraints (10) and (11) declare the scope of the feasible variables. It is obvious that constraints (7-8) are nonlinear, thus **P1** is a Mixed-Integer-Nonlinear Programming (MINLP) problem, which is known as an NP-hard one[34].

3.2. Robust Microservice Instance Placement

After getting the AI microservice orchestration plan, mobile application vendors will submit their deployment requests (usually represented by YMAL files) to the edge infrastructure provider to build container-based microservice applications. Generally, an edge

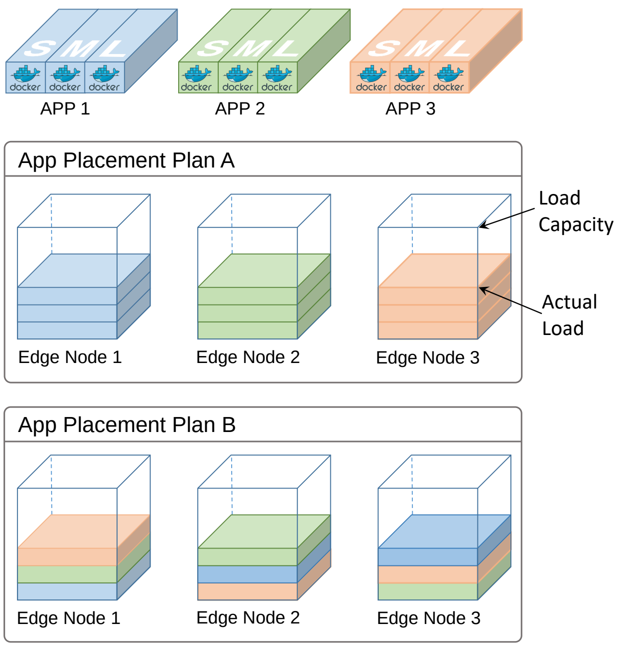


Figure 4: Robust deployment of microservice instances.

infrastructure provider could receive multiple deployment requests from multiple mobile application vendors for different edge AI microservices. Restricted by the failure-prone edge notes connectivity and the fluctuant edge server performance, how to place the microservice instances from different mobile applications to suitable edge nodes to reduce the QoS deterioration caused by edge network or edge nodes failures is a key problem.

Edge servers under high load for a long time are prone to crash, which leads to user-perceived QoS deterioration, thus the load balance of edge server is an important metric to evaluate a microservice instances placement plan. Besides, when an edge server crash or lost connection happens, all their hosted microservice instances will fail and out of service. These instances could belong to different mobile applications from multiple mobile application vendors. As shown in Figure 4, we say placement plan B is better than A because the server failure, crash, or lost connection risks are shared by all applications, which avoids a rapid QoS deterioration of a certain mobile application. Therefore, the load balance of a single mobile application among all edge servers is also an important key indicator to evaluate a placement plan.

Suppose an edge infrastructure provider manages an edge resource pool with m edge servers $E = \{e_1, e_2, \ldots, e_m\}$, and provider Container-as-a-Service (CaaS) to mobile application vendors. We use c_k and r_k to identify the CPU cores and memory size of edge server e_k respectively. There are total l microservice orchestration plans $\boldsymbol{P} = \{P_1, P_2, \ldots, P_l\}$ of mobile application providers' edge AI microservices

$\boldsymbol{S} = \{S_1, S_2, ..., S_l\}$ has been submitted to this provider. We use $I_h = \{s_h^1, s_h^2, ...\}$ to denote all instances that to be launched of AI microservice S_h, and it can be obtained from S_h' orchestration plan $P_h = (\mathcal{B}_h, \mathcal{W}_h)$.

We use function $T(s)$ to identify the type of container that microservice instance s is going to be launched, functions $C(j)$ and $M(j)$ to identify the vCPU (i.e., virtual processors) cores and the memory required by the j-th container configuration, and boolean function $\mathcal{P}(s,k) \in \{0,1\}$ to indicate whether the container that host microservice instance s is placed to edge server e_k. As some edge resource management platform as Kuberedge and K3S did, in this study, we consider the resource utilization of an edge server as:

$$u_k = \frac{1}{2}\left[\frac{\sum_{k=1}^{m}\sum_{s\in S_h}\mathcal{P}(s,k)\cdot C(T(s))}{c_k} + \frac{\sum_{k=1}^{m}\sum_{s\in S_h}\mathcal{P}(s,k)\cdot M(T(s))}{r_k}\right]. \quad (12)$$

Similarly, the resource occupation rate of application S_h's microservice instances on edge server e_k can be evaluated as:

$$u_k^h = \frac{1}{2}\left[\frac{\sum_{s\in S_h}\mathcal{P}(s,k)\cdot C(T(s))}{c_k} + \frac{\sum_{s\in S_h}\mathcal{P}(s,k)\cdot M(T(s))}{r_k}\right]. \quad (13)$$

Given a set of edge AI microserevice orchestration plan P, for an edge infrastructure provide, the robust microserevice instance placement problem can be formulated as:

$$\textbf{P2} \quad \min_{(\mathcal{P})} : \frac{1}{m}\sum_{k=1}^{m}(\bar{u}-u_k)^2, \quad (14)$$

$$\frac{1}{m}\sum_{k=1}^{m}\|\boldsymbol{v_k}-\boldsymbol{t}\|, \quad (15)$$

$$\text{s.t. } u_k \le 1, \qquad \forall k \quad (16)$$

$$u_k^h \le 1. \qquad \forall k,h \quad (17)$$

where $\bar{u} = \frac{1}{m}\sum_{k=1}^{m}u_k$ is the mean value of the resource utilization rates of all edge servers, $\boldsymbol{v_k} = (u_k^1, u_k^2, ..., u_k^l)$ the resource occupation rate of different applications on server e_k, $\boldsymbol{t} = (t_1, t_2, ..., t_l)$ the ratio of total resource required by different applications, where

$$t_h = \frac{1}{2}\left[\frac{\sum_{s\in S_h}\mathcal{P}(s,k)\cdot C(T(s))}{\sum_{k=1}^{m}\sum_{s\in S_h}\mathcal{P}(s,k)\cdot C(T(s))} + \frac{\sum_{s\in S_h}\mathcal{P}(s,k)\cdot M(T(s))}{\sum_{k=1}^{m}\sum_{s\in S_h}\mathcal{P}(s,k)\cdot M(T(s))}\right]. \quad (18)$$

and $\|\boldsymbol{t}\| = 1$.

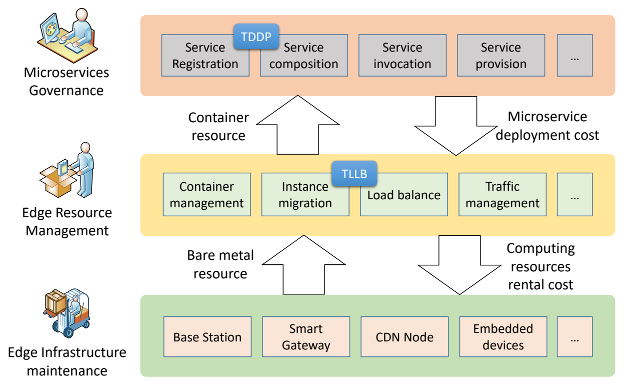


Figure 5: Integration and deployment of proposed approaches.

As shown in (14) and (15), the targets of the placement problem **P2** is to achieve load balance among edge servers and also a load balance among mobile applications. Minimizing the variance of edge servers' resource utilization helps to achieve a load balance among edge servers, and reduce their failure rate. While maintaining the resource occupation ratios of different applications on the some edge servers close to their total load ratios helps to spread the risk of failures fairly among all applications, reduce the server crash impact on a certain application, and avoid to cascading failures incurred by the rapid deterioration of the response time of a certain application. This is a combinatorial optimization problem which can be formulated to a multiple knapsack problem by performing problem reduction, which is known as a NP-hard one[7]. Therefore, the proposed robust multiple AI microservice instance placement problem is also NP-hard.

4. Solutions

For the two problems formulated above sections, we propose a Three Dimension Dynamic Programming-based algorithm (shorts for TDDP) and a Three Level Load Balance-based heuristic algorithm (shorts for TLLB) respectively to obtain the orchestration plans and the placement decisions.

As shown in Figure 5, the proposed TDDP algorithm can be integrated into the service dynamic scaling module of microservice governance frameworks (e.g., Eureka in Spring Cloud and Nacos in Spring Cloud Alibaba) to generate a cost-effective deployment plan (i.e., multiple DNN model mixed orchestration plan in this study) under the constraints of required QoS. The proposed TLLB algorithm can be integrated into the container scheduling module of edge resource management systems (e.g., Kube-scheduler in KubeEdge and K3S) to generate robust container (i.e., microservice instance) placement plans to cope with the high dynamic edge network and failure-prone edge resources.

