HyDK: A Hybrid DRL-KKT Framework for Latency-Critical Service Placement With Multi-Source Synchronization

Zhenli He, *Senior Member, IEEE*, Xiaolong Zhai, *Student Member, IEEE*, Mingxiong Zhao, *Member, IEEE*, Wei Zhou, and Keqin Li, *Fellow, IEEE*

***Abstract*—In mission-critical IoT-MEC environments, jointly optimizing service placement and resource allocation is intractable due to the high-dimensional coupling of discrete topological decisions with continuous resource dimensioning. Furthermore, traditional methods oversimplify dependencies, overlooking multi-source "Wait-for-All" synchronization and the stochastic variance of bursty workloads. To bridge these gaps, we propose HyDK, a variance-aware framework synergizing Deep Reinforcement Learning (DRL) with convex optimization. The core innovation is our Action Space Pruning mechanism. We theoretically decompose the hybrid decision space by solving the continuous sub-problem to optimality via a KKT-based convex optimization routine. This acts as a deterministic optimality backstop, effectively pruning continuous dimensions and allowing the agent to focus exclusively on the complex discrete topological search. To address physical realities, we construct a fine-grained Directed Acyclic Graph (DAG) model to capture data aggregation bottlenecks and integrate an M/G/1 queuing model incorporating the second moment of service time to mitigate long-tail latency risks. Trace-driven simulations using the Edge-IIoTset demonstrate that HyDK improves system responsiveness by up to 25.6% with significantly tighter confidence intervals compared to existing baselines.**

***Index Terms*—Action space pruning, deep reinforcement learning, directed acyclic graph, Karush–Kuhn–Tucker conditions, multi-source synchronization.**

Received 1 November 2024; revised 13 April 2026; accepted 18 April 2026. Date of publication 21 April 2026; date of current version 12 June 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62362068, Grant 62361056, and Grant 62301222, in part by the Applied Basic Research Foundation of Yunnan Province under Grant 202301AT070194 and Grant 202301AT070422, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant 2023SE208, and in part by the Innovation Foundation of the Engineering Research Center of Integration and Application of Digital Learning Technology, Ministry of Education, China, under Grant 1431007. *(Corresponding author: Mingxiong Zhao.)*

Mingxiong Zhao is with the Engineering Research Center of Cyberspace, Yunnan Key Laboratory of Software Engineering, Kunming 650500, China, also with the School of Software, Yunnan University, Kunming 650500, China, and also with the Engineering Research Center of Integration and Application of Digital Learning Technology, Ministry of Education, Beijing 100039, China (e-mail: jimmyzmx@gmail.com).

Zhenli He and Wei Zhou are with the Engineering Research Center of Cyberspace, Yunnan Key Laboratory of Software Engineering, Kunming 650500, China, also with the School of Software, Yunnan University, Kunming 650500, China (e-mail: hezl@ynu.edu.cn; zwei@ynu.edu.cn).

Xiaolong Zhai is with the School of Software, Dalian University of Technology, Dalian 116600, China (e-mail: zhaixiaolong@mail.dlut.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TSC.2026.3686168, provided by the authors.

Digital Object Identifier 10.1109/TSC.2026.3686168

I. Introduction

A. Motivation

THE exponential proliferation of Internet of Things (IoT) devices is reshaping the landscape of modern digital infrastructure, with projections estimating 30.9 billion connected devices by 2025 [1]. Integral to mission-critical domains such as smart cities, industrial automation, healthcare, and intelligent transportation [2], these devices generate massive streams of heterogeneous data demanding ultra-low latency processing. Mobile Edge Computing (MEC) has emerged as a paradigm shift to mitigate the latency bottleneck by pushing computation and storage capabilities closer to the network edge [3]. However, in resource-constrained IoT-MEC environments [4], the integration of diverse IoT services with limited edge capacities introduces a mathematically intractable challenge: the joint optimization of *service placement* and *resource allocation*.

Consider a latency-critical scenario in intelligent manufacturing [5], where a Compute-type Service (ComS) for anomaly detection depends on real-time telemetry from multiple sensors (e.g., temperature, pressure, and vibration). Strategically placing Cache-type Services (CSs) near the ComS is pivotal to minimizing data retrieval latency. Concurrently, resource allocation must dynamically partition the finite computing power among contending services to maximize throughput [6]. This problem is compounded by three fundamental challenges: the high-dimensional coupling of decision variables, the stochasticity of workload variance, and the strict synchronization constraints of multi-source dependencies.

First, existing literature often oversimplifies service dependencies. Despite advancements in optimization strategies [7], [8], [9], [10], [11], many studies model task relations as linear chains or independent mappings. This fails to capture the "Wait-for-All" synchronization constraints inherent in Industrial IoT (IIoT), where a ComS cannot commence execution until it aggregates data from *all* prerequisite CSs [5]. In such multi-source topologies, system performance is bottlenecked by the slowest data retrieval path. Neglecting these synchronization semantics leads to suboptimal placements where the spatial separation of a single critical data source can violate the stringent timing constraints of the entire control loop [12].

Second, the stochastic nature of IoT traffic is frequently underestimated. Traditional approaches often rely on mean-value

analysis (e.g., M/M/1 queuing models) to estimate resource usage, ignoring the *variance* (second moment) of service times. In high-concurrency environments like collision avoidance systems, traffic burstiness can cause transient queue build-ups even when average utilization is low [13], [14]. Optimization strategies that fail to account for this stochastic variance risk severe "long-tail" latency events, compromising the reliability of safety-critical applications.

Third, the joint optimization problem presents a severe algorithmic hurdle due to the heterogeneous coupling of discrete topological decisions (placement) and continuous numerical variables (resource allocation). Pure Deep Reinforcement Learning (DRL) agents struggle to converge effectively in such hybrid action spaces, often suffering from the curse of dimensionality. Conversely, traditional convex optimization methods lack the scalability to handle the combinatorial explosion of discrete placement options in dynamic networks. Therefore, a structural decomposition of the decision space is imperative to achieve both convergence speed and theoretical optimality. Taken together, these observations suggest that an effective solution should not treat placement and allocation as a monolithic black-box optimization task, but should instead jointly address synchronization-aware dependency modeling, variance-aware latency estimation, and structure-aware decomposition of the hybrid decision space.

B. Our Contributions

Motivated by the above observations, we propose **HyDK**, a variance-aware hybrid DRL-KKT framework for latency-critical service placement with multi-source synchronization. Rather than forcing a single learning agent to optimize the entire hybrid decision space, HyDK decomposes the problem according to its intrinsic structure: DRL is used to explore the discrete placement topology, while a KKT-based solver determines the continuous resource allocation for a given placement. This design reduces the learning burden of the agent while preserving analytical tractability in the continuous subproblem. Our specific contributions are as follows:

- *Synergistic Optimization via Action Space Pruning:* We propose a novel architectural decomposition where the continuous resource allocation sub-problem is solved via a rigorously derived bisection method based on KKT conditions. This mechanism acts as a deterministic "optimality backstop" embedded within the environment, effectively **pruning** the continuous dimensions from the DRL agent's action space. As a result, the agent only needs to focus on the high-level discrete placement search, which improves convergence efficiency while preserving feasibility in resource allocation.
- *Multi-Source Synchronization and Variance-Aware Modeling:* We construct a fine-grained **Directed Acyclic Graph (DAG)** model to explicitly capture the "Wait-for-All" dependency constraints, identifying and mitigating latency bottlenecks caused by multi-source data aggregation. Furthermore, we integrate an **M/G/1 queuing model** that incorporates the **second moment of service time**. This combination enables HyDK to capture both synchronization bottlenecks and long-tail latency risks caused by bursty workloads, which are not well handled by simplified dependency models or mean-value-only analyses.
- *Rigorous Validation with Statistical Evidence:* We designed simulation experiments based on the real-world **Edge-IIoTset** dataset to evaluate HyDK. Beyond standard mean-value comparisons, we also report confidence intervals to assess robustness under stochastic fluctuations. The results demonstrate that HyDK not only improves average system responsiveness compared to existing baselines but also delivers significantly tighter confidence intervals, proving its robustness against network jitter and stochastic fluctuations.

The remainder of this paper is organized as follows: Section II reviews the existing literature. Section III introduces the system model and problem formulation. Section IV details the proposed HyDK framework. Section V presents the experimental evaluation, and Section VI concludes this work.

II. Related Work

In this section, we review the state-of-the-art in service placement and resource allocation. We classify existing works based on their modeling granularity and algorithmic structures, highlighting the critical gaps that motivate our hybrid framework. Due to space limitations, a comprehensive analysis of related work is provided in Section I of the supplementary material, available online.

A. Service Placement and Dependency Modeling

Extensive research on IoT-MEC service placement has historically addressed either CSs or ComSs in isolation.

For ComSs, Chen et al. [7] proposed a decentralized placement algorithm for MEC systems, optimizing utility via techniques like Gibbs sampling and graph coloring. Badri et al. [9] introduced a multi-stage stochastic optimization approach balancing energy efficiency and QoS in dynamic user environments. Zhang et al. [10] employed clustering and nonlinear programming to optimize edge server deployment and service placement, while Bi et al. [11] proposed a multi-objective model for joint service placement and scheduling to reduce costs and improve service quality.

For CSs, Chen et al. [15] developed a deep-learning-based resource allocation framework for energy-efficient cache placement in MEC, while Xia et al. [16] employed Lyapunov optimization to develop an online collaborative caching algorithm. Chen et al. [17] introduced a TTL-based content placement framework in edge caching networks that demonstrated superior performance with YouTube data.

