Task-Aware Collaborative Inference and Fine-Grained DNN Partitioning in MEC Networks

Guanlei Zhang, Qiyang Zhang, *Member, IEEE*, Lei Feng, *Member, IEEE*, Fanqin Zhou, *Member, IEEE*, Praveen Kumar Donta, *Senior Member, IEEE*, and Schahram Dustdar, *Fellow, IEEE*

***Abstract*—Mobile devices (MDs) are increasingly incorporating deep neural network (DNN) inference into their systems due to the rapid growth of intelligent applications. Mobile edge computing-based distributed DNN collaborative inference has gained popularity due to limited on-device computation and energy budgets. However, the resource competition among MDs, along with the coupling of collaborative inference tasks across MDs and servers, creates significant challenges for efficient resource management. This issue is further exacerbated by the complexity of directed acyclic graph (DAG)-structured DNNs. Most prior studies do not jointly address the dual challenges of partitioning complex-structured DNNs and leveraging advanced optimization for collaborative inference, and their resilience to channel condition fluctuations remains underexplored. To address these challenges, we propose a novel task-aware collaborative inference framework. First, we devise a fine-grained partitioning point search algorithm based on a bidirectional graph linked list, which enables one-dimensional and flexible partitioning of DAG-structured DNNs. We then reformulate the problem of minimizing collaborative inference energy consumption and latency as a task-aware Markov decision process (MDP), which partitions each user's inference task queue into consecutive task windows for resource allocation. Building on this, we propose an Embedded Multi-Agent Hybrid Proximal Policy Optimization (EMH-PPO) algorithm to learn effective policies. Extensive experiments conducted across diverse network scenarios reveal that, compared to local DNN inference on MDs, our proposed method reduces inference latency by up to 64% and energy consumption by up to 46%.**

***Index Terms*—Mobile edge computing, distributed inference, deep reinforcement learning, model partitioning, resource allocation.**

Received 10 January 2025; revised 4 November 2025; accepted 28 December 2025. Date of publication 5 January 2026; date of current version 7 May 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62571057, in part by the Beijing Natural Science Foundation of China under Grant L254061, and in part by the Fundamental Research Funds for the Beijing University of Posts and Telecommunications under Grant 2025TSQY03. Recommended for acceptance by D. Roy. *(Corresponding authors: Qiyang Zhang; Lei Feng.)*

Guanlei Zhang, Lei Feng, and Fanqin Zhou are with the State Key Laboratory of Network and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: zhangguanlei@bupt.edu.cn; fenglei@bupt.edu.cn; fqzhou2012@bupt.edu.cn).

Qiyang Zhang is with the School of Computer Science, Peking University, Beijing 100871, China (e-mail: qiyangzhang@pku.edu.cn).

Praveen Kumar Donta is with the Department of Computer and Systems Sciences, Stockholm University, 16425 Kista, Sweden (e-mail: praveen@dsv.su.se).

Schahram Dustdar is with the Distributed Systems Group, TU Wien, A-1040 Wien, Austria, and also with UPF, 08018 Barcelona, Spain (e-mail: dustdar@dsg.tuwien.ac.at).

Digital Object Identifier 10.1109/TMC.2025.3650680

I. Introduction

In recent years, the rapid advancement of artificial intelligence (AI) has been propelled by significant enhancements in computational power and the expanding availability of data [1]. An increasing number of Internet of Things (IoT) devices are integrating AI algorithms, particularly deep neural network (DNN)-based intelligent applications, achieving significant performance improvements in AI-related tasks such as image understanding, natural language processing, and content generation [2]. However, performing computation-intensive DNN-based tasks typically demands substantial computational resources, which leads to challenges related to inference latency and energy consumption when deploying these models on resource-constrained mobile devices (MDs) [3].

To address these challenges, collaborative inference in mobile edge computing (MEC) networks has been proposed. By utilizing the distributed computing resources and deploying DNNs across these resources [4], it is possible to flexibly offload part or all of the DNN inference computation to MEC servers [5]. With the dense deployment of edge servers in 5G networks [6], collaborative inference in MEC networks enables efficient completion of intensive DNN inference tasks, significantly reducing overhead and improving the user experience.

Several existing works explore DNN inference offloading and resource allocation for collaborative inference. However, current methods remain limited in two respects. First, most studies consider a single MD-server pair [7], whereas practical edge environments feature multiple servers concurrently serving multiple MDs. This multi-user contention for server resources makes server-side costs a non-negligible factor that is sometimes overlooked in the overall optimization. Second, some studies propose advanced optimization algorithms to solve such NP-hard problems, including deep reinforcement learning (DRL)-based methods [8]. However, these methods are not well suited for the collaborative inference of complex-structured DNNs and typically rely on a time-based Markov decision process (MDP) formulation [9], i.e., a discrete-time MDP with uniform time discretization, which shows limited adaptability in dynamic MEC networks. On one hand, research on partitioning complex-structured DNNs frequently relies on mathematical approaches such as graph min-cut, which are computationally expensive [10], [11], [12]. On the other hand, while studies focusing on resource allocation have introduced efficient DRL-based methods [13], [14], [15], these methods

typically adopt a time-based MDP formulation. This approach, where decisions are made at fixed time slots, can be misaligned with the variable completion times of inference tasks in dynamic MEC networks. Such a temporal mismatch may reduce the effectiveness of DRL-based methods. Furthermore, these approaches often only tackle coarse-grained partitioning for chained DNNs.

In this paper, we propose a multi-agent collaborative inference framework involving multiple MDs and servers, focusing on the partitioning of directed acyclic graph (DAG)-structured DNNs and the resource scheduling for collaborative inference to optimize inference latency and energy consumption for MDs. However, it also poses several difficulties. First, the interdependencies in this scenario and the complex structure of DNNs pose significant challenges for efficient collaborative inference. For example, there are competitive and interdependent relationships among MDs and between MDs and servers. Moreover, as the depth of DNNs increases, traditional chain-like topologies struggle to meet performance demands [16] (e.g., AlexNet [17], VGGNet [18]), prompting the evolution of DNNs towards a DAG topology [19]. Consequently, partitioning these models is no longer feasible through linear segmentation [20], as their complex multi-dimensional structures introduce more potential partitioning points, further complicating resource allocation. Second, while DRL-based methods show strong promise for these complex optimization problems, their success critically depends on the design of the underlying MDP. Since DRL optimizes a policy by maximizing the expected return of a given MDP, the pivotal challenge is to formulate an MDP that robustly handles the variable completion times of inference tasks in dynamic MEC networks.

To address these challenges, we propose a task-aware collaborative inference framework for multi-user and multi-server scenarios. This framework includes an operator-grained partitioning method tailored for DAG-structured DNNs and an embedded multi-agent proximal policy optimization (EMH-PPO) algorithm. First, we introduce a bidirectional graph linked list based algorithm to represent partitioning points of DAG-structured DNNs in a one-dimensional form, transforming the problem into a linear partitioning task similar to that of chained DNNs. To enable robust policy learning in dynamic MEC networks, we propose a task-aware MDP formulation. Finally, we design an EMH-PPO algorithm with an information-sharing mechanism to solve the optimization problem. Experiments conducted on three representative models, ResNet-50 [23], GoogLeNet [24], and MobileNetV2 [25], demonstrate the generality and scalability of our framework. The results show that our framework significantly reduces inference latency and energy consumption on MDs. The main contributions of this work are summarized as follows:

- We formulate a task-aware collaborative inference framework for a multi-user, multi-server MEC network scenario and propose an operator-grained partitioning point search algorithm based on a bidirectional graph linked list for DAG-structured DNNs. This approach simplifies DAG partitioning to a one-dimensional partitioning scheme while preserving flexibility in partitioning and resource scheduling.
- We reformulate the problem as a task-aware MDP with decision periods triggered by task completion rather than fixed slots, improving robustness to time-varying channel conditions in dynamic MEC networks. Building on this, we propose the EMH-PPO algorithm, which enhances information sharing and collaboration among agents to address resource allocation for collaborative inference.
- We conduct extensive experiments to evaluate the performance of proposed method. Results demonstrate that compared to local DNN inference, our method can reduce inference latency by up to 64% and decrease the energy consumption of MDs by up to 46%.

The remainder of this paper is organized as follows. Section II reviews related work. Section III introduces the system model and formulates the original optimization problem. Section IV reformulates the original optimization problem as a task-aware MDP, provides a theoretical analysis of its equivalence to the original problem, and then introduces the proposed EMH-PPO algorithm. Section V showcases our experimental results and analysis. Finally, Section VI concludes the paper.

II. Related Work

A. DNN Partitioning

Existing DNN partitioning methods can be generally categorized into vertical (model-level) partitioning and horizontal (data-level) partitioning. A pioneering work in linear vertical partitioning is Neurosurgeon [20], which demonstrates how a chained DNN can be split into two parts, thereby laying the groundwork for partial offloading and distributed deployment. Since this approach does not alter the fundamental DNN architecture, many subsequent works have followed a similar strategy. Horizontal partitioning methods, such as those in [26], [27], [28], primarily focus on slicing feature maps to enhance computational parallelism. However, these approaches are generally limited to convolutional neural networks (CNNs) and, due to restricted information exchange among slices, require careful evaluation of their potential impact on model accuracy. Additionally, some studies modify the original DNN architecture to reduce redundant computations [29]. Laskaridis et al. [30] proposed SPINN, which integrates vertical partitioning with an early-exit mechanism for CNNs, enabling adaptation to dynamic conditions and user demands. Similarly, Ren et al. [31] introduced an early-exit mechanism for chained DNNs, dynamically reducing redundant computations based on accuracy requirements. EDDI [10] integrates early-exit mechanisms into DAG-structured DNNs and formulates DNN partitioning optimization as a minimum cut problem on the DAG. PDD [22] utilizes topological sorting to transform DAG-topology DNNs into equivalent chained DNN structures. Although many studies have explored DNN partitioning methods from various perspectives, fine-grained partitioning techniques in dynamic MEC networks still need further investigation.

TABLE I
COMPARISON BETWEEN THE PROPOSED METHOD AND EXISTING WORKS

Ref.	Scenario	Partitioning Method	Channel Condition Dynamics	Optimization Objective(s)	Optimization Method	MDP Formulation
[21]	Single-user / single-server	Linear partitioning	—	Latency	Integer linear programming	—
[22]	Single-user / multi-server	Layer-grained DAG	—	Latency	Greedy	—
[12]	Multi-user / single-server	Layer-grained DAG	Bandwidth fluctuations	Latency	Iterative heuristic algorithm	—
[13]	Multi-user / single-server	Layer-grained DAG	—	Latency, energy	Dynamic programming, DRL (SAC)	Discrete-time
[7]	Multi-user / single-server	Linear partitioning	—	Latency, energy	DRL (MAHPPO)	Discrete-time
[15]	Multi-user / multi-server	Linear partitioning	—	Latency	DRL (Multi-Task Learning based A3C)	Discrete-time
Ours	Multi-user / multi-server	Operator-grained DAG	User mobility, shadow fading	Latency, energy	DRL (EMH-PPO)	**Task-aware**

B. Collaborative DNN Inference

Extensive research has investigated the collaborative inference problem of DNNs in wireless scenarios [32]. Such studies typically focus on the load distribution of DNN segments and resource scheduling issues within edge networks. Edgent [33] employs regression-based approaches and an online change point detection algorithm to perform resource scheduling in response to bandwidth fluctuations. MCIA [34] constructs multiple compressed versions of the original DNN, optimizing both version selection and partitioning. Additionally, it introduces a DRL-based approach to balance accuracy and latency. Liu et al. [35] accelerated end-edge-cloud collaborative inference by developing a DNN-based latency prediction model that identifies two partitioning points. Li et al. [21] proposed JALAD, a model partitioning and intermediate data quantization transmission scheme tailored for chained CNN architectures, which employs an iterative search to identify the optimal decision. [36] and [37] designed distinct autoencoders to compress the intermediate data in CNN model partitioning, proving highly effective for image-related tasks. [7] and [15] propose multi-agent DRL algorithms to optimize resource allocation for collaborative inference in multi-device scenarios, significantly reducing latency and energy costs. However, these methods are limited to chained DNNs. Previous works often overlook server-side processing overhead, and applying advanced optimization methods to collaborative inference with modern DNNs remains challenging. Additionally, the effectiveness of their time-based MDP formulations in dynamic MEC networks has not been adequately discussed. Table I summarizes the distinctions between our work and previous studies.