4.1. Three Dimension Dynamic Programming-based Algorithm for P1

In problem **P1**, the weight of request load that is allocated to a certain set of microservice instances with the same DNN model and container type w_{ij} is a continuous variable. However, in real-world production environments, this load distribution information is usually represented in the form of a discrete level (e.g., the gateway weight configuration in Spring Cloud [1]) and has a minimal granularity (e.g., 1%, 2%, 5%, etc). Therefore, as the Gateway configuration specification in Spring Cloud, in this study, we consider the minimal load allocation granularity to $g = 0.01$, i.e., 1%, and propose a dynamic programming-based algorithm (TDDP) to find the optimal multi-DNN-Model edge AI microservice orchestration plans.

Algorithm 1: TDDP for **P1**

```
Input: Edge AI microservice $S$; Multiple DNN
       models $\boldsymbol{M}$ for $S$; Request strength $\lambda$;
       QoS constraint $\phi$; Response time
       constraint $\tau$
Output: Orchestration plan $P$; $P$'s deployment
        cost $C$
1  $\mathbb{M}, \mathbb{W}, \mathbb{Q} \leftarrow \emptyset$
2  $n \leftarrow |\boldsymbol{M}|$
3  $C \leftarrow \mathrm{DP}(n, 1, \phi)$
4  $\mathcal{W}, \mathcal{B} \leftarrow \mathrm{PlanRetrieval}(\mathbb{W}, \mathbb{Q}, n, \lambda)$
5  return $P \leftarrow (\mathcal{W}, \mathcal{B}),\ \ C$

6  Function DP($i, j, k$)
7      if $q_i \cdot j < k$ then
8          return $\infty$
9      if $i = 1$ then
10         $c, \boldsymbol{w}, \boldsymbol{b} \leftarrow \mathrm{OptSubPlan}(i, j \cdot \lambda, \tau)$
11         $\mathbb{M}[i, j, k] \leftarrow c$, $\mathbb{W}[i, j, k] \leftarrow j$
12         $\mathbb{Q}[i, j, k] \leftarrow k$
13         return $c$
14     $S \leftarrow \emptyset,\ \ v \leftarrow \infty$
15     for $x \leftarrow 0$ to $j$ with step $g$ do
16         $c, \boldsymbol{x}, \boldsymbol{b} \leftarrow \mathrm{OptSubPlan}(i, x \cdot \lambda, \tau)$
17         if $\mathbb{M}[i-1, j-x, k-q_i \cdot j] \neq \emptyset$ then
18             $s \leftarrow \mathbb{M}[i-1, j-x, k-q_i \cdot x] + c$
19         else
20             $s \leftarrow \mathrm{DP}(i-1, j-x, k-q_i \cdot x) + c$
21             $\mathbb{M}[i-1, j-x, k-q_i \cdot j] \leftarrow s$
22         $S \leftarrow S \cup s$
23     if $S \neq \emptyset$ then
24         $v, \mathbb{M}[i, j, k] \leftarrow \min(S)$
25         $\mathbb{W}[i, j, k] \leftarrow g\cdot \mathrm{argmin}(S)$
26         $\mathbb{Q}[i, j, k] \leftarrow k - q_i \cdot g\cdot \mathrm{argmin}(S)$
27     return $v$
```

Algorithm 2: OptSubPlan

```
Input: DNN model ID $i$; Allocated request load
       $\lambda_i$; Response time constraint $\tau$;
       Container type count $o$
Output: Sub-deployment cost $c$, Load ratios
        on different type of containers $\boldsymbol{w}$,
        Instances count of different type of
        containers $\boldsymbol{b}$
1  $c \leftarrow 0,\ \ \boldsymbol{b}, \boldsymbol{w}, \boldsymbol{g}, \boldsymbol{h} = O_{1\times o}$
2  for $j \leftarrow 1$ to $o$ do
3      $g_j \leftarrow \mu_{ij} - 1/\tau$
4      $h_j \leftarrow (\mu_{ij} - 1/\tau)/p_j$
5  $\boldsymbol{d} \leftarrow$ get the descending order of $\boldsymbol{h}$
6  $\boldsymbol{h}, \boldsymbol{g} \leftarrow$ rearrange $\boldsymbol{h}$ and $\boldsymbol{g}$ according to order $\boldsymbol{d}$
7  while $\lambda_i \neq 0$ do
8      $t \leftarrow d_1$
9      $b_t \leftarrow \lfloor \lambda_i / g_t \rfloor$
10     $\boldsymbol{d}, \boldsymbol{g}, \boldsymbol{h} \leftarrow$ remove $d_1$, $g_1$, $h_1$ from $\boldsymbol{d}$, $\boldsymbol{g}$, $\boldsymbol{h}$
11     $\lambda_i \leftarrow \lambda_i - b_t \cdot g_t$
12     $c \leftarrow c + b_t \cdot p_t$
13 for $j \leftarrow 1$ to $o$ do
14     $w_j \leftarrow (b_j \cdot \mu_{ij}) / \sum_{j=1}^{o}(b_j \cdot \mu_{ij})$
15 return $c, \boldsymbol{w}, \boldsymbol{b}$
```

Algorithm 3: PlanRetrieval

```
Input: Weight path $\mathbb{W}$; QoS path $\mathbb{Q}$; Number
       of DNN models $n$; Request strength $\lambda$;
       Response time constraint $\tau$
Output: Load distribution plan $\mathcal{W}$;
        Microservice instance lunch count $\mathcal{B}$
1  $\boldsymbol{p} \leftarrow O_{1\times n},\ \ j \leftarrow 1,\ \ k \leftarrow \phi$
2  for $i \leftarrow n$ to $1$ do
3      $p_i \leftarrow \mathbb{W}[i, j, k]$
4      $k \leftarrow \mathbb{Q}[i, j, k]$
5      $j \leftarrow j - p_i$
6  for $i \leftarrow 1$ to $n$ do
7      $c, \boldsymbol{w}, \boldsymbol{b} \leftarrow \mathrm{OptSubPlan}(i, p_i \cdot \lambda, \tau)$
8      $\mathcal{W}[i, :] \leftarrow \boldsymbol{w} \cdot p_i$
9      $\mathcal{B}[i, :] \leftarrow \boldsymbol{b}$    // $\mathcal{W}[i, :]$ and $\mathcal{B}[i, :]$ are
           the $i$-th row of matrix $\mathcal{W}$ and $\mathcal{B}$
10 return $\mathcal{W}, \mathcal{B}$
```

Algorithm 1 shows the process of the proposed TDDP algorithm, it can be seen that it starts with initializing a lookup table $\mathbb{M}$ for the recursion of the dynamic programming, a weight distribution path table $\mathbb{W}$, and a QoS constraint path table $\mathbb{Q}$ with empty set (as shown in line 1). Then, it steps to the dynamic programming recursion procedure (as shown in line 3). After that, it gets the cost of optimal orchestration plan C and begins to fetch the details of the optimal plan (i.e., load distribution plan $\mathcal{W}$ and instance launch plan $\mathcal{B}$) by revisiting the search path $\mathbb{W}$ and $\mathbb{Q}$ (as shown in line 4). And finally, it returns the optimal orchestration plan and its deployment cost (as shown in line 5).

Recursive equation is the key component of designing a dynamic programming method[9], and that of the proposed TDDP method can be described as:

$$\begin{cases} F_i(w,\phi) = \min\limits_{0\le w_i\le 1}\Big\{ f_i(w_i,\phi_i) + \\ \qquad\qquad F_{i-1}(w-w_i, \phi - w_i\cdot q_i)\Big\}, \; i>1 \\ F_1(w,\phi_1) = f_1(w,\phi_1) \end{cases}$$

?
(19)

where $F_i(w)$ denotes the deployment cost of allocating the proportion of w load to the microservice instances that employ the first i DNN models and $f_i(w_i)$ denotes the cost of allocating the proportion of w_i load to the instances that employ the i-th DNN model. The recursion process is shown as function *DP* in algorithm 1 (as shown in lines 6-28), and $f_i(w_i)$ is implemented by function *OptSubPlan* as shown in Algorithm 2.

As represented in the recursive equation (19), the function *OptSubPlan* takes the target DNN model type and the allocated request load as input, it first calculates the maximum load of different container types that can bear within the constraint of response time τ, and evaluate the cost-performance of different container types (as shown in lines 2-4 in Algorithm 2). Then, it sorts the maximum load vector $\boldsymbol{g}$ and the cost-performance vector $\boldsymbol{h}$ of different container types according to the descending order of $\boldsymbol{h}$ (as shown in lines 5-6). After that, it begins to pick instances with the highest cost-performance in turn to fulfill the allocated request load λ_i (as shown in lines 7-12). And finally, it fairly distributes the load λ_i to a different type (i.e., container type) of instances according to their actual inference speed (as shown in lines 13-14).