While effective for specific sub-problems, these decoupled approaches neglect the inherent data dependencies in modern IoT applications, where computation cannot proceed without the requisite data. Recognizing this limitation, recent research has increasingly considered the interdependencies between CSs and ComSs.

Yu et al. [18] proposed a collaborative offloading and caching method to reduce execution times in multi-user MEC scenarios. Ndikumana et al. [19] developed a model integrating communication, computation, and caching to optimize resource allocation, while Bi et al. [20] addressed computation offloading with a focus on energy consumption. Gao and Casale [21] proposed JCSP, a joint caching and service placement model that improves response times using layered queueing networks and genetic algorithms.

However, a significant limitation persists in these joint optimization works: they predominantly assume a simplified linear (one-to-one) dependency. This implies that a compute service requests data from a single specific cache source. Such oversimplification fails to capture the complexity of industrial scenarios where a single analytic task often requires synchronized data aggregation from multiple heterogeneous sensors. Unlike prior studies, this work advances the field by accurately modeling these Multi-source dependency constraints (Wait-for-all). We explicitly address the latency bottleneck caused by the slowest data retrieval path, thereby enhancing system performance through reduced data retrieval delays.

B. Stochastic Performance Modeling in DAG Architectures

DAG-based models offer valuable insights for microservice dependencies in distributed applications. Xu et al. [22] proposed the DSPLS algorithm to optimize task dependencies and completion times, while Liu et al. [23] developed GenDoc for efficient task placement with theoretical error analysis. Qu et al. [24] optimized placement by balancing node load and latency through an ILP model, while Goudarzi et al. [8] introduced a distributed approach using layered queueing networks.

Despite these advancements, a critical gap remains in how these models handle network dynamics. Most existing DAG approaches rely on deterministic or mean-value analysis, focusing solely on average response times. This overlooks the stochastic variance (jitter) of IoT workloads, where traffic bursts can cause transient queue build-ups even under low average load. By integrating a variance-aware queuing model into the optimization loop, our approach moves beyond mean-value estimation to explicitly mitigate the long-tail latency risks inherent in high-concurrency environments.

C. Algorithmic Evolution in Hybrid Decision Spaces

Various algorithms have been proposed for IoT-MEC service placement, including optimization and heuristic approaches.

Maia et al. [25] advanced a genetic algorithm with heuristic initialization for balanced resource allocation, while Lin et al. [26] proposed a mixed-integer non-linear programming approach enhanced by ADMM. Farhadi et al. [27] introduced a two-time-scale framework for service placement that optimizes dedicated resources, enhancing system performance in edge clouds. DRL-based methods have also been explored; Goudarzi et al. [8] achieved a 30% cost reduction, and Liu et al. [28] proposed a PDQN method to minimize service dwell time. However, a critical limitation persists across these studies: heuristics often lack scalability in large-scale instances, while standard DRL agents struggle to converge stably when facing hybrid action spaces that couple discrete placement with continuous resource dimensioning.

To overcome these convergence challenges, we propose a HyDK that advances beyond simple algorithmic combinations. Unlike prior methods that force a single agent to learn coupled decisions, our approach introduces a novel "Action Space Pruning" mechanism. Recognizing that the resource allocation sub-problem is convex for a fixed placement, we utilize KKT conditions as a deterministic optimality guarantee embedded within the environment. This mathematically solves the continuous resource dimension, effectively pruning it from the DRL agent's action space. By decoupling the problem structure, our framework allows the DRL agent to focus exclusively on the discrete topological search for service placement, thereby achieving significantly faster convergence and guaranteed optimality for resource allocation.

TABLE I
SUMMARY OF NOTATIONS AND DEFINITIONS

Notation	Definition
The application layer	
$\mathcal{M}$	The heterogeneous applications set
M	The number of heterogeneous applications
m	The application m
K_m	Total number of CSs in application m
V_m	The set of service components of application m
$v_{m,0}$	The ComS of application m
$v_{m,k}$	The k-th CS of application m
λ_m	Poisson arrival rate of tasks for application m
$\overline{r_m}$	Expected CPU cycles required to execute $v_{m,0}$
$\overline{t_m}$	Average service time for application m
$\overline{t_m^2}$	Second moment of service time for application m
$\overline{W_m}$	Average queuing waiting time for application m
$\overline{T_m}$	Average task sojourn time ($\overline{t_m} + \overline{W_m}$)
ρ_m	Utilization rate of application m ($\lambda_m \overline{t_m}$)
f_m	Computing resources allocated to $v_{m,0}$
$e_{m,k}$	Data volume transmitted from $v_{m,k}$ to $v_{m,0}$
$\omega_{m,k}$	Storage size of the service image for $v_{m,k}$
$d_{m,k}$	Size of cached data required by $v_{m,k}$
$s_{m,k}$	Total storage requirement of $v_{m,k}$ ($\omega_{m,k} + d_{m,k}$)
$T_{m,k}^{trans}$	Transmission latency from $v_{m,k}$ to $v_{m,0}$
T_m^{trans}	Bottleneck data retrieval time ($\max_k\{T_{m,k}^{trans}\}$)
T_m^{total}	Weighted total sojourn time for application m per unit time
The edge layer	
$\mathcal{H}$	The edge servers set
h	The edge server h
H	The number of edge servers
S_h	Maximum storage capacity of edge server h
F_h	Maximum computing capacity of edge server h
R_h	Bandwidth between edge server h and the cloud data center
$R_{h,h'}$	Bandwidth between edge server h and h'
T^{ini}	System-wide service initialization time
$x(m,k,h,t)$	Binary decision for placing component $v_{m,k}$ on server h at t
Q_h	Total number of ComSs deployed on server h
$f_{h,q}$	Computing resources allocated to the q-th ComS on server h
Δt	Unit of time

III. System Model and Problem Formulation

This section delineates the fundamental assumptions, notations, and definitions pertinent to our study, along with a comprehensive overview of the IoT-MEC environment.

A. IoT-MEC Environment

In this section, we introduce the IoT-MEC environment that forms the basis of our study. The mathematical symbols used in this paper are summarized in Table I .

The IoT-MEC environment is illustrated in Fig. 1. We consider a set of H heterogeneous edge servers (ESs), denoted by

TABLE II
DEFAULT PARAMETER SETTINGS

The application layer			
M	9	K_m	[1, 3]
λ_m	[0, 50]	r_m	[0.01, 0.02] $\times$ 10^9 CPU cycles
$\omega_{m,0}$	[20, 35] GB	$\omega_{m,k}$	[15, 25] GB
$d_{m,k}$	[20, 45] GB	$e_{m,k}$	[6, 10] KB
R_h	[125, 150] Mbps	$R_{h,h'}$	[125, 150] Mbps
The edge layer			
H	{4,6}	S_h	[512, 1024] GB
F_h	[2, 3.5] GHz	Δt	1 s

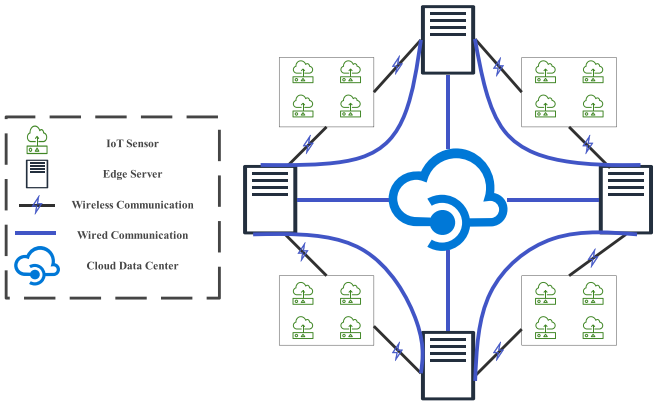

Fig. 1. The IoT-MEC environment encompasses multiple IoT devices, numerous edge servers, and a cloud data center.

$\mathcal{H} = \{0, \ldots, H-1\}$, used to host various applications. When a service is deployed, it is migrated from the cloud data center (CDC) to an ES.

Suppose there are M applications in the system, denoted by the set $\mathcal{M} = \{0, \ldots, M-1\}$. Each application $m \in \mathcal{M}$ is decomposed into a unified set of service components $\mathcal{V}_m = \{v_{m,0}, v_{m,1}, \ldots, v_{m,K_m}\}$, where $v_{m,0}$ represents the ComS and $v_{m,k}$ $(1 \leq k \leq K_m)$ denotes the k-th CS. To represent the placement decisions, we define a binary variable $x(m,k,h,t) \in \{0,1\}$, which equals 1 if component $v_{m,k}$ is deployed on edge server $h \in \mathcal{H}$ at time slot t, and 0 otherwise. This component-based modeling framework allows for a consistent mathematical description of both computing and storage resource requirements across the heterogeneous IoT-MEC environment.

The execution dependencies within the component set $\mathcal{V}_m$ follow a DAG structure. In this model, $v_{m,0}$ requires both computing and storage resources for execution, whereas each $v_{m,k}$ primarily consumes storage resources, with only minimal computing overhead for maintenance. Consequently, our resource allocation policy focuses on assigning computing resources specifically to $v_{m,0}$. Upon triggering a request for $v_{m,0}$, the associated components $v_{m,k}$ immediately transfer the required data to the ES hosting $v_{m,0}$. Since the control signaling is negligible, its transmission latency is omitted.

To analyze the system dynamics, the operational horizon is discretized into T equally spaced time slots, $\mathcal{T} = \{1, \ldots, T\}$. The placement decision is formally expressed as:

$$x(m,k,h,t) = \begin{cases} 1, & \text{if } v_{m,k} \text{ is on ES } h, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In this environment, service redeployment is assumed to be infrequent, occurring only during significant events such as updates to production requirements or substantial shifts in task arrival rates and resource demands.

Within the unit time interval Δt, the arrival of tasks requesting application $m \in \mathcal{M}$ is modeled as a Poisson process. We assume that the arrival rate λ_m remains approximately constant over multiple consecutive time slots, such that the system operates under quasi-stationary conditions and the arrival process can be characterized by a stationary Poisson process with rate λ_m. To accurately reflect the diverse nature of computational tasks, the CPU cycles required for the execution of ComSs in application $m \in \mathcal{M}$ are assumed to follow a general distribution, where $\overline{r_m}$ denotes the expected value of the required cycles. Since a dedicated VM is instantiated for each placed service, the processing of requests within each VM is modeled as an M/G/1 queue. This approach is particularly advantageous as it captures the impact of service time variability on system performance, allowing for a more flexible and realistic estimation of the average sojourn time for heterogeneous applications.

In practice, hosting all services on a single ES is infeasible due to the stringent storage constraints of individual servers $h \in \mathcal{H}$. Additionally, given the finite computational capacity of each ES, an inefficient placement strategy may violate the diverse requirements of heterogeneous tasks. Therefore, it is essential to jointly optimize service placement and resource allocation to ensure that all task demands are satisfied within the resource-constrained IoT-MEC environment.

To ensure the feasibility of service placement, we define the physical resource limitations of the edge nodes. Let S_h and F_h denote the total storage and computational capacities of ES $h \in \mathcal{H}$, respectively. The ComS $v_{m,0}$ of $\mathcal{V}_m$ is allocated with f_m computing resources, and also occupies a storage space of $s_{m,0}$, which represents the size of the service itself. Each CS $v_{m,k}(1 \leq k \leq K_m)$ has a storage requirement $s_{m,k}$. For a CS, the storage requirement is the sum of its image size $\omega_{m,k}$ and cached data size $d_{m,k}$, i.e., $s_{m,k} = \omega_{m,k} + d_{m,k}$. Consequently, the following constraints must be satisfied for each ES h at any time slot t:

$$\sum_{m \in \mathcal{M}} x(m,0,h,t) \cdot f_m \leq F_h, \quad \forall h \in \mathcal{H}, \tag{2}$$

$$\sum_{m \in \mathcal{M}} \sum_{k=0}^{K_m} x(m,k,h,t) \cdot s_{m,k} \leq S_h, \quad \forall h \in \mathcal{H}. \tag{3}$$

In order to better describe the computation of various types of times, we first introduce the data flow in the IoT-MEC environment and then elaborate on the execution of service requests.

As shown in Fig. 2, the complete data flow process is as follows: in order to process the task requests initiated by IoT devices, the related service software and database are first migrated from the CDC to the ES. After collecting the data, the sensor continuously transmits it to the ES where the corresponding CS is deployed. The complete execution of the application is as follows: when the ComS needs a calculation, it first requests the required data from the corresponding CSs; the CSs send the data to the ComS as soon as they receive the request. When ComS is idle (no task is executed) and receives all the data, it can start to execute the task.

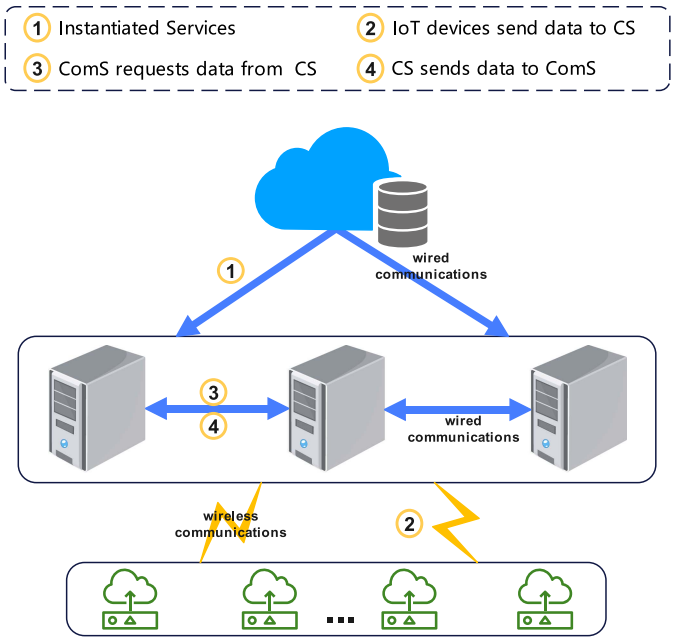

Fig. 2. The complete data flow process in our model.

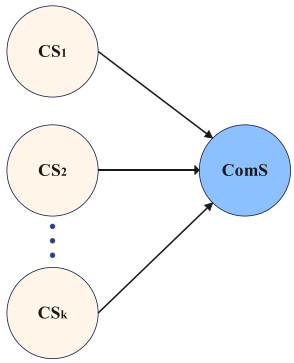

Fig. 3. Data dependencies.

B. Service Initialization Time

Service deployment at the edge requires instantiating VMs and downloading necessary service images from the CDC, incurring an initialization latency. Let R_h denote the transmission bandwidth between h and the CDC. Based on the placement decisions $x(m,k,h,t)$, the initialization time for h is determined by the total volume of data downloaded:

$$T_h^{\text{ini}} = \frac{1}{R_h} \sum_{m \in \mathcal{M}} \left(x(m,0,h,t) \cdot s_{m,0} + \sum_{k=1}^{K_m} x(m,k,h,t) \cdot \omega_{m,k} \right). \tag{4}$$

Since ESs can perform downloads in parallel, the system-wide service initialization time is defined by the bottleneck server:

$$T^{\text{ini}} = \max_{h \in \mathcal{H}} \left(T_h^{\text{ini}} \right). \tag{5}$$

C. Task Sojourn Time

Heterogeneous applications in an IoT-MEC environment consist of a varying number of services, and the execution of the ComS in each application relies on the data requested from all the relevant CSs. The dependencies between the CSs and the ComS are shown in Fig. 3. To describe our model more clearly, we put ourselves in the shoes of the application and analyze the sojourn time of tasks requesting services from different applications. Due to space limitations, the justification for adopting the DAG structure over other data structures is provided in Section II of the supplementary material, available online.

For the average sojourn time, as mentioned earlier, we assume the requests for heterogeneous $\mathcal{V}_m, m \in \mathcal{M}$ follow a Poisson distribution with rate λ_m, while the required CPU cycles follow a general distribution with expectation $\overline{r_m}$. Consequently, the request processing is modeled as an M/G/1 queue. This modeling choice captures the performance impact of service time variability and provides a robust framework for heterogeneous tasks, specifically by accounting for the variance in task execution which is critical for accurate latency estimation in cloud environments. Accordingly, the average sojourn time for $\mathcal{V}_m$ can be expressed as:

$$\overline{T_m} = \overline{t_m} + \overline{W_m}, \tag{6}$$

where $\overline{W_m}$ is the average waiting time and $\overline{t_m}$ is the average service time, the average service time can be expressed as:

$$\overline{t_m} = \frac{\overline{r_m}}{f_m} + T_m^{trans}, \tag{7}$$

where f_m (CPU cycles/s) denotes the computing resources allocated to the task by the ES, $\frac{\overline{r_m}}{f_m}$ denotes the average execution time of the ComS of the application $\mathcal{V}_m$ on the ES, and T_m^{trans} denotes the total transmission time for data from all CSs of application $\mathcal{V}_m$ to the ComS, and is represented by:

$$T_{m,k}^{\text{trans}} = \frac{e_{m,k}}{R_{h,h'}} \cdot JF(h,h'), \tag{8}$$

where $e_{m,k}$ denotes the size of the response data transmitted to ComS $v_{m,0}$ by the CS $v_{m,k}$ of application m. $R_{h,h'}$ is the bandwidth between ES h' (where $v_{m,k}$ is placed) and ES h (where $v_{m,0}$ is placed). $JF(h,h')$ denotes a judgment function where $JF(h,h') = 0$ if h and h' denote the same ES, and $JF(h,h') = 1$ otherwise. Furthermore, since each ComS involves multiple CSs transmitting data to it, the overall transmission time for application m can be expressed as:

$$T_m^{\text{trans}} = \max_{k \in \{1,\ldots,K_m\}} (T_{m,k}^{\text{trans}}). \tag{9}$$

The term $\overline{W_m}$ in (6) represents the queuing delay caused by resource contention. It is calculated as:

$$\overline{W_m} = \frac{\lambda_m \overline{t_m^2}}{2(1-\rho_m)}, \tag{10}$$

where λ_m denotes the average task arrival rate, $\overline{t_m^2}$ is the second moment of the service time, and $\rho_m = \lambda_m \overline{t_m}$ represents the utilization of the application m. The $\overline{t_m^2}$ can be expressed as:

$$\overline{t_m^2} = \frac{\overline{r_m^2}}{f_m^2} + {T_m^{trans}}^2 + 2\left(\frac{\overline{r_m} T_m^{trans}}{f_m}\right), \tag{11}$$

where $\overline{r_m^2}$ denotes the second moment of the computational requirement $\overline{r_m}$, which is modeled as $\overline{r_m^2} = 1.7\overline{r_m}^2$ [29].

D. Problem Formulation

The number of access requests to the m per unit of time Δt follows a Poisson distribution of λ_m, which implies that the average number of accesses to the m per unit of time is λ_m, and therefore, the average total sojourn time of the tasks requesting m per unit time can be expressed as:

$$T_m^{total} = \lambda_m \overline{T_m}. \tag{12}$$

Then for any task, the average sojourn time can be expressed as:

$$\overline{T} = \frac{\sum_{m=0}^{(M-1)} \lambda_m \overline{T_m}}{\sum_{m=0}^{(M-1)} \lambda_m}. \tag{13}$$

In this paper, our goal is to minimize the average sojourn time of a task. To achieve this goal, we adopt a comprehensive optimization approach to solve the joint service placement and resource allocation optimization problem using DRL and KKT. Mathematically, the problem of minimizing the average sojourn time of a task and its constraints can be formulated as follows:

$$P: \min_{p^*, f^*} \quad \overline{T} \tag{14a}$$

$$\text{s.t.} \begin{cases} \sum_{m \in \mathcal{M}} x(m,0,h,t) \cdot f_m \le F_h, \quad \forall h \in \mathcal{H} & \text{(a)} \\ \sum_{m \in \mathcal{M}} \sum_{k=0}^{K_m} x(m,k,h,t) \cdot s_{m,k} \le S_h, \quad \forall h \in \mathcal{H} & \text{(b)} \\ \overline{T}_m \le \frac{\Delta t}{\lambda_m}, & \text{(c)} \\ \sum_{h \in \mathcal{H}} x(m,k,h,t) = 1, & \text{(d)} \\ x(m,k,h,t) \in \{0,1\}, & \text{(e)} \end{cases} \tag{14b}$$

where (14a) denotes our optimization objective, $p*$ denotes the optimal placement decision, and f^* denotes the optimal resource allocation policy. (14b) denote the constraints received by this optimization problem, where (a) (b) denote that the placement decision must satisfy the resource constraints of the ESs; (c) indicates the time limit $\frac{\Delta t}{\lambda_m}$ as a soft constraint to ensure fairness, preventing any application from being compromised for the sake of the global average; and (d) and (e) ensure that each service component is deployed on exactly one ES.

IV. Proposed Approach

In IoT-MEC environments, service placement and resource allocation are interdependent processes that necessitate a coordinated optimization strategy [28]. To address the challenge of simultaneously tackling these coupled problems, we propose **HyDK**, a joint solution integrating the Proximal Policy Optimization (PPO) algorithm [30] with the KKT method. This approach optimizes global strategies to minimize the average application sojourn time and significantly reduce service initialization time.

In order to better elucidate the reward function of the PPO algorithm in the subsequent discussion, we first present the resource allocation algorithm we designed.

A. Resource Allocation Strategy

To facilitate the transition from application-specific requirements to server-local resource management, we link each application $m \in \mathcal{M}$ to its physical deployment on an edge server $h \in \mathcal{H}$. Specifically, if the ComS of application m is deployed as the q-th service on server h, we map its parameters to server-local variables such that $\overline{r_{h,q}} = \overline{r_m}$, $\lambda_{h,q} = \lambda_m$, $f_{h,q} = f_m$, $\rho_{h,q} = \rho_m$, $\overline{W_{h,q}} = \overline{W_m}$, and $\overline{t_{h,q}} = \overline{t_m}$. Here, $q \in \{0, \ldots, Q_h - 1\}$ represents the ComS index within server h.

In our designed IoT-MEC environment, we assume that the CSs require negligible computing resources, which we disregard. Therefore, computing resources are allocated only to ComSs. Before finding the optimal resource allocation, we need to introduce the notion that queue stability presupposes a utilization rate $\rho < 1$. For application m, the condition for the flow of tasks requesting application m to reach stability can be expressed as:

$$\rho_m = \lambda_m \left(T_m^{trans} + \frac{\overline{r_m}}{f_m} \right) < 1. \tag{15}$$

From (15), the lower bound of the computing resources f_m required to ensure queue stability is derived as:

$$f_m > \frac{\overline{r_m} \lambda_m}{1 - \lambda_m T_m^{trans}} = f_m^{min}. \tag{16}$$

Specifically, f_m must exceed this minimum threshold to prevent queue overflow. Generalizing to the edge layer, the following condition must be satisfied to ensure the stability of all services deployed on edge server h:

$$\sum_{q=0}^{Q_h - 1} f_{h,q} \le F_h^{max}, \tag{17}$$

where Q_h is the total number of ComSs hosted on server h. This implicitly requires that a valid placement strategy must ensure the server's total capacity can satisfy the minimum resource requirements of all hosted services. All subsequent analyses are based on the premise that $\rho < 1$ is rigorously satisfied.

In each ES h, our goal is to determine the resource allocation policy $\mathbf{f}_h = \{f_{h,0}, \ldots, f_{h,(Q_h-1)}\}$, aiming to minimize the average sojourn time under the constraint (17). This constitutes a convex optimization problem solvable via the Lagrange multiplier method.

Assuming that we have obtained a service placement strategy based on the current situation, we instantiate a VM for each ComS and allocate the available resources to that VM to exclusively handle the ComSs placed on it. The resource allocation is performed to minimize the average sojourn time of the ComSs on that server, and the objective function can be expressed as follows:

$$\min \alpha = \frac{\sum_{q=0}^{(Q_h-1)} \lambda_{h,q} \overline{T_{h,q}}}{\sum_{q=0}^{(Q_h-1)} \lambda_{h,q}}, \tag{18}$$

where all variables except $f_{h,q}$ are constants once the service placement strategy is determined. Since the total task arrival rate $\sum_{q=0}^{(Q_h-1)} \lambda_{h,q}$ acts as a constant scaling factor, minimizing α is mathematically equivalent to minimizing the numerator alone. This transformation converts the fractional objective into a standard additive form, which simplifies the optimality analysis and facilitates the efficient computation of the optimal solution using KKT conditions:

$$\min \alpha' = \sum_{q=0}^{Q_h-1} \lambda_{h,q} \overline{T_{h,q}} = \sum_{q=0}^{Q_h-1} \lambda_{h,q} (\overline{t_{h,q}} + \overline{W_{h,q}}), \tag{19}$$

We first analyze the convexity of the objective function. For the objective function, its first-order partial derivative for any $f_{h,q}$ can be expressed as:

$$\frac{\partial \alpha'}{\partial f_{h,q}} = \lambda_{h,q}\left(\frac{\partial \overline{t_{h,q}}}{\partial f_{h,q}} + \frac{\partial \overline{W_{h,q}}}{\partial f_{h,q}}\right). \tag{20}$$

Specifically, the partial derivatives of these two terms can be calculated as follows:

$$\frac{\partial \overline{t_{h,q}}}{\partial f_{h,q}} = -\frac{\overline{r_{h,q}}}{f_{h,q}^2}, \tag{21}$$

$$\frac{\partial \overline{W_{h,q}}}{\partial f_{h,q}} = \frac{(1-\rho_{h,q})\lambda_{h,q}\frac{\partial \overline{t_{h,q}^2}}{\partial f_{h,q}} + \lambda_{h,q}\overline{t_{h,q}^2}\frac{\partial \rho_{h,q}}{\partial f_{h,q}}}{2(1-\rho_{h,q})^2}. \tag{22}$$

Finally, the $\frac{\partial \overline{t_{h,q}^2}}{\partial f_{h,q}}$ and $\frac{\partial \overline{\rho_{h,q}}}{\partial f_{h,q}}$ are derived as:

$$\frac{\partial \overline{t_{h,q}^2}}{\partial f_{h,q}} = -2\left(\frac{\overline{r_{h,q}^2}}{f_{h,q}^3} + \frac{\overline{r_{h,q}}T_{h,q}^{trans}}{f_{h,q}^2}\right), \tag{23a}$$

$$\frac{\partial \overline{\rho_{h,q}}}{\partial f_{h,q}} = -\frac{\lambda_{h,q}\overline{r_{h,q}}}{f_{h,q}^2}. \tag{23b}$$

Due to the complexity of the first-order derivative, analytically deriving the second-order derivatives is non-trivial. Hence, we validate the convexity of the objective function by plotting the trends of the first-order derivative. Due to space limitations, the convexity analysis of the objective function is provided in Section III of the supplementary material, available online. The results demonstrate that the objective function is convex, allowing us to utilize the KKT method to determine the optimal resource allocation for the ES. Specifically, we construct the Lagrangian function as follows:

$$\mathcal{L}(\mathbf{f}_h, \beta) = \sum_{q=0}^{Q_h-1}(\lambda_{h,q}\overline{T_{h,q}}) + \beta\left(\sum_{q=0}^{Q_h-1} f_{h,q} - F_h^{max}\right), \tag{24}$$

where β is the Lagrange multiplier. According to KKT conditions [31]:

$$\begin{cases} \frac{\partial \alpha'}{\partial f_{h,q}} + \beta = 0, \\ \sum_{q=0}^{(Q_h-1)} f_{h,q} - F_h^{max} \le 0, \\ \beta \ge 0, \\ \beta\left(\sum_{q=0}^{(Q_h-1)} f_{h,q} - F_h^{max}\right) = 0. \end{cases} \tag{25}$$

Since deriving a closed-form solution is intractable, leveraging the convexity of the objective function, we employ the Bisection Method to numerically identify the optimal Lagrange multiplier, thereby determining the optimal resource allocation strategy. The detailed procedure is summarized in Algorithm 1.

Algorithm 1: KKT-Based Resource Allocation.

Input: Service set $\mathcal{S}_h = \{0, \ldots, Q_h - 1\}$, total capacity F_h^{max}, tolerance δ.
Output: Optimal allocation $\mathbf{f}_h^* = \{f_{h,1}, \ldots, f_{h,(Q_h-1)}\}$.
Initialize $\beta_{\min} \leftarrow 0$, $\beta_{\max} \leftarrow \beta_{init}$.
while $\beta_{\max} - \beta_{\min} > \delta$ **do**
  $\beta_{mid} \leftarrow (\beta_{\min} + \beta_{\max})/2$.
  Resource Calculation:
    For ComS of application m, solve $f_{h,q}$ satisfying $\frac{\partial \alpha'}{\partial f_{h,q}} + \beta_{mid} = 0$.
    Calculate total demand $F_{sum} \leftarrow \sum_{q=0}^{Q_h-1} f_{h,q}$.
  Update Multiplier:
  if $F_{sum} > F_h^{max}$ **then**
    $\beta_{\min} \leftarrow \beta_{mid}$
  else
    $\beta_{\max} \leftarrow \beta_{mid}$
  end if
end while
return $\mathbf{f}_h^*$ derived from β_{mid}.

B. Service Placement Strategy

After conducting the aforementioned analysis, we conclude that, given the service placement strategies and the circumstances of service arrivals, it is possible to determine the optimal resource allocation for each ES. Subsequently, to address the service placement challenge, we propose a DRL-based algorithm for service placement.

To design a service placement algorithm leveraging DRL, the problem must first be modeled as a Markov Decision Process (MDP). In computer science, a standard MDP is represented as a tuple $< S, A, P, R, \gamma >$, comprising five key elements: S for the set of states, A for the set of actions, P as the transition probability function, R for the reward function, and γ as the discount factor. This tuple succinctly encapsulates the core components defining the MDP's dynamics and characteristics, facilitating a formal and comprehensive framework for computational modeling and analysis.

In dynamic environments, intelligent agents must first observe the current state before deciding on the optimal action. Following the acceptance of the chosen action by the environment, a reward is issued, marking the end of an interaction cycle. This process repeats, with the agent aiming to optimize its behavior by maximizing the cumulative rewards from ongoing interactions with the environment. The ultimate objective is to learn and execute actions leading to the most favorable outcomes, aligning with the goal of achieving the highest cumulative rewards.

Hence, it is imperative to model the service placement optimization problem as a MDP. Given the IoT-MEC environment addressed in this study is model-free, it obviates the need for transition probabilities. Thus, the MDP is conceptualized in three components: the state space S, the action space A, and the reward function R. To tailor this model for edge computing scenarios, the components of reinforcement learning are specifically designed as follows.

- *State space:* The current state s_t encapsulates the most recent data obtained from the IoT-MEC framework, and the DRL system generates decisions on service placement relying on this data, consequently securing rewards from the environment. Within our framework, each state is viewed as comprising two distinct segments, namely the information contained in the CSs and the information contained in the ComSs, which can be expressed as:

$$s_t = (s_t^{\text{CSs}}, s_t^{\text{ComSs}}). \tag{26}$$

Specifically, these state features encompass the storage resource demands of both the CSs and ComSs, the computational requirements exclusively for the ComSs, and the real-time task arrival rates within the current time slot.

- *Action space:* In each time slot t, the edge node needs to make a service placement strategy a_t according to the current state.

$$a_t = (a_t^{\text{CSs}}, a_t^{\text{ComSs}}), \tag{27}$$

where a_t^{CSs} represents a placement action on CSs, a_t^{ComSs} represents a placement action on ComSs, and $a_t^{\text{ComSs}}, a_t^{\text{CSs}} \in \mathcal{H}$.

- *Reward function:* After taking a_t action in state s_t, the edge node will receive a reward r_t from the edge computing system. In our model, our main goal is to minimise the average sojourn time of the service, while we want to make the service initialization time as short as possible, so we set the reward function as:

$$R_t = \begin{cases} -(\overline{\mathrm{T}} + \nu \mathrm{T}^{\text{ini}}), & \text{(a)} \\ \Psi, & \text{(b)} \\ -\eta(\overline{\mathrm{T}} + \nu \mathrm{T}^{\text{ini}}). & \text{(c)} \end{cases} \tag{28}$$

The three cases are defined to handle successful placement, resource violations, and Quality of Service constraints, respectively. In case (a), when a service is successfully placed within the system's resource limits, the reward is calculated as the negative sum of the average sojourn time $\overline{\mathrm{T}}$ and the weighted initialization time T^{ini}. Since T^{ini} is typically significantly larger than the average sojourn time of an arbitrary application, we introduce a scaling factor $\nu < 1$ to attenuate the influence of the initialization phase, preventing the model from over-optimizing for startup speed at the expense of long-term service efficiency. In case (b), if the placement policy fails to satisfy the system's hard resource constraints, the agent receives a large penalty term Ψ, which serves to discourage the selection of infeasible actions and accelerates the convergence toward valid placement strategies. Finally, case (c) integrates a soft QoS constraint where a penalty factor η is applied to amplify the negative reward if the execution time surpasses the maximum threshold $\frac{\Delta t}{\lambda_m}$. This mechanism effectively guides the algorithm away from solutions that yield unacceptable delays for individual tasks, ensuring the system meets strict latency requirements. In practice, parameters ν, Ψ, and η are empirically tuned prior to training to ensure penalties dominate upon constraint violations, so that the learned policies strictly satisfy the boundary constraints.

C. Detailed Analysis of PPO Training Process

The PPO algorithm is a widely used reinforcement learning policy optimization method proposed by OpenAI [30]. It improves the stability of the training process by limiting the magnitude of policy updates, preventing performance fluctuations caused by each update being too large. Compared with the Trust Region Policy Optimization (TRPO) algorithm [32], which requires complex second-order derivative computation and constrained optimization, PPO is simpler in terms of implementation and debugging.

To enhance the understanding of PPO, this section delves into the mathematical foundation and step-by-step training process of PPO. We detail the surrogate objective function, gradient ascent approach, and the update mechanisms for both the actor and critic networks. The training process involves iteratively optimizing the networks over multiple epochs using the collected trajectory data, incorporating an entropy regularization term to maintain policy diversity and encourage exploration. This comprehensive breakdown serves as a guide for implementing PPO in complex reinforcement learning tasks.

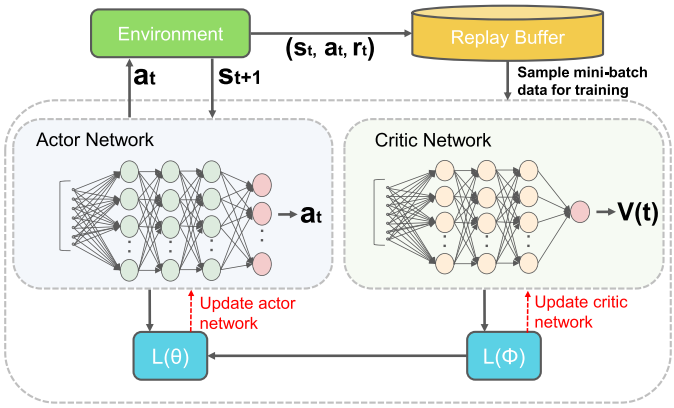

Fig. 4. The overall PPO algorithmic flow.

The overall algorithmic workflow of PPO is shown in Fig. 4. The PPO algorithm employs a stochastic gradient ascent approach to optimize the surrogate objective function. To prevent premature convergence and ensure sufficient exploration, an entropy bonus is integrated into the actor's objective function:

$$L^{actor}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) + \iota S[\pi_\theta](s_t)], \tag{29}$$

where $S[\pi_\theta](s_t)$ denotes the entropy of the policy distribution at state s_t, and ι represents the entropy coefficient. At time step t, $\hat{A}_t$ denotes the approximation of the advantage function, ϵ signifies the clipping parameter, and $r_t(\theta)$ denotes the probability ratio between the updated policy and the previous policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{30}$$

$\hat{A}_t$ can be expressed as

$$\hat{A}_t = \delta_t + (\gamma\lambda_{GAE})\delta_{t+1} + \cdots + (\gamma\lambda_{GAE})^{T'-t+1}\delta_{T'-t+1}, \tag{31}$$

where t specifies the time index in $[0, T']$, T' denotes the length of the collected trajectory, and δ_t is the temporal difference (TD) error.

The state-value function $V(s_t)$ is estimated by the critic network. The critic network is updated by minimizing the mean squared error between the predicted value and the target value computed from the collected rewards:

$$L^{VF}(\phi) = \hat{\mathbb{E}}_t\left[\left(V_\phi(s_t) - \hat{V}_t^{target}\right)^2\right], \tag{32}$$

where $\hat{V}_t^{target}$ denotes the target value, calculated as $\hat{V}_t^{target} = \hat{A}_t + V_\phi(s_t)$, and is treated as a constant during the optimization of the critic network.

The actor network parameters are updated via stochastic gradient ascent, while the critic network parameters are optimized

Algorithm 2: PPO Policy Update Process.

Input: Trajectory Buffer $\mathcal{D}$ (containing $\{s_t, a_t, r_t\}$), Number of epochs K, and entropy coefficient ι.
Output: Updated network weights θ and ϕ.
Data Preparation Phase:
  Extract state, action, and reward sequences from $\mathcal{D}$.
  Compute advantage estimates $\hat{A}_t$ and TD targets $\hat{V}_t^{target}$ using data in $\mathcal{D}$.
Optimization Phase:
for epoch $k = 1, \ldots, K$ **do**
  Train actor network:
    Calculate objective $L^{actor}(\theta)$ using batch data from $\mathcal{D}$ per (29).
    Update actor network parameters θ using (33).
  Train critic network:
    Calculate value loss $L^{VF}(\phi)$ using batch data from $\mathcal{D}$ per (32).
    Update critic network parameters ϕ using (34).
end for
Update old policy parameters: $\theta_{\text{old}} \leftarrow \theta$.
Clear Trajectory Buffer $\mathcal{D}$.

using stochastic gradient descent:

$$\theta \leftarrow \theta + lr_a \nabla_\theta L^{actor}(\theta), \tag{33}$$

$$\phi \leftarrow \phi - lr_c \nabla_\phi L^{VF}(\phi). \tag{34}$$

The learning rates for the actor and critic networks are denoted by lr_a and lr_c, respectively.

D. The HyDK Algorithm

Algorithm 3 presents the overall execution framework of the proposed HyDK method. To handle the hybrid action space efficiently, the decision process in each time slot t is hierarchically decoupled: the discrete service placement a_t is first determined by the actor network, which immediately triggers the calculation of the optimal resource allocation $\mathbf{f}_h^*$ via the KKT-based solver (Algorithm 1). This joint action $(a_t, \mathbf{f}_h^*)$ is then executed in the environment. Following an episodic on-policy setting, the algorithm accumulates transition data into a trajectory buffer $\mathcal{B}$ throughout the episode and performs a unified policy update using Algorithm 2 only after the episode terminates.

Algorithm 3: The HyDK Algorithm: Joint Service Placement and Resource Allocation.

Input: The observed state s_t, and fixed time horizon $T = 25$.
Output: Service placement a_t and optimal computing resource allocation $\mathbf{f}_h^*$.
Initialize policy network π_θ and value network V_ϕ.
for episode $ep = 1, \ldots, EP$ **do**
  Initialize an empty Trajectory Buffer $\mathcal{D}$.
  Reset environment and observe initial state s_1.
  for time slot $t = 1, \ldots, T$ **do**
    Step 1: Action Sampling
      Input s_t into π_θ and sample service placement action $a_t \sim \pi_\theta(\cdot|s_t)$.
    Step 2: Hybrid Resource Allocation
      Given a_t, compute optimal resource allocation $\mathbf{f}_h^*$ via Algorithm 1.
    Step 3: Execution
      Execute $(a_t, \mathbf{f}_h^*)$, receive reward r_t and next state s_{t+1}.
    Step 4: Storage
      Store transition (s_t, a_t, r_t) into Buffer $\mathcal{D}$.
      Update state $s_t \leftarrow s_{t+1}$.
  end for
  Step 5: Policy Update
    Update networks π_θ and V_ϕ using trajectory buffer $\mathcal{D}$ via Algorithm 2.
end for

V. Performance Evaluation

In this section, we present the baseline algorithms and the default parameter configurations used in our experiments. We also analyze and demonstrate the performance of our proposed algorithm from various perspectives.

A. Simulation Setup

Following methodologies similar to [33], [34], and [35], we utilize the Edge-IIoTset dataset to train our decision model. The Edge-IIoTset is a comprehensive, real-world cybersecurity dataset tailored for IoT and Industrial IoT (IIoT) applications, designed for both centralized and federated learning models. Generated from a purpose-built IoT/IIoT testbed, it encompasses a variety of devices, sensors, protocols, and cloud/edge configurations. The dataset includes data from more than 10 IoT devices, such as low-cost digital sensors for temperature and humidity sensing, ultrasonic sensors, water level detection sensors, pH sensors, soil moisture sensors, heart rate sensors, and flame sensors.

Specifically, we extracted the sensor data generation frequencies and network traffic attributes from the Edge-IIoTset to represent the workload profiles in our scenario. Based on this historical information, we analyzed the realistic task arrival rates (λ_m) of nine different sensors and utilized these arrival patterns to simulate task requests in our IoT-MEC environment. The main experimental parameters used in our IoT-MEC environment are summarized in Table II.

We compare the performance achieved by the proposed method with the following baseline algorithms.

- *GA:* Genetic Algorithm (GA) utilizes selection, crossover, and mutation mechanisms to iteratively evolve the service placement strategy. This discrete strategy is then combined with the KKT-based algorithm for optimal resource allocation to derive the final joint decision.
- *PSO:* Particle Swarm Optimization (PSO) solves the placement sub-problem by updating particle positions based on personal and global historical bests. The resulting placement policy is integrated with the KKT-based algorithm to generate the joint service placement and resource allocation strategy.
- *SA:* Simulated Annealing (SA) employs a probabilistic cooling schedule to escape local optima while searching for the global service placement strategy. Coupled with the KKT-based algorithm for resource allocation, it achieves the optimal joint decision.

TABLE III
PARAMETER SETTINGS FOR GA, PSO, SA AND DRL-BASED ALGORITHMS

Algorithm	Value
Genetic Algorithm	
Population Size	50
Max Generations	100
Number of Parents for Mating	10
Particle Swarm Optimization	
Number of Particles	50
Cognitive Factor	1
Social Factor	2
Inertia Weight	0.729
Max Iterations	100
Simulated Annealing	
Initial Temperature	1
Max Iterations	100
DRL-based	
Learning Rate	3×10^{-5}(PPO) or 3×10^{-4}(A2C)
ϵ	0.15 (PPO-based)
Discount Factor	0.1
Entropy Coefficient	0.01(PPO-based)

- *A2C:* Advantage Actor-Critic (A2C) is a DRL-based method that learns the placement policy via an actor-critic architecture to maximize rewards. The corresponding computing resource allocation is determined using a KKT-based algorithm.
- *PPO-PRA:* This method combines the PPO algorithm with the Proportional Resource Allocation (PRA) resource allocation strategy. PPO is employed to make decisions on service placement, while computing resources are allocated to services proportionally based on their demand levels.

Furthermore, to validate the effectiveness of the KKT-based resource allocation strategy, we also compared it with three classic resource allocation algorithms:

- *PRA:* The PRA method allocates computing resources to each service proportionally according to its task arrival rate and workload.
- *EEA:* The Equal Excess Allocation (EEA) algorithm first satisfies the minimum resource demand of each service for stability and then distributes the remaining capacity equally.
- *IGA:* The Iterative Greedy Allocation (IGA) heuristic incrementally assigns resources to the service that provides the maximum reduction in total system latency at each step.

The algorithm parameters are set as shown in Table III. Next, we analyze the convergence behavior of the proposed algorithms under various hyperparameter settings and evaluate their performance in service placement using metrics such as average sojourn time and service initialization time. All simulations were implemented using the PyTorch deep learning framework and executed on a high-performance server equipped with an NVIDIA GeForce RTX 4090 GPU. Due to space limitations, the hyperparameter tuning process (including the rationale for A2C's different learning rate) and additional confidence interval analyses are provided in Section IV of the supplementary material, available online.

B. Average Sojourn Time

As previously discussed, our primary objective is to minimize the average sojourn time. In this section, we compare the performance of our proposed algorithm with that of the benchmark algorithm in terms of average sojourn time across different environmental settings.

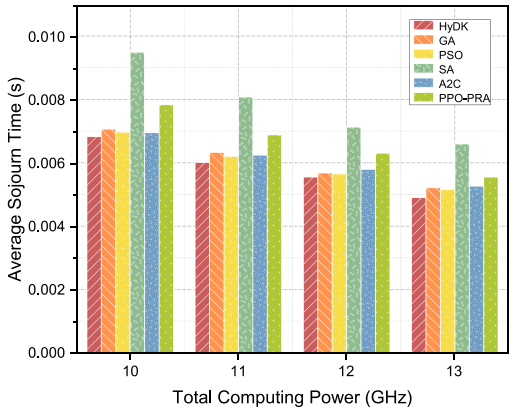

Fig. 5. Performance under different computing resources.

To comprehensively evaluate the effectiveness of the proposed algorithm in addressing the service placement problem, we conducted experiments across eight distinct environments. Specifically, these environments were configured by systematically varying the total computing power (ranging from 10 GHz to 13 GHz) and the total storage resources (ranging from 2304 GB to 3072 GB) to simulate diverse edge capacities. To deeply investigate the algorithm's sensitivity to environmental resource variations, we applied three identical, standardized service variation patterns across all eight environments, rather than generating random workloads for each scenario.

To ensure a fair comparison and reproducibility, we implemented a rigorous seed synchronization protocol for the four independent trials. Specifically, for the **DRL-based methods (PPO, A2C)**, we trained four separate models for each environment, where the random seeds for neural network initialization were explicitly set to $\{0, 1, 2, 3\}$. Correspondingly, for the **heuristic baselines (GA, PSO, SA)**, we executed four independent optimization runs, with the random seeds for initializing the candidate solutions (or populations) also set to $\{0, 1, 2, 3\}$. This configuration guarantees that all algorithms operate under strictly controlled stochastic conditions, ensuring that performance differences are attributable solely to the algorithmic design.

Fig. 5 illustrates the average sojourn time of different algorithms with respect to varying total computing power, ranging from 10 GHz to 13 GHz. As anticipated, a general downward trend is observed across all schemes, indicating that increased computing resources effectively reduce the system's average sojourn time. Among the evaluated algorithms, our proposed **HyDK** consistently achieves the lowest sojourn time across all resource settings, demonstrating its superiority in resource scheduling efficiency. Specifically, compared to the baseline algorithms (GA, PSO, SA, A2C and PPO-PRA), **HyDK** reduces the average sojourn time by approximately 4.16%, 2.92%, 25.60%, 3.96% and 12.25%, respectively. This substantial margin underscores the robustness of our **hybrid framework** in optimizing system performance.

Fig. 6 examines the impact of varying storage resources on the average sojourn time, with capacity decreasing from 3072 GB

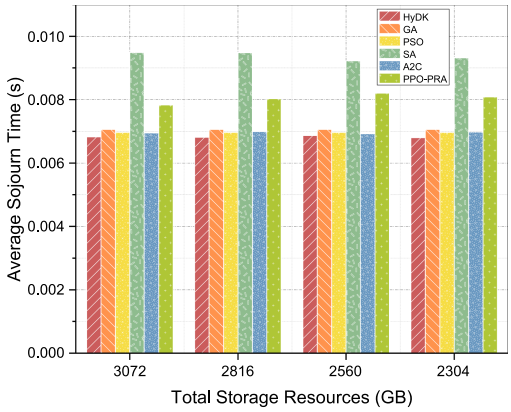

Fig. 6. Performance under different storage resources.

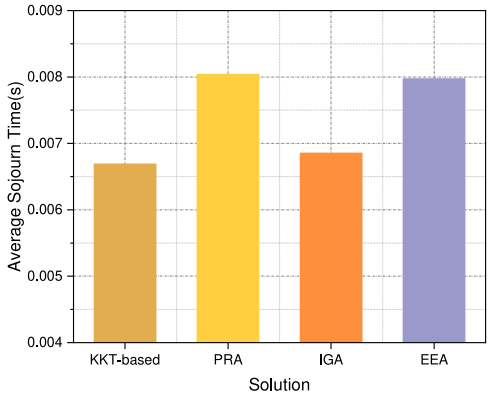

Fig. 8. Performance comparison of average sojourn time between the proposed KKT-based solution and the benchmark algorithms (PRA, EEA, and IGA).

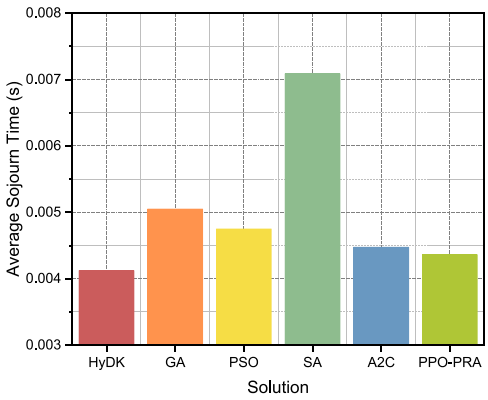

Fig. 7. Performance after adding ESs.

to 2304 GB. It is observed that the performance of all evaluated algorithms remains relatively stable, indicating that the system's average sojourn time is insensitive to storage variations within this specific range. Despite this general stability across all schemes, **HyDK** consistently achieves the lowest sojourn time compared to the baseline algorithms (GA, PSO, SA, A2C and PPO-PRA). This demonstrates that while storage constraints have a limited impact on the overall trend, the **HyDK** framework maintains a distinct performance advantage and ensures optimal service efficiency under all tested conditions.

To investigate the scalability and robustness of our proposed **HyDK** framework within a larger action space, we expanded the network scale to a total of 6 ESs. In this scenario, the complexity of the service placement problem scales exponentially with the increase in available ESs, posing a significant challenge for efficient decision-making. The corresponding average sojourn time is presented in Fig. 7. Despite the exponentially expanding action space, **HyDK** maintains superior performance. Specifically, compared to the baseline algorithms (GA, PSO, SA, A2C and PPO-PRA), **HyDK** achieves average improvements of 18.42%, 13.26%, 41.81%, 7.83%, and 5.5%, respectively. Notably, as the dimensionality of the action space increases, the performance gap between DRL-based approach and traditional heuristic algorithms becomes increasingly pronounced, highlighting the advantage of **HyDK** in handling high-dimensional optimization problems.

The experimental results across various resource settings, including increased computing resources, expanded storage capacity, and a larger number of ESs, consistently demonstrate the superior performance of **HyDK** in minimizing average sojourn time.

Table IV details the mean sojourn time and the corresponding confidence intervals across different resource conditions. It is worth noting that in most experimental settings, the confidence intervals of **HyDK** are narrower compared to the baseline algorithms. This indicates that **HyDK** demonstrates superior stability and consistency compared to the baseline algorithms.Additionally, it is observed that PSO and GA exhibit identical performance and confidence intervals in certain environments. This phenomenon is attributed to the fact that identical initialization random seeds were used for both algorithms, and the service variation patterns were fixed during the experiments. Such an observation does not indicate insufficient exploration, but rather reflects the controlled experimental setting.

To further validate the effectiveness of the proposed resource allocation algorithm, we conducted a comparative analysis among the KKT-based solver of **HyDK**, PRA, EEA, and IGA, while maintaining an identical service placement strategy to ensure a fair comparison. As illustrated in Fig. 8, our KKT-based solution consistently outperforms all baseline methods in terms of average sojourn time. Quantitatively, this approach achieves sojourn time reductions of approximately **16.82%** and **16.11%** compared to PRA and EEA, respectively. Furthermore, even when compared to the near-optimal IGA, our solution still exhibits a **2.4%** reduction in average sojourn time. These results confirm the superior efficacy of the KKT-based optimization module within **HyDK** in achieving efficient resource allocation.

C. *Service Initialisation Time*

Our overall goal is to minimize both the average sojourn time of tasks and the service initialization time incurred by service placement. Therefore, we compare the service initialization time of each approach. Fig. 9 and Table V illustrate the performance in terms of service initialization time. It is evident that **HyDK** consistently achieves the shortest average initialization time compared to the baseline algorithms. Furthermore, in most experimental environments, **HyDK** exhibits narrower confidence

TABLE IV
COMPARISON OF CONFIDENCE INTERVALS FOR AVERAGE SOJOURN TIME UNDER DIFFERENT ENVIRONMENTS

Condition	Value	OURS	GA	PSO	SA	A2C	PPO-PRA
		(Mean ± Confidence Radius) $\times 10^{-3}$					
Total Computing Resources (GHz)	10	6.822 ± 0.411	7.059 ± 0.465	6.967 ± 0.449	9.490 ± 0.622	6.948 ± 0.457	7.834 ± 0.510
	11	6.012 ± 0.350	6.325 ± 0.388	6.202 ± 0.367	8.076 ± 0.518	6.239 ± 0.388	6.880 ± 0.428
	12	5.548 ± 0.327	5.683 ± 0.331	5.648 ± 0.325	7.126 ± 0.485	5.790 ± 0.349	6.300 ± 0.373
	13	4.904 ± 0.282	5.216 ± 0.296	5.158 ± 0.282	6.595 ± 0.434	5.261 ± 0.284	5.550 ± 0.317
Total Storage Resources (GB)	3072	6.822 ± 0.411	7.059 ± 0.465	6.967 ± 0.449	9.490 ± 0.622	6.948 ± 0.457	7.834 ± 0.510
	2816	6.815 ± 0.414	7.059 ± 0.465	6.967 ± 0.449	9.484 ± 0.645	6.992 ± 0.443	8.089 ± 0.499
	2560	6.867 ± 0.425	7.059 ± 0.465	6.967 ± 0.449	9.224 ± 0.737	6.927 ± 0.437	8.204 ± 0.505
	2304	6.799 ± 0.412	7.059 ± 0.465	6.967 ± 0.449	9.321 ± 0.659	6.984 ± 0.474	8.034 ± 0.500

TABLE V
COMPARISON OF CONFIDENCE INTERVALS FOR SERVICE INITIALISATION TIME UNDER DIFFERENT ENVIRONMENTS

Condition	Value	OURS	GA	PSO	SA	A2C	PPO-PRA
		(Mean ± Confidence Radius)$\times 10^{3}$					
Total Computing Resources (GHz)	10	6.688 ± 0.239	6.918 ± 0.285	7.031 ± 0.297	9.345 ± 0.445	6.535 ± 0.254	6.727 ± 0.245
	11	6.616 ± 0.248	6.769 ± 0.287	7.035 ± 0.309	9.101 ± 0.419	7.081 ± 0.282	6.745 ± 0.240
	12	7.108 ± 0.276	6.744 ± 0.276	6.906 ± 0.284	8.521 ± 0.409	6.869 ± 0.264	6.702 ± 0.254
	13	6.341 ± 0.234	6.747 ± 0.285	6.907 ± 0.289	9.300 ± 0.384	6.905 ± 0.270	6.973 ± 0.285
Total Storage Resources (GB)	3072	6.688 ± 0.239	6.918 ± 0.285	7.031 ± 0.297	9.345 ± 0.445	6.535 ± 0.254	6.727 ± 0.245
	2816	6.832 ± 0.256	6.918 ± 0.285	7.031 ± 0.297	9.397 ± 0.416	6.351 ± 0.236	6.906 ± 0.272
	2560	6.687 ± 0.238	6.918 ± 0.284	7.031 ± 0.297	9.475 ± 0.422	7.052 ± 0.269	6.744 ± 0.273
	2304	6.309 ± 0.222	6.918 ± 0.285	7.031 ± 0.297	8.995 ± 0.408	7.002 ± 0.273	6.812 ± 0.265

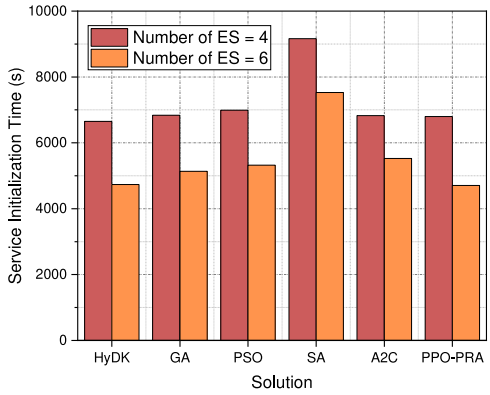

Fig. 9. Service initialization time performance of different methods.

intervals, which indicates superior stability and reliability in the service startup process.

Based on the analysis of the experimental results, the performance gap between the PPO-PRA algorithm and **HyDK** confirms the superiority of our framework in resource allocation. Furthermore, the comparison with the A2C algorithm demonstrates that the PPO-based learning mechanism within **HyDK** is more suitable for our environment, proving capable of training robust decision-making models for discrete problems even when dealing with high-dimensional action spaces. Finally, compared to GA, PSO, and SA, **HyDK** significantly outperforms the heuristic baselines in terms of both service initialization time and average sojourn time.

VI. CONCLUSION AND FUTURE WORK

In this paper, we addressed the tractability challenge of jointly optimizing service placement and continuous resource allocation in IoT-MEC environments. By analyzing the coupling between topological dependencies and resource dimensioning, we proposed the HyDK framework. This approach synergistically combines Deep Reinforcement Learning (DRL) with KKT conditions to effectively prune the action space, allowing the agent to focus on complex topological searches while guaranteeing numerical optimality for resource allocation. Furthermore, by integrating M/G/1 model, our method explicitly mitigates the latency jitter inherent in high-concurrency IoT workloads. Extensive simulations demonstrate that HyDK significantly outperforms existing baselines in terms of convergence speed and average sojourn time reduction, proving its robustness in dynamic network environments.

Future work will focus on bridging the gap between theoretical modeling and practical deployment. A key priority is to transition from simulation environments to a real-world experimental testbed. We intend to implement the HyDK algorithms on physical edge infrastructure to validate their adaptability to real-time network dynamics and hardware variability. Beyond physical validation, we also plan to enhance the model's scalability by incorporating containerized VNF orchestration and addressing critical non-functional requirements such as energy consumption and privacy-preserving computation.

REFERENCES

[1] A. Rawat and A. Pandey, "Recent trends in IoT: A review," *J. Manage. Serv. Sci.*, vol. 2, no. 2, pp. 1–12, Nov. 2022. [Online]. Available: https://jmss.a2zjournals.com/index.php/mss/article/view/21

[2] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2410–2422, Dec. 2020.

[3] Z. He, K. Li, and K. Li, "Cost-efficient server configuration and placement for mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 9, pp. 2198–2212, Sep. 2022.

[4] C. Kim and S. Kim, "Optimizing logging and monitoring in heterogeneous cloud environments for IoT and edge applications," *IEEE Internet Things J.*, vol. 10, no. 24, pp. 22611–22622, Dec. 2023.

[5] M. Soori, B. Arezoo, and R. Dastres, "Internet of Things for smart factories in Industry 4.0, A review," *Internet Things Cyber- Phys. Syst.*, vol. 3, pp. 192–204, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667345223000275

[6] J. Li, W. Liang, W. Xu, Z. Xu, Y. Li, and X. Jia, "Service home identification of multiple-source IoT applications in edge computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1417–1430, Mar./Apr. 2023.

[7] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 377–390, Feb. 2021.
[8] M. Goudarzi, M. Palaniswami, and R. Buyya, "A distributed deep reinforcement learning technique for application placement in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2491–2505, May2023.
[9] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.
[10] X. Zhang, Z. Li, C. Lai, and J. Zhang, "Joint edge server placement and service placement in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 11261–11274, Jul. 2022.
[11] R. Bi, T. Peng, J. Ren, X. Fang, and G. Tan, "Joint service placement and computation scheduling in edge clouds," in *Proc. IEEE Int. Conf. Web Serv.*, 2022, pp. 47–56.
[12] Y. Chen, Y. Sun, B. Yang, and T. Taleb, "Joint caching and computing service placement for edge-enabled IoT based on deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 19501–19514, Oct. 2022.
[13] J. Liu, C. Liu, B. Wang, G. Gao, and S. Wang, "Optimized task allocation for IoT application in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10370–10381, Jul. 2022.
[14] K. Moghaddasi and S. Rajabi, "Double deep Q-learning networks for energy-efficient IoT task offloading in D2D MEC environments," in *Proc. 7th Int. Conf. Internet Things Appl.*, 2023, pp. 1–6.
[15] J. Chen, H. Xing, X. Lin, A. Nallanathan, and S. Bi, "Joint resource allocation and cache placement for location-aware multi-user mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 24, pp. 25698–25714, Dec. 2022.
[16] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 281–294, Feb. 2021.
[17] L. Chen, L. Song, J. Chakareski, and J. Xu, "Collaborative content placement among wireless edge caching stations with time-to-live cache," *IEEE Trans. Multimedia*, vol. 22, no. 2, pp. 432–444, Feb. 2020.
[18] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11098–11112, Nov. 2018.
[19] A. Ndikumana et al., "Joint communication, computation, caching, and control in Big Data multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1359–1374, Jun. 2020.
[20] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, Jul. 2020.
[21] Y. Gao and G. Casale, "JCSP: Joint caching and service placement for edge computing systems," in *Proc. IEEE/ACM 30th Int. Symp. Qual. Serv.*, 2022, pp. 1–10.
[22] J. Xu et al., "Service placement strategy for joint network selection and resource scheduling in edge computing," *J. Supercomput.*, vol. 78, pp. 14504–14529, 2022.
[23] L. Liu et al., "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Int. Symp. Qual. Serv.*, 2019, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3326285.3329055
[24] Z. Qu, X. Zhang, H. Huang, Y. Li, and W. Wang, "Community and priority-based microservice placement in collaborative vehicular edge computing networks," in *Proc. 2024 IEEE Wireless Commun. Netw. Conf.*, 2024, pp. 1–6.
[25] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. Franklin de Castro, "An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing," *Comput. Netw.*, vol. 194, 2021, Art. no. 108146. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621002127
[26] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7257–7271, Nov. 2021.
[27] V. Farhadi et al., "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, Apr. 2021.
[28] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang, "Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 3870–3881, Jul. 2023.
[29] Z. He, Y. Xu, M. Zhao, W. Zhou, and K. Li, "Priority-based offloading optimization in cloud-edge collaborative computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 3906–3919, Nov./Dec. 2023.
[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: https://arxiv.org/abs/1707.06347
[31] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge Univ. Press, 2004.
[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html
[33] A. Aljuhani et al., "A deep learning integrated blockchain framework for securing industrial IoT," *IEEE Internet Things J.*, vol. 11, no. 5, pp. 7817–7827, Mar. 2024.
[34] T.-M. Hoang, T.-T. Nguyen, T.-A. Pham, and V.-N. Nguyen, "An IDS-based DNN model deployed on the edge network to detect industrial IoT attacks," in *Proc. Int. Conf. Intell. Things*, 2023, pp. 307–319.
[35] S. Ullah, W. Boulila, A. Koubâa, and J. Ahmad, "MAGRU-IDS: A multi-head attention-based gated recurrent unit for intrusion detection in IIoT networks," *IEEE Access*, vol. 11, pp. 114590–114601, 2023.

Zhenli He (Senior Member, IEEE) received the BS degree in software engineering from Yunnan University, Kunming, China, in 2010, and the MS and PhD degrees in software engineering and systems analysis and integration from Yunnan University, Kunming, China, in 2012 and 2015, respectively. From 2019 to 2021, he was a postdoctoral researcher with Hunan University, Changsha, China. He is currently an associate professor and head with the Department of Software Engineering, School of Software, Yunnan University. He was selected as a Young Talent of the Yunnan Provincial "Xingdian Talent Support Program" and is a recipient of the Hongyun Gardener Award for teaching excellence. His research interests include edge computing, energy-efficient computing, heterogeneous computing, and edge-AI techniques.

Xiaolong Zhai (Student Member, IEEE) received the MS degree in software engineering from Yunnan University, Kunming, China, in 2025. He is currently working toward the PhD degree in software engineering with the Dalian University of Technology, Dalian, China. His research focuses on mobile edge computing and deep reinforcement learning.

Mingxiong Zhao (Member, IEEE) Mingxiong Zhao received the BS degree in electrical engineering and the PhD degree in information and communication engineering from the South China University of Technology, Guangzhou, China, in 2011 and 2016, respectively. He was a visiting PhD student with the University of Minnesota, Twin Cities, MN, USA, from 2012 to 2013 and also with the Singapore University of Technology and Design, Singapore, from 2015 to 2016. He is currently a full professor and Donglu young scholar with Yunnan University, Kunming, China. He is also the vice dean of the National Pilot School of Software. His research interests include network security, mobile edge computing, and edge AI. He has been recognized as an Outstanding Young Talent of Yunnan Province since 2019. He serves as the youth editor of the Journal of Information and Intelligence and committee member of Technical Committee on Data Security of the China Communications Society.

Wei Zhou received the PhD degree from the University of Chinese Academy of Sciences. He is currently a full professor with the Software School, Yunnan University. His research interests include distributed data-intensive computing and bioinformatics. He is a fellow of the China Communications Society, a member of the Yunnan Communications Institute, and a member of the Bioinformatics Group of the Chinese Computer Society. He received the Wu Daguan Outstanding Teacher Award from Yunnan University in 2016 and was selected for the Youth Talent Program of Yunnan University in 2017. He has led several projects funded by the National Natural Science Foundation.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston, in 1990. He is a SUNY distinguished professor with the State University of New York and a National distinguished professor with Hunan University (China). He has authored or co-authored more than 1280 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top few most influential scientists in parallel and distributed computing, regarding single-year impact (ranked #2) and career-long impact (ranked #3) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars and is among the top 0.02% out of more than 20 million scholars worldwide based on top-cited publications in the last ten years. He is listed in ScholarGPS Highly Ranked Scholars and is among the top 0.002% out of more than 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is a member of Academia Europaea (MAE), a member of the European Academy of Sciences and Arts (EASA), a fellow of the American Association for the Advancement of Science (AAAS), a member of the USA National Academy of Artificial Intelligence (NAAI), a fellow of the International Academy of Artificial Intelligence Sciences (AAIS), a fellow of the International Artificial Intelligence Industry Alliance (AIIA), a member of the World Academy of Sciences (WAS), a fellow of the World Academy of Engineering & Technology (WAET), a fellow of the Asia Computational Intelligence Society (ACIS), and a member of the SUNY Distinguished Academy.