TABLE II
IMPORTANT NOTATIONS

Notation	Description
$\mathcal{S}$	The number of partitioning points
$\mathcal{C}$	The number of wireless channels
$\mathcal{N}$	The number of MDs
$\mathcal{M}$	The number of computing MEC servers
$\mathcal{X}_n$	The number of total DNN inference tasks for MD n
π	The optimization policy
$R_n(\pi)$	The uplink data transmission rate of MD n
$I_n(\pi)$	The intermediate data size of MD n
$W(\pi)$	The expected average size of the task window
W_{max}	The maximum number of tasks allocated to each MD within a task window
$\boldsymbol{L}_{n,k}(\pi)$	The inference latency of MD n at task window k
$\boldsymbol{E}_k(\pi)$	The total energy consumption of all MD at task window k
$L_{wait,k}(\pi)$	The average wait time at task window k
$M_n(\pi)$	The number of MAC operations of MD n
α	The balance factor of waiting time penalty
β	The energy consumption penalty factor
ζ	The energy coefficient
$\mathbf{X}_k$	The number of DNN inference tasks remaining up to task window k
$\mathbf{d}_k$	The distance between MDs and computing MEC servers at task window k
$\mathbf{s}_k$	The partitioning point selections for MDs at task window k
$\mathbf{c}_k$	The wireless channel selections for MDs at task window k
$\mathbf{m}_k$	The computing MEC server selections for MDs at task window k
$\mathbf{P}_k$	The transmission power of MDs at task window k
$\mathbf{W}_k$	The task window sizes of MDs at task window k
$\boldsymbol{\theta}$	The parameter set of the agent networks
ϕ	The parameters of the embedding networks
w	The parameters of the critic networks

III. SYSTEM MODEL

In this section, we first describe the architecture of the proposed system and then detail each subsystem. We conclude with the problem formulation. The key notations used throughout this paper are summarized in Table II.

A. System Framework and Workflow

System Framework: The overall architecture of the proposed task-aware collaborative inference framework is depicted in Fig. 1. The system consists of MDs and MEC servers. MDs, such as smartphones and in-vehicle computing platforms, are often constrained by computing resources, energy, and mobility. These limitations result in low efficiency or inability to perform DNN inference locally. We split DNNs into two parts, deploying one part locally on the MD and the other on the computing MEC server. Based on the status of the MDs, the scheduling MEC server allocates appropriate DNN partitioning points and offloading decisions. Following these decisions, the MDs complete local DNN inference and then send the intermediate data to the designated computing MEC servers, which subsequently process the offloaded inference tasks. Furthermore, the scheduling server and computing servers are interconnected through optical fibers. For simplicity, we omit the wireless base station (BS) in the system and integrate its functions into MEC servers.

Workflow: Each MD first initializes its own queue of DNN inference tasks. At the beginning of each decision period, the scheduling MEC server evaluates the status of MDs, allocates the number of tasks to be completed (referred to as the task window, which represents a subset of tasks from the overall task queue) based on their current capacity, and formulates specific offloading strategies (e.g., DNN partitioning, radio resource allocation) for these tasks. All offloading decisions made throughout the

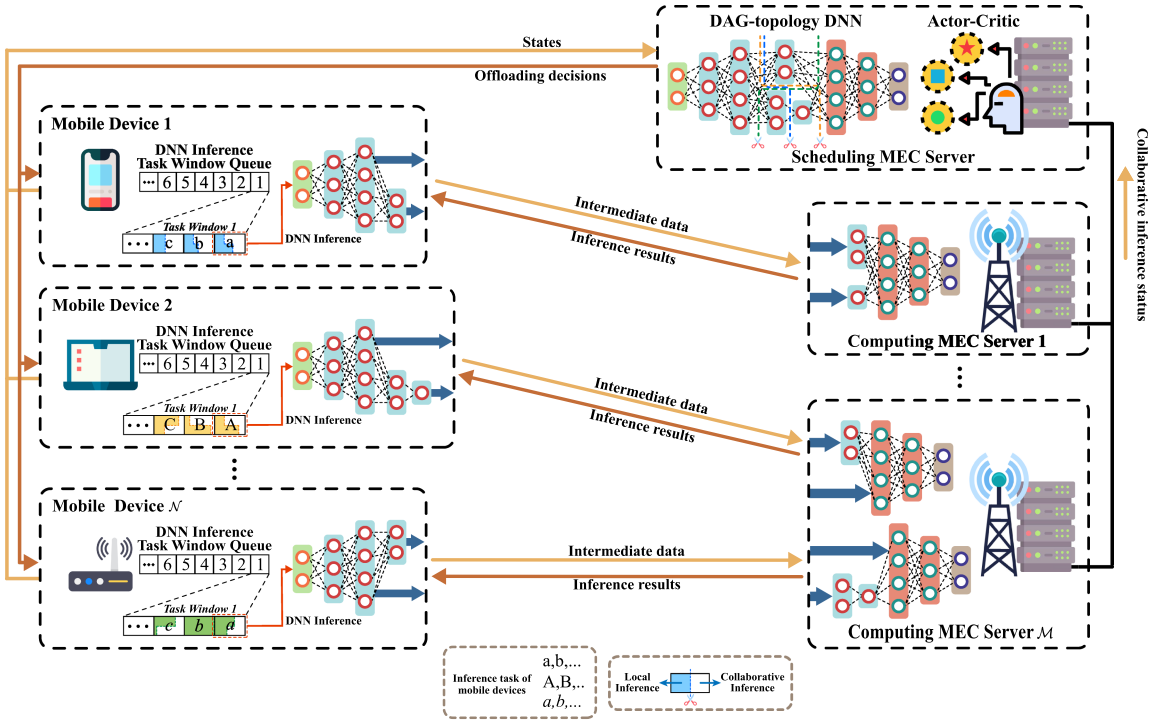

Fig. 1. Overview of the task-aware collaborative inference framework. The DNN inference task queue of the MD is divided into multiple task windows, each containing several DNN inference tasks. MDs perform partial DNN inference based on offloading decisions (e.g., partitioning points, channels, computing MEC servers), with the DNN split into two parts.

optimization process are collectively represented as the policy set π. Subsequently, MDs commence local DNN inference based on the directives provided by the scheduling MEC server and transmit intermediate data via wireless channels to the assigned computing MEC servers. These servers complete the remainder of the DNN inference process and transmit the task status to the scheduling MEC server. Our primary objective is to find an optimal strategy π enabling the scheduling MEC server to guide all MDs toward completing all DNN inference tasks as quickly and energy-efficiently as possible.

B. DNN Partitioning Model

As most DNNs do not exhibit cyclic dependencies, we can readily construct a DAG representation based on their forward pass.[1] As illustrated in Fig. 2, using the Inception3a module from GoogLeNet [24] and the ResBlock from ResNet-50 [23] as examples, the upper parts shows the model topology, while the lower parts presents the operator-grained DAG representation. In this context, a partitioning point in a DAG-structured DNN can be represented by a set of operators that collectively block all forward data flow.[2] For instance, in Fig. 2(b), potential

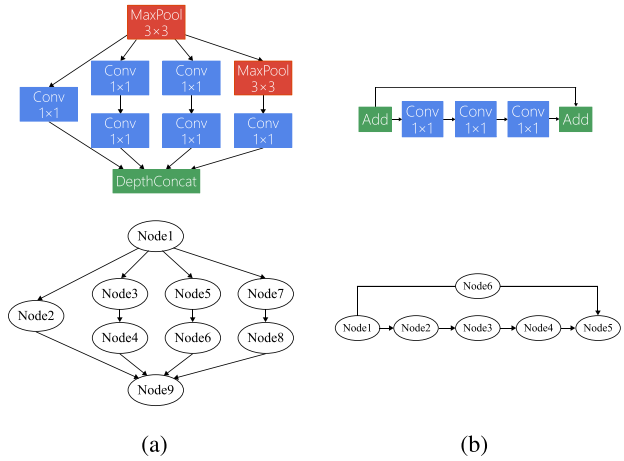

Fig. 2. Layer-wise and operator-wise DAG representations of DNN modules: (a) Inception3a module from GoogLeNet [24]; (b) ResBlock from ResNet-50 [23].

partitioning points can include numerous combinations, such as (Node2, Node6) or (Node3, Node6), among others.

The extensive combinations and interdependencies among operators make identifying all possible partitioning points complex, especially for collaborative inference in dynamic MEC networks, where channel quality and server-side load vary over time, requiring online adaptation of partitioning decisions. Moreover, the graph structure hinders a straightforward mathematical representation of partitioning decisions.

[1] In this paper, we exclude certain operators with comparatively small computation workloads (e.g., pooling) and integrate their effects (the change in transmitted data size) into surrounding operators, as they have negligible impact on our partitioning decisions.

[2] Under this representation, each partitioning point in a chained DNN contains only a single operator, allowing our approach to be readily extended to chained DNNs in previous studies.

Algorithm 1: DAG-Structured DNN Partitioning Point Search Algorithm.

input : DNN model node pool P, $\{p_i \in P | i = 1, \ldots, N\}$;
output: Partitioning point set S;

Initialization: empty set S;
for $i = 1, \ldots, N$ **do**
  $P_1 \leftarrow P \setminus (p_i \cup p_i^{parent} \cup p_i^{child})$;
  if P_1 *is empty* **then**
    $S \leftarrow S \cup p_i$;
  else
    Initialization: empty sibling node pool P_b and empty combination set P_c;
    foreach $n \in P_1$ **do**
      $P_2 \leftarrow P_1 \setminus (n \cup n^{parent} \cup n^{child})$;
      $P_b \leftarrow P_b \cup (P_1 \setminus P_2)$;
    $P_c = P_b(1) \times P_b(2) \cdots \times P_b(length(P_b))$; // × denotes Cartesian product
    foreach $c \in P_c$ **do**
      Add the set $(p_i \cup c)$ to partitioning point set S.

We introduce an innovative bidirectional graph linked list to represent operator-grained DAGs, where each node in the list corresponds to an operator in the DAG. Each such node encapsulates several key attributes, including references to its parent(s) and child(ren) to facilitate bidirectional traversal (an operator can have multiple parents or children, thereby forming the graph structure), its computation workload, its associated inference latency (for subsequent resource allocation). Building on this structure, we propose an algorithm that searches and organizes all potential partitioning points into a one-dimensional sequence, facilitating systematic enumeration and analysis while ensuring the original DNN model structure is not changed, as outlined in Algorithm 1. For any node p_i in the linked list, we begin by performing a bidirectional traversal of its parent and child nodes, then remove them from the complete set of nodes. If the remaining set contains only p_i, this indicates that p_i alone forms a partitioning point. Otherwise, we repeat the bidirectional traversal on the remaining nodes to identify any parallel branch nodes associated with p_i. We then collect the node sets for each parallel branch, and by computing their Cartesian product, obtain all partitioning points that include p_i. Ultimately, we represent all partitioning points in a one-dimensional form, $s \in \{1, \ldots, \mathcal{S}\}$, thereby reducing the DAG-structured DNN partitioning to a linear partitioning scheme analogous to chained DNNs while preserving partitioning flexibility. Here, s denotes a partitioning point encompassing multiple operators, and we let $I_n(\pi)$ represent the cumulative output feature size of all operators in the partitioning decision under policy π for MD n, which quantifies the amount of intermediate data that must be transmitted over the wireless channel. $s = 1$ and $s = \mathcal{S}$ denote the partitioning points at input and output, respectively. The former scenario involves sending the original input data from the MD to the computing MEC server for inference, while the latter indicates that the MD performs complete local DNN inference.