We can obtain the deployment cost of the optimal multi-DNN-Model orchestration plan after the recursion of dynamic programming (as shown in line 3 in Algorithm 1). However, to get the detailed plan (i.e., $\mathcal{W}$ and $\mathcal{B}$), we need to revisit the decision path of the optimal solution. A backtracking-based orchestration plan retrieval method is shown in Algorithm 3.

For the *PlanRetrieval* procedure (i.e., Algorithm 2), the time complexity of initializing vector b, w, g, h is $O(o)$ (as shown in line 1), where o is the count of available container types. The time complexity of evaluating the cost-performance of different container types is also $O(o)$ (as shown in line 2~4). The time complexity of rearranging the index of h, g according to the descending order of h is $O(o \log o)$ (as shown in line 5~6). Finally, the time complexity of fulfilling the allocated λ_i and deciding the load distributed to different types of instances are both $O(o)$. Therefore, the total time complexity of the *OptSubPlan* procedure is $O(o \log o)$;

For the *PlanRetrieval* procedure (i.e., Algorithm 3), the time complexity of initializing vector p is $O(o)$ (as shown in line 1). The time complexity of revisiting the decision path to get the load distribution among different DNN models is $O(n)$ (as shown in lines 2~5), where n is the count of candidate DNN models for an edge AI microservice. Finally, the time complexity of building the orchestration plan in a backtracking way is $O(no \log o)$. Therefore, the total time complexity of the *PlanRetrieval* procedure is $O(no \log o)$.

As shown in Algorithm 1, the time complexity of initializing searching path table $\mathbb{M}$, $\mathbb{W}$, and $\mathbb{Q}$ is $O(n \cdot m \cdot (1/g))$, where n is the count of candidate DNN models for a certain edge AI microservice, m the discretized QoS value, and g the granularity of loads allocation (set to 1% in this paper according to the spring cloud gateway configuration). Due to $1/g$ equals 100 in this paper, which is a constant and thus the initialization time complexity can be represented as $O(nm)$. According to the recursive equation defined in E.q. (19), the time complexity of the proposed dynamic programming procedure is $O(n(1/g)o \log o)$, which can be represented as $O(no \log o)$. Finally, due to the time complexity of *PlanRetrieval* procedure is $O(no \log o)$, therefore the total time complexity of the proposed TDDP algorithm is $O(nm{+}no \log o)$. In real-world production environments, the candidate DNN models count n is usually less than 50; the QoS metrics (i.e., precision and accuracy) are usually correct to two decimal places, which means m less than 10^4; and the available container types o are usually less than 20 (Amazon Elastic Container Service provides 5 kinds of configurations). Therefore, the time complexity of the TDDP algorithm would not become the bottleneck of the proposed approach, the follow-up experiments based on real-world applications also show that it is capable of making orchestration decisions at the milliseconds level.

4.2. Three Level Load Balance-based Algorithm for P2

For the robust multiple-edge AI microservice instance placement problem, we aim to achieve a three-level-load-balance (i.e., load balance among edge servers, load balance among AI microservices, and load balance among different DNN models for a certain microservice) to reduce the impact on user-perceived QoS resulted by the edge servers' failures or edge network's disconnection.

As shown in Algorithm 4, the proposed TLLB methods first calculate the resource proportion of every edge server n E (as shown in lines 2-3), this is to estimate the targets load of edge servers to guide the follow-up placement. Then, it begins to perform the placement step for every orchestration plan that has been submitted (as shown in lines 4-25).

In the placement procedure for a certain plan (i.e., an edge AI microservice), it first calculates the optimal

Algorithm 4: TLLB for **P2**

```
Input: Submitted orchestration plans $\boldsymbol{P}$; $I_h$ the set of microservice instances of plan $P_h$ ;Set of edge servers $E$
Output: Placement plan $\mathcal{P}$
1  $\boldsymbol{z}, \boldsymbol{t}, \boldsymbol{s} \leftarrow O_{1\times|E|}$,  $\mathcal{P} \leftarrow O_{|I|\times|E|}$,  $d \leftarrow 0$
2  for $e_k \in E$ do
3      $z_k \leftarrow$ get the resource proportion of server $e_k$ in $E$
4  for $h \leftarrow 1$ to $|P|$ do
5      $L_h \leftarrow$ calculate the total load of instances in $P_h$
6      $\boldsymbol{R}_h \leftarrow$ calculate load ratios of the instances with different DNN models in $P_h$
7      $I_h \leftarrow$ sort $I_h$ in a descending order according to instances' required resources
8      foreach $e_k \in E$ do
9          $t_k \leftarrow L_h \cdot z_k$
10     while $|I_h| \neq 0$ do
11         $r \leftarrow$ get the resource required by $s_h^1$
12         foreach $e_k \in E$ do
13             $s_k \leftarrow t_k - r$
14         $C \leftarrow$ find all candidate edge servers in $E$ with the same score: max($\boldsymbol{s}$)
15         foreach $e_k \in C$ do
16             if no instance with the same DNN model of $s_h^1$ that has been placed to $e_k$ then
17                 $d \leftarrow k$
18                 break
19         if $d = 0$ then
20             foreach $e_k$ in $C$ do
21                 $s_k \leftarrow$ calculate the score of $e_k$ according to E.q. (20)
22             $d \leftarrow \underset{1\leq k\leq |C|}{\text{argmin}}\, s_k$
23         $t_k \leftarrow t_k - r$
24         $\mathcal{P}(s_h^1, d) \leftarrow 1$
25         $I_h \leftarrow$ remove $s_h^1$ from $I_h$
```

loads of different edge servers for the current microservice to achieve the first two load balances (as shown in lines 8-9). Then, it begins the loop of placing the instance with the highest required resource, i.e., s_h^1, in turn to its most suitable server. To find the target edge serves for every microservice instance, it first calculates the distance of an edge server's load after placement to its optimal load calculated in the previous step (as shown in lines 11-13). The proposed TLLB method tends to place the current instance to the edge serve with the shortest such distance, the reason for doing this is because the later instances to be placed have lower required resources, and this strategy helps to fill up the target loads of edge servers more smoothly. Note that, there could be multiple edge servers that have the same distance, and we use C to denote them. Under this circumstance, we consider the load balance among instances that employ different DNN models to finalize the target edge server. It can be seen that it first tries to find an edge server without being placed any instances with the same DNN model (as shown in lines 15-18). If there is no edge server qualified, TLLB will evaluate a score for every server in C and place s_h^1 to the server with a minimum score (as shown in 19-25). Let $\boldsymbol{R}_h$ and $\boldsymbol{\gamma}_k$ denote the load ratios of the instances with different DNN models in plan P_h and edge server e_k respectively, to achieve the third level of load balance, we define the score of server in C to the cosine distance between $\boldsymbol{R}_h$ and $\boldsymbol{\gamma}_k$:

$$s_k = \frac{\boldsymbol{R}_h \cdot \boldsymbol{\gamma}_k}{\|\boldsymbol{R}_h\|\|\boldsymbol{\gamma}_k\|}. \tag{20}$$

It can be seen that TLLB prefers the edge server with better load balances on instances with different DNN models. That is because different DNN models have different QoS, and the fluctuation of user perceived QoS incurred by server crashes or network failures can be alleviated by performing third load balance.

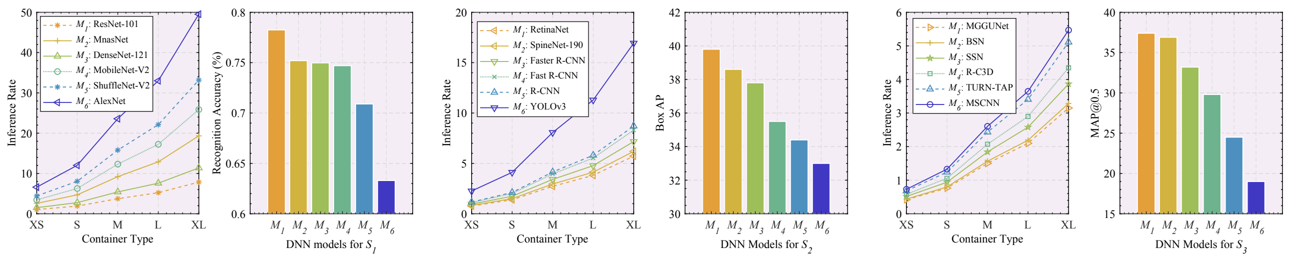
For a submitted orchestration plan P_h, the time complexity of calculating the total load, load ratios with different DNN models, and sorting microservice instance set I_h is $O(|I_h|\log|I_h|)$ (as shown in lines 5~7 in Algorithm 4). The time complexity of finding all candidate edge servers C in E for a certain instance is $O(|E|\log|E|)$ (as shown in lines 11~14). The time complexity of placing a microservice instance to the most suitable candidate is $O(|C|)$, where $|C| \leq |E|$. Therefore, the total time complexity of generating the microservice instance placement solution for a submitted orchestration plan is $O(|I_h| \cdot (\log|I_h| + |E|\log|E|))$.

