# Dependent Task Offloading in Vehicular Edge Computing Using Trajectory-Aware Deep Reinforcement Learning

Sangrez Khan *, Marios Avgeris †, Amir Ali-Pour * Julien Gascon-Samson *, Aris Leivadeas *
* Department of Software and IT Engineering, École de Technologie Supérieure, Montreal, Canada
Email: sangrez.khan.1@ens.etsmtl.ca, amir.ali-pour@etsmtl.ca, julien.gascon-samson@etsmtl.ca, aris.leivadeas@etsmtl.ca
† Informatics Institute, Faculty of Science, University of Amsterdam (UvA), Amsterdam, Netherlands
Email: m.avgeris@uva.nl

***Abstract*—Vehicular Edge Computing (VEC) enables latency sensitive applications by offloading computation from moving vehicles to roadside units (RSUs). Scheduling dependent tasks in such environments is challenging because vehicles move quickly through RSU coverage zones and connectivity windows are short. We propose a trajectory aware task offloading framework that unifies short-term mobility forecasting with deep reinforcement learning. A mobility-aware transformer predicts the vehicle's future trajectory and derives RSU coverage intervals, while a Graph Attention Network (GAT) based actor–critic, trained via Proximal Policy Optimization (PPO), decides where each task in a directed acyclic graph (DAG) should execute. Experiments with real highway trajectories and a large synthetic DAG library show that the proposed approach consistently outperforms baselines in terms of delay, and energy.**

***Index Terms*—Vehicular Edge Computing, Task Offloading, Deep Reinforcement Learning, Mobility Prediction**

## I. Introduction

The rapid evolution of connected and autonomous vehicles has intensified the demand for low-latency, high-reliability computation to support applications such as real-time navigation, cooperative perception and augmented reality [1]. Vehicular Edge Computing (VEC) addresses this need by enabling vehicles to offload computational tasks to nearby RSUs equipped with edge servers. By processing data close to the source, VEC reduces end-to-end latency and alleviates the load on remote cloud infrastructures [2].

However, task offloading in vehicular environments faces distinct challenges. Vehicles move at high speeds, causing rapidly fluctuating wireless channels, brief RSU coverage periods, and frequent handoffs [3]. These conditions make scheduling decisions more complex, particularly for applications with dependent subtasks represented as Directed Acyclic Graphs (DAGs), where the order of execution and data dependencies must be strictly followed [4], [5]. Poor scheduling in such scenarios can result in longer execution times, higher energy usage, and unnecessary handoffs.

Traditional offloading strategies, including heuristic rules and optimization based approaches, struggle to adapt to the fast changing topology of vehicular networks. While Deep Reinforcement Learning (DRL) has shown promise in enabling adaptive and data driven decision making, most existing works consider independent tasks and ignore future vehicle locations [6]. Consequently, these solutions overlook the link between predicted RSU coverage windows and the structure of DAG based applications. To address these limitations, we propose a trajectory aware DAG based task offloading framework that jointly considers predicted mobility patterns and task dependencies. Our contribution is threefold:

- We develop an enhanced Transformer model to predict short horizon vehicle trajectories from real highway data. The predicted positions are utilized to define RSU coverage intervals, giving the agent foresight into future connectivity.
- We design a Graph Attention Network (GAT) based actor–critic policy, trained with Proximal Policy Optimization (PPO), that makes DAG level offloading decisions by jointly encoding task features, RSU states and predicted coverage information.
- We conduct extensive experiments using real mobility traces and a large library of synthetic DAG workloads. Our method consistently outperforms all-local, all-remote, random and compute-aware strategies in terms of delay, and energy.

The rest of the paper is organized as follows, Section II reviews related work. Section III presents the system model and problem formulation. Section IV describes the proposed framework, Section V the experimental setup and results, and Section VI concludes.

## II. Related Work

Mobility prediction is a fundamental enabler for efficient VEC offloading. Early works have relied on simple models such as constant velocity or Markov chains to estimate future vehicle positions [7]. While computationally inexpensive, these models fail to capture the non-linear dynamics of real traffic scenarios, particularly under varying acceleration and lane-change behaviors.

More recent approaches employ machine learning-based predictors, including Recurrent Neural Networks (RNNs) and

Convolutional Neural Networks (CNNs) [8], [9]. These methods improve prediction accuracy but often struggle with long-term dependencies and irregular sampling intervals in vehicular data. Existing DAG scheduling methods in edge computing include list scheduling [10] and critical-path heuristics [11], but these often require precise a priori knowledge of task execution times and network conditions, which is unrealistic in vehicular contexts. Some studies have proposed integer linear programming (ILP) formulations for DAG offloading [12], achieving optimