

HTBM: A Hybrid-Transformer-Based Method with Double Deep Q-Learning and Reptile Meta-Learning for Dynamic MEC Task Offloading

Nouhaila Moussammi^{1*}, Mohamed El Ghmary²

Department of Computer Sciences-Faculty of Sciences, Mohammed V University in Rabat, Morocco¹

ENSAM, Mohammed V University in Rabat, Morocco²

Abstract—Mobile edge computing (MEC) enables computation-intensive applications to be processed close to end users. In dynamic MEC environments, task-offloading decisions must account for changing communication conditions, heterogeneous computing resources, and dependencies among fine-grained subtasks. Conventional reinforcement-learning approaches may require substantial interaction data and may suffer from unstable value estimation. This study proposes a meta-reinforcement-learning framework, termed HTBM (Hybrid-Transformer-Based Method), for latency-aware task offloading in dynamic MEC systems. HTBM combines a hybrid-transformer sequential representation mechanism, Double Deep Q-Network (DDQN) for discrete offloading decisions, and the first-order Reptile algorithm for rapid adaptation across related MEC environments. The system model follows a fine-grained directed acyclic graph (DAG) representation in which applications are decomposed into dependent subtasks that can be executed locally or at an edge server. Experimental evaluation is conducted under a representative MEC simulation scenario. Across transmission-rate experiments, a latency reduction of approximately 2.00% was observed under the evaluated run configuration compared with GASTO, an existing graph-based meta-reinforcement-learning offloading method. Additional experiments examine rapid adaptation, robustness across DAG topologies, generalization to an unseen topology, inference time, and an ablation study isolating the contribution of each HTBM component.

Keywords—Mobile edge computing; task offloading; Double Deep Q-Network (DDQN); hybrid transformer; deep reinforcement learning; meta-reinforcement learning; reptile; Directed Acyclic Graph (DAG); latency minimization

I. INTRODUCTION

Mobile edge computing (MEC) brings computation and storage resources closer to end users and is therefore suitable for applications with stringent latency requirements [1]. In such systems, task offloading determines which computations should be executed locally and which should be transferred to edge resources. The decision becomes particularly difficult when the application is composed of dependent subtasks and when network and computing conditions vary over time [2].

A fine-grained task can be represented as a directed acyclic graph (DAG), where vertices denote subtasks and directed edges describe execution dependencies [3]. An offloading decision must consequently account not only for the computational

requirements of individual subtasks, but also for the availability of predecessor outputs, communication delays, and the queue state of edge servers, a challenge also addressed through hierarchical DRL formulations for dependent-task offloading [4].

Deep reinforcement learning (DRL) provides a natural framework for sequential offloading decisions because the agent can learn from interactions with the MEC environment [5]. Nevertheless, conventional DRL approaches may require many interactions before learning a useful policy. Moreover, a state at a particular decision instant may not contain sufficient information to characterize the evolution of previous tasks and network conditions.

Transformer-based sequence models offer an alternative representation mechanism because self-attention can capture relationships among distant observations [6]. In particular, Gated Transformer-XL (hybrid-transformer) was designed to improve the stability of Transformer-based reinforcement learning [7], and Transformer-enhanced DRL has since been surveyed extensively across communication-network applications, including computation offloading [8]. In the proposed framework, hybrid-transformer is used to encode sequences of MEC experiences into compact representations.

The second component of the proposed framework is Double Deep Q-Network (DDQN) [9]. Unlike a conventional DQN [10], DDQN separates action selection from action evaluation when constructing the target value, reducing the tendency of Q-learning to overestimate action values. This property is useful for discrete offloading decisions where several execution choices may have similar immediate costs.

Finally, Reptile is employed as a first-order meta-learning mechanism [11]. Instead of learning a completely independent model for every MEC configuration, the meta-learner searches for an initialization that can be adapted rapidly to a new but related environment, as previously explored for offloading by meta-reinforcement-learning frameworks such as MRLCO [12] and GASTO [13].

The main contributions of this work are summarized as follows:

- A hybrid-transformer-based sequential representation mechanism is integrated with a Double DQN decision module for latency-oriented MEC task offloading.

*Corresponding author.

- A first-order Reptile meta-learning procedure is incorporated so that the learned model can rapidly adapt to previously unseen MEC tasks and environmental configurations.
- The original fine-grained DAG task structure is retained, allowing dependencies among subtasks to influence the offloading and execution process.
- A comprehensive experimental evaluation is conducted to study rapid adaptation, convergence, latency under different transmission rates, DAG-topology generalization, and inference time.
- An ablation study is conducted to isolate the individual contribution of the hybrid-transformer representation, the Double DQN target rule, and the Reptile meta-initialization to the overall latency performance of HTBM.

The remainder of this study is organized as follows. Section II discusses related work. Section III presents the MEC system model and problem formulation. Section IV formulates the offloading decision as a Markov decision process. Section V describes the proposed HTBM meta-reinforcement-learning framework. Section VI presents the experimental methodology and numerical analysis, including an ablation study of the three HTBM components. Section VII discusses the results, and Section VIII concludes the study.

II. RELATED WORK

A. Heuristic and Optimization-Based Offloading

Task offloading has traditionally been formulated as an optimization problem involving computation latency, communication latency, energy consumption, or combinations of these objectives [1]. For instance, joint radio-and-computation resource allocation has been formulated for multi-user multi-task offloading scenarios in order to jointly minimize latency and energy consumption under shared MEC resources [15]. Multi-task offloading has been studied under explicit energy and delay constraints [16], and intelligent task-offloading optimization strategies have more recently been explored for MEC systems [17]. Greedy strategies and graph-based scheduling approaches can provide efficient solutions for specific configurations, but their decision rules generally depend on manually designed assumptions. Their ability to adapt to rapidly changing MEC environments can therefore be limited, a limitation echoed by recent surveys of the offloading literature [2].

B. Deep Reinforcement Learning for MEC Offloading

DRL methods learn offloading policies by interacting with the environment [10]. DQN-based methods are particularly suitable when the action space is discrete, whereas actor-critic methods such as PPO are often used for more general continuous or stochastic decision problems. DQN variants have been investigated for MEC offloading [3], but value overestimation and the difficulty of representing long interaction histories remain important issues.

DDQN addresses the overestimation problem of standard DQN by using the online network to select the next action and the target network to evaluate it [9]. Several recent works apply

DDQN specifically to MEC offloading and resource allocation, reporting improved convergence and reduced latency relative to conventional DQN and fixed-policy baselines [5]. Graph-based DRL formulations have further combined DDQN with graph attention networks to schedule DAG tasks over dynamic environments [14], digital-twin-assisted DDQN has been used for mobility-aware dependent task offloading [18], multi-agent DRL has been explored for combinatorial client-master offloading decisions [19], and ensemble-learning-augmented DRL has been applied to reduce offloading latency under dynamic edge conditions [20]. Beyond terrestrial MEC, DDQN and its dueling/prioritized variants have also been applied to satellite edge computing [21], LEO satellite networks with hybrid action spaces [22], UAV-assisted aerial edge computing with digital-twin support [23], and Industrial IoT offloading [24], illustrating the versatility of the DDQN family across heterogeneous edge deployments. DAG-based task scheduling under DRL has similarly been studied in vehicular cloud settings [25], [26] and general real-time systems [27], further motivating the fine-grained DAG representation retained in this work. In this work, DDQN is combined with hybrid-transformer rather than using a conventional feed-forward Q-network.

C. Transformer-Based Reinforcement Learning

Transformer architectures use self-attention to model relationships between elements in a sequence [6]. Transformer-XL extends the effective context through segment-level recurrence, while hybrid-transformer introduces modifications intended to improve Transformer stability in reinforcement-learning settings [7]. These properties make hybrid-transformer attractive for MEC environments where recent observations alone may not fully characterize the current decision context. Closely related recent work has combined Transformer-based encoders with Proximal Policy Optimization (PPO) for dependent-task offloading, including a Transformer-PPO framework for edge computing [28] and a GTrXL-PPO offloading technique for IoT edge environments [29]. Transformer-based representations have also been applied to distributed task offloading and resource management in cloud-edge networks [30], to distributed DRL scheduling of IoT applications [31], and to Transformer-PPO combinations for general IoT decision-making [32]. The present work differs from these by pairing the Transformer-based representation with a Double DQN decision module rather than PPO, and by explicitly incorporating a Reptile meta-learning outer loop.

D. Meta-Reinforcement Learning for MEC

Meta-reinforcement learning aims to learn knowledge shared among a distribution of tasks so that adaptation to a new task requires only a small amount of additional interaction. Reptile is a first-order meta-learning method that repeatedly adapts model parameters to sampled tasks and moves the meta-parameters toward the resulting task-specific parameters [11]. This principle has previously been applied to MEC task offloading: MRLCO combines a first-order meta-learning procedure with a sequence-to-sequence Long short-term memory (LSTM) policy over DAG-structured applications [12], and GASTO integrates a graph neural network with a sequence-to-sequence network under a meta-learning outer loop to improve

adaptation speed and robustness across dynamic MEC environments [13]. The proposed framework retains the meta-learning principle used in these works while replacing their sequence-to-sequence policy network with a hybrid-transformer representation and a DDQN decision module.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. MEC Network Model

As shown in Fig. 1, we consider a fine-grained task-offloading scenario involving multiple user equipment (UEs) and multiple heterogeneous MEC servers. Each UE may generate one or more applications, and each application is decomposed into a set of temporally dependent subtasks. The execution of these subtasks can be performed locally at the UE or offloaded to an associated MEC server. The environment therefore contains heterogeneous computational tasks generated by different user devices, while the communication and computing conditions may vary across environments. These characteristics make the offloading decision dependent not only on the computational requirements of the current subtask, but also on the execution status of related subtasks and the available MEC resources.

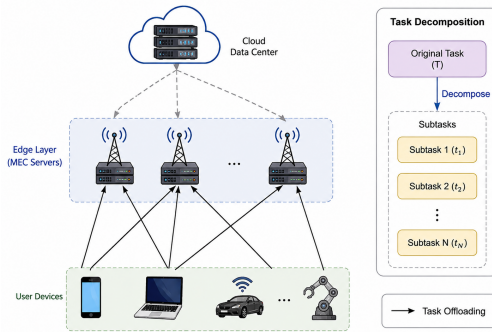


Fig. 1. Overall MEC system model.

The set of user equipment is denoted by:

$$\mathcal{U} = \{u_1, u_2, \dots, u_U\}, \quad (1)$$

and the set of MEC servers is denoted by:

$$\mathcal{K} = \{e_1, e_2, \dots, e_K\}. \quad (2)$$

Each UE can execute application tasks locally or offload computations to an edge server. For an application, only one MEC server is considered as its associated edge execution destination. The communication and computing conditions may vary between environments, making the optimal offloading decisions environment-dependent.

B. DAG-Based Fine-Grained Applications

Each application is modeled as a directed acyclic graph (DAG):

$$G = (V, E), \quad (3)$$

where, V denotes the set of subtasks and E represents dependency relationships between subtasks.

A subtask is represented by:

$$T_{i,j} = \{S_{i,j}, C_{i,j}, O_{i,j}\}, \quad (4)$$

where, $S_{i,j}$ is the input-data size, $C_{i,j}$ is the required number of CPU cycles, and $O_{i,j}$ is the output-data size.

The edge $(v_a, v_b) \in E$ indicates that subtask v_b cannot begin until the required output of predecessor subtask v_a becomes available. Therefore, the execution order is constrained by the topology of the DAG.

C. Local Execution

For a subtask executed by UE i , its computation latency is:

$$t_{i,j}^{\text{ue}} = \frac{C_{i,j}}{f_i^{\text{ue}}}, \quad (5)$$

where, f_i^{ue} is the CPU frequency of the UE. Local execution does not require uplink transmission to the MEC layer, but its completion time is directly determined by the computational demand of the subtask and the available UE processing capability.

D. Edge Execution

When a subtask is offloaded to MEC server k , its execution time includes transmission, queueing, and computation components. It can be expressed generally as:

$$t_{i,j}^k = \frac{S_{i,j}}{R_{i,k}^{\text{ul}}} + t_{i,j}^{k,\text{queue}} + \frac{C_{i,j}}{f_k}, \quad (6)$$

where, $R_{i,k}^{\text{ul}}$ is the uplink transmission rate, $t_{i,j}^{k,\text{queue}}$ is the queueing delay at server k , and f_k is the computing frequency of the MEC server.

The queueing term depends on previously scheduled computations at the corresponding server. Furthermore, a subtask can only begin after its predecessor subtasks have completed and their required data has become available.

E. Subtask Completion and DAG Latency

Let $ST(v)$ and $FT(v)$ denote the start and finish times of subtask v . The start time is constrained by predecessor completion times:

$$ST(v) = \max_{p \in \text{Pred}(v)} FT(p), \quad (7)$$

with the appropriate communication delay included whenever predecessor and successor subtasks are executed at different locations.

The finish time is:

$$FT(v) = ST(v) + t(v). \quad (8)$$

Let F denote the set of terminal nodes of the DAG. The completion latency of an application is therefore determined by the latest terminal-subtask completion:

$$T_{\text{DAG}} = \max_{v \in F} FT(v). \quad (9)$$

F. Optimization Objective

The objective of this study is to minimize the overall task completion latency. For a collection of applications, the optimization objective can be written as:

$$\min_{\mathbf{a}} \sum_{i=1}^U T_{\text{DAG},i}, \quad (10)$$

subject to DAG precedence constraints, execution-location constraints, and MEC resource constraints.

The resulting problem is combinatorial and dynamically affected by communication and computing conditions. We therefore formulate it as a Markov decision process.

IV. MARKOV DECISION PROCESS

A. State Space

At decision step t , the environment provides an observation containing the relevant task and MEC information. The instantaneous state is denoted by:

$$s_t \in \mathcal{S}. \quad (11)$$

A sequence of observations:

$$\mathcal{H}_t = (s_{t-L+1}, \dots, s_t) \quad (12)$$

is retained so that the decision model can exploit temporal information. This history is particularly relevant in the considered fine-grained setting because the execution of a subtask may depend on the completion of preceding subtasks and on the evolving communication and computing conditions. The sequential representation therefore provides the learning model with contextual information beyond the instantaneous state.

B. Action Space

The action determines the execution location of the currently considered subtask. For a simplified local/edge decision,

$$a_t \in \mathcal{A}, \quad (13)$$

where, an action corresponds to local execution or an eligible MEC execution choice.

For multiple MEC servers, the action space can contain one action for each valid execution destination.

C. Reward

Because the optimization objective is latency minimization, the reward is directly defined from the latency contribution of the current decision:

$$r_t = -\Delta T_t, \quad (14)$$

where, ΔT_t denotes the latency contribution associated with the current decision. No additional reward normalization or shaping is applied. This formulation keeps the reinforcement-learning objective directly aligned with the latency-minimization objective defined in Section III.

V. PROPOSED HTBM FRAMEWORK

A. Framework Overview

The proposed HTBM architecture consists of three main components. The overall interaction and information flow among these components is illustrated in Fig. 2. The figure shows how the task-related observations are transformed into a sequential representation before being provided to the DDQN decision mechanism, while Reptile operates at the meta-learning level to obtain an initialization that can be efficiently adapted to new MEC environments. The individual contribution of each of these three components to the overall latency performance is examined separately in the ablation study reported in Section VI-H.

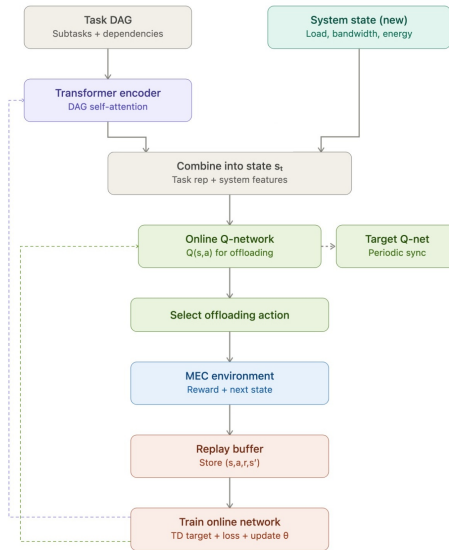


Fig. 2. Workflow of the proposed HTBM meta-reinforcement-learning framework.

- The hybrid-transformer module encodes a sequence of MEC observations and extracts a representation containing information from the interaction history.
- DDQN maps the encoded representation to action-value estimates and selects the offloading action.
- Reptile learns an initialization across a distribution of MEC tasks, enabling rapid adaptation to unseen configurations.

B. Hybrid-Transformer Sequential Representation

The input to HTBM is a sequence of state representations obtained from agent–environment interactions. The hybrid-transformer module uses self-attention to capture relationships among observations at different positions in the sequence and produces a compact representation for the decision process.

Let $h_{i,n}$ denote the hidden representation corresponding to the n -th element of the sequence for task i . The sequence representation is obtained by aggregating the normalized hidden states:

$$c_i = \frac{1}{N} \sum_{n=1}^N \text{LayerNorm}(h_{i,n}). \quad (15)$$

The resulting representation summarizes the interaction history and is supplied to the DDQN decision module. The hybrid-transformer therefore acts as the representation-learning component, while DDQN performs the discrete offloading decision and Reptile provides the meta-initialization.

C. Double Deep Q-Network

Let $Q_\theta(s, a)$ denote the online Q-network and $Q_{\theta^-}(s, a)$ denote the target Q-network. DDQN selects the next action using the online network:

$$a^* = \arg \max_{a'} Q_\theta(s_{t+1}, a'). \quad (16)$$

The target value is then evaluated using the target network:

$$y_t = r_t + \gamma Q_{\theta^-}(s_{t+1}, \arg \max_{a'} Q_\theta(s_{t+1}, a')), \quad (17)$$

for nonterminal transitions. For terminal transitions,

$$y_t = r_t. \quad (18)$$

The DDQN loss is

$$L(\theta) = \frac{1}{B} \sum_{b=1}^B (y_b - Q_\theta(s_b, a_b))^2, \quad (19)$$

where, B is the mini-batch size.

The target network is periodically synchronized with the online network:

$$\theta^- \leftarrow \theta. \quad (20)$$

D. Experience Replay

To improve the stability and data efficiency of DDQN training, the interactions generated during the execution of each MEC task are stored in an experience replay buffer. Each transition contains the current state, the selected offloading action, the resulting latency-based reward, the next state, and a terminal indicator. The replay memory is defined as:

$$\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1}, d_t)\}, \quad (21)$$

where, $d_t \in \{0, 1\}$ indicates whether the transition terminates an episode. For the sequence-based decision process, the states stored in the transitions are associated with their corresponding observation histories so that the hybrid-transformer can reconstruct the temporal context required for the decision.

Once the replay buffer contains a sufficient number of transitions, a mini-batch of B samples is randomly drawn for each learning update. The sampling operation can be expressed as:

$$\mathcal{B} = \{(s_b, a_b, r_b, s'_b, d_b)\}_{b=1}^B, \quad (s_b, a_b, r_b, s'_b, d_b) \sim \mathcal{D}. \quad (22)$$

Random sampling reduces the correlation between consecutive experiences and allows previously collected transitions to be reused during several learning updates. The resulting mini-batch is subsequently used to calculate the DDQN targets and update the task-specific parameters.

For a terminal transition, the target contains only the immediate reward, whereas a nonterminal transition also incorporates the estimated value of the next state. Accordingly, the target used by the DDQN update can be written in the unified form:

$$y_b = r_b + \gamma(1 - d_b)Q_{\theta^-}(s'_b, \arg \max_{a'} Q_\theta(s'_b, a')). \quad (23)$$

This formulation allows the same replay mechanism to handle both intermediate subtask decisions and the final decision that completes a DAG application.

E. Reptile Meta-Learning

The MEC environment is viewed as a distribution of related tasks

$$\mathcal{T} \sim \rho(\mathcal{T}). \quad (24)$$

Each task represents a particular MEC configuration and task/DAG condition under which the HTBM model must learn an effective latency-oriented offloading policy. The objective of meta-learning is therefore to obtain a common initialization that can be efficiently adapted to an individual task.

Let θ denote the shared meta-parameters of the hybrid-transformer and DDQN components. For each sampled task $\mathcal{T}_j$, the task-specific parameters are initialized from the current meta-parameters:

$$\theta_j^0 \leftarrow \theta. \quad (25)$$

The initialized model interacts with $\mathcal{T}_j$, stores the resulting transitions in its replay buffer, and performs a number of DDQN inner-loop updates. After adaptation, the resulting parameters are denoted by θ'_j .

The task-specific adaptation can be represented by the generic update:

$$\theta_j^{k+1} = \theta_j^k - \alpha_{\text{in}} \nabla_{\theta_j^k} L_{\mathcal{T}_j}(\theta_j^k), \quad (26)$$

where, α_{in} is the inner-loop learning rate and $L_{\mathcal{T}_j}$ is the DDQN loss evaluated using experiences collected from task $\mathcal{T}_j$. After K inner-loop updates,

$$\theta'_j = \theta_j^K. \quad (27)$$

The Reptile outer loop then moves the shared initialization toward the task-adapted parameters. For a meta-batch containing N sampled tasks, the update is:

$$\theta \leftarrow \theta + \epsilon \left(\frac{1}{N} \sum_{j=1}^N \theta'_j - \theta \right), \quad (28)$$

where, ϵ denotes the outer-loop learning rate. Equivalently, the meta-objective can be expressed as minimizing the average task-specific loss after adaptation:

$$\min_{\theta} \mathcal{L}_{\text{meta}}(\theta) = \mathbb{E}_{\mathcal{T} \sim \rho(\mathcal{T})} [L_{\mathcal{T}}(\theta'_\mathcal{T})]. \quad (29)$$

Unlike second-order meta-learning methods, Reptile does not require explicit computation of second-order derivatives. Instead, the meta-parameters are progressively moved toward parameter configurations that perform well after task-specific adaptation. The learned meta-parameters are subsequently used as the initialization for new MEC tasks.

F. Meta-Training

Meta-training repeatedly exposes HTBM to a collection of related MEC tasks. At the beginning of each meta-iteration, a meta-batch of N tasks is sampled from $\rho(\mathcal{T})$. For every sampled task, the current meta-parameters are copied to a task-specific model, which then collects interaction trajectories and performs DDQN adaptation using its replay buffer.

For task $\mathcal{T}_j$, the inner-loop learning process can be summarized as:

$$\theta_j^{k+1} = \theta_j^k - \alpha_{\text{in}} \nabla_{\theta_j^k} L_{\mathcal{T}_j}(\theta_j^k), \quad k = 0, \dots, K-1, \quad (30)$$

where, K is the number of inner-loop updates. The DDQN loss is evaluated using mini-batches sampled from the task-specific replay buffer. After the K updates, the adapted parameters are stored as $\theta'_j = \theta_j^K$.

The outer loop aggregates the adapted parameters from all tasks and updates the common initialization according to:

$$\theta^{\text{new}} = \theta + \epsilon \left(\frac{1}{N} \sum_{j=1}^N \theta'_j - \theta \right). \quad (31)$$

The updated parameters θ^{new} are then used to initialize the next meta-iteration. Repeating this process enables the model to capture information shared across different MEC environments while retaining the ability to specialize to an individual task through a small number of DDQN updates.

The complete meta-training procedure is summarized in Algorithm 1.

Algorithm 1 Meta-Training of HTBM

```

1: Initialize meta-parameters  $\theta$ 
2: for  $i = 1, \dots, M$  do
3:   Sample  $N$  tasks from  $\rho(\mathcal{T})$ 
4:   for  $j = 1, \dots, N$  do
5:     Initialize task parameters  $\theta_j \leftarrow \theta$ 
6:     Initialize the replay buffer  $\mathcal{D}_j$ 
7:     Interact with task  $\mathcal{T}_j$  and collect trajectories
8:     Encode trajectory sequences using hybrid-transformer
9:     Store transitions in  $\mathcal{D}_j$ 
10:    for  $k = 1, \dots, K$  do
11:      Sample a mini-batch from  $\mathcal{D}_j$ 
12:      Select next actions using the online Q-network
13:      Evaluate selected actions using the target Q-network
14:      Compute the DDQN loss and update  $\theta_j$ 
15:    end for
16:     $\theta'_j \leftarrow \theta_j$ 
17:  end for
18:  Update  $\theta$  using the Reptile interpolation rule
19: end for

```

G. Meta-Testing

During meta-testing, the outer-loop parameters remain fixed. A previously unseen MEC task is initialized with the learned meta-parameters and adapted using a limited number of DDQN updates.

Algorithm 2 Meta-Testing of HTBM

```

1: Select an unseen MEC task  $\mathcal{T}_{\text{test}}$ 
2: Initialize  $\theta_{\text{test}} \leftarrow \theta$ 
3: Collect a small amount of task-specific experience
4: Encode the observed sequence using hybrid-transformer
5: for  $k = 1, \dots, L$  do
6:   Sample replay transitions
7:   Compute DDQN targets
8:   Update  $\theta_{\text{test}}$ 
9: end for
10: Evaluate the adapted model on  $\mathcal{T}_{\text{test}}$ 

```

VI. EXPERIMENTAL METHODOLOGY AND NUMERICAL ANALYSIS

A. Simulation Settings

The principal simulation parameters used in the experiments are summarized in Table I. The experimental environ-

TABLE I. EXPERIMENTAL PARAMETER SETTINGS USED IN THE HTBM EVALUATION

Parameter Definition	Value
Number of Meta-Training Tasks (Count)	100
Number of Meta-Testing Tasks (Count)	100
Communication-to-Computation Ratio (CCR)	$0.3 \sim 0.5$
DAG Fat (Fat)	$\{0.1, 0.3, 0.4, 0.5, 0.7\}$
DAG Density (Density)	$\{0.4, 0.5, 0.6, 0.7, 0.8\}$
Inner-loop learning rate	0.1
Outer-loop learning rate	0.01

ment contains four MEC servers and twenty user devices. Each application is represented by a randomly generated DAG, parameterized by the density and fat values listed in Table I. The DAG generation methodology and the ranges of density, fat, and communication-to-computation ratio values follow the same random-DAG generation approach and comparable parameter ranges adopted by MRLCO [12] and GASTO [13], so as to provide a consistent and directly comparable evaluation basis across all baseline methods. The UE CPU frequency is set to 1×10^9 cycles/s and the edge-server CPU frequency is set to 8×10^9 cycles/s.

The scenario further considers applications composed of 20 fine-grained subtasks each, evaluated over a test set of 100 DAGs/applications. Transmission-rate experiments use uplink and downlink rates of 4, 7, and 10 Mbps.

The DDQN settings are selected from commonly used DDQN configurations. In particular, a learning rate of 10^{-4} , discount factor $\gamma = 0.99$, mini-batch size of 32, replay-buffer capacity of 100,000 transitions, initial exploration rate $\epsilon_{\text{start}} = 1.0$, minimum exploration rate $\epsilon_{\text{min}} = 0.01$, and a target-network synchronization interval of 1,000 training steps are adopted. These values are consistent with an established Double DQN implementation configuration, which uses a 10^{-4} learning rate, $\gamma = 0.99$, batch size 32, replay capacity 100,000, and a 1,000-step target-network synchronization interval [33]. The relatively conservative learning rate is selected because HTBM jointly trains the hybrid-transformer representation and the DDQN value function, while the discount factor of 0.99 allows the agent to account for future latency costs. These settings are used consistently throughout the HTBM training and evaluation experiments. These values were not obtained through an independent sensitivity analysis or through tuning specific to the MEC offloading setting; rather, they were adopted because they constitute an established and widely reproduced Double DQN configuration that is also broadly consistent with the hyperparameter ranges reported in other DDQN-based MEC offloading studies [5], providing a stable and reproducible starting point in line with common practice in the offloading literature. Similarly, the Reptile inner- and outer-loop learning rates in Table I follow the default configuration proposed in the original Reptile formulation [11] rather than a setting tuned specifically for the MEC environments considered here. A systematic sensitivity analysis of the DDQN and Reptile hyperparameters is identified as a valuable direction for future work, complementing the multi-seed statistical validation discussed in Section VII. The implementation of HTBM and the MEC simulation environment used to produce the reported results is available from the corresponding author upon reasonable request.

B. Evaluation Metrics

The evaluation focuses on the following metrics:

- Task completion latency: total latency required to complete the DAG applications.
- Rapid adaptation: number of iterations or interactions required for the adapted model to reach a stable performance level.
- Generalization: performance under DAG topologies and task configurations not used during meta-training.
- Inference time: time required to obtain offloading decisions from the trained neural model.
- Latency reduction: relative reduction in average task-completion latency compared with a selected baseline, computed as

$$R_{\text{lat}} = \frac{\bar{T}_{\text{base}} - \bar{T}_{\text{HTBM}}}{\bar{T}_{\text{base}}} \times 100\%. \quad (32)$$

C. Rapid Adaptation Under Different Transmission Rates

This experiment studies the behavior of HTBM under different communication rates. The evaluation uses uplink and downlink rates of 4, 7, and 10 Mbps. The latency trajectories are shown in Fig. 3. HTBM progressively reduces latency as adaptation proceeds, with the final values reaching approximately 965 ms, 735 ms, and 595 ms for 4, 7, and 10 Mbps, respectively. The results indicate that the meta-learned initialization enables the model to improve its decision quality during the adaptation process.

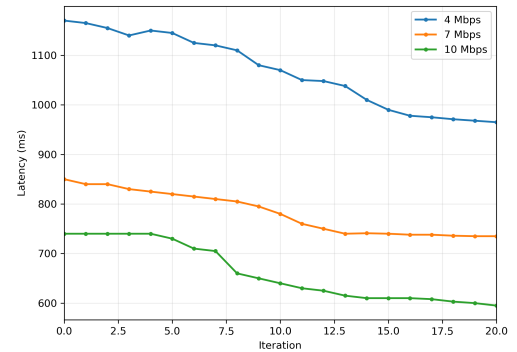


Fig. 3. Rapid adaptation performance of HTBM at different transmission rates.

D. Comparison with Alternative Offloading Algorithms

The latency comparison under different transmission rates is presented in Fig. 4 and summarized in Table II. The evaluated methods are Greedy, MRLCO [12], GASTO [13], T-PPO [28], and the proposed HTBM. At 4 Mbps, HTBM obtains a latency of 965 ms, compared with 980 ms for GASTO. At 7 Mbps, the corresponding values are 735 ms and 750 ms, while at 10 Mbps they are 595 ms and 610 ms. Thus, HTBM provides reductions of 1.53%, 2.00%, and 2.46% relative to GASTO at 4, 7, and 10 Mbps, respectively. Therefore, an average task-completion latency reduction of

TABLE II. TASK-COMPLETION LATENCY UNDER DIFFERENT
TRANSMISSION RATES

Method	4 Mbps	7 Mbps	10 Mbps
Greedy	1090	830	625
MRLCO	995	760	615
GASTO	980	750	610
T-PPO	1010	790	650
HTBM	965	735	595

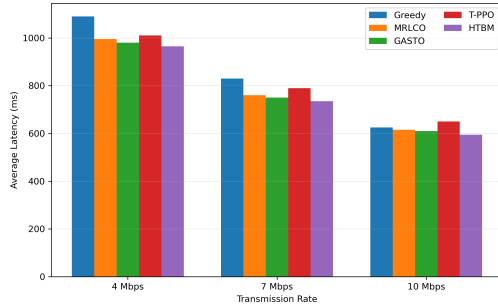


Fig. 4. Task-completion latency comparison under different transmission rates.

approximately 2.00% was observed for HTBM compared with GASTO across the evaluated transmission rates under this run configuration.

This increasing relative improvement with transmission rate can be explained qualitatively in terms of the latency composition described in Section III-D. At lower transmission rates (e.g., 4 Mbps), the transmission term $S_{i,j}/R_{i,k}^{ul}$ in Eq. (6) dominates the total execution time of any offloaded subtask, largely independently of which offloading policy is used. Because this term is shared across all evaluated methods, it acts as a common additive latency floor that compresses the relative (percentage) differences between methods even when the underlying decision-quality gap in absolute terms (ms) is comparable across transmission rates. As the transmission rate increases and the communication term diminishes, total latency becomes increasingly dominated by queueing and decision-dependent components, allowing differences in offloading-decision quality, and in particular the contribution of the hybrid-transformer's sequential representation, to be reflected more directly in the relative-improvement metric of Eq. (32). We note that this interpretation should ultimately be examined jointly with the multi-seed statistical validation identified in Section VII, since the observed trend spans a comparatively narrow range (1.53%–2.46%) that should be confirmed to lie outside run-to-run variance.

E. Generalization Across DAG Topologies

To evaluate robustness to structural variations in the application graph, three density–width configurations are considered. The first uses density 0.7 and width 0.4, the second uses density 0.4 and width 0.6, and the third uses density 0.1 and width 0.5. The corresponding latency trajectories are presented in Fig. 5, 6, and 7, respectively. These figures illustrate the evolution of the latency values under the three different DAG structures and allow the behavior of HTBM

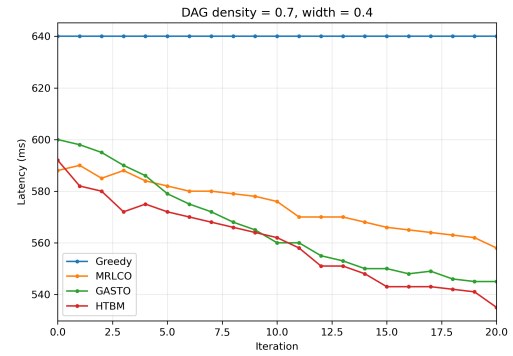


Fig. 5. Latency evolution for DAG density 0.7 and width 0.4.

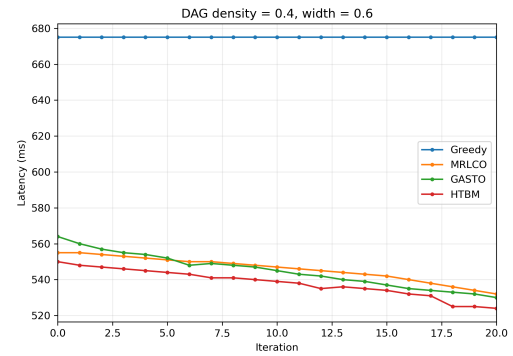


Fig. 6. Latency evolution for DAG density 0.4 and width 0.6.

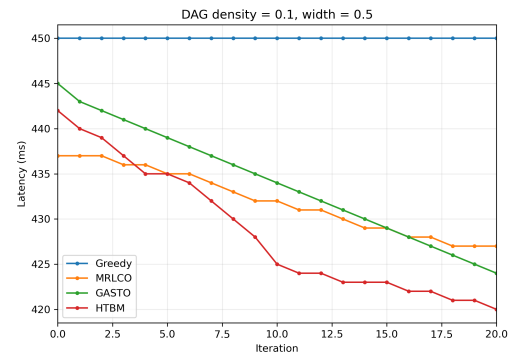


Fig. 7. Latency evolution for DAG density 0.1 and width 0.5.

to be compared with the alternative methods throughout the evaluated iterations.

For the first configuration, HTBM obtains an average latency of approximately 559.90 ms, compared with 566.14 ms for GASTO, corresponding to a reduction of approximately 1.10%. For the second configuration, HTBM achieves approximately 538.05 ms compared with 544.95 ms for GASTO, corresponding to a 1.27% reduction. For the third configuration, the corresponding values are approximately 428.57 ms and 434.05 ms, giving a 1.26% reduction. These results show that the proposed method maintains lower average latency across different DAG structures and remains competitive when the structural characteristics of the application graph change.

F. Generalization to an Unseen DAG Topology

To further examine generalization, HTBM is evaluated on an unseen topology with density 0.4 and width 0.9, as shown in Fig. 8. The average latency of HTBM is approximately 566.48 ms, whereas GASTO and MRLCO obtain approximately 580.33 ms and 593.38 ms, respectively. Thus, HTBM reduces average latency by approximately 2.39% relative to GASTO and 4.53% relative to MRLCO on the unseen topology.

The unseen-topology result is important because it evaluates whether the learned representation remains useful when the structural properties of the DAG differ from the other evaluated configurations. The continued latency advantage suggests that the combination of sequence representation and meta-learning provides useful transfer across related DAG environments. Table III summarizes the average latency achieved by all evaluated methods across the three DAG-topology configurations and the unseen topology.

TABLE III. AVERAGE LATENCY ACROSS DAG-TOPOLOGY EXPERIMENTS

Method	(0.7,0.4)	(0.4,0.6)	(0.1,0.5)	Unseen
Greedy	640.00	675.00	450.00	720.00
MRLCO	574.57	545.90	431.95	593.38
GASTO	566.14	544.95	434.05	580.33
HTBM	559.90	538.05	428.57	566.48

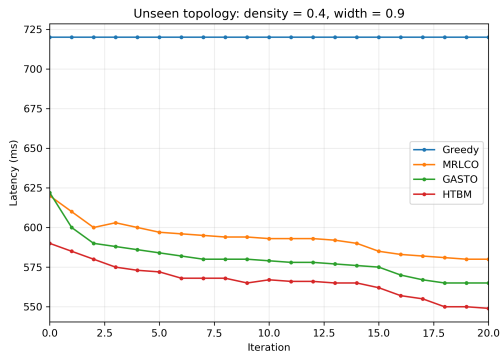


Fig. 8. Generalization performance on an unseen DAG topology with density 0.4 and width 0.9.

G. Comparison of Network Architectures

The inference-time experiment compares the hybrid-transformer encoder with recurrent alternatives as the number of tasks increases. The measured inference times are shown in Fig. 9. The hybrid-transformer requires approximately 0.26–0.32 ms, while LSTM requires approximately 3.30–3.40 ms and GRU approximately 2.05–2.20 ms over the evaluated task counts. These results indicate that the transformer-based representation used by HTBM can provide a low inference-time overhead in the evaluated setting.

H. Ablation Study

To assess the individual contribution of each of the three components of HTBM, three ablated variants are evaluated: HTBM without the Reptile meta-initialization (HTBM-NM), HTBM without the hybrid-transformer sequence encoder (HTBM-MLP), and HTBM without the Double DQN

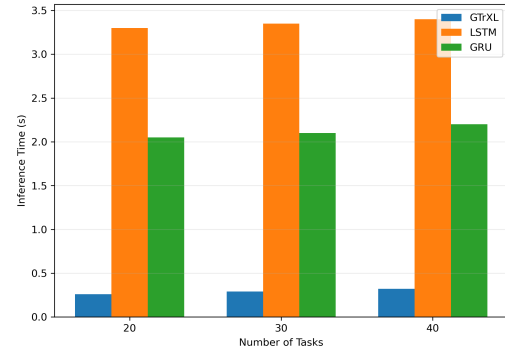


Fig. 9. Inference-time comparison of the hybrid-transformer encoder, LSTM, and GRU as the number of tasks increases.

target rule (HTBM-DQN). Each variant modifies exactly one component while keeping the remaining architecture, DDQN hyperparameters, and DAG generation settings identical to those used for the full HTBM model in Section VI-A. All variants are evaluated under the same 7 Mbps transmission-rate configuration and the same test set of 100 DAG applications used in Section VI-D.

For HTBM-NM, the network is initialized randomly rather than from the meta-trained parameters θ , and is then adapted using the same number of inner-loop DDQN updates specified in Algorithm 2. For HTBM-MLP, the hybrid-transformer encoder is replaced by a feed-forward encoder that maps the instantaneous state s_t directly to the representation used by DDQN, without access to the observation history $\mathcal{H}_t$. For HTBM-DQN, the target value in Eq. (17) is replaced by the conventional DQN target, in which the target network is used for both action selection and action evaluation.

Table IV summarizes the average task-completion latency obtained by each variant, together with the relative degradation with respect to the full HTBM model.

TABLE IV. ABLATION RESULTS AT 7 MBPS TRANSMISSION RATE

Variant	Component Removed	Avg. Latency	Degradation vs. HTBM
HTBM (full)	—	735.10	—
HTBM-NM	Reptile meta-init	779.40	+6.04%
HTBM-MLP	Hybrid-transformer	758.85	+3.24%
HTBM-DQN	Double Q target	747.60	+1.71%

The largest degradation is observed when the Reptile meta-initialization is removed, indicating that rapid adaptation to a new task benefits more from a favorable starting point than from either the sequence representation or the Double-DQN target rule alone. Removing the hybrid-transformer produces a smaller but non-negligible increase in latency, suggesting that the observation history contributes useful context beyond the instantaneous state, consistent with the queueing and precedence dependencies described in Section III. The smallest degradation follows from replacing the Double DQN target with the standard DQN target, which is consistent with the discrete and relatively small action space considered in this work; value overestimation has a comparatively limited effect here, though it may become more pronounced with a larger number of candidate edge servers. These results indicate that

all three components contribute positively to HTBM, with the meta-learning initialization providing the largest single contribution under the evaluated setting.

VII. DISCUSSION

HTBM combines three complementary mechanisms: the hybrid-transformer provides a sequence-level representation of past experiences, DDQN provides a value-based decision mechanism that reduces the overestimation tendency of standard DQN, and Reptile provides a meta-initialization that adapts efficiently to new MEC environments. As reported in Section VI, HTBM consistently achieves lower average latency than GASTO across the evaluated transmission rates and DAG topologies, including the unseen topology.

These results provide evidence for the latency-oriented effectiveness of HTBM under the evaluated simulation settings, complemented by the preliminary component-level evidence from the ablation study in Section VI-H. Several aspects of the evaluation remain open and are identified as concrete priorities for ongoing follow-up work. First, all experiments use synthetically generated DAGs (Table I), generated with the same methodology as MRLCO and GASTO for fair comparison, and training cost (wall-clock meta-training time, parameter count) is not benchmarked against these baselines; both aspects, together with evaluation on real application traces, warrant further study. Second, the reported comparisons (Table II, Table III, Fig. 3–9) reflect single training runs per configuration; given the well-known run-to-run variance of deep reinforcement learning, a multi-seed evaluation (five or more independent runs, mean $\pm$ standard deviation, and a paired *t*-test for the HTBM-vs-GASTO comparisons) is planned, together with a similar statistical treatment of the ablation results. Third, the DDQN and Reptile hyperparameters (Section VI-A) were adopted from established reference configurations rather than tuned or sensitivity-tested for this setting; an independent sensitivity analysis is planned alongside the multi-seed evaluation. Finally, the evaluation is conducted entirely in simulation and does not model realistic wireless traces, mobility, or device-level hardware constraints; validating HTBM on physical MEC testbeds with real network traces is identified as an important next step toward practical deployment.

VIII. CONCLUSION

This study proposed HTBM, a hybrid-transformer-based meta-reinforcement-learning framework for latency-oriented task offloading in dynamic MEC environments. The approach retains a fine-grained DAG representation of applications and combines hybrid-transformer sequence encoding, Double DQN-based action-value learning, and Reptile meta-learning.

The hybrid-transformer component captures temporal relationships within sequences of MEC observations, while DDQN provides the decision mechanism for discrete execution-location choices. Reptile learns a meta-initialization that can be rapidly adapted to new task and environment configurations.

Consistent with the results reported in Section VI, HTBM achieves lower average task-completion latency than GASTO across the evaluated transmission rates, DAG topologies, and

the unseen topology. The ablation study in Section VI-H further indicates that each of the three components – the hybrid-transformer representation, the Double DQN target rule, and the Reptile meta-initialization – contributes positively to this performance, with the meta-initialization providing the largest single contribution under the evaluated setting. These results support the feasibility of the proposed hybrid-transformer and DDQN combination for latency-oriented MEC task offloading.

REFERENCES

- [1] M. Y. Akhlaqi and Z. B. Mohd Hanapi, "Task offloading paradigm in mobile edge computing—current issues, adopted approaches, and future directions," *J. Netw. Comput. Appl.*, vol. 212, 2023.
- [2] S. Dong, J. Tang, K. Abbas, R. Hou, J. Kamruzzaman, L. Rutkowski, and R. Buyya, "Task offloading strategies for mobile edge computing: A survey," *Comput. Netw.*, p. 110791, 2024.
- [3] J. Liu, Y. Mi, X. Zhang, and X. Li, "Task graph offloading via deep reinforcement learning in mobile edge computing," *Future Gener. Comput. Syst.*, vol. 158, pp. 545–555, 2024.
- [4] X. You, H. Yan, Y. Xu, L. Wang, and L. Dai, "Hierarchical reinforcement learning empowered task offloading in V2I networks," *arXiv:2405.11352*, 2024.
- [5] C. Zhang, C. Peng, M. Lin, Z. Du, and C. Wu, "Double DQN reinforcement learning-based computational offloading and resource allocation for MEC," in *Proc. 13th EAI Int. Conf. MONAMI*, Springer, 2024.
- [6] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," *arXiv:1901.02860*, 2019.
- [7] E. Parisotto *et al.*, "Stabilizing transformers for reinforcement learning," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 7487–7498.
- [8] Luong, Nguyen Cong, *et al.* "Transformer-Enhanced Reinforcement Learning: Fundamentals and Applications in Communication Networks." IEEE Communications Surveys & Tutorials 2026.
- [9] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, 2016, pp. 2094–2100.
- [10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
- [11] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv:1803.02999*, 2018.
- [12] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, 2021.
- [13] Y. Li, J. Li, Z. Lv, H. Li, Y. Wang, and Z. Xu, "GASTO: A fast adaptive graph learning framework for edge computing empowered task offloading," *IEEE Trans. Netw. Serv. Manag.*, vol. 20, no. 2, pp. 932–944, 2023.
- [14] Z. Liu, L. Huang, Z. Gao, M. Luo, S. Hosseinalipour, and H. Dai, "GA-DRL: Graph neural network-augmented deep reinforcement learning for DAG task scheduling over dynamic vehicular clouds," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, no. 4, pp. 4226–4242, 2024.
- [15] N. Moussammi, M. El Ghmary, and A. Idrissi, "Multi-Task Multi-User Offloading in Mobile Edge Computing," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 12, 2022, doi: 10.14569/IJACSA.2022.01312108.
- [16] N. Moussammi, M. El Ghmary, and A. Idrissi, "Multi-task offloading to a MEC server with energy and delay constraint," in *The Int. Conf. Artif. Intell. Smart Environ.*, Cham: Springer International Publishing, 2022.
- [17] N. Moussammi, M. El Ghmary, and A. Idrissi, "Optimization strategies in mobile edge computing through intelligent task offloading," in *Modern Artificial Intelligence and Data Science 2024: Tools, Techniques and Systems*, Cham: Springer Nature Switzerland, 2024, pp. 539–553.
- [18] X. Chen, J. Cao, Y. Sahni, M. Zhang, Z. Liang, and L. Yang, "Mobility-aware dependent task offloading in edge computing: A digital twin-assisted reinforcement learning approach," *IEEE Trans. Mob. Comput.*, vol. 24, no. 4, pp. 2979–2994, 2025.

- [19] T. Z. Gebrekidan, S. Stein, and T. J. Norman, "Combinatorial client-master multiagent deep reinforcement learning for task offloading in mobile edge computing," in *Proc. 23rd Int. Conf. Auton. Agents Multiagent Syst. (AAMAS)*, 2024, pp. 2273–2275.
- [20] Y. Qin, J. Chen, L. Jin, R. Yao, and Z. Gong, "Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning," *Sci. Rep.*, vol. 15, no. 1, pp. 211–219, 2025.
- [21] S. Xu, J. Liu, J. Tang, X. Liu, and Z. Li, "Multi objective reinforcement learning driven task offloading algorithm for satellite edge computing networks," *Sci. Rep.*, vol. 15, 2025.
- [22] X. Yang, H. Fang, Y. Gao, X. Wang, K. Wang, and Z. Liu, "Computation offloading and resource allocation based on P-DQN in LEO satellite edge networks," *Sensors*, vol. 23, no. 24, p. 9885, 2023.
- [23] B. Li, Y. Liu, L. Tan, H. Pan, and Y. Zhang, "Digital twin assisted task offloading for aerial edge computing and networks," arXiv:2208.00834, 2022.
- [24] W. Cheng, X. Liu, X. Wang, and G. Nie, "Task offloading and resource allocation for Industrial Internet of Things: A double-dueling deep Q-network approach," *IEEE Access*, vol. 10, pp. 103111–103120, 2022.
- [25] Z. Liu, M. Liwang, S. Hosseinalipour, H. Dai, Z. Gao, and L. Huang, "RFID: Towards low latency and reliable DAG task scheduling over dynamic vehicular clouds," arXiv:2208.12568, 2022.
- [26] B. Guo, M. Liwang, X. Xia, L. Li, Z. Jiao, S. Hosseinalipour, and X. Wang, "Seamless graph task scheduling over dynamic vehicular clouds: A hybrid methodology for integrating pilot and instantaneous decisions," arXiv:2502.12557, 2025.
- [27] B. Sun, M. Theile, Z. Qin, D. Bernardini, D. Roy, A. Bastoni, and M. Caccamo, "Edge generation scheduling for DAG tasks using deep reinforcement learning," arXiv:2308.14647, 2023.
- [28] N. Gholipour, M. D. de Assuncao, P. Agarwal, J. Gascon-Samson, and R. Buyya, "TPTO: A Transformer-PPO based task offloading solution for edge computing environments," arXiv:2312.11739, 2023.
- [29] P. Phan-Trung *et al.*, "GTPO: A stabilizing task offloading technique based on GTrXL and PPO in IoT edge computing environments," in *Proc. IEEE Conf.*, 2024.
- [30] M. Han, X. Sun, X. Wang, W. Zhan, and X. Chen, "Transformer-based distributed task offloading and resource management in cloud-edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 9, pp. 2938–2953, 2025.
- [31] Z. Wang, M. Goudarzi, and R. Buyya, "TF-DDRL: A transformer-enhanced distributed DRL technique for scheduling IoT applications in edge and cloud computing environments," *IEEE Trans. Serv. Comput.*, vol. 18, no. 2, pp. 1039–1053, 2025.
- [32] G. Rjoub, S. Islam, J. Bentahar, M. A. Almaiah, and R. Alrawashdeh, "Enhancing IoT intelligence: A transformer-based reinforcement learning methodology," arXiv:2404.04205, 2024.
- [33] D. Byrne, "Double Deep Q-network (DDQN) implementation," PyTorch Lightning Bolts, 2020. [Online]. Available: https://lightning-bolts.readthedocs.io/en/stable/models/reinforce_learn.html