5. Experiments and Analysis

To verify the effectiveness and efficiency of the proposed solution, a series experiments based on real-world edge server configurations and AI microservices performances are conducted. In this section, we first introduce the settings of our experiments, then show the results of comparisons between ours and the state-of-the-art ones.

5.1. Experiments Settings

We consider 4 kinds of real-world edge servers for edge infrastructure providers to build their edge resource pool, Table 1 detail their configurations and count. Edge AI application vendors acquire edge resources in a container way and we consider 5 kinds of container types as Amazon AWS Fargate did, Table 2 shows their configurations and prices. There are



(a) Image classification service S_1. (b) Object detection service S_2. (c) Action recognition service S_3.

Figure 6: Inference rate and QoS of different DNN models on containers with different configurations.

Table 1
Edge servers configurations

Server Type	Cores	RAM	Storage	Network	Count
Huawei TaiShan 2280E	16	32 GB	2TB	4×10 GE	10
ZTE E5430 G4 MEC	32	64 GB	2TB	2×10 GE	4
Dell PowerEdge XE2420	32	96 GB	8TB	2×10 GE	4
Lenovo ThinkServer SE550	48	128 GB	12TB	2×10 GE	2

Table 2
Container configurations and prices

Container Type	vCPU	RAM	Per vCPU per hour (USD)	Per GB per hour (USD)
XS	0.25	0.5 GB	0.0032	0.0004
S	0.5	1 GB	0.0064	0.0007
M	1	2 GB	0.0129	0.0014
L	2	4 GB	0.0258	0.0028
XL	4	8 GB	0.0515	0.0057

3 kinds of edge AI microservices (i.e., image classification S_1, object detection S_2, and action recognition S_3) and we consider 6 candidate DNN models for each of them. Specifically, ResNet 101, MnasNet, DenseNet 121, MobileNet V2, ShuffleNet V2, and AlexNet models are available for image classification applications; RetinaNet, SpineNet-190, Faster R-CNN, Fast R-CNN, R-CNN, and YOLOv3 for object detection; MGGUNet, BSN, SSN, R-C3D, TURN-TAP, and MSCNN for action recognition. As shown in Figure 6, we collect the QoS (i.e., response time and accuracy) of different edge AI microservices that employ different DNN models on different kinds of containers to conduct our experiment.

The proposed approaches and all baselines are implemented with MATLAB R2021b and all experiments are conducted on a PC with Intel Core i5 (Quad-Core, 3.8GHz), 8Gb RAM, and 512Gb storage. All cases are evaluated 50 times and the mean performances are recorded.

5.2. Deployment Cost Evaluation

For the multiple DNN model orchestration problem of edge AI microservice, we consider the following methods as baselines:

- **DFS** [28]: finding the optimal orchestration plan by performing a Depth-First-Search (DFS);
- **BGA + BAR** [25]: a meta-heuristic algorithm that first employ the binary genetic algorithm to find the preliminary solution, then a bottleneck-analysis-based solution is performed to finalize the orchestration plan;
- **SG** [15]: a simple greedy heuristic that only chooses one DNN model that satisfies the QoS constraints and always selects the container type with the lowest price;
- **BG** [10]: a bidirectional greedy heuristic that always chooses the DNN model with the highest QoS inference time ratio and cheapest container type that fulfills the QoS constraints at different stages.
- **GrandSLAm** [13]: a heuristic microservice execution framework that only consider a single container type (i.e., M size in our experiments), it firsts pushes candidate microservice instances into an ascending priority queue in terms of estimated slack time, then increases the batch count of instances until QoS constraints are about to be unsatisfied.

The QoS constraints and request load of 3 edge AI microservices in our experiment environments are shown in Table 3.

Results: Figure 7 compares the deployment cost incurred by the proposed TDDP approach and its peers under different load situations and QoS constraints, and Figure 8 shows the decision time (i.e., CPU elapsed time) of them. It can be seen that the proposed TDDP approach achieves the lowest deployment in all three microservice cases, more specifically, the deployment cost of TDDP are 1.8%, 1.1%, and 0.67% lower than DFS in three cases on average; 67.7%, 68.66%,

Table 3
QoS constraints and request load settings

Edge AI Microservice	QPS	Response time constraint (s)	QoS constraint
S_1: Image classification	50 ∼ 600	0.2 ∼ 1	65% ∼ 74% (Accuracy rate)
S_2: Object detection	50 ∼ 600	0.2 ∼ 1	34 ∼ 39 (Box AP)
S_3: Action recognition	50 ∼ 300	0.8 ∼ 1.6	22 ∼ 34 (MAP@0.5)

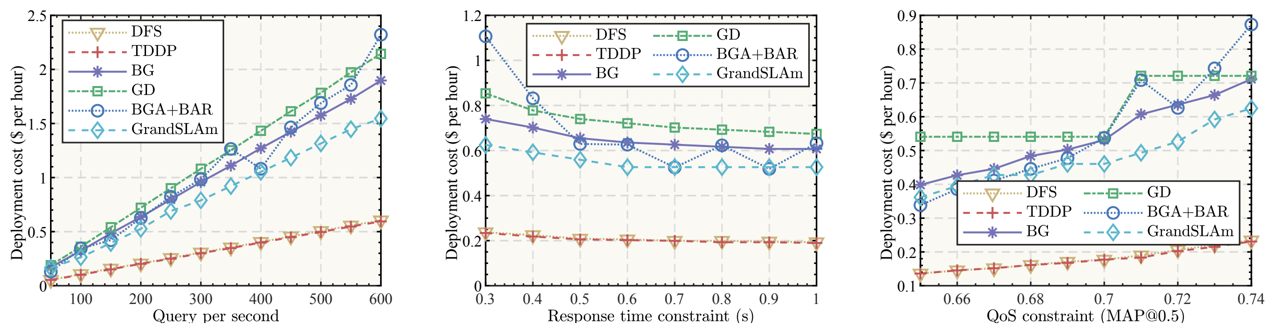
and 65.79% lower than BG in three cases on average; 71%, 73.15%, and 70.99% lower than GD on average; 70.76%, 35.54%, and 41.20% lower than BGA + BAR on average; 62.73%, 56.6% and 56.63% lower than GrandSLAm on average. DFS performs close to the proposed TDDP deployment cost but at the cost of a higher decision time. We can learn from the Figure 8 that the BG and SG approach gets the lowest decision time, the proposed TDDP costs slightly higher time than them but much lower time than DFS and BGA + BAR methods, more specifically, the decision time of TDDP are 57.39%, 62.16%, and 73.25% lower than DFS in three cases on average; 94.04%, 80.88%, and 83.4% lower than BGA + BAR on average.
Analysis: The reason why the proposed TDDP gets the lowest deployment cost lies in that it investigates every possible multi-DNN model orchestration plan as DFS did. But different from the exhaustive search of DFS, the dynamic programming and pruning features of the proposed TDDP can significantly reduce the search scope and thus shorten the decision time. The TDDP outperforms the BG because it leverages the advantages of different DNN models and employs multiple DNN models to fulfill the QoS requirement of a single edge AI microservices. Through the GD heuristic gets the lowest decision time, its simple greedy deployment strategy limits its capability of finding high-quality deployment plans. Same as a heuristic method, the BGA + BAR achieves a better performance than GD, however, its iterative optimization suffers from high time complexity and results in the highest CPU elapsed time. The reason why TDDP outperforms the GrandSLAm lies in that TDDP considers various container types with different configurations to build orchestration plans, which helps to smoothly fulfill the various request demands. While the GrandSLAm only considers one type of container, which may lead to unnecessary resource waste and finally result in high deployment costs.

5.3. Effects Analysis of Resources Failure

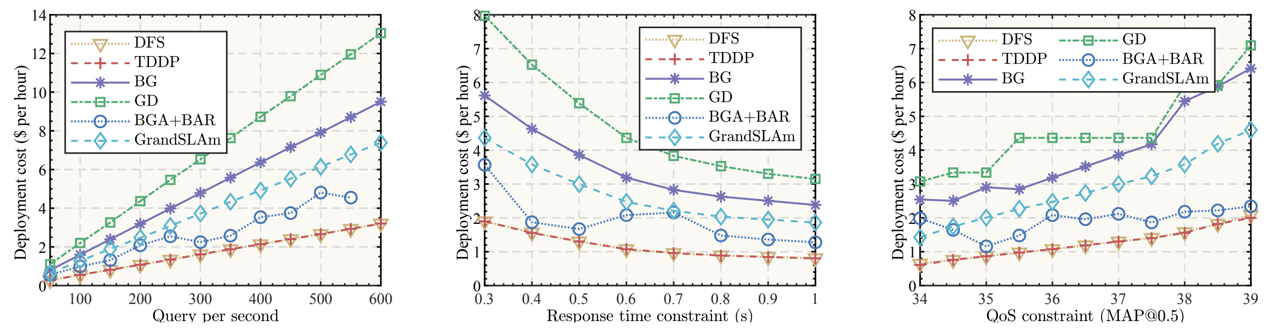
In our experiments, we also evaluate the effect of different microservice instance placement approaches when encountering edge resource failures. As related studies[37, 3] did, we randomly choose a certain proportion of edge nodes and disable them to simulate edge resource failures. For the robust edge AI microservice placement problem, we consider the following real-world container scheduling policies as baselines:

- **LRP [5]**: Least-Requested-Priority placement policy that always schedules pods (i.e., new microservice instances) to edge nodes with the lowest resource utilization rate;
- **LRP-E [5]**: similar to LRP except that the pods are preferentially placed to the edge nodes with the lowest load;
- **Spread [20]**: it prefers placing newly arriving instances to the edge node with the lowest number of hosted containers;
- **SSP [19]**: Selector-Spread-Priority placement policy that aims at distributing the instances of the same microservice to different edge nodes;
- **Random**: a random placement policy that randomly selects edge nodes to host microservice instances.

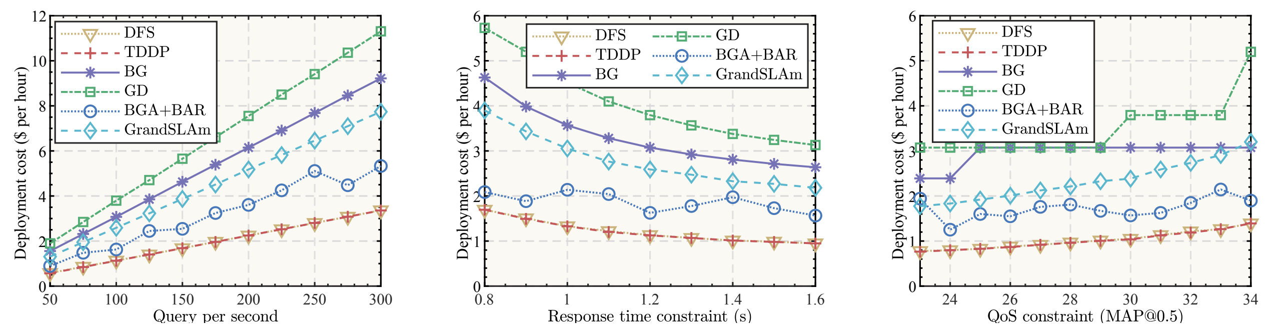
where LPR, LPR-E, and SSP are candidate pod placement policies of Kubernetes, and Spread is the default container placement policy of Swarm. We control the proportion (from 10% to 50%) of edge resources that encounter failures and observe the impact of these failures on the QoS of deployed edge AI microservices.
Results: Figure 9 shows the change in response time and accuracy rates of the deployed 3 edge AI microservices faced with different scales of failures. As in the previous experiment, the deployed 3 applications are image classification application (IC APP); Object detection application (OD APP); and Action recognition application (AR APP); It can be seen that with the increase of unavailable edge resources, the response time of all three microservices shows an increasing trend, and the response time fluctuation of the proposed TLLB placement strategy is much lower than other policies. Besides, the fluctuation of the accuracy rates under the TLLB placement strategy is also much lower than other policies. Table 4 shows the numerical analysis of them, and it can be seen that the proposed TLLB placement strategy achieves the lowest load standard deviations on both the edge server view and application view, which indicates less and limited QoS variation when an encounter with edge resources failures.



(a) Image classification microservice.



(b) Object detection microservice.



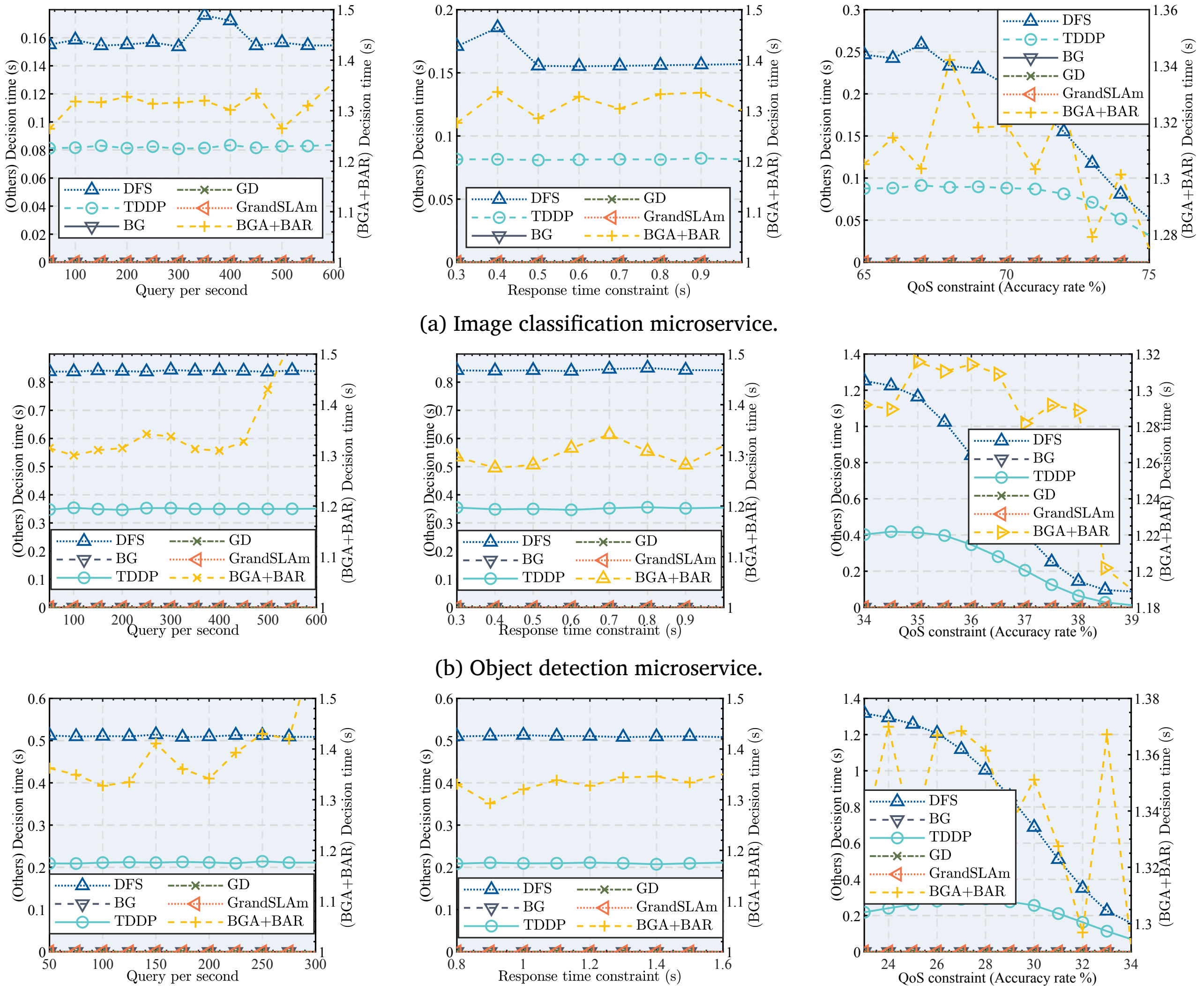
(c) Action recognition microservice.

Figure 7: Comparison of deployment cost.

Analysis: The proposed TLLB placement approach outperforms its peers because it not only considers the load balance between edge servers but also balances the load among different applications. In this way, the deployed application share the risk of edge resources failures and the potential cascade crashes are avoided if they have a dependent relationship. Besides, it also considers the load balance among different DNN models for a certain microservice, which further guarantees the stationarity of deployed edge AI microservices, i.e., minimizing the QoS fluctuation as far as possible when encountering edge resource failures. To sum up, the three levels load balance strategy of the proposed TLLB placement approach guarantees the fairness of risk and load in the edge server, deployed application(microservice), and DNN model levels, which helps to eliminate the impact of edge resources failures as far as possible (as shown in the lowest response time in Table 4).

6. Related Works

In recent years, the philosophy of cloud-native has been widely accepted by mainstream software providers and become a guideline to fully embrace the cloud ecosystem in the post-cloud era. The key idea of building an application that meets the cloud-native requirement is to employ the microservice architecture to improve flexibility and maintainability, and fully leverage the cloud infrastructure to achieve elastic scaling, dynamic scheduling, and higher resource utilization [16]. In this section, we first review the related studies on AI microservices and the recent progress of microservices governance at the edge. Then, we analyze the limitation of current studies and discuss the meaning/necessity of investigating the cost-effective and robust deployment of AI microser-


(a) Image classification microservice.

(b) Object detection microservice.

(c) Action recognition microservice.

Figure 8: Comparison of execution time.

Table 4
Numerical analysis of the edge resource failure experiment

Algorithm	Std of Edge servers' load	Std of App1's load	Std of App2 load's	Std of App3's load	Average response time
TLLB	**0.0078**	**0.0066**	**0.0067**	**0.0031**	**16.3313**
LRP	0.3778	0.2660	0.4245	0.3728	18.3556
LRP2	0.4326	0.2660	0.4084	0.3794	20.8695
Spread	0.5134	0.1751	0.1625	0.2056	23.0064
SSP	0.5511	0.1751	0.1746	0.2171	23.6266
Random	0.4690	0.1712	0.1921	0.2191	20.1012

vice in the edge environment.

6.1. AI Microservices

With the increasing popularity of AI-based applications among end-users, more and more application providers begin to deliver AI services in a microservice manner, and the management of AI microservice thus becomes a hot issue. For example, Kannan *et al.* [13] proposed an SLA-guaranteed microservice execution framework for AI and ML applications, shorts for GrandSLAm. They first analyze the difference between classic and AI/ML-based microservices in terms

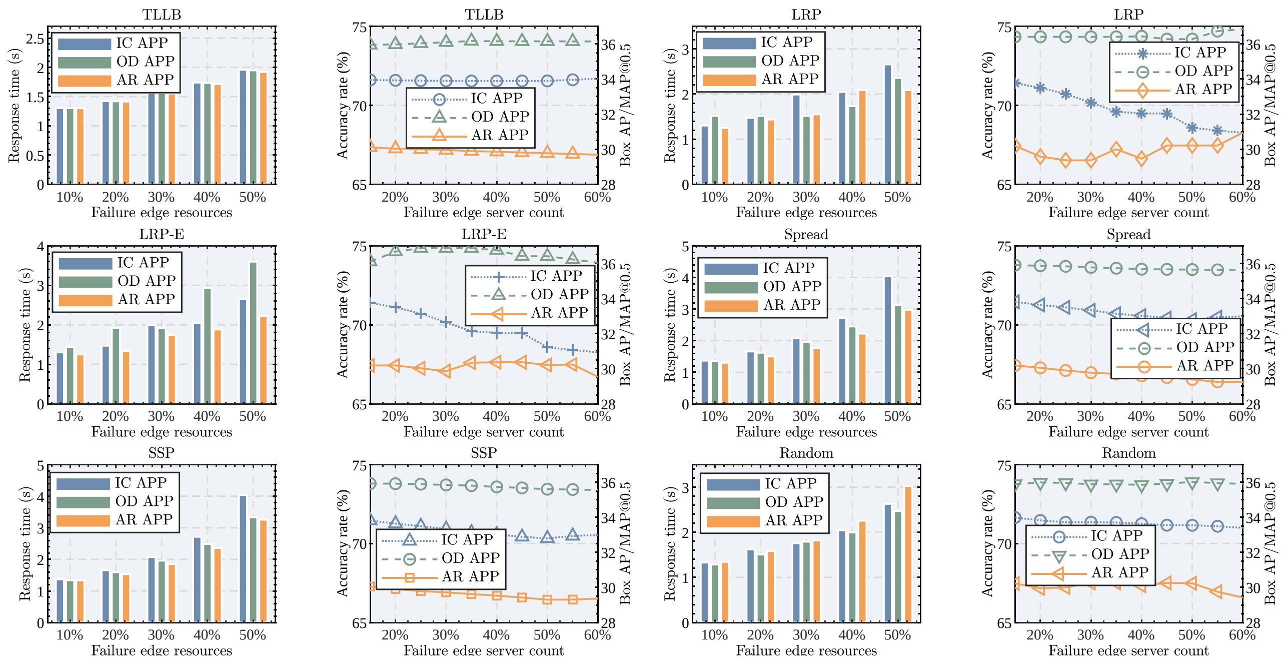


Figure 9: QoS variation after encounter edge resource failures.

of specific SLA metrics. Then, they train a regression-based model to estimate the completion time of the coming request. And finally, they derive the individual stage SLAs for each microservice/stage based on the predicted response time and developed a dynamic batching algorithm to schedule requests while meeting the SLA constraints. To achieve elastic and scalable Network Function Virtualization (NFC), Nekovee *et al.* [22] proposed an AI-enabled microservice architecture and analyzed its potential features for live streaming, smart safety, and enterprise VPN. Chang *et al.* [2] developed a composable and self-evolving microservices-based approach to transform AI-accelerated applications into secure, scalable, and composable enterprise microservices. And Rausch *et al.* [27] investigated the possibility of realizing a seamless end-to-end intelligent edge system by employing the AI-empowered cyber-physical fabric. For the secure and reliable deployment of AI microservices, Muthusamy *et al.* [21] employed AI methods to understand the relation between AI models and business Key Performance Indicators (KPIs), and proposed a data-driven approach to detect the potential deployment risks. While Zhao *et al.* [49] targeted the AI model sharing scenarios and proposed a microservices-based ML model packaging/sharing platform (shorts for Acumos).

6.2. Microservices Governance at Edge

Meanwhile, to smoothly deliver QoS-guaranteed services at the edge and improve the utilization of precious edge computing resources, the smart governance of microservices at the edge has attracted a lot of attractions. For example, to continuously provision QoS-guaranteed services and achieve self-adaptive system governance when faced with massive requests and personalized demands, He *et al.* [11] proposed EPF4M (a programming framework) and EI4MS (an infrastructure for self-adaptive microservice systems) to build a cloud–edge environment-based microservice architecture. In their approach, service systems are allowed to redeploy their microservices with the changes of the fluctuant QoS requirements, and they proposed a two-phase strategy to minimize the redeployment overhead. In order to achieve a user-mobility-aware microservice redeployment, they also proposed three heuristics [12] and evaluated their performances by integrating them into Kubernetes. Samanta *et al.* [29] considered the network delay and network price as microservice scheduling targets and proposed a Lagrangian multiplier-based dynamic microservice scheduling framework. Similarly, Wang *et al.* [38] took the mobility of end-users into consideration and considered the service delay and cost as scheduling targets. They first proposed a dynamic programming-based offline microservice coordination algorithm to yield global optimal schedules. However, this offline algorithm heavily relies on prior knowledge and suffers from low efficiency. Then, they formulated the microservice scheduling problem to a Markov decision process (MDP) and proposed a reinforcement learning-based online solution. Liu *et al.* [18] proposed a fuzzy-control-based algorithm (FSODM) to autonomously scale the deployed edge microservice in the runtime. Zhao *et al.* [45] investigated the redundant deployment of microservices in distributed edge environments. They proposed a Genetic algorithm-based algorithm for the edge server selection and a Monte Carlo simulation-based approach for the redundant placement framework. However, their solution only supports edge microservice applications with a sequential combinatorial structure. Filip *et al.* [8] proposed a novel microservice scheduling model by performing a particular mathematical formulation on heterogeneous cloud-edge environments. Villari *et al.* [35] proposed user-location-aware edge microservice orchestration deployment approach by introduce server geographical constraints.

A careful investigation into the aforementioned studies shows that they are still limited in three ways: 1) the scheduling granularity of most existing AI microservice governance solutions still remains at the service level, seldom touching the DNN model level. However, dynamically changing the orchestration plan of different DNN models to fulfill the QoS requirement of the microservices could further reduce the deployment cost of EAPs; 2) most current microservice placement algorithms only consider the load balance and deployment affinity on the edge nodes level or application level. However, when we consider the mixed orchestration of different DNN models to implement a microservice, applying these traditional placement strategies may fail to fulfill the QoS constraints; 3) most of the existing studies mainly focus on one of the microservice orchestration or placement problems on the edge environments only. However, in real-world application scenarios, we usually need to integrate both of them into an edge AI microservice governance solution, and composing orchestration and placement strategies with different design goals and philosophies may result in low system efficiency and additional operational overhead. Thus, the edge AI microservice orchestration and placement problems need to be solved together and the corresponding strategies should supplement each other to reduce internal friction.

Therefore, the coherent and collaborative orchestration and placement algorithms, which consider the mixed orchestration of different DNN models to implement an edge AI microservice, are in high demand to solve the cost-effective and robust deployment of AI microservice in the MEC environments.

7. Conclusion and Further studies

This study proposed a novel cost-effective multiple DNN model mixed orchestration problems for edge AI microservice deployment for mobile application providers and its corresponding robust microservice instance placement problem for edge infrastructure providers. To solve the first problem, we developed a three-dimension-dynamic-programming-based algorithm that can yield the optimal deployment plan of edge AI microservice when considering multiple DNN model orchestration. For the microservice instance placement problem, we proposed a three-level balance method that is capable of balancing the load between servers, applications, and DNN models. The proposed algorithms can easily be integrated into current popular microservice governance and edge resources management platforms (e.g., Spring Cloud, Dubbo, KubeEdge, etc). The experiment based on real-world edge AI applications and DNN models has demonstrated that the proposed orchestration and placement methods can significantly reduce the deployment cost of edge AI service and the performance degradation when encountering failures compared with traditional approaches.

For our further studies, we will address the following concerns: 1) some temporal data mining methods (e.g., LSTNet and TPA-LSTM) can be employed to predict end-users future request strength, and based on this information, the corresponding runtime edge AI services automatic scaling approaches can be developed to save more deployment costs for mobile application providers; 2) we only consider public edge resource providers in this study, for our future works, a mixed edge resource pool constructed by private edge cloud and public edge cloud will be well investigated to achieve a more cost-effective edge AI service governance; 3) more Quality-of-Experience (QoE) metrics of edge AI microservices such as service satisfaction, lagging time, and reliability should be well investigated to build a user-experience-centric edge AI service provision model.

References

[1] Carnell, J., Sánchez, I.H., 2021. Spring microservices in action. Simon and Schuster.

[2] Chang, R.N., Bhaskaran, K., Dey, P., Hsu, H., Takeda, S., Hama, T., 2020. Realizing a composable enterprise microservices fabric with ai-accelerated material discovery api services, in: 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), IEEE. pp. 313–320.

[3] Chantre, H.D., da Fonseca, N.L., 2018. Multi-objective optimization for edge device placement and reliable broadcasting in 5g nfv-based small cell networks. IEEE Journal on Selected Areas in Communications 36, 2304–2317.

[4] Cisco, U., 2020. Cisco annual internet report (2018–2023) white paper. Cisco: San Jose, CA, USA .

[5] Da Silva, G.F., Brasileiro, F., Lopes, R., Morais, F., Carvalho, M., Turull, D., 2020. Qos-driven scheduling in the cloud. Journal of Internet Services and Applications 11, 1–36.

[6] De Lauretis, L., 2019. From monolithic architecture to microservices architecture, in: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE. pp. 93–96.

[7] Feng, Y., Wang, G.G., 2022. A binary moth search algorithm based on self-learning for multidimensional knapsack problems. Future Generation Computer Systems 126, 48–64.

[8] Filip, I.D., Pop, F., Serbanescu, C., Choi, C., 2018. Microservices scheduling model over heterogeneous cloud-edge environments as support for iot applications. IEEE Internet of Things Journal 5, 2672–2681.

[9] Forootani, A., Iervolino, R., Tipaldi, M., Dey, S., 2022. Transmission scheduling for multi-process multi-sensor remote esti-

mation via approximate dynamic programming. Automatica 136, 110061.

[10] Haugland, D., 2007. A bidirectional greedy heuristic for the subspace selection problem, in: International Workshop on Engineering Stochastic Local Search Algorithms, Springer. pp. 162–176.

[11] He, X., Tu, Z., Xu, X., Wang, Z., 2019. Re-deploying microservices in edge and cloud environment for the optimization of user-perceived service quality, in: International Conference on Service-Oriented Computing, Springer. pp. 555–560.

[12] He, X., Tu, Z., Xu, X., Wang, Z., 2021. Programming framework and infrastructure for self-adaptation and optimized evolution method for microservice systems in cloud–edge environments. Future Generation Computer Systems 118, 263–281.

[13] Kannan, R.S., Subramanian, L., Raju, A., Ahn, J., Mars, J., Tang, L., 2019. Grandslam: Guaranteeing slas for jobs in microservices execution frameworks, in: Proceedings of the Fourteenth EuroSys Conference 2019, pp. 1–16.

[14] Kjorveziroski, V., Filiposka, S., 2022. Kubernetes distributions for the edge: serverless performance evaluation. The Journal of Supercomputing , 1–28.

[15] Lai, P., He, Q., Cui, G., Xia, X., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., Yang, Y., 2019. Edge user allocation with dynamic quality of service, in: International Conference on Service-Oriented Computing, Springer. pp. 86–101.

[16] Leppanen, T., Savaglio, C., Lovén, L., Jarvenpaa, T., Ehsani, R., Peltonen, E., Fortino, G., Riekki, J., 2019. Edge-based microservices architecture for internet of things: Mobility analysis case study, in: 2019 IEEE Global Communications Conference (GLOBECOM), IEEE. pp. 1–7.

[17] Li, X.Y., Lin, W., Chang, J.M., Jia, X., 2022. Transmission failure analysis of multi-protection routing in data center networks with heterogeneous edge-core servers. IEEE/ACM Transactions on Networking .

[18] Liu, C.C., Huang, C.C., Tseng, C.W., Yang, Y.T., Chou, L.D., 2019. Service resource management in edge computing based on microservices, in: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), IEEE. pp. 388–392.

[19] Mao, Y., Fu, Y., Zheng, W., Cheng, L., Liu, Q., Tao, D., 2021. Speculative container scheduling for deep learning applications in a kubernetes cluster. IEEE Systems Journal .

[20] Menouer, T., 2021. Kcss: Kubernetes container scheduling strategy. The Journal of Supercomputing 77, 4267–4293.

[21] Muthusamy, V., Slominski, A., Ishakian, V., 2018. Towards enterprise-ready ai deployments minimizing the risk of consuming ai models in business applications, in: 2018 First International Conference on Artificial Intelligence for Industries (AI4I), pp. 108–109. doi:10.1109/AI4I.2018.8665685.

[22] Nekovee, M., Sharma, S., Uniyal, N., Nag, A., Nejabati, R., Simeonidou, D., 2020. Towards ai-enabled microservice architecture for network function virtualization, in: 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), IEEE. pp. 1–8.

[23] Panek, G., Fajjari, I., Tarasiuk, H., Bousselmi, A., Toukabri, T., 2022. Application relocation in an edge-enabled 5g system: Use-cases, architecture and challenges. IEEE Communications Magazine .

[24] Peng, Q., Wu, C., Xia, Y., Ma, Y., Wang, X., Jiang, N., 2021. Dosra: A decentralized approach to online edge task scheduling and resource allocation. IEEE Internet of Things Journal 9, 4677–4692.

[25] Peng, Q., Xia, Y., Wang, Y., Wu, C., Luo, X., Lee, J., 2019. Joint operator scaling and placement for distributed stream processing applications in edge computing, in: International Conference on Service-Oriented Computing, Springer. pp. 461–476.

[26] Porambage, P., Okwuibe, J., Liyanage, M., Ylianttila, M., Taleb, T., 2018. Survey on multi-access edge computing for internet of things realization. IEEE Communications Surveys & Tutorials 20, 2961–2991.

[27] Rausch, T., Dustdar, S., 2019. Edge intelligence: The convergence of humans, things, and ai, in: 2019 IEEE International Conference on Cloud Engineering (IC2E), pp. 86–96. doi:10.1109/IC2E.2019.00022.

[28] Sadeghiram, S., Ma, H., Chen, G., 2021. Priority-based selection of individuals in memetic algorithms for distributed data-intensive web service compositions. IEEE Transactions on Services Computing .

[29] Samanta, A., Tang, J., 2020. Dyme: Dynamic microservice scheduling in edge computing enabled iot. IEEE Internet of Things Journal 7, 6164–6174.

[30] Sanchez-Gallegos, D.D., Gonzalez-Compean, J., Carretero, J., Marin, H., Tchernykh, A., Montella, R., 2022. Puzzlemesh: A puzzle model to build mesh of agnostic services for edge-fog-cloud. IEEE Transactions on Services Computing .

[31] Štefanič, P., Cigale, M., Jones, A.C., Knight, L., Taylor, I., Istrate, C., Suciu, G., Ulisses, A., Stankovski, V., Taherizadeh, S., et al., 2019. Switch workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications. Future Generation Computer Systems 99, 197–212.

[32] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D., 2017. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. IEEE Communications Surveys & Tutorials 19, 1657–1681.

[33] Tang, J., Jalalzai, M.M., Feng, C., Xiong, Z., Zhang, Y., 2022. Latency-aware task scheduling in software-defined edge and cloud computing with erasure-coded storage systems. IEEE Transactions on Cloud Computing .

[34] Tang, L., D'Ariano, A., Xu, X., Li, Y., Ding, X., Samà, M., 2021. Scheduling local and express trains in suburban rail transit lines: Mixed–integer nonlinear programming and adaptive genetic algorithm. Computers & Operations Research 135, 105436.

[35] Villari, M., Celesti, A., Tricomi, G., Galletta, A., Fazio, M., 2017. Deployment orchestration of microservices with geographical constraints for edge computing, in: 2017 IEEE Symposium on Computers and Communications (ISCC), IEEE. pp. 633–638.

[36] Wang, F.Y., Qin, R., Wang, X., Hu, B., 2022. Metasocieties in metaverse: Metaeconomics and metamanagement for metaenterprises and metacities. IEEE Transactions on Computational Social Systems 9, 2–7.

[37] Wang, H., Xu, H., Huang, H., Chen, M., Chen, S., 2021. Robust task offloading in dynamic edge computing. IEEE Transactions on Mobile Computing .

[38] Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., Shen, X.S., 2019. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. IEEE Transactions on Mobile Computing .

[39] Wang, X., Han, Y., Leung, V.C., Niyato, D., Yan, X., Chen, X., 2020. Edge AI: Convergence of edge computing and artificial intelligence. Springer.

[40] Waseem, M., Liang, P., Shahin, M., 2020. A systematic mapping study on microservices architecture in devops. Journal of Systems and Software 170, 110798.

[41] Xing, T., Barbalace, A., Olivier, P., Karaoui, M.L., Wang, W., Ravindran, B., 2022. H-container: Enabling heterogeneous-isa container migration in edge computing. ACM Transactions on Computer Systems (TOCS) .

[42] Xu, J., Palanisamy, B., Ludwig, H., Wang, Q., 2017. Zenith: Utility-aware resource allocation for edge computing, in: 2017 IEEE international conference on edge computing (EDGE), IEEE. pp. 47–54.

[43] Xu, X., Liu, Q., Luo, Y., Peng, K., Zhang, X., Meng, S., Qi, L., 2019. A computation offloading method over big data for iot-

enabled cloud-edge computing. Future Generation Computer Systems 95, 522–533.

[44] Zhang, X., Zhang, J., Peng, C., Wang, X., 2022. Multimodal optimization of edge server placement considering system response time. ACM Transactions on Sensor Networks (TOSN) .

[45] Zhao, H., Deng, S., Liu, Z., Yin, J., Dustdar, S., 2019. Distributed redundant placement for microservice-based applications at the edge. arXiv preprint arXiv:1911.03600 .

[46] Zhao, L., Li, B., Tan, W., Cui, G., He, Q., Xu, X., Xu, L., Yang, Y., 2022. Joint coverage-reliability for budgeted edge application deployment in mobile edge computing environment. IEEE Transactions on Parallel and Distributed Systems 33, 3760–3771.

[47] Zhao, L., Li, Z., Al-Dubai, A.Y., Min, G., Li, J., Hawbani, A., Zomaya, A.Y., 2021a. A novel prediction-based temporal graph routing algorithm for software-defined vehicular networks. IEEE Transactions on Intelligent Transportation Systems .

[48] Zhao, L., Zheng, T., Lin, M., Hawbani, A., Shang, J., Fan, C., 2021b. Spider: a social computing inspired predictive routing scheme for softwarized vehicular networks. IEEE Transactions on Intelligent Transportation Systems .

[49] Zhao, S., Talasila, M., Jacobson, G., Borcea, C., Aftab, S.A., Murray, J.F., 2018. Packaging and sharing machine learning models via the acumos ai open platform, in: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 841–846. doi:10.1109/ICMLA.2018.00135.

[50] Zhou, Y., Zhang, D., Xiong, N., 2017. Post-cloud computing paradigms: a survey and comparison. Tsinghua Science and Technology 22, 714–732.

Highlights

- Edge computing is a promising paradigm to accelerate novel mobile applications
- Edge computing paradigm is still restricted by limited resources and error-prone network
- Edge resources are usually provisioned in a Container-as-a-Service way
- Multiple DNN models with mixed deployment help to reduce the operating costs of edge application providers
- Considering multilevel load balance guarantees the robust of deployed edge AI microservices

Author contributions

Use this form to specify the contribution of each author of your manuscript. A distinction is made between five types of contributions: Conceived and designed the analysis; Collected the data; Contributed data or analysis tools; Performed the analysis; Wrote the paper.

For each author of your manuscript, please indicate the types of contributions the author has made. An author may have made more than one type of contribution. Optionally, for each contribution type, you may specify the contribution of an author in more detail by providing a one-sentence statement in which the contribution is summarized. In the case of an author who contributed to performing the analysis, the author's contribution for instance could be specified in more detail as 'Performed the computer simulations', 'Performed the statistical analysis', or 'Performed the text mining analysis'.

If an author has made a contribution that is not covered by the five pre-defined contribution types, then please choose 'Other contribution' and provide a one-sentence statement summarizing the author's contribution.

Manuscript title: Towards Cost-Effective and Robust AI Microservice Deployment in Edge Computing Environments

Author 1: Chunrong Wu

☒ Conceived and designed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☒ Collected the data
Specify contribution in more detail (optional; no more than one sentence)

☒ Contributed data or analysis tools
Specify contribution in more detail (optional; no more than one sentence)

☒ Performed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☒ Wrote the paper
Specify contribution in more detail (optional; no more than one sentence)

☐ Other contribution
Specify contribution in more detail (required; no more than one sentence)

Author 2: Qinglan Peng

☐ Conceived and designed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☒ Collected the data
Specify contribution in more detail (optional; no more than one sentence)

☒ Contributed data or analysis tools
Specify contribution in more detail (optional; no more than one sentence)

☒ Performed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Wrote the paper
Specify contribution in more detail (optional; no more than one sentence)

☐ Other contribution
Specify contribution in more detail (required; no more than one sentence)

Author 3: Yunni Xia

☒ Conceived and designed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Collected the data
Specify contribution in more detail (optional; no more than one sentence)

☐ Contributed data or analysis tools
Specify contribution in more detail (optional; no more than one sentence)

☒ Performed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Wrote the paper
Specify contribution in more detail (optional; no more than one sentence)

☐ Other contribution
Specify contribution in more detail (required; no more than one sentence)

Author 4: Yong Jin

☐ Conceived and designed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Collected the data
Specify contribution in more detail (optional; no more than one sentence)

☒ Contributed data or analysis tools
Specify contribution in more detail (optional; no more than one sentence)

☒ Performed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Wrote the paper
Specify contribution in more detail (optional; no more than one sentence)

☐ Other contribution
Specify contribution in more detail (required; no more than one sentence)

Author 5: Zhentao Hu

☐ Conceived and designed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☒ Collected the data
Specify contribution in more detail (optional; no more than one sentence)

☐ Contributed data or analysis tools
Specify contribution in more detail (optional; no more than one sentence)

☐ Performed the analysis
Specify contribution in more detail (optional; no more than one sentence)

☐ Wrote the paper
Specify contribution in more detail (optional; no more than one sentence)

☐ Other contribution
Specify contribution in more detail (required; no more than one sentence)

Chunrong Wu, received the B.S. degree in software engineering from Xinjiang University, Xinjiang, China, in 2016, the M.Eng. degree in software engineering from Zhejiang Univesity, Zhejiang, China, in 2018, and the Ph.D. degree in software engineering from Chongqing University, Chongqing, China, in 2022. She is currently a Lecturer with the School of Artificial Intelligence, Henan University. She has authored or coauthored more than 10 research publications. Her research interests are in edge computing, service computing, and data mining.

Qinglan Peng, received the B.S. degree in software engineering from Xinjiang University, Xinjiang, China, in 2016, the M.Eng. degree in software engineering from Zhejiang Univesity, Zhejiang, China, in 2018, and the Ph.D. degree in software engineering from Chongqing University, Chongqing, China, in 2022. He is currently a Lecturer with the School of Artificial Intelligence, Henan University. He has authored or coauthored more than 20 research publications. His research interests are in edge computing, service computing, and cloud computing.

Yunni Xia, received the B.S. degree in computer science from Chongqing University, Chongqing, China, in 2003, and the Ph.D. degree in computer science from Peking University, Beijing, China, in 2008. He is currently a Professor with the College of Computer Science, Chongqing University. He has authored or coauthored more than 100 research publications. His research interests are in service computing, cloud computing, edge computing, intelligent data processing, and software dependability.

Yong Jin, received the Ph.D. degree in computer science from Northwestern Polytechnical University, Xian, China. He is currently a Professor the School of Artificial Intelligence, Henan University. His research interests are in distributed computation and wireless sensor network.

Zhentao Hu, received the Ph.D. degree in computer science from Northwestern Polytechnical University, Xian, China. He is currently a Professor the School of Artificial Intelligence, Henan University. His research interests are in Intelligent Information Processing, Modeling and estimation of complex systems, and moving targets tracking.

Author Credit Statement

Chunrong Wu: Conceived and designed the analysis; Collected the data; Contributed data or analysis tools; Performed the analysis; Wrote the paper.

Qinglan Peng: Collected the data; Contributed data or analysis tools; Performed the analysis; Visualization; Methodology.

Yunni Xia: Conceived and designed the analysis; Performed the analysis.

Yong Jin: Contributed data or analysis tools; Performed the analysis.

Zhentao Hu: Collected the data; Supervision.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: