# Joint DNN Partition and Thread Allocation Optimization for Energy-Harvesting MEC Systems

Yizhou Shi[1], Liying Li[1], Yue Zeng[1,4], Peijin Cong[1], and Junlong Zhou[1,2,3]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China.
[2]National Mobile Communications Research Laboratory, Southeast University, Nanjing 211111, China.
[3]Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, Shanghai 201804, China.
[4]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China.
{yizhou_shi,liyingli,zengyue,cpj,jlzhou}@njust.edu.cn

*Abstract*—**Deep neural networks (DNNs) have demonstrated exceptional performance, leading to diverse applications across various mobile devices (MDs). Considering factors like portability and environmental sustainability, an increasing number of MDs are adopting energy harvesting (EH) techniques for power supply. However, the computational intensity of DNNs presents significant challenges for their deployment on these resource-constrained devices. Existing approaches often employ DNN partition or offloading to mitigate the time and energy consumption associated with running DNNs on MDs. Nonetheless, existing methods frequently fall short in accurately modeling the execution time of DNNs, and do not consider to use thread allocation for further latency and energy consumption optimization. To solve these problems, we propose a dynamic DNN partition and thread allocation method to optimize the latency and energy consumption of running DNNs on EH-enabled MDs. Specifically, we first investigate the relationship between DNN inference latency and allocated threads and establish an accurate DNN latency prediction model. Based on the prediction model, a DRL-based DNN partition (DDP) algorithm is designed to find the optimal partitions for DNNs. A thread allocation (TA) algorithm is proposed to reduce the inference latency. Experimental results from our test-bed platform demonstrate that compared to four benchmarking methods, our scheme can reduce DNN inference latency and energy consumption by up to 37.3% and 38.5%.**

***Keywords—Mobile edge computing, task offloading, energy harvesting systems, DNN partition, thread allocation.***

## I. Introduction

Deep neural networks (DNNs) have seen widespread application across numerous fields, including computer vision, natural language processing, and robotics [1]. DNNs are widely deployed in mobile devices (MDs) such as smartwatches and drones due to their remarkable performance [2]. In recent years, many MDs have started adopting energy harvesting (EH) techniques for power supply, driven by requirements such as portability and environmental sustainability [3]–[5]. EH techniques enable MDs to capture energy from renewable sources like solar and wind, enhancing their operational longevity and reducing reliance on conventional batteries [6]. However, deploying DNNs, which are computationally intensive, on resource-limited MDs remains challenging. On the one hand, MD users expect prompt responses to their requests. High response time degrades user satisfaction, especially in applications requiring real-time interaction. On the other hand, the energy consumption of running DNNs on MDs cannot be ignored since excessive energy consumption reduces device lifespan and degrades user experience.

Mobile edge computing (MEC) has emerged as a prevalent paradigm for addressing these challenges. In this paradigm, edge servers (ESs) are placed near MDs, allowing MDs to offload complex DNNs, partially or entirely, to nearby servers via wireless connections. This setup enables MDs to utilize the computing power of ESs to execute DNNs, reducing computation time and energy consumption [7], [8]. However, adopting MEC architecture introduces additional communication overhead, especially when transmitting data and intermediate results to ESs during DNN execution. Therefore, reducing latency and energy consumption is still particularly critical for MDs utilizing EH techniques for energy supply.

Some researchers are dedicated to reducing latency and energy consumption through DNN partition and resource allocation strategies. Kakolyris *et al.* [9] designed a DNN partition and offloading framework for heterogeneous edge systems based on a collaborative filtering technique. Duan *et al.* [10] studied offloading pipelines for multiple DNN inference jobs and jointly optimized the DNN partition and pipeline scheduling. In [11], the authors designed a runtime framework, which considers resource availability and co-located interference for the parallel execution of multiple DNNs. However, these approaches do not take the energy consumption as a constraint, which is crucial to EH-enabled MEC systems.

Some existing works considered the energy consumption of EH-enabled MDs in MEC, and attempted to improve the latency. Hu *et al.* [12] proposed a mobility-aware offloading

Corresponding author: Junlong Zhou. This work was supported in part by the National Natural Science Foundation of China under Grants 62172224, 62302221, 62302216, 62402226, in part by the Natural Science Foundation of Jiangsu Province under Grants BK20220138, BK20230913, BK20230912, BK20241453, in part by the Fundamental Research Funds for the Central Universities under Grants 30922010318, 30924010815, 30924010817, in part by the open research fund of National Mobile Communications Research Laboratory, Southeast University (No. 2024D07), in part by the open research fund of National Key Laboratory for Novel Software Technology, Nanjing University (No. KFKT2024B17), and in part by the open research fund of Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, under Grant ESSCKF2024-07.

and resource allocation algorithm using Lyapunov optimization and semidefinite programming. Wang *et al.* [13] explored decentralized DNN task partition and offloading control in an MEC system with multiple wireless devices powered by renewable energy sources. Zhang *et al.* [14] presented a deep reinforcement learning (DRL) algorithm to determine optimal computation offloading and server selection decisions. However, these approaches ignore the characteristics of DNN, and cannot calculate or approximate DNN execution time accurately to support DNN partition and offloading.

Real-time measurement during runtime, and static pre-recording of DNN latency are both impractical [15]. The former affects task responsiveness, while the latter ignores dynamic factors such as hardware states and runtime frameworks. An effective solution is to build a DNN inference latency prediction model based on factors influencing DNN inference. Existing methods, inspired by Neurosurgeon in [16], mostly rely on network layer types [2], [10]. In [9], the authors consider hardware types in the DNN latency prediction. However, currently, no work has explored the relationship between DNN inference latency and threads. According to the results observed experimentally (see Figs. 2-3), the DNN inference latency exhibits a marginal diminishing effect with the number of allocated threads. This indicates the DNN inference latency can be further optimized by using thread allocation, and we can estimate latency based on allocated threads. Though, Li *et al.* [17] explored to minimize inference latency via thread allocation, this method ignores the energy consumption, rendering it unsuitable for EH-enabled MDs.

To solve the above problems, this paper aims to optimize the latency and energy consumption of executing DNNs on EH-enabled MDs under the MEC framework. To this end, we first investigate the relationship between DNN inference latency, allocated threads, and model partition proportion and establish a DNN inference latency prediction model. We then propose a DRL-based DNN partition (DDP) algorithm to judiciously determine optimal partition points for DNNs. We design a thread allocation (TA) algorithm to optimize the parallel inference capability of ES and to further reduce the inference latency. Experiments conducted on a real test-bed platform show that our scheme outperforms four benchmarking methods in reducing DNN inference latency and energy consumption.

## II. Preliminary

### A. MEC Network

Fig. 1 illustrates the MEC network [18] that mainly consists of a base station (BS) and an edge server (ES). In the network coverage area, there is a set $\mathcal{N} = \{1, 2, \cdots, n, \cdots, N\}$ of MDs, where $N$ denotes the total number of MDs. Each MD is embedded with an EH module. Each MD generates tasks of various types, with each type aligning with a DNN that can be represented as a linear combination of multiple virtual layers. A virtual layer consists of multiple layers in series or parallel layers. We denote the DNN from the $n$-th MD as $\mathcal{F}_n = (D_n^{max}, S_n, G_n)$, where $D_n^{max}$ represents the maximum tolerable latency, $S_n$ represents the maximum number of virtual layers and $G_n = \{L_{n,1}, L_{n,2}, \cdots, L_{n,S_n}\}$ represents the abstraction of the DNN model. The $l$-th virtual layer $L_{n,l}$'s input data size (in bits) is denoted as $d_{n,l}$.

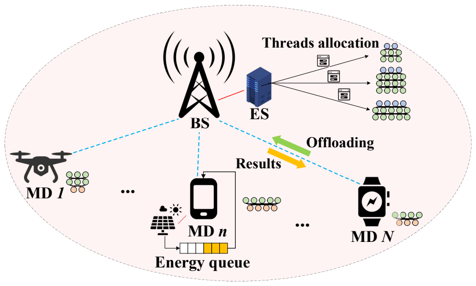


Fig. 1: MEC network framework

We divide the continuous timeline into discrete time slots, each characterized by a duration of $\tau$. The wireless channel is used for communication between the ES and MDs. In each time slot $t$, let $B(t)$ denote the wireless uplink bandwidth. Meanwhile, tasks arriving from the $n$-th MD follow a Poisson distribution, with an arrival rate of $\lambda_n(t)$. Since we focus on DNN inference in this paper, which requires much less computing resources compared to DNN training, CPUs are capable enough for DNN inference [11], [17]. Each CPU core is associated with a thread in the ES, and each DNN inference task is assigned threads. The number of available threads on the ES during time slot $t$ can be represented as $K(t)$.

In each time slot, we need to determine the partition strategy $\boldsymbol{l}(t) = (l_1(t), l_2(t), \cdots, l_N(t))$ for each DNN inference task. This strategy indicates that the execution covers from the first virtual layer to the $l_n(t)$-th virtual layer at the $n$-th MD, while the remaining layers are offloaded to the ES. Additionally, we also need to make a thread allocation strategy $\boldsymbol{k}(t) = (k_1(t), k_2(t), \cdots, k_N(t))$ to determine the number of threads allocated for each DNN to execute the inference task on the ES, where $k_n(t)$ represents the number of threads allocated for DNN from the $n$-th MD.

### B. E2E Inference Latency

As shown in Fig. 1, the end-to-end (E2E) latency of DNN inference tasks comprises local inference latency, wireless transmission latency, queuing latency, and server inference latency. Then, the E2E latency of the DNN task on $n$-th MD at the time slot $t$ can be expressed as

$$D_n^{e2e}(t) = D_n^{loc}(t) + D_n^{tran}(t) + D_n^{que}(t) + D_n^{ser}(t), \quad (1)$$

where $D_n^{loc}(t)$ denotes local computation latency, $D_n^{tran}(t)$ denotes wireless transmission latency, $D_n^{que}(t)$ denotes queuing latency, and $D_n^{ser}(t)$ denotes server inference latency[1].

***Wireless transmission latency.*** In time slot $t$, the DNN corresponding to the inference task on $n$-th MD is partitioned at the $l_n(t)$-th virtual layer, and the data size sent to the ES is denoted as $d_{n,l_n(t)+1}$. Hence, the wireless transmission latency of the $n$-th MD at time slot $t$ can be expressed as

$$D_n^{tran}(t) = \frac{d_{n,l_n(t)+1}}{B(t)}. \quad (2)$$

[1]The details of $D_n^{loc}(t)$ and $D_n^{ser}(t)$ will be presented in Section III-A.

***Queuing latency.*** The tasks that have arrived at the ES but cannot be processed immediately are cached in a queue waiting to be executed. Similar to queuing theory [19], we model the cache queue as an M/D/1 queue, and the queuing latency of the $n$-th MD at time slot $t$ is expressed as

$$D_n^{que}(t) = \begin{cases} 0, & l_n(t) = S_n, \\ \frac{\lambda_n(t)}{2(\mu_n(t))^2(1-\rho_n(t))}, & 0 \leq l_n(t) < S_n, \end{cases} \quad (3)$$

where $\mu_n(t) = 1/D_n^{ser}(t)$ denotes the service rate of the ES, and $\rho_n(t) = \lambda_n(t)/\mu_n(t)$ denotes the queue utilization rate.

## C. Energy Queue

The energy level of each MD in the current time slot depends on the remaining energy from the previous time slot, the energy consumption, and the harvested energy at the current time. Therefore, we establish an energy queue $E_n(t)$ to characterize the energy dynamics of the $n$-th MD, which is updated according to the following rule:

$$E_n(t+1) = \min\left\{E_n(t) - E_n^{con}(t) + E_n^{harv}(t), E_n^{max}\right\}, \quad (4)$$

where $E_n^{con}(t)$ is energy consumption, $E_n^{harv}(t)$ is harvested energy, $E_n^{max}$ is the maximum battery capacity of the $n$-th MD, and the initial value of $E_n(t)$ is $E_n(0) = E_n^{max}$.

***Energy consumption.*** Energy consumption of MDs primarily comes from inference computation and data transmission.

$$E_n^{con}(t) = E_n^{loc}(t) + E_n^{tran}(t). \quad (5)$$

The energy consumption during local inference $E_n^{loc}(t)$ is $E_n^{loc}(t) = P_n^{loc}(t) \cdot D_n^{loc}(t)$, where the inference power of the $n$-th MD is denoted by $P_n^{loc}(t)$. The transmission energy consumption $E_n^{tran}(t)$ is $E_n^{tran}(t) = P_n^{tran}(t) \cdot D_n^{tran}(t)$, where $P_n^{tran}(t)$ is the transmission power of the $n$-th MD.

***Energy harvesting.*** In this paper, we assume that the energy source is solar energy, so the corresponding energy intensity is represented by the solar irradiance $R(t)$ $(W/m^2)$. The efficiency of the energy conversion in the EH module of the $n$-th MD is represented by $\alpha_n(t)$. The energy harvested by the $n$-th MD during time slot $t$ is computed as follows

$$E_n^{harv}(t) = R(t) \cdot \alpha_n(t) \cdot \tau, \quad (6)$$

where $R(t)$ varies with the time of day, while $\alpha(t)$ is related to the hardware parameters of the EH module.

## III. Proposed Scheme

### A. DNN Inference Latency Prediction Model

The inference latency depends on the allocated threads, the model partition proportion, however, the association between them is difficult to be modeled directly. To explore the association between them, we conducted experiments on three models as shown in Figs. 2-3. From the results, we extract the following observations that help in modeling.

**Observation 1:** *The inference latency exhibits a marginal decrease as the number of allocated threads increases.*

As shown in Fig. 2, in all three DNNs, the inference latency shows a marginal decrease as the number of assigned threads

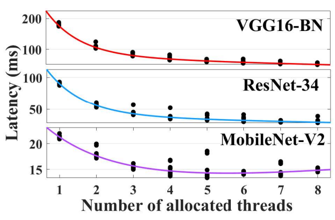


Fig. 2: DNN Inference latency with respect to the number of allocated threads.

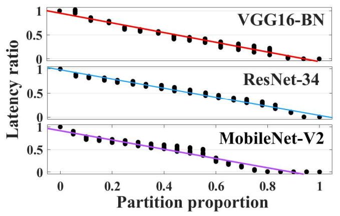


Fig. 3: Latency ratio of the partitioned DNN compared to that of the complete DNN.

increases. Based on this observation, we model the entire inference latency of the DNN under different numbers of allocated threads using a second-order exponential function:

$$D_n(k_n(t)) = c_{n,0} \cdot \exp(c_{n,1} \cdot k_n(t)) + c_{n,2} \cdot \exp(c_{n,3} \cdot k_n(t)), \quad (7)$$

where $c_{n,0}$, $c_{n,1}$, $c_{n,2}$ and $c_{n,3}$ represent the constants obtained through regression fitting.

**Observation 2:** *As the model partition proportion decreases on the ES, the ratio of the partitioned DNN inference latency to the entire DNN inference latency decreases linearly.*

As shown in Fig. 3, in all three DNNs, the latency ratio and partition proportion maintain a stable and linear decreasing relationship. This observation motivates us to model the partitioned DNN inference latency by directly multiplying the entire inference latency by the partition proportion.

***Server inference latency.*** Given the allocated threads $k_n(t)$ and partition point $l_n(t)$, the entire DNN inference latency at the ES is $D_n(k_n(t))$, and the latency ratio is $(1 - \frac{l_n(t)}{S_n})$. Thus, the server inference latency $D_n^{ser}(t)$ can be calculated as

$$D_n^{ser}(t) = D_n(k_n(t)) \cdot (1 - \frac{l_n(t)}{S_n}). \quad (8)$$

***Local inference latency.*** The inference capability of the MD remains constant, so we can measure the entire DNN inference latency at the MD $n$ in advance, which is denoted by $D_n^{loc}$. Therefore, the latency ratio on the MD side is $\frac{l_n(t)}{S_n}$, and we can calculate the local inference latency $D_n^{loc}(t)$ as

$$D_n^{loc}(t) = D_n^{loc} \cdot \frac{l_n(t)}{S_n}. \quad (9)$$

### B. Problem Formulation

The objective of this paper is to minimize latency and energy consumption. To this end, we formalize the utility function as follows to evaluate the collaborative optimization of inference latency and energy consumption of all MDs in a one-time slot:

$$\mathcal{U}(t) = V \cdot \sum_{n \in \mathcal{N}} \frac{E_n^{max} - E_n(t)}{E_n^{max}} + (1-V) \cdot \sum_{n \in \mathcal{N}} \frac{D_n^{max} - D_n^{e2e}(t)}{D_n^{max}}, \quad (10)$$

where $\frac{E_n^{max} - E_n(t)}{E_n^{max}}$ represents the normalized residual energy, and $\frac{D_n^{max} - D_n^{e2e}(t)}{D_n^{max}}$ represents the normalized saved latency.

The balance coefficient $V \in (0,1)$ balances the importance of latency and energy consumption.

***Long-term Average Utility Maximization Problem:*** Based on the utility function $\mathcal{U}(t)$, our objective is to determine partition strategy $\boldsymbol{l}(t)$ and thread allocation strategy $\boldsymbol{k}(t)$ of all MDs at each time slot, and to maximize long-term average utility. So, we formulate the long-term average utility maximization problem as follows

$$\textbf{P1:} \quad \max_{\boldsymbol{l}(t),\boldsymbol{k}(t)} \lim_{t\to\infty} \frac{1}{T}\sum_{t=0}^{T} \mathbb{E}\left[\mathcal{U}(t)\right] \tag{11}$$

$$\textit{s.t.} \quad E_n(t) > 0, \quad \forall n \in \mathcal{N}, t \in \mathcal{T}, \tag{12}$$

$$\sum_{n=1}^{N} k_n(t) = K(t), \quad \forall t \in \mathcal{T}, \tag{13}$$

$$k_n(t) \geq k_{min}, \quad \forall n \in \mathcal{N}, t \in \mathcal{T}, \tag{14}$$

$$0 \leq l_n(t) \leq S_n, \quad \forall n \in \mathcal{N}, t \in \mathcal{T}. \tag{15}$$

Eq. (12) represents that the energy of all MDs cannot be depleted at any time. Eq. (13) denotes that the sum of allocated threads for all DNN tasks in each time slot should equal the currently available threads at the ES. Eq. (14) is used to ensure fairness among tasks, preventing tasks with large resource demands from monopolizing threads and leaving no threads for tasks with smaller demands. $k_{min}$ represents the minimum threads each type of task should receive, and Eq. (15) specifies the range for selecting the partition point.

The problem constructed above is obviously an integer linear programming, and its solution space exhibits exponential growth, making it difficult to solve in polynomial time. Furthermore, our defined problem aims to maximize the long-term utility of multiple time slots, which further complicates the problem. DRL, as a method designed to solve multi-stage decision-making problems that capture long-term rewards while extracting complex rules hidden in the environment using neural networks [20], inspires us to design a DRL-based program in the next section.

## IV. DRL-based Algorithm Design

### A. Key Components of DRL

A DRL agent consists of three key components: state, action, and reward, as detailed below.

- *State:* The system state includes DNN task arrival rates and energy queues of all MDs, network bandwidth, and available threads of the ES.

$$s_t = \{\lambda_1(t), \cdots, \lambda_N(t), E_1(t), \cdots, E_N(t), B(t), K(t)\}. \tag{16}$$

- *Action:* The system action includes all MDs' partition and thread allocation strategies.

$$a_t = \{l_1(t), \cdots, l_N(t), k_1(t), \cdots, k_N(t)\}. \tag{17}$$

- *Reward:* At each time slot $t$, following the execution of a potential action $a_t$ within a specific state $s_t$, the agent will receive a corresponding reward $r_t$. The objective of DRL is to maximize the expected discounted cumulative total reward $\mathbb{E}\left(\sum_{t=0}^{T}\gamma^t r_t\right)$. Therefore, the reward function should be aligned with the objective function. Here, we define the reward $r_t$ as our utility function.

$$r_t = \mathcal{U}(t). \tag{18}$$

### B. Thread Allocation Algorithm

Although DRL can effectively deal with complex long-term decision-making problems, however, it may face long training time and inference latency when confronted with massive decision variables. Inspired by [21], designing a rule-based scheme to handle partial simple decisions and leaving the complex part to DRL can effectively accelerate training process. In the studied problem, as shown in Eq. (8), the thread allocation strategy only affects the inference latency in the offloading part of the DNN, which is much easier to handle. Therefore, we here design an expert experience-based scheme to deal with threading decisions as follows.

We propose a thread allocation (TA) algorithm that, given $\boldsymbol{l}(t)$ provided by the agent, assigns $\boldsymbol{k}(t)$ to offloaded DNN tasks in order to minimize latency. The TA algorithm adjusts the thread allocation strategy iteratively, whose flow is illustrated in **Algorithm 1**. Lines 1-4 are the initialization of the process. Lines 5-21 update the allocated threads of each DNN task iteratively. In lines 7-15, the thread allocation strategy is pre-adjusted by reducing and increasing the number of threads for all DNN tasks, resulting in $k_n^{drop}$ and $k_n^{gain}$. Based on $k_n^{drop}$ and $k_n^{gain}$, $D_n^{up}$ and $D_n^{down}$ are pre-computed, respectively. Then, for all $1 \leq n \leq N$, the increment in latency due to resource reduction, $\delta D_n^{up}$, and the decrement in latency due to resource increase, $\delta D_n^{down}$, is calculated. In lines 16-19, the DNN task with the maximum decrement in latency, $D_{n^{down}}^{down}$, and the DNN task with the minimum increment in latency, $D_{n^{up}}^{up}$, are selected to update the threads, if $D_{n^{down}}^{down} > D_{n^{up}}^{up}$. The algorithm is considered to be converged when the variance in accumulated inference latency between two successive iterations drops below a threshold $\epsilon$ (lines 20-21).

### C. DRL-based DNN Partition Algorithm

As the input to the TA algorithm includes the partition strategy $\boldsymbol{l}(t)$, there is a need for a DRL method capable of generating explicit policy to determine $\boldsymbol{l}(t)$ at each time slot. Furthermore, the various DNN types among MDs and the complex structure of existing DNNs with dozens of virtual layers create a huge action space. Hence, we employ Deep Deterministic Policy Gradient (DDPG) [20] to train the action policy for determining $\boldsymbol{l}(t)$, which is a classic DRL method capable of learning explicit policies in environments with continuous state and large-scale discrete action.

DDPG consists of four neural networks: 1) actor network $\pi\left(s_t|\theta_\pi\right)$; 2) critic network $Q\left(s_t, a_t|\theta_Q\right)$; 3) target actor network $\pi^{'}\left(s_t|\theta_\pi^T\right)$; 4) target critic network $Q^{'}\left(s_t, a_t|\theta_Q^T\right)$. The actor network generates policy, i.e., actions $a_t$ based on state $s_t$ at each time slot. Since we need to generate partition strategy for at most $N$ MDs, the actor network's number of output

**Algorithm 1:** Thread Allocation Algorithm

```
Input: Partition strategy $\boldsymbol{l}(t)$, available threads $K(t)$
Output: Thread allocation strategy $\boldsymbol{k}(t)$
1  Initialize $k_n(t) \leftarrow k_{min}, 1 \leq n \leq N$;
2  if $N \cdot k_{min} < K(t)$ then
3      Randomly allocate the remaining threads;
4  Calculate $D_n^{e2e,*}(t), 1 \leq n \leq N$ according to Eq. (1);
5  while $True$ do
6      $D_n^{e2e}(t) \leftarrow D_n^{e2e,*}(t), 1 \leq n \leq N$;
7      for $n = 1$ to $N$ do
8          $k_n^{drop} \leftarrow k_n(t) - 1, k_n^{gain} \leftarrow k_n(t) + 1$;
9          if $k_n^{drop} \leq k_{min} - 1$ then
10             Calculate $D_n^{up}$ by using $k_n^{drop}$;
11         else
12             $D_n^{up}(t) \leftarrow +\infty$;
13         Calculate $D_n^{down}$ by using $k_n^{gain}$;
14         $\delta D_n^{up} \leftarrow D_n^{up} - D_n^{e2e,*}(t)$;
15         $\delta D_n^{down} \leftarrow D_n^{e2e,*}(t) - D_n^{down}$;
16     $n^{up} \leftarrow \arg\min_n \delta D_n^{up}, n^{down} \leftarrow \arg\max_n \delta D_n^{down}$;
17     if $D_{n^{down}}^{down} > D_{n^{up}}^{up}$ then
18         $k_{n^{down}}(t) \leftarrow k_n^{gain}, k_{n^{up}}(t) \leftarrow k_n^{drop}$;
19         Compute $D_n^{e2e,*}(t), 1 \leq n \leq N$ using updated $\boldsymbol{k}(t)$;
20     if $\sum_{n=1}^{N} D_n^{e2e}(t) - \sum_{n=1}^{N} D_n^{e2e,*}(t) \leq \epsilon$ then
21         break;
```

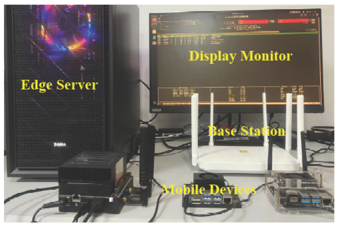
**Algorithm 2:** DRL-based DNN Partition Algorithm

```
1  Initialize networks $\theta_\pi$, $\theta_Q$, $\theta_\pi^T$, $\theta_Q^T$, and replay buffer $\mathcal{B}$;
2  for each episode do
3      Reset state $s_0$;
4      for each time slot $t$ do
5          Generate action $a_t$;
6          Map $a_t$ to $\boldsymbol{l}(t)$ by using Eq. (19);
7          $\boldsymbol{k}(t) \leftarrow$ call Algorithm 1;
8          Compute $r_t$ and update state $s_{t+1}$;
9          Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$;
10         if Replay buffer $\mathcal{B}$ is full then
11             Sample minibatch of transitions;
12             Compute target value $y_t$;
13             Update $\theta_Q$ with loss $L(\theta_Q)$;
14             Update $\theta_\pi$ with policy gradient $\nabla_{\theta_\pi} J$;
15             Softly update target networks $\theta_\pi^T$, $\theta_Q^T$;
```

neurons is set to $N$, with each neuron corresponding to a partition strategy for an MD. To map the continuous output values to specific partition points, we use the tanh to map the output values to the interval between $(-1, 1)$. Then, according to the following formula, we convert the numerical values into specific partition points:

$$l_n(t) = \left\lfloor \frac{\tanh \mathcal{P}_n(t) + 1}{2} \cdot S_n \right\rfloor, \quad 1 \leq n \leq N, \forall t \qquad (19)$$

where $\mathcal{P}_n(t)$ represents the value of the $n$-th output neuron. The critic network is tasked with approximating the true action-value function and producing the action value $Q(s_t, a_t)$ for a given state-action pair $(s_t, a_t)$.



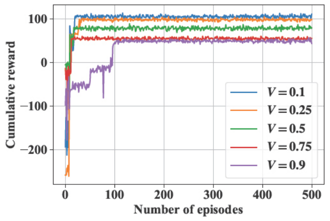


Fig. 4: Real test-bed platform. Fig. 5: Cumulative reward.

Following the design of DDPG, we propose the DRL-based DNN partition (DDP) algorithm. **Algorithm 2** shows the detail of the training process of our DDP algorithm.

## V. Evaluation

**Experimental Setup.** We conducted experiments based on our real test-bed platform (as shown in Fig. 4), whose detailed hardware configurations are shown in Table I. We use Python 3.9 and Pytorch v1.9.0 as our deep learning running framework. Three DNNs are adopted as tasks for offloading in our experiments: VGG-16BN, ResNet-34, and MobileNet-V2. The task arrival rate $\lambda_n(t)$ of MD $n$ at time slot $t$ is randomly set from the interval $[20, 30]$. The duration of a time slot $\tau$ is set to 30s. The maximum tolerable latency $D_n^{max}$ is set to 90% of the latency when the DNN task is executed on MD $n$. According to [11], the wireless network bandwidth per time slot is randomly set from $[10, 20]$Mbps to simulate Wi-Fi and is set from $[20, 40]$Mbps to simulate 5G. We use the *wondershaper* to control the bandwidth. The number of threads that can be allocated on the ES is set from $[12, 20]$. According to the monitoring data from *jtop*, the average data transmission power is 6.5W, and the average DNN inference power is 8.6W. As for EH data, we adopt the dataset provided by the NREL Solar Radiation Research Laboratory [22]. The sampling rate of solar irradiance $R(t)$ is one minute.

TABLE I: Real test-bed platform configuration.

| Type | Device | Comp |
|---|---|---|
| MD | Jetson Orin NX | Arm® Cortex®-A78AE CPU |
| | Jetson Orin Nano | Arm® Cortex®-A57 CPU |
| | Jetson AGX Xavier | NVIDIA Carmel Arm® CPU |
| ES | Workstation | Intel® i7-12700KF CPU |
| BS | Redmi Router AX6000 | MediaTek Filogic 830 |

**Convergence Evaluation.** We evaluate the convergence of the proposed DDP algorithm's training process. Fig. 5 shows the cumulative reward under different values of $V$ during the training process.The figure illustrates that the reward initially rises with the number of training episodes and subsequently stabilizes. This indicates that the proposed DDP algorithm can converge. Moreover, the results also show that a smaller $V$ can achieve a higher cumulative reward. We set the value of $V$ to 0.5 in the following experiments.

**Performance Comparison.** We compare our scheme with four other benchmarking schemes, which are described as

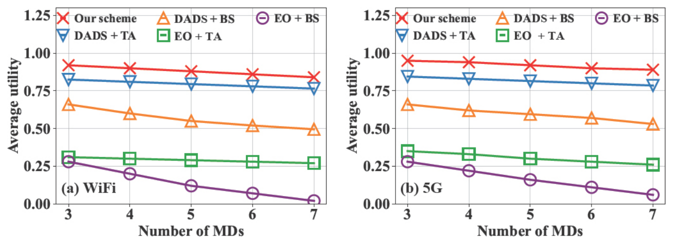


Fig. 6: Average utility versus number of MDs.

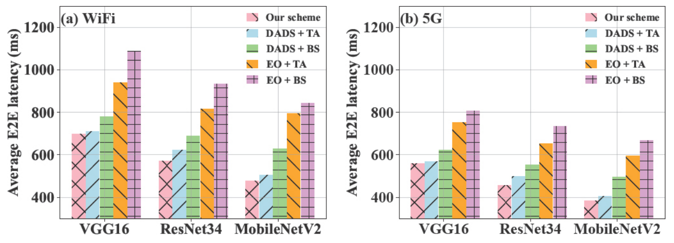


Fig. 7: Average E2E latency versus different types of DNNs.

follows: 1) *Dynamic Adaptive DNN Surgery (DADS) + Binary Search (BS):* DADS is a state-of-the-art DNN partition method for DAG-based DNNs [17]. It aims to minimize the inference latency by the max-flow min-cut theorem. Then, it uses binary search to find the minimum number of threads. 2) *DADS + TA:* This scheme first partitions DNNs through DADS, then allocates threads on the ES using our proposed TA algorithm. 3) *Edge Only (EO) + TA:* EO + TA method offloads DNNs to the ES without partition, and uses the TA algorithm for thread allocation. 4) *EO + BS:* EO + BS method offloads entire DNNs to the ES, and allocates threads using the BS algorithm.

We first compare our approach with four benchmarking schemes in terms of average utility values when the number of MDs changes. Fig 6(a) and 6(b) give the average utility under the scenarios of WiFi and 5G networks, respectively. It can be seen from the figures that our approach can achieve the highest average utility values under different numbers of MDs in both WiFi and 5G networks. It can improve the average utility by up to 11.5% compared with four benchmarking methods.

To validate the universality of the proposed method, we then compare our scheme with the other four schemes under different types of DNNs. Specifically, we set the number of MDs to 3 and deployed VGG-16, ResNet-34, and MobileNet-V2 on separate MDs. We then run the MEC system for 120 time slots with different schemes and network settings and finally calculate the average E2E latency for each type of DNN under WiFi and 5G networks. It can be seen from Fig. 7(a) and Fig. 7(b) that our scheme achieves the minimum average E2E latency for various types of DNN inference tasks. Especially for VGG-16, our scheme reduces the average E2E latency by up to 1.6%, 10.5%, 25.8%, and 37.3% compared with DADS + TA, DADS + BS, EO + TA, and EO + BS, respectively.

We also evaluate the scalability of our scheme by comparing it with four benchmarking approaches. We adjust the task

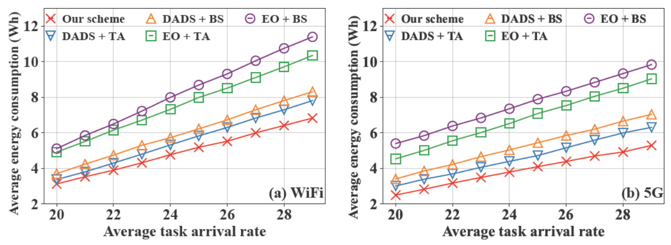


Fig. 8: Average energy consumption versus task arrival rate.

arrival rate for each MD in every time slot, and calculate the average energy consumption of MDs after the MEC system operates for 120 time slots. The experiment results are given in Figs. 8(a)-8(b). Our scheme can achieve the lowest average energy consumption as the task arrival rate increases. Specifically, our scheme reduces average energy consumption by up to 12.7%, 17.9%, 27.7%, and 38.5% compared with DADS + TA, DADS + BS, EO + TA, and EO + BS, respectively.

**Prediction Accuracy Evaluation.** The efficiency of our scheme depends on how accurate the prediction model is for DNN inference latency, so we further evaluate the predicted latency of the three DNN models (VGG-16BN, ResNet-34, MobileNet-V2) mentioned in this paper in terms of Root Mean Squared Error (RMSE) and compare with the *Neurosurgeon* [16], which is a classic method for the prediction of DNN inference latency. The results shown in Table II demonstrate that the prediction errors of the three DNN models are within an acceptable range, with RMSE being close to 0. Additionally, our scheme achieves up to 8.31× less RMSE compared to *Neurosurgeon*. This is because our prediction model takes into account the runtime conditions of DNN inference, rather than predicting the inference latency of individual DNN layers in isolation, resulting in more accurate results.

TABLE II: Accuracy of the inference latency prediction model.

| | VGG-16BN | ResNet-34 | MobileNet-V2 |
|---|---|---|---|
| Our scheme | 3.7449 | 2.8021 | 0.8009 |
| *Neurosurgeon* [16] | 26.9913 | 24.9800 | 6.6516 |

## VI. Conclusions

This paper investigates the DNN partition and thread allocation strategy to jointly reduce DNN inference latency and energy consumption of EH-enabled MDs in the MEC network. We first model an MEC system and formulate a long-term utility maximization problem. To solve this problem, we establish a prediction model to evaluate DNN inference latency based on the partition proportion of the DNN model and thread allocation pattern. Subsequently, we design a TA algorithm tailored for the edge server, enabling the allocation of threads to offloaded DNNs. Additionally, we present a DDP algorithm to dynamically determine DNN partition points. The results of experiments conducted on our real test-bed platform reveal that our proposed scheme effectively realizes collaborative optimization of DNN inference latency and energy consumption in the long run.

## REFERENCES

[1] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[2] X. Chen, M. Li, H. Zhong, Y. Ma, and C.-H. Hsu, "DNNOff: offloading DNN-based intelligent IoT applications in mobile edge computing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2820–2829, 2022.

[3] J. Zhou, K. Cao, X. Zhou, M. Chen, T. Wei, and S. Hu, "Throughput-conscious energy allocation and reliability-aware task assignment for renewable powered in-situ server systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 516–529, 2021.

[4] X. Hou, J. Zhou, M. Zhao, L. Li, P. Cong, Z. Wu, and S. Hu, "ILRM: Imitation learning based resource management for integrated CPU-GPU edge systems with renewable energy sources," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024, doi: 10.1109/TCAD.2024.3513892.

[5] P. Cong, J. Zhou, L. Li, K. Cao, T. Wei, and K. Li, "A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–44, 2020.

[6] S. Islam, J. Deng, S. Zhou, C. Pan, C. Ding, and M. Xie, "Enabling fast deep learning on tiny energy-harvesting IoT devices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 921–926.

[7] J. Li, W. Liang, M. Huang, and X. Jia, "Reliability-aware network service provisioning in mobile edge-cloud networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1545–1558, 2020.

[8] F. Jiang, K. Wang, L. Dong, C. Pan, W. Xu, and K. Yang, "Deep-learning-based joint resource scheduling algorithms for hybrid MEC networks," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6252–6265, 2020.

[9] A. K. Kakolyris, M. Katsaragakis, D. Masouros, and D. Soudris, "RoaD-RuNNer: Collaborative DNN partitioning and offloading on heterogeneous edge systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[10] Y. Duan and J. Wu, "Optimizing job offloading schedule for collaborative DNN inference," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 3436–3451, 2023.

[11] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, "HiTDL: High-throughput deep learning inference at the hybrid mobile edge," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4499–4514, 2022.

[12] H. Hu, Q. Wang, R. Q. Hu, and H. Zhu, "Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 541–17 556, 2021.

[13] F. Wang, S. Cai, and V. K. Lau, "Decentralized DNN task partitioning and offloading control in MEC systems with energy harvesting devices," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, pp. 173–188, 2022.

[14] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9303–9317, 2020.

[15] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2019.

[16] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[17] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 3017–3030, 2023.

[18] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in DNN-task enabled mobile edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2435–2445, 2021.

[19] G. F. Newell, *Applications of queueing theory*, 2013, vol. 4.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016, pp. 1–14.

[21] Y. Su, W. Fan, L. Gao, L. Qiao, Y. Liu, and F. Wu, "Joint DNN partition and resource allocation optimization for energy-constrained hierarchical edge-cloud systems," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 3, pp. 3930–3944, 2022.

[22] "MIDC: NREL Solar Radiation Research Laboratory (BMS)." [Online]. Available: https://midcdmz.nrel.gov/