In terms of computational complexity, let N denote the total number of operator nodes in the DAGstructured DNN, and let M be the cardinality of the parallel branch node set associated with an arbitrary node. The outer loop of Algorithm 1 visits all N nodes, resulting in a computational complexity of $\mathcal{O}(N)$. For each node, the algorithm enumerates all admissible combinations of its parallel branch nodes, which contain 2^M possibilities, leading to a computational complexity of $\mathcal{O}(2^M)$. Thus, the overall computational complexity is $\mathcal{O}(N \cdot 2^M)$. In practical DNNs, the computational graph is typically deep yet narrow, so most layers have only a few concurrent branches (i.e., $M \ll N$). This structural property ensures that the exponential term 2^M remains bounded by a modest constant (e.g., $M \leq 4$ for ResNet-like architectures). Hence, the proposed partitioning point search algorithm achieves near-linear scalability with respect to network size N.

C. Communication Model

An MD needs to transmit the intermediate data output by its local sub-model to the corresponding computing MEC server for collaborative inference. The wireless channels available in the system are represented as $c \in \{1, \ldots, \mathcal{C}\}$, and the MDs are denoted as $n \in \{1, \ldots, \mathcal{N}\}$. We consider a scenario where physical channels can be reused by multiple MDs, in keeping with the trend in future mobile networks toward higher spectral efficiency [38]. Using the Shannon formula, the uplink data transmission rate of MD n can be calculated as

$$R_n(\pi) = B_c \log_2 \left(1 + \frac{P_n(\pi) G_n}{\sigma_c + \sum_{i \in \{1, \ldots, \mathcal{N}\} \setminus \{n\}} P_i(\pi) G_i} \right), \quad (1)$$

where B_c is the bandwidth of channel c, $P_n(\pi)$ and G_n are the transmission power and channel gain of MD n. σ_c is the background Gaussian white noise power of channel c.

D. Inference Latency Model

In the inference latency model, the total inference latency consists of three components: the local computation latency of MDs L_{local}, the transmission latency of intermediate data L_{trans}, and the computation latency of the computing MEC server L_{server}. We ignore the transmission delay of user state and decision information, since these control messages are negligible in size compared to the intermediate data generated by collaborative inference and thus have virtually no effect on policy performance. Among these components, L_{local} and L_{server} can be readily monitored, while L_{trans} can be calculated based on the communication model. We denote the computing MEC servers as $m \in \{1, \ldots, \mathcal{M}\}$. For L_{server}, we consider the actual operating mode of the GPU/NPU, meaning that L_{server} is proportional to the number of MDs utilizing the same computing MEC server. The proportionality factor is the baseline inference latency L_0 of the server. The overall inference latency $\mathbf{L}(\pi)$ of MD n can be represented as

$$\begin{aligned} \mathbf{L}_n(\pi) &= L_{local,n}(\pi) + L_{trans,n}(\pi) + L_{server,n}(\pi) \\ &= L_{local,n}(\pi) + \frac{I_n(\pi)}{R_n(\pi)} + L_0 \cdot \sum_{i=1}^{\mathcal{N}} \mathbb{I}(m_i(\pi) = m_n(\pi)), \quad (2) \end{aligned}$$

where the baseline inference latency L_0 denotes the latency when a computing MEC server exclusively serves a single MD under unloaded conditions. $\mathbb{I}(\cdot)$ represents the indicator function,

which equals 1 if the enclosed condition holds, and 0 otherwise. And $m_i(\pi)$ indicates the index of computing MEC server to which MD i offloads its task under policy π.

E. Energy Consumption Model

In the energy consumption model, we consider the energy consumption of MDs, as many of them are energy-constrained. It comprises two parts: local computation energy consumption and intermediate data transmission energy consumption. Transmission energy consumption can be easily calculated as the product of transmission power and time. As for the computation energy consumption of MDs, we estimate it based on the number of multiply-accumulate (MAC) operations performed by the DNN sub-model on the MD side, scaled by an energy coefficient ζ (representing energy per MAC operation) that depends on the chip architecture [39]. The total energy consumption of MD n can be expressed as

$$\mathbf{E}_n(\pi) = E_{trans,n}(\pi) + \zeta M_n(\pi), \tag{3}$$

where $E_{trans,n}(\pi) = P_n(\pi) \cdot L_{trans,n}(\pi)$, and $M_n(\pi)$ denotes the computation workload executed on the MD side. Specifically, $M_n(\pi)$ is the sum of MAC operations for all operators in the MD-side sub-model, as determined by the partitioning point s specified in optimization policy π.

F. Problem Formulation

Building on the modeling described above, we introduce the optimization objective for original problem. Let $\mathcal{X}_n \in \{\mathcal{X}_1, \ldots, \mathcal{X}_{\mathcal{N}}\}$ represent the total number of DNN inference tasks for MDs. Our goal is to minimize inference latency and energy consumption of MDs, which can be formulated as follows

$$\begin{aligned} \min_{\pi} \ & \left\{ \max_{n} \sum_{x=1}^{\mathcal{X}_n} \mathbf{L}_{n,x}(\pi) + \frac{\beta}{\mathcal{N}} \sum_{x=1}^{\mathcal{X}_n} \sum_{n=1}^{\mathcal{N}} \mathbf{E}_{n,x}(\pi) \right\} \\ \text{s.t.} \ & C1: \beta > 0,\ \zeta > 0,\ \forall \beta, \zeta \in \mathbb{R}^+, \\ & C2: 0 \le P_n(\pi) \le P_{limit},\ \forall n \in \{1, \ldots, \mathcal{N}\}, \end{aligned} \tag{4}$$

where $\mathbf{L}_{n,x}(\pi)$ and $\mathbf{E}_{n,x}(\pi)$ represent the inference latency and energy consumption of MD n for executing inference task x. Constraint $C1$ ensures that the energy consumption penalty factor β and the energy coefficient ζ are positive, reflecting the balance between latency and energy. Constraint $C2$ specifies a transmission power limit P_{limit}, maintaining the transmission power of MDs within a reasonable range.

IV. Task-Aware Embedded Multi-Agent DRL Approach

In this section, we introduce a task-aware embedded multi-agent DRL approach, termed EMH-PPO. First, we discuss the Markov property of the original optimization problem and the motivation for constructing an MDP from the task dimension, and propose a novel task-aware MDP reformulation. We then analyze the equivalence of this reformulation to the original problem. Building on this reformulation, we detail the design of states, actions, and rewards. Lastly, we outline the EMH-PPO framework, including the actor-critic network design and the policy optimization process. Additionally, we provide a computational complexity analysis for EMH-PPO.

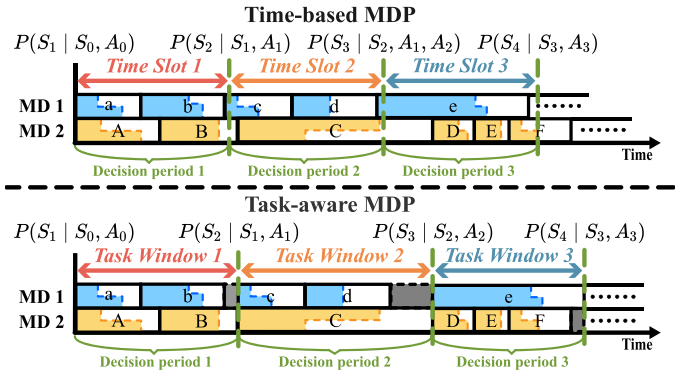

Fig. 3. Comparison of the time-based MDP (slot-based decisions) and the task-aware MDP (task-window decisions), illustrated with a two-MD scenario.

A. Task-Aware MDP Reformulation

The original optimization problem described in (4) is a combinatorial problem with a non-convex, mixed-integer nonlinear objective function and is NP-hard [40]. DRL provides an attractive way to address such problems by casting them as sequential decision making on an MDP. However, applying DRL effectively requires formulating the problem as a valid MDP, and the conventional time-based approach presents inherent challenges in this context. A key limitation of the time-based MDP formulation is its critical dependence on a pre-specified decision period (i.e., the time-slot length) [9], which is difficult to select a priori in dynamic MEC networks [7]. This difficulty arises because task completion times are inherently variable under time-varying channel conditions. With short slots, a single task may span multiple slots, making the transition dynamics sensitive to past states and actions, thereby violating the Markov assumption and increasing learning instability. With long slots, the decision frequency is reduced, leading to less responsive resource allocation and diminished control flexibility.

To resolve this, we propose a task-aware MDP formulation that divides each user's task queue into consecutive task windows. Fig. 3 compares the two MDP formulations and shows how short slots in the time-based MDP make transitions history-dependent. By aligning decision periods with task windows, each period's duration becomes variable and adapts to the completion time of the corresponding inference task. Since the inference task queue can be specified a priori and remains unchanged under network dynamics, this task-synchronous timing decouples policy decisions from latency fluctuations induced by channel condition variability, thereby ensuring stable policy learning in dynamic MEC networks. At each decision period, the DRL agent assigns to each MD n a task window containing $W_n(\pi)$ inference tasks, with $W_n(\pi)$ being capped at a maximum of $W_{\max}$. Within every task window, all MDs begin their respective inference tasks. The first of these tasks for each MD is initiated simultaneously, due to the assumption that the

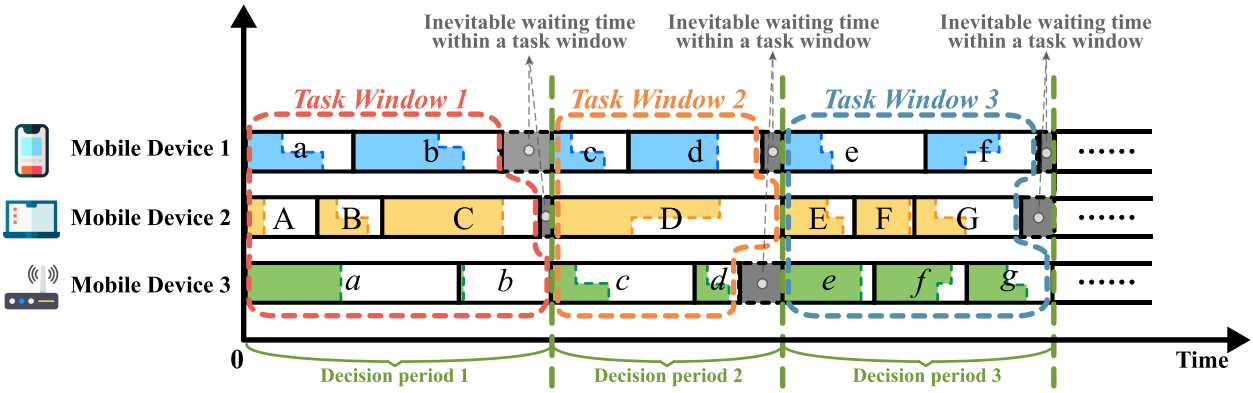

Fig. 4. An illustrative example of task window allocations involving three MDs, where each MD processes a maximum of $W_{\max} = 3$ tasks within each task window. Letters in different fonts represent inference tasks from different MDs, with the dashed lines outlining the colored segments allocated for local processing, while the remaining white regions indicate workloads offloaded to servers. The gray areas denote inevitable waiting times within the task window, and the task window concludes when the MD taking the longest to complete all its assigned tasks finishes.

transmission latency of the decision information is negligible. Fig. 4 illustrates an example of task window allocations.

Due to the division of inference task queues, $K(\pi)$ task windows are required to complete all inference tasks. We reformulate the optimization objective as minimizing the average cost of inference tasks

$$\min_{\pi} \left\{ \sum_{k=1}^{K(\pi)} \frac{\max_n \mathbf{L}_{n,k}(\pi) + \alpha L_{wait,k}(\pi)}{W(\pi)} + \frac{\beta}{\mathcal{N}} \sum_{k=1}^{K(\pi)} \frac{\mathbf{E}_k(\pi)}{W(\pi)} \right\}$$

$$\text{s.t.} \quad C1: \alpha > 0,\ \beta > 0,\ \zeta > 0,\ \forall \alpha, \beta, \zeta \in \mathbb{R},$$

$$C2: 0 \le P_n(\pi) \le P_{limit},\ \forall n \in \{1, \ldots, \mathcal{N}\},$$

$$C3: W(\pi) > 0. \tag{5}$$

Variations in MDs' channel conditions mean that tasks within a task window rarely finish simultaneously, leading to brief waiting periods, as illustrated by the gray areas in Fig. 4. This inter-device waiting is undesirable because it lowers resource utilization in our collaborative inference framework. To alleviate the effect, we add a waiting time penalty $L_{wait,k}(\pi)$ weighted by a balance factor α. This penalty represents the average waiting time in task window k and encourages the policy to minimize mutual waiting. The inference latency of MD n in task window k is $\mathbf{L}_{n,k}(\pi) = \sum_{x=1}^{W_{n,k}(\pi)} \mathbf{L}_{n,k,x}(\pi)$, where $W_{n,k}(\pi)$ is the number of tasks assigned to MD n in that window. The total energy consumption of all MDs in task window k is $\mathbf{E}_k(\pi) = \sum_{x=1}^{W_{n,k}(\pi)} \sum_{n=1}^{\mathcal{N}} \mathbf{E}_{n,x,k}(\pi)$. And $W(\pi) = \mathbb{E}[\frac{\sum_{n=1}^{\mathcal{N}} W_{n,k}(\pi)}{\mathcal{N}}] = \mathbb{E}[W_k(\pi)]$ represents the expected average size of the task windows.

We define the state S_k and action A_k as

$$S_k = \{\mathbf{X}_k, \mathbf{d}_k\} = \{(X_{n,k}, d_{n,k})\}_{n=1}^{\mathcal{N}}, \tag{6}$$

$$\begin{aligned} A_k &= \{\mathbf{s}_k, \mathbf{c}_k, \mathbf{m}_k, \mathbf{P}_k, \mathbf{W}_k\} \\ &= \{(s_{n,k}, c_{n,k}, m_{n,k}, P_{n,k}, W_{n,k})\}_{n=1}^{\mathcal{N}} \\ &= \left\{ \left(\{s_{n,k,x},\ c_{n,k,x},\ m_{n,k,x},\ P_{n,k,x}\}_{x=1}^{W_{n,k}},\ W_{n,k} \right) \right\}_{n=1}^{\mathcal{N}}. \end{aligned} \tag{7}$$

The state S_k encompasses relevant information about all MDs, such as the number of remaining inference tasks $\mathbf{X}_k$ and the distance to the BS $\mathbf{d}_k$. The action A_k includes resource allocation decisions, such as the partitioning point $\mathbf{s}_k$, the selection of wireless channel $\mathbf{c}_k$ and computing MEC server $\mathbf{m}_k$, the transmission power $\mathbf{P}_k$, and the size of task window $\mathbf{W}_k$. The reward $\boldsymbol{r}_k$ can be represented as the negative of the optimization objective in (5), in which case maximizing the cumulative reward for an episode is equivalent to minimizing latency and energy consumption

$$\boldsymbol{r}_k(\pi) = -\left(\frac{\max_n \mathbf{L}_{n,k}(\pi) + \alpha L_{wait,k}(\pi)}{W_k(\pi)} + \frac{\beta}{\mathcal{N}} \frac{\mathbf{E}_k(\pi)}{W_k(\pi)} \right). \tag{8}$$

In this context, we use the sample $W_k(\pi)$ of $W(\pi)$ as an estimate of $W(\pi)$.

B. Equivalence Analysis

Next, we analyze the equivalence of our reformulated task-aware MDP with the original problem. First, we express the original problem in the form of task windows

$$\begin{aligned} J_1(\pi) &= \max_n \sum_{x=1}^{\mathcal{X}_n} \mathbf{L}_{n,x}(\pi) + \frac{\beta}{\mathcal{N}} \sum_{x=1}^{\mathcal{X}_n} \sum_{n=1}^{\mathcal{N}} \mathbf{E}_{n,x}(\pi) \\ &= \max_n \sum_{k=1}^{K(\pi)} \sum_{x=1}^{W_{n,k}(\pi)} \mathbf{L}_{n,k,x}(\pi) \\ &\quad + \frac{\beta}{\mathcal{N}} \sum_{k=1}^{K(\pi)} \sum_{x=1}^{W_{n,k}(\pi)} \sum_{n=1}^{\mathcal{N}} \mathbf{E}_{n,x,k}(\pi) \\ &= \max_n \sum_{k=1}^{K(\pi)} \mathbf{L}_{n,k}(\pi) + \frac{\beta}{\mathcal{N}} \sum_{k=1}^{K(\pi)} \mathbf{E}_k(\pi). \end{aligned} \tag{9}$$

Similarly, we can represent the optimization objective of our reformulated objective as

$$J_2(\pi)=\sum_{k=1}^{K(\pi)}\frac{\max_n \mathbf{L}_{n,k}(\pi)+\alpha L_{wait,k}(\pi)}{W(\pi)}+\frac{\beta}{\mathcal{N}}\sum_{k=1}^{K(\pi)}\frac{\mathbf{E}_k(\pi)}{W(\pi)}. \tag{10}$$

By optimizing $J_2(\pi)$, we can obtain a new policy π' that satisfies $J_2(\pi') \leq J_2(\pi)$. Our goal is to establish the equivalence between the reformulated MDP and the original optimization objective, specifically determining whether optimizing $J_2(\pi)$ effectively optimizes $J_1(\pi)$. We divide the analysis into two steps. First, we establish the relationship between total resource consumption and average resource consumption. The total resource consumption is given by

$$J_3(\pi)=\sum_{k=1}^{K(\pi)}\max_n \mathbf{L}_{n,k}(\pi)+\frac{\beta}{\mathcal{N}}\sum_{k=1}^{K(\pi)}\mathbf{E}_k(\pi). \tag{11}$$

Expanding and rearranging $J_2(\pi') \leq J_2(\pi)$ yields

$$\begin{aligned} J_3(\pi')-J_3(\pi) \leq\ & J_3(\pi)\left[\frac{W(\pi')}{W(\pi)}-1\right] \\ & +\alpha W(\pi')\left[\hat{L}_{wait}(\pi)-\hat{L}_{wait}(\pi')\right], \end{aligned} \tag{12}$$

where $\hat{L}_{wait}(\pi)=\sum_{k=1}^{K(\pi)}\frac{L_{wait,k}(\pi)}{W(\pi)}$.

The conditions for the inequality in (12) to hold are clearly influenced by the average size of the task windows. We summarize these conditions in the following theorem.

Theorem 1: There exists a sufficiently small α such that the policy update for $J_2(\pi)$ also optimizes $J_3(\pi)$.

Proof: We discuss two scenarios:

1) *When $W(\pi') < W(\pi)$:* Given that $J_3(\pi)$ represents the resource consumption in terms of time and energy, it is positive. Consequently, it is always possible to find an α such that the right-hand side of (12) is less than zero.
2) *When $W(\pi') \geq W(\pi)$:* Since $\hat{L}_{wait}(\pi)$ not only depends on the policy π but also on the environmental state, directly comparing the magnitudes of $\hat{L}_{wait}(\pi)$ is impractical. In practice, we utilize a small learning rate to update the policy π, ensuring that $W(\pi')$ and $W(\pi)$ remain very similar during the updating process. Consequently, the first term on the right side of (12) will be a very small positive number. If $\hat{L}_{wait}(\pi) < \hat{L}_{wait}(\pi')$, a sufficiently small α exists such that the right side of (12) becomes negative. Conversely, if $\hat{L}_{wait}(\pi) > \hat{L}_{wait}(\pi')$, a small α can adjust the right side to be approximately zero.

Thus, we can obtain

$$J_3(\pi') \leq J_3(\pi). \tag{13}$$

Thus, we prove the equivalence between $J_2(\pi)$ and $J_3(\pi)$. Given that $\max_n \mathbf{L}_{n,k}(\pi)$ represents the longest latency for MD n to complete tasks within a task window, we can establish the relationship between the latency components of $J_1(\pi)$ and $J_3(\pi)$. Additionally, since the energy consumption terms in both $J_1(\pi)$ and $J_3(\pi)$ are identical, the following condition is satisfied

$$\max_n \sum_{k=1}^{K(\pi)}\mathbf{L}_{n,k}(\pi) \leq \sum_{k=1}^{K(\pi)}\max_n \mathbf{L}_{n,k}(\pi), \tag{14a}$$

$$J_1(\pi) \leq J_3(\pi). \tag{14b}$$

We represent the reduction in the updated optimization objective $J_3(\pi)$ with a positive constant C, then we can obtain the following inequalities

$$\begin{aligned} & J_1(\pi') \leq J_3(\pi') = J_3(\pi) - C, \\ & J_1(\pi) \leq J_3(\pi). \end{aligned} \tag{15}$$

Finally, according to (15), by optimizing $J_2(\pi)$, we can continuously reduce the upper bound of $J_1(\pi)$ and find a more optimal policy π' that satisfies $J_1(\pi') \leq J_1(\pi)$.

C. Embedded Actor-Critic Architecture

As described in Section IV-A, we present a multi-objective optimization problem with a hybrid action space. Inspired by previous works [7], [41], we propose the embedded multi-agent hybrid proximal policy optimization algorithm, namely the EMH-PPO, with its architecture illustrated in Fig. 5. EMH-PPO features an actor-critic architecture. We use θ_n, ϕ, and ω to represent the parameters of the agent network n, the embedding network, and the critic network, where $\boldsymbol{\theta}=\{\theta_1,\theta_2,\ldots,\theta_{\mathcal{N}}\}$. Compared to conventional DRL algorithms, our EMH-PPO introduces two innovative improvements to address our optimization problem:

- *Actor Network:* Our actor network consists of two parts: a global embedding network and multiple agent networks. The global embedding network first maps the input state to a shared embedding. Subsequently, each agent network processes this representation to generate the specific offloading decisions for its corresponding MD.
- *Critic Network:* For each task window, the critic network estimates the value for every inference task contained therein, based on the current state. This per-task value estimation helps to more accurately fit the value function and effectively guide the training of the actor network.

In the actor network, the state S_k is formed by concatenating the states of all MDs, which are first encoded through the global embedding network. The encoded data is then processed individually by each agent. The agent network comprises six output branches, corresponding to the action set $A_k=(\mathbf{s}_k,\mathbf{c}_k,\mathbf{m}_k,\mathbf{P}_k,\mathbf{W}_k)$. For discrete actions, the output is represented by a categorical distribution, while for continuous actions, it is represented by a Gaussian distribution with mean μ_k and standard deviation σ_k. The final action is then obtained through sampling

$$\begin{aligned} & \pi_{\theta^d,\phi}(A_k^d|S_k) \sim \text{Categorical}(\mathbf{p}_k), \\ & \pi_{\theta^c,\phi}(A_k^c|S_k) \sim \mathcal{N}(\mu_k,\sigma_k^2), \end{aligned} \tag{16}$$

where $\mathbf{p}_k$ is a probability vector, θ^d and θ^c represent the network parameters for the discrete branch and the continuous branch, respectively.

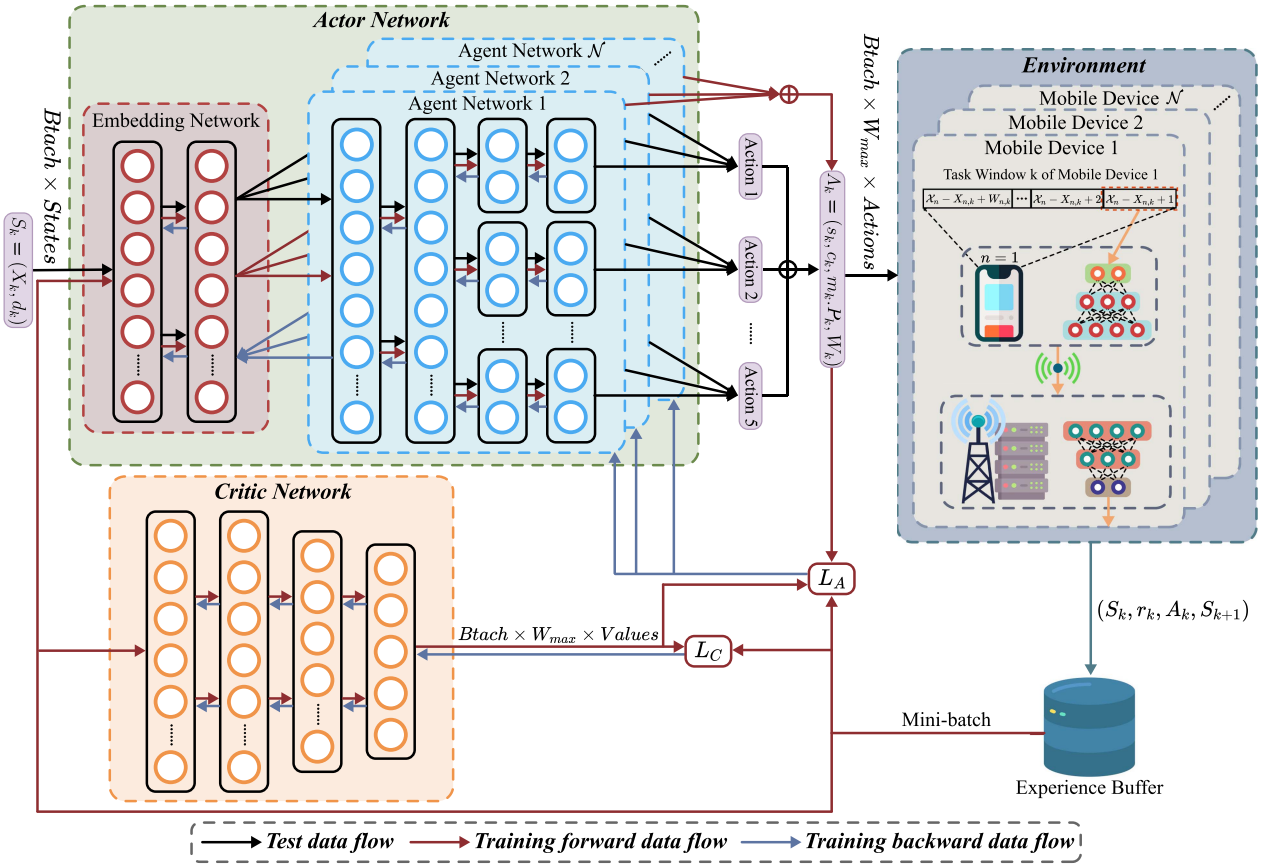

Fig. 5. Overview of the proposed EMH-PPO algorithm. The left side of the diagram showcases the architecture of our innovative actor-critic network, while the right side depicts environmental interactions and processing. Data flows during training and testing phases are highlighted in distinct colors.

For the critic network, it learns the value function $V_\omega(S_k)$ of the current environment, assisting the actor network update process. The EMH-PPO explicitly incorporates task-level decision-making through a multi-dimensional action space. Unlike conventional DRL methods that generate scalar actions per time step, our actor-critic network outputs per-task decision vectors $\{s_{n,k,x}, c_{n,k,x}, m_{n,k,x}, P_{n,k,x}\}_{x=1}^{W_{n,k}}$, which parametrizes the offloading strategy for task x in task window k. This structural innovation requires simultaneous optimization across all $W_{n,k}$ observable tasks, instead of issuing a single aggregated action for each state as in conventional DRL.

D. Policy Optimization

Then, we introduce the policy optimization process in EMH-PPO. In EMH-PPO, we use the PPO-clip form for policy optimization. The clipped loss is as follows

$$L^{\text{CLIP}}(\theta,\phi) = \hat{\mathbb{E}}_k[\min(r_k(\theta,\phi)\hat{A}_k, \text{clip}(r_k(\theta,\phi), 1-\epsilon, 1+\epsilon)\hat{A}_k)], \quad (17)$$

where $r_k(\theta,\phi) = \frac{\pi_{\theta',\phi'}(A_k|S_k)}{\pi_{\theta,\phi}(A_k|S_k)}$ is the probability ratio of the new and old policies, and the *clip* function constrains the range of $r_k(\theta,\phi)$ within $[1-\epsilon, 1+\epsilon]$. $\hat{A}_k$ is the estimate of the advantage function, used to evaluate the value difference between the new and old policies. The expressions for $\hat{A}_k$ are given by

$$\hat{A}_k = \sum_{l=0}^{K} (\gamma\lambda)^l \delta_{k+l}, \quad (18)$$

Algorithm 2: EMH-PPO Algorithm.

Initialization: the embedding network, agent networks and critic network with parameters ϕ, $\theta_{n,\forall n\in\mathcal{N}}$ and ω; initial step $i = 0$;
// training iterations
while *step:* $i < I$ **do**
  Initialization: initial state $S \leftarrow S_0 = (X_0, d_0)$ and transition dict D;
  for *episode step:* $k = 1, \ldots, K$ **do**
    $S_k \leftarrow S$;
    Sample actions $A_{n,k} \sim \pi_{\theta_n,\phi}(A_{n,k}|S_{n,k})$;
    Execute $A_k = A_{n,k,\forall n\in\mathcal{N}}$ and add tuple (S_k, A_k, S_{k+1}, R_k) into D;
    $S \leftarrow S_{k+1}$;
  // updating EMH-PPO parameters
  for *updating step:* $j = 1, \ldots, J$ **do**
    Randomly sample a mini-batch from D;
    Compute $L_A(\boldsymbol{\theta}, \phi)$ and $L_C(\omega)$ with (17) and (18);
    $\omega \leftarrow \omega + \eta\nabla_\omega L_C(\omega)$;
    $\theta_n \leftarrow \theta_n + \eta\nabla_{\theta_n} L_A(\boldsymbol{\theta}, \phi)$, $\forall n \in \{1, \ldots, \mathcal{N}\}$;
    $\phi \leftarrow \phi + \eta\nabla_\phi L_A(\boldsymbol{\theta}, \phi)$;
  $i \leftarrow i + K$.

when calculating $\hat{A}_k$, we use Generalized Advantage Estimation (GAE) [42]. The hyperparameter $\lambda \in [0, 1]$, introduced by GAE, controls the number of steps considered in the advantage estimation. Here, $\delta_k = \boldsymbol{r}_k + \gamma V(S_{k+1}) - V(S_k)$ represents temporal difference (TD) error. Therefore, the loss function for the actor network can be represented as

$$L_A(\boldsymbol{\theta}, \phi) = \sum_{n=1}^{\mathcal{N}} \frac{L^{\text{CLIP}}(\theta_n, \phi)}{W_{\max}}, \tag{19}$$

where $W_{\max}$ denotes the maximum number of tasks allocated to MDs within a task window.

For the critic network, we employ the mean squared error of TD as the loss function

$$L_C(\omega) = \frac{1}{2} \left[\boldsymbol{r}_{k+1} + \gamma V_\omega(S_{k+1}) - V_\omega(S_k)\right]^2. \tag{20}$$

As described above, we have elaborated on the policy optimization process in detail. We summarize the workflow of EMH-PPO in Algorithm 2, which outlines the key steps in the algorithm.

E. Computational Complexity Analysis

The computational complexity of the proposed EMH-PPO primarily depends on the architecture of its actor-critic networks, which consists of three components: (1) an embedding network with l^ϕ fully-connected (FC) layers, (2) $\mathcal{N}$ parallel agent networks (each agent n has l^θ FC layers), and (3) a critic network with l^ω FC layers. For a FC network, the computational complexity scales as $\mathcal{O}(\sum_{x=0}^{l-1} \vartheta_x \vartheta_{x+1})$ [43], where ϑ_x denotes the number of neurons at layer x. Let ϑ_x^ϕ, ϑ_x^θ, and ϑ_x^ω represent the neuron numbers in layer x of the embedding network, agent network, and critic network, respectively. During synchronous parameter updates, the total computational complexity is $\mathcal{O}(\sum_{x=0}^{l^\phi-1} \vartheta_x^\phi \vartheta_{x+1}^\phi + \mathcal{N} \sum_{x=0}^{l^\theta-1} \vartheta_x^\theta \vartheta_{x+1}^\theta + \sum_{x=0}^{l^\omega-1} \vartheta_x^\omega \vartheta_{x+1}^\omega)$.

V. Experiments

In this section, we present comprehensive comparative experiments to validate the feasibility and effectiveness of the proposed operator-grained partitioning method and the EMH-PPO algorithm.

A. Implementation Details

Simulation settings: The simulation parameters for the system and the radio environment are presented in Table III. We assume that MDs perform DNN inference tasks using ResNet-50 [23]. L_0 is obtained on a server equipped with an Intel Xeon Silver 4108 processor, a NVIDIA Tesla M40 24GB GPU, and 64GB of memory. We then set $L_{local} = 5L_0$. The computational load of DNNs is evaluated with a 224×224 RGB input, and the results are reported in MMACs (Million MACs). At the start of an episode, the number of tasks for MDs is initialized following a Poisson distribution $\mathcal{X}_n \sim Poisson(50)$. Meanwhile, the distance $d_{n,k}$ between MD n and the BS in k-th state is independently sampled from a uniform distribution $U[1, 100]$ meters at each state transition to emulate user mobility [44]. In the radio environment, the path loss is calculated as $(\hat{d}/d_{n,k})^h$, where $h = 3$ is the path loss exponent, $\hat{d} = 1$ m is the reference distance, and the standard deviation of shadow fading is 8 dB [45].

TABLE III
Simulation Parameters

Parameter	Value
Number of MDs ($\mathcal{N}$)	3
Number of computing MEC servers ($\mathcal{M}$)	2
Number of channels ($\mathcal{C}$)	2
Inference model	ResNet-50, GoogLeNet, MobileNetV2
The number of inference tasks for MDs ($\mathcal{X}_n$)	Poisson distribution with parameter $\lambda = 50$
Distance between MD n and base station in k-th task window ($d_{n,k}$)	Uniformly distributed in [1, 100] meters
Maximum transmission power of MDs (P_{limit})	1 W
Bandwidth of channel c (B_c)	1 MHz
Background gaussian white noise power (σ_c)	10^{-9}W
Path loss exponent (h)	3
Reference distance ($\hat{d}$)	1 m
Maximum size of task windows ($W_{\max}$)	5
Energy coefficient (ζ)	10^{-5}
energy consumption penalty factor (β)	50
waiting time penalty balance factor (α)	0.05
Reward discount factor (γ)	0.95
PPO-clip parameter (ϵ)	0.2
Learning rate	10^{-4}
Optimizer	Adam [48]
Batch size	256
Training steps	1500000

EMH-PPO's actor-critic network consists of FC layers. The embedding network comprises two hidden layers with 512 and 256 neurons, and utilizes GELU [46] activation function. Both the agent network and the critic network consist of three hidden layers and one output layer. The hidden layers contain 256, 128, and 64 neurons, and utilize the ReLU [47] activation function. In the performance testing phase, we fix the number of tasks and the distance to their expected and approximate expected values, both set to 50, to ensure a fair comparison.

Metrics and baselines: We primarily use two performance metrics: *1) Total Inference Latency*, which represents the time required for all MDs to complete their inference tasks; *2) Average Energy Consumption*, which indicates the average energy consumed by an MD to complete an inference task. We compare our method with the following four baseline schemes:

- *Local Inference Only:* The full model is executed on the MD only, without uplink transmission or server-side computation. In this case the total inference latency coincides with the MD-side computation latency.
- *H-PPO [41]:* a classic DRL method capable of handling hybrid action spaces.
- *MAHPPO [7]:* a multi-agent enhanced version of H-PPO.
- *AgileNN [49]:* A recent work on DNN collaborative inference for extremely weak devices proposes splitting the intermediate features, with partial feature computation

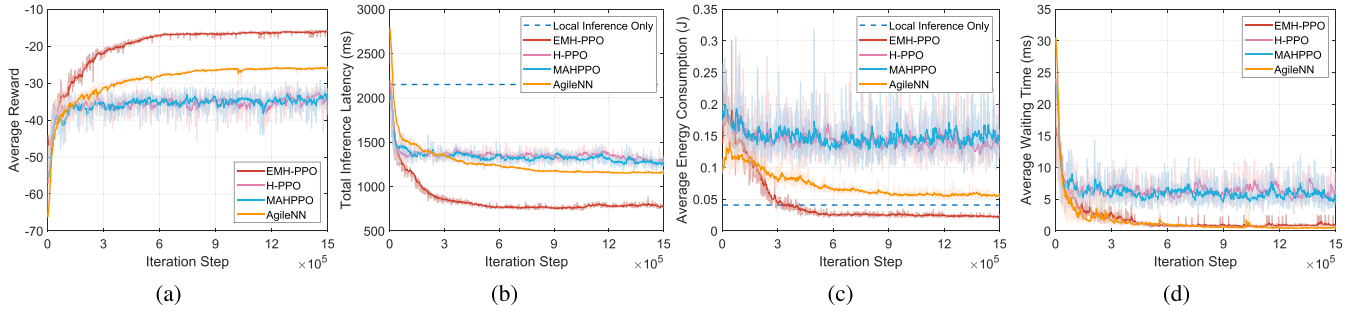

Fig. 6. Convergence performance of the proposed EMH-PPO algorithm compared with MAHPPO [7], H-PPO [41], and AgileNN [49]. The metrics evaluated include: (a) average reward; (b) total inference latency; (c) average energy consumption; (d) average waiting time.

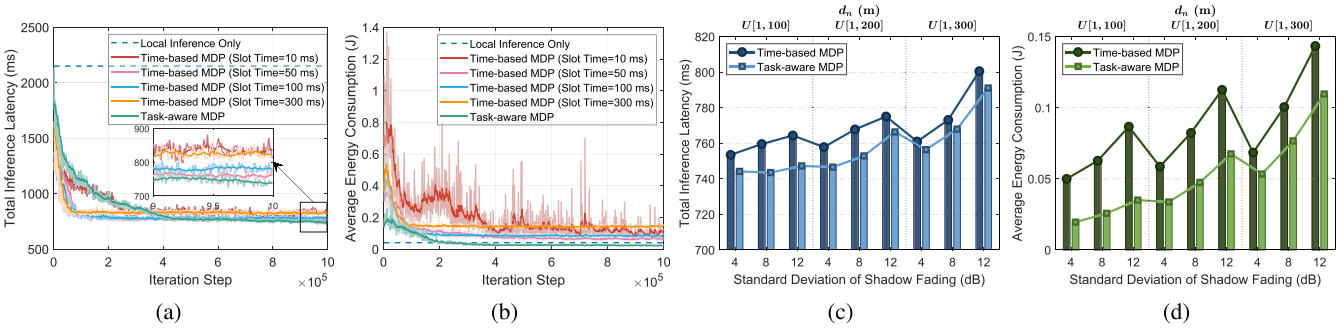

Fig. 7. Convergence performance of the EMH-PPO algorithm under the time-based MDP and the proposed task-aware MDP: (a)–(b) effect of slot time on convergence; (c)–(d) performance under varying channel conditions.

retained on a small Local NN of the embedded device, while the remaining features are processed by a larger Remote NN. In our simulation, we set the split ratio of the intermediate data to 0.5, configure the Local NN and the Remote NN with the same architecture, and disregard any impact on accuracy.

B. Convergence Analysis

Fig. 6 illustrates the convergence performance of the proposed EMH-PPO compared to baselines. As shown, our approach consistently demonstrates superior performance across various metrics. Fig. 6(a) reports the average reward under the task-aware MDP, a composite that combines latency and energy in a single objective. EMH-PPO achieves an average reward about $1.6\times$ that of AgileNN and nearly $2\times$ that of MAHPPO and H-PPO. Fig. 6(b) and (c) present comparisons of total inference latency and average energy consumption. Compared with fully local execution on MDs, EMH-PPO simultaneously reduces total latency and average energy by up to 64% and 46%. In contrast, AgileNN, MAHPPO, and H-PPO reduce total inference latency by approximately 46%, 41%, and 37%, respectively. MAHPPO and H-PPO consume more energy than local inference, and AgileNN is only comparable to it. These results indicate that the baselines are unable to jointly optimize latency and energy. Fig. 6(b) further reports the average waiting time per inference task, which measures the additional time to complete a task (i.e., the inevitable waiting time in Fig. 4). EMH-PPO attains substantially shorter waiting times than the baselines, with an additional per-task waiting time as low as 0.76 ms, which is negligible compared with the MD-side DNN inference time. AgileNN is close to EMH-PPO, whereas MAHPPO and H-PPO exhibit waiting times roughly $7.7\times$ those of EMH-PPO. Taken together, the shorter waiting time and the joint improvements in latency and energy suggest higher resource utilization and greater overall efficiency for EMH-PPO.

To isolate the effect of the problem formulation, we compare EMH-PPO under the proposed task-aware MDP with EMH-PPO under a time-based MDP, as shown in Fig. 7. To ensure a fair comparison, the output dimension of EMH-PPO under the time-based MDP is aligned with that of standard PPO. All other network, optimizer, and training settings are kept identical. Since the reward definitions differ between the two approaches (the time-based MDP does not include a waiting time penalty), we directly compare inference latency and energy consumption. As shown in Fig. 7(a) and (b), the slot time in the time-based MDP strongly affects convergence. With a well-tuned slot time (e.g., 50 ms), the total inference latency is comparable to that of our method, whereas the average energy consumption is approximately $3\times$ as high. In contrast, overly short slot times (e.g., 10 ms) yield sparse rewards, inflate the variance of advantage estimates, and misalign slot-based discounting with actual task completion time. This misalignment overweights short-term returns and destabilizes convergence. Conversely, excessively long slot times (e.g., 300 ms) reduce decision granularity, resulting in an inference latency about 11.9% higher than that of our method.

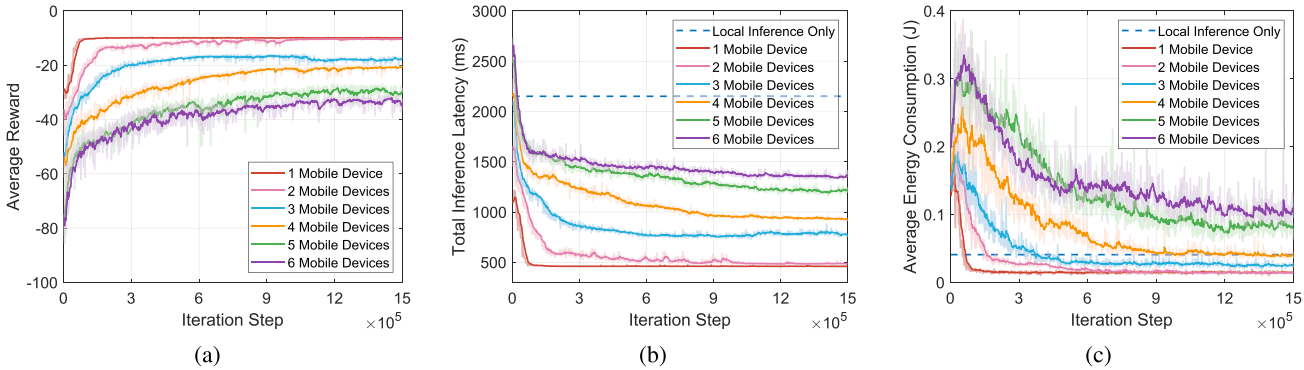

Fig. 8. Convergence performance with varying numbers of MDs: (a) average reward; (b) total inference latency; (c) average energy consumption.

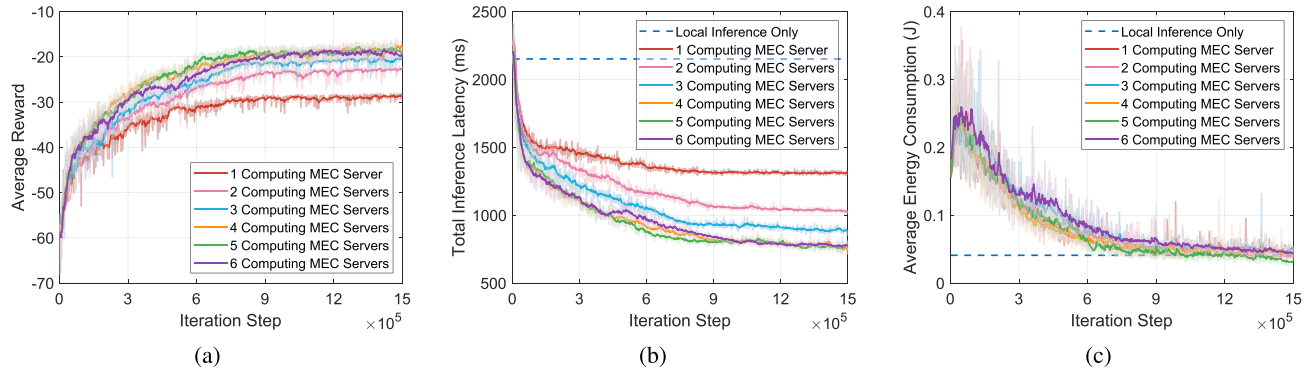

Fig. 9. Convergence performance with varying numbers of computing MEC servers: (a) average reward; (b) total inference latency; (c) average energy consumption.

The corresponding energy consumption is roughly $6.2\times$ that of our method and $3.5\times$ that of local inference. In addition, we evaluated the sensitivity of the two formulations to channel condition variability with the slot time in the time-based MDP fixed at 50 ms. The results are depicted in Fig. 7(c) and (d). We emulate varying degrees of channel condition fluctuations by adjusting the distance between the MD and the BS, as well as the standard deviation of shadow fading. Across all evaluated channel conditions, the policy learned by EMH-PPO under the task-aware MDP consistently outperforms its time-based counterpart. Our method maintains a stable advantage in inference latency. Moreover, as the standard deviation of shadow fading increases, the growth in energy consumption under our method is noticeably smaller than that observed under the time-based MDP. These results substantiate the effectiveness of the task-aware MDP reformulation with EMH-PPO in dynamic wireless environments.

C. Convergence Analysis Under Different Parameter Settings

In this subsection, we conduct comparative experiments to quantify the impact of system parameters on convergence. To ensure stable and reproducible evaluations on our simulation platform, we vary the numbers of MDs and MEC servers from 1 to 6. Figs. 8 and 9 report the convergence behavior of the proposed EMH-PPO under varied parameter settings. We also benchmark the baselines under the same configurations, with results summarized in Figs. 10, 11, and Table IV.

1) Different Numbers of MDs: Fig. 8 illustrates the impact of varying the number of MDs on EMH-PPO convergence performance. As shown in Fig. 8(a), as the number of MDs increases from 1 to 6, the intensified contention for limited radio and edge computing resources leads to a steady decline in average reward. In scenarios with surplus resources (i.e., $\mathcal{N} \leq 2$), our results show clear marginal effects, as illustrated in Fig. 8(b) and (c). EMH-PPO converges to nearly the same reward. As the number of MDs increases, contention for MEC compute and radio resources intensifies and convergence performance degrades. Even with six MDs, however, our method reduces latency by about 37.1% relative to fully local execution, whereas additional energy savings are impractical under fixed infrastructure. This reflects a resource-induced bottleneck and a trade-off between latency and energy, rather than an algorithmic limitation. Fig. 10 also compares EMH-PPO against baseline algorithms under different MD counts. Across all scenarios, although all methods exhibit an almost linear decline in reward, EMH-PPO consistently outperforms the baselines by a significant margin. Specifically, MAHPPO and H-PPO deliver about 52% lower performance than EMH-PPO, while AgileNN exhibits suboptimal resource utilization even under ample resource availability. Moreover, as

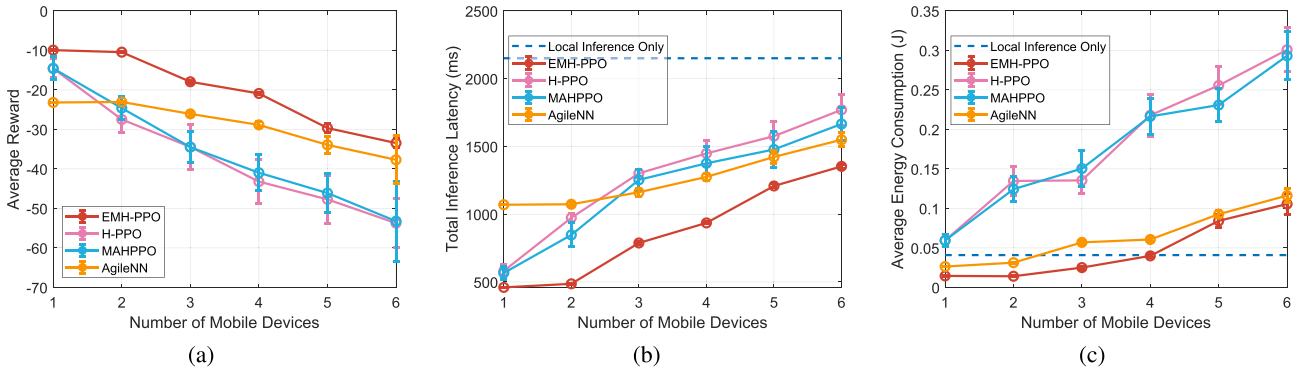

Fig. 10. Performance comparison of different algorithms with varying numbers of MDs: (a) average reward; (b) total inference latency; (c) average energy consumption.

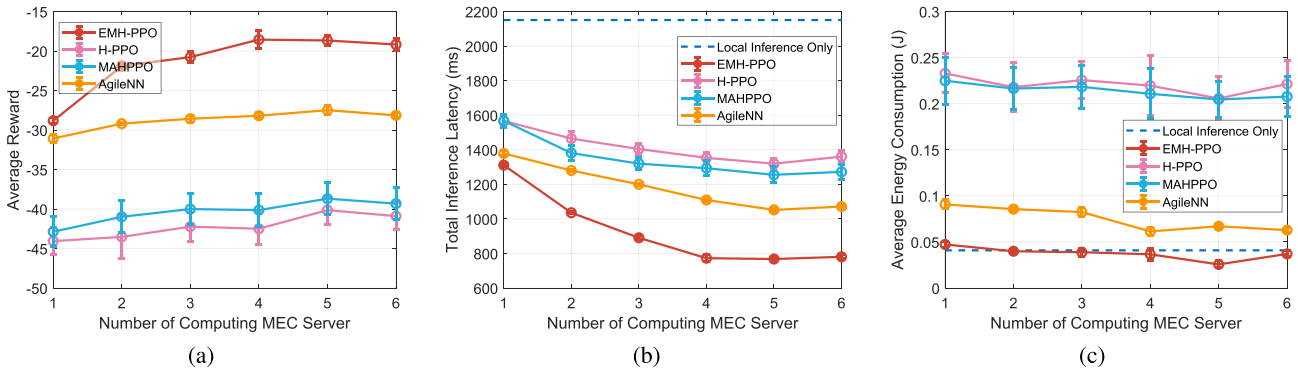

Fig. 11. Performance comparison of different algorithms with varying numbers of computing MEC servers: (a) average reward; (b) total inference latency; (c) average energy consumption.

TABLE IV
COMPARISON OF AVERAGE REWARD, INFERENCE LATENCY (MS), AND AVERAGE ENERGY CONSUMPTION (J) UNDER VARYING NUMBERS OF INFERENCE TASKS

Number of Inference Tasks	Average Reward				Total Inference Latency (ms)					Average Energy Consumption (J)				
	EMH-PPO	H-PPO	MAHPPO	AgileNN	Local Only	**EMH-PPO**	H-PPO	MAHPPO	AgileNN	Local Only	**EMH-PPO**	H-PPO	MAHPPO	AgileNN
50	**-17.94**	-34.44	-34.49	-26.07	2150.75	**787.81**	1301.59	1253.45	1161.95	0.0409	**0.0251**	0.1355	0.1504	0.0570
100	**-18.10**	-33.90	-34.61	-26.08	4301.50	**1582.88**	2534.95	2498.67	2323.49	0.0409	**0.0252**	0.1408	0.1531	0.0572
150	**-18.06**	-33.54	-34.13	-26.04	6452.25	**2379.29**	3783.07	3776.00	3489.03	0.0409	**0.0251**	0.1399	0.1482	0.0559
200	**-17.97**	-33.01	-33.84	-26.06	8603.00	**3164.64**	5046.86	5000.18	4649.34	0.0409	**0.0253**	0.1411	0.1492	0.0566

the number of MDs increases, our method incurs the smallest increase in average energy consumption.

2) Different Numbers of Computing MEC Servers: The balance between the number of computing MEC servers and MDs determines whether the system bottleneck lies in limited computational resources or communication capacity. To investigate how the relative availability of edge computing resources affects convergence performance, we fix the number of MDs at 4 and vary the number of computing MEC servers from 1 to 6. As depicted in Fig. 9, the convergence of EMH-PPO exhibits a clear pattern of diminishing returns. When the system is computation-limited ($\mathcal{M} < \mathcal{N}$), each additional MEC server yields substantial gains by relieving compute contention and enabling greater parallelism. As the number of MEC servers matches or exceeds the number of MDs ($\mathcal{M} \geq \mathcal{N}$), the primary bottleneck shifts from computation to radio resources, and performance gains plateau as throughput becomes constrained by wireless data transmission rather than server-side processing. Fig. 11 also compares EMH-PPO against baselines. With more MEC servers available for parallel computation, all methods perform better. Although all methods benefit from additional servers, EMH-PPO achieves the largest reward improvement of approximately 35.6%, whereas AgileNN, MAHPPO, and H-PPO improve by only 11.5%, 9.8%, and 8.9% respectively. Notably, only EMH-PPO reduces average energy consumption below that of fully local inference. This disparity highlights EMH-PPO's superior ability to exploit available resources to minimize contention and maximize throughput.

3) Different Numbers of DNN Inference Tasks: As summarized in Table IV, we assess the effect of varying the number of inference tasks on convergence performance. Given that our MDP formulation explicitly optimizes total inference latency

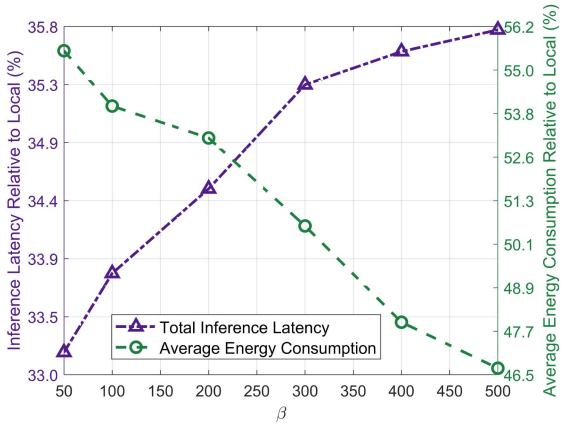

Fig. 12. Trade-offs between inference latency and energy consumption under various β settings, presented as percentages relative to the default experimental setup of local inference only.

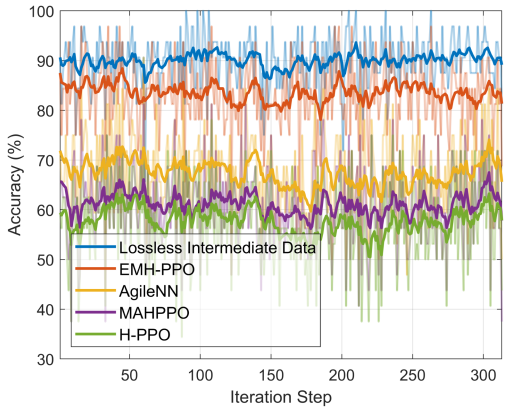

Fig. 13. Comparison of ResNet-50 inference accuracy on CIFAR-10 under varying intermediate data precision loss across different algorithms using the proposed DNN partitioning method.

and per-task energy consumption, all methods exhibit latency scaling following the same linear trend as fully local inference, though with varying scaling coefficients. Specifically, EMH-PPO exhibits the lowest per-task latency increase at approximately 15.8 ms per task, compared with about 25.0 ms per task for MAHPPO and 25.3 ms per task for H-PPO. Moreover, average energy consumption per task remains essentially constant across all task counts.

D. Trade-Offs Between Latency and Energy Consumption

The energy penalty factor β reflects the relative weight of the energy term in the reward function. By tuning β, we can realize different latency-energy trade-offs, thereby tailoring collaborative inference strategies to diverse application scenarios and user requirements. Fig. 12 illustrates the mutually restrictive trade-off between inference latency and energy consumption under different β settings. As β increases, the weight of the energy term in the optimization objective also rises, leading to decreased energy consumption at convergence. Moreover, it is observable that as β escalates, the advantages of optimizing energy consumption outweigh the latency penalties. For instance, when β is increased to 500, energy consumption is reduced to about 47% of that observed with local inference, whereas the degradation in latency performance compared to a β of 50 is merely about 2.5%. This demonstrates that our method can dynamically adjust the trade-off between latency and energy consumption based on requirements, achieving significant energy savings at the cost of a relatively small increase in latency.

E. Comparison of DNN Model Partitioning Methods

To further substantiate the advantages of our operator-grained partitioning scheme, we compare it with the traditional chain-based vertical scheme, Neurosurgeon [20], and evaluate performance under different numbers of MDs. To ensure generality, we evaluate three representative models: ResNet-50 [23], GoogLeNet [24], and the later lightweight, multi-branch MobileNetV2 [25]. ResNet-50 and GoogLeNet are seminal architectures that have shaped many subsequent DNN designs, whereas MobileNetV2 enables us to verify the scalability of the proposed partitioning method on a modern, resource-efficient network. This selection spans diverse architectural configurations for a representative assessment. Table V summarizes the average reward, total inference latency, and average energy consumption for each setting. In every user scenario and for all models, the proposed method achieves higher reward, lower latency, and lower energy than linear vertical partitioning. On average, the operator-grained scheme reduces latency by 3–4% and energy consumption by 12–15% while also improving the reward. These results confirm that finer-grained partitioning provides a larger optimization space for collaborative inference, improving adaptability and resource efficiency in dynamic MEC networks.

F. Impact of Latency Tolerance on Inference Accuracy

To evaluate the potential accuracy impact of the proposed approach, we introduce a latency-aware precision loss model. Under the default experimental settings, the probability of degrading the precision of intermediate data is set equal to the ratio between an algorithm's end-to-end inference latency and the latency of fully local inference, thereby reflecting the user's tolerance for delay. Using the proposed partitioning method, we measure the inference accuracy of ResNet-50 on the CIFAR-10 test dataset for all algorithms. As shown in Fig. 13, compared to the 90.2% accuracy achieved with lossless intermediate data, EMH-PPO incurs only a 6.35% degradation, substantially outperforming the baseline methods, which suffer accuracy losses of 22.74%, 29.06%, and 32.63%. These results underscore the robustness of EMH-PPO to intermediate data precision degradation and its ability to preserve high inference accuracy under latency-driven data loss.

G. Experimental Testbed

In this experiment, we validate the real-world feasibility of our operator-grained partitioning method and EMH-PPO. As

TABLE V
COMPARISON OF PROPOSED AND VERTICAL DNN PARTITIONING STRATEGIES IN TERMS OF AVERAGE REWARD, INFERENCE LATENCY (MS), AND ENERGY CONSUMPTION (J)

Number of MDs	DNN Model	Proposed Method			Vertical Partitioning [20]		
		Reward	Latency	Energy	Reward	Latency	Energy
1	ResNet-50	-9.93	460.06	0.0145	-10.31	473.48	0.0170
	GoogLeNet	-6.44	300.56	0.0089	-6.84	313.49	0.0106
	MobileNetV2	-5.43	250.42	0.0085	-5.83	262.38	0.0102
2	ResNet-50	-10.45	486.65	0.0141	-10.57	489.37	0.0149
	GoogLeNet	-9.11	423.29	0.0163	-9.85	438.09	0.0187
	MobileNetV2	-8.20	415.58	0.0124	-9.40	430.87	0.0144
3	ResNet-50	-17.94	787.83	0.0251	-18.63	811.05	0.0294
	GoogLeNet	-12.99	564.59	0.0237	-13.23	582.67	0.0266
	MobileNetV2	-10.77	503.39	0.0137	-11.23	520.66	0.0158
4	ResNet-50	-20.90	935.89	0.0399	-20.97	929.48	0.0424
	GoogLeNet	-14.72	671.75	0.0243	-15.40	692.18	0.0273
	MobileNetV2	-12.60	589.45	0.0155	-13.07	608.34	0.0177
5	ResNet-50	-29.65	1208.37	0.0842	-30.80	1246.86	0.0987
	GoogLeNet	-18.66	817.10	0.0427	-19.74	840.54	0.0473
	MobileNetV2	-15.84	695.83	0.0263	-16.04	717.10	0.0295
6	ResNet-50	-33.45	1352.92	0.1056	-34.72	1394.60	0.1236
	GoogLeNet	-21.05	891.49	0.0552	-21.74	929.40	0.0599
	MobileNetV2	-16.90	752.16	0.0325	-17.63	774.20	0.0361

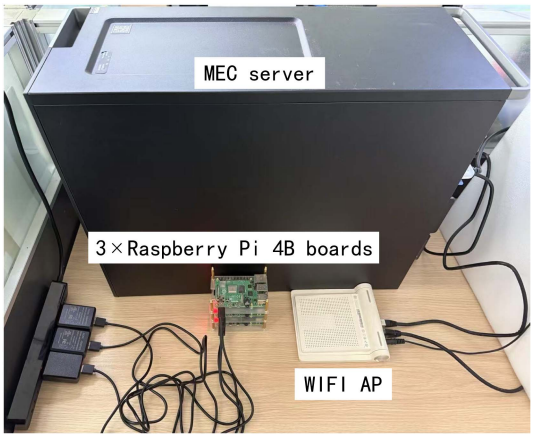

Fig. 14. Our experimental testbed consists of one MEC server and three Raspberry Pi 4B boards.

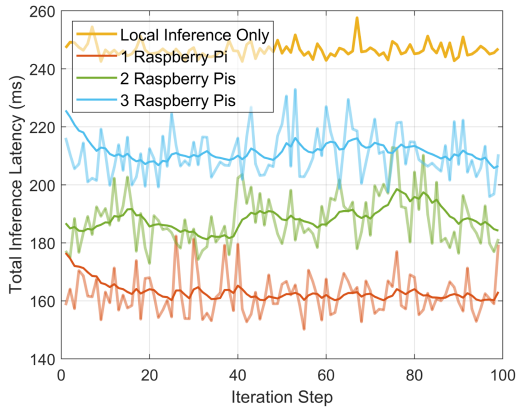

Fig. 15. End-to-end latency for 100 inference tasks on the experimental testbed comprising one MEC server and varying numbers of Raspberry Pis.

shown in Fig. 14, the testbed consists of three Raspberry Pi 4B boards and an MEC server equipped with an Intel Xeon Silver 4210R CPU and an NVIDIA GeForce RTX 3090 GPU. We partition MobileNetV2 between the Raspberry Pis and the MEC server, then measure the average inference latency for 100 concurrent collaborative tasks on one, two, and three Raspberry Pis, as shown in Fig. 15. Compared with local inference on Raspberry Pis, partitioning between one Raspberry Pi and the MEC server reduces inference latency by approximately 35%. When two Raspberry Pis are employed, latency is cut by about 28%, and with three devices, the reduction remains around 24%. These results demonstrate that our strategy can be effectively applied in real-world edge computing environments.

VI. CONCLUSION

In this paper, we propose a task-aware collaborative inference framework that enables efficient DNN collaborative inference across multi-user, multi-server edge environments. First, we introduce a DNN partitioning algorithm based on a bidirectional graph linked list, enabling operator-grained and flexible

partitioning of DAG-structured DNNs while mapping partition points to a one-dimensional sequence. Next, we reformulate the problem into a task-aware MDP that aligns decision periods with task completions, thereby insulating decision timing from variations in inference latency caused by channel condition fluctuations and stabilizing policy learning in dynamic MEC networks. Finally, we present the EMH-PPO algorithm, which incorporates a global information-sharing mechanism to optimize collaborative inference decisions. Experimental results demonstrate that, compared with local inference on MDs, our method reduces inference latency by up to 64% and energy consumption by up to 46%. In future work, we will extend our framework to large-scale MEC networks with tens to hundreds of devices and systematically evaluate scalability and robustness to non-stationary channels.

REFERENCES

[1] S. Dustdar, V. C. Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4092–4105, Apr. 2023.

[2] H. Yang, A. Alphones, Z. Xiong, D. Niyato, J. Zhao, and K. Wu, "Artificial-intelligence-enabled intelligent 6G networks," *IEEE Netw.*, vol. 34, no. 6, pp. 272–280, Nov./Dec. 2020.

[3] S. Duan et al., "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 591–624, Firstquarter 2023.

[4] P. K. Donta, B. Sedlak, I. Murturi, V. C. Pujol, and S. Dustdar, "Human-based distributed intelligence in computing continuum systems," *IEEE Internet Comput.*, vol. 29, no. 1, pp. 61–68, Mar./Apr. 2025.

[5] Z. Liu, M. Tian, M. Dong, X. Wang, C. Qiu, and C. Zhang, "MoEI: Mobility-aware edge inference based on model partition and service migration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9437–9450, Oct. 2024.

[6] M. Ding, D. Lopez-Perez, H. Claussen, and M. A. Kaafar, "On the fundamental characteristics of ultra-dense small cell networks," *IEEE Netw.*, vol. 32, no. 3, pp. 92–100, May/Jun. 2018.

[7] Z. Hao, G. Xu, Y. Luo, H. Hu, J. An, and S. Mao, "Multi-agent collaborative inference via DNN decoupling: Intermediate feature compression and edge learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 6041–6055, Oct. 2023.

[8] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[9] G. Zhou, W. Tian, R. Buyya, R. Xue, and L. Song, "Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions," *Artif. Intell. Rev.*, vol. 57, no. 5, 2024, Art. no. 124.

[10] L. Yang, X. Shen, C. Zhong, and Y. Liao, "On-demand inference acceleration for directed acyclic graph neural networks over edge-cloud collaboration," *J. Parallel Distrib. Comput.*, vol. 171, pp. 79–87, 2023.

[11] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1423–1431.

[12] Y. Li et al., "Real-time adaptive partition and resource allocation for multi-user end-cloud inference collaboration in mobile environment," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 13076–13094, Dec. 2024.

[13] T. Kim, H. Park, Y. Jin, S.-s. Lee, and S. Lee, "Partition placement and resource allocation for multiple DNN-based applications in heterogeneous IoT environments," *IEEE Internet Things J.*, vol. 10, no. 11, pp. 9836–9848, Jun. 2023.

[14] Z. Liu et al., "Hastening stream offloading of inference via multi-exit DNNs in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 535–548, Jan. 2024.

[15] H. Li, X. Li, Q. Fan, Q. He, X. Wang, and V. C. Leung, "Distributed DNN inference with fine-grained model partitioning in mobile edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9060–9074, Oct. 2024.

[16] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "REPVGG: Making VGG-style convnets great again," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 13733–13742.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 84–90.

[18] K. Simonyan, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[19] O. Elharrouss, Y. Akbari, N. Almadeed, and S. Al-Maadeed, "Backbones-review: Feature extractor networks for deep learning and deep reinforcement learning approaches in computer vision," *Comput. Sci. Rev.*, vol. 53, 2024, Art. no. 100645.

[20] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.

[21] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *Proc. IEEE 24th Int. Conf. Parallel Distrib. Syst.*, 2018, pp. 671–678.

[22] L. Wu, G. Gao, J. Yu, F. Zhou, Y. Yang, and T. Wang, "PDD: Partitioning DAG-topology DNNs for streaming tasks," *IEEE Internet Things J.*, vol. 11, no. 6, pp. 9258–9268, Mar. 2024.

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

[24] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

[26] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, Apr. 2021.

[27] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "DeepSlicing: Collaborative and adaptive CNN inference with low latency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 9, pp. 2175–2187, Sep. 2021.

[28] L. Yang, C. Zheng, X. Shen, and G. Xie, "OFPCNN: On-demand fine-grained partitioning for CNN inference acceleration in heterogeneous devices," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 12, pp. 3090–3103, Dec. 2023.

[29] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–30, 2022.

[30] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic progressive inference of neural networks over device and cloud," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–15.

[31] P. Ren, X. Qiao, Y. Huang, L. Liu, C. Pu, and S. Dustdar, "Fine-grained elastic partitioning for distributed DNN towards mobile web AR services in the 5G ERA," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3260–3274, Nov./Dec. 2022.

[32] G. Zhu et al., "Pushing ai to wireless network edge: An overview on integrated sensing, communication, and computation towards 6G," *Sci. China Inf. Sci.*, vol. 66, no. 3, 2023, Art. no. 130301.

[33] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge AI: On-demand accelerating deep neural network inference via edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 447–457, Jan. 2020.

[34] H. Qi, F. Ren, L. Wang, P. Jiang, S. Wan, and X. Deng, "Multi-compression scale DNN inference acceleration based on cloud-edge-end collaboration," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 1, pp. 1–25, 2024.

[35] G. Liu et al., "An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing," *Future Gener. Comput. Syst.*, vol. 140, pp. 422–435, 2023.

[36] N. Li, A. Iosifidis, and Q. Zhang, "Attention-based feature compression for CNN inference offloading in edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2023, pp. 967–972.

[37] S. Yao et al., "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proc. 18th Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 476–488.

[38] R. Narmeen, P. Mach, Z. Becvar, and I. Ahmad, "Joint exit selection and offloading decision for applications based on deep neural networks," *IEEE Internet Things J.*, vol. 11, no. 23, pp. 38098–38112, Dec. 2024.

[39] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1387–1395.
[40] P. Bonami, M. Kilinç, and J. Linderoth, "Algorithms and software for convex mixed integer nonlinear programs," in *Mixed Integer Nonlinear Programming*. Berlin, Germany: Springer, 2011, pp. 1–39.
[41] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *Proc. 28th Int. Joint Conf. Artif. Intell.. Int. Joint Conf. Artif. Intell. Org.*, 2019, pp. 2279–2285.
[42] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.
[43] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 855–863.
[44] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," *EURASIP J. Wireless Commun. Netw.*, vol. 2020, no. 1, 2020, Art. no. 188.
[45] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1935–1949, Mar. 2021.
[46] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.
[47] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[48] D. P. Kingma, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[49] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: Agile offloading with explainable AI," in *Proc. 28th Annu. Int. Conf. Mobile Comput. Netw.*, 2022, pp. 200–213.

Guanlei Zhang received the BEng degree in communication engineering from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2021. He is currently working toward the PhD degree in computer science with the Beijing University of Posts and Telecommunications. His research interests include edge computing and distributed intelligence.

Qiyang Zhang (Member, IEEE) received the PhD degree in computer science from the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, in 2024. He is currently an postdoctoral researcher with the School of Computer Science, Peking University. From 2022 to 2023, he was a visiting student with Distributed Systems Group, TU Wien. He has authored or coauthored papers in WWW, INFOCOM, and *IEEE Transactions on Mobile Computing*. His research focuses on mobile edge computing.

Lei Feng (Member, IEEE) received the BEng and PhD degrees in communication and information systems from the Beijing University of Posts and Telecommunications (BUPT), in 2009 and 2015, respectively. He is currently an associate professor with the School of Computer Science, BUPT. He has coauthored about 80 technical peer-reviewed papers published in academic journals and conferences. His research interests include resources management in wireless network and signal processing.

Fanqin Zhou (Member, IEEE) received the BEng degree in electronical information science and technology from the School of Electric Engineering, Beijing University of Posts and Telecommunications (BUPT), Beijing, China in 2012, and the PhD degree in information and communication engineering from the Institute of Network Technology, BUPT in 2019. He is currently a lecturer with the School of Computer Science, BUPT. His research interests include network management and optimization and mobile edge networks.

Praveen Kumar Donta (Senior Member, IEEE) received the PhD degree from the Department of CSE, IIT (ISM), Dhanbad, India, in 2021. He was a visiting PhD degree student for six months during PhD with Mobile & Cloud Laboratory, University of Tartu, Estonia. He is currently senior lecturer with the Department of Computer and Systems Sciences, Stockholm University, Sweden. He is also a former postdoctoral researcher with the Distributed Systems Group, TU Wien, Austria.

Schahram Dustdar (Fellow, IEEE) is currently full professor of computer science heading the Research Division of Distributed Systems with TU Wien, Austria. He is also a part-time ICREA professor with UPF, Barcelona. He is the co-editor-in-chief of *ACM Transactions on Internet of Things* and the editor-in-chief of *Computing* (Springer). He is an associate editor for *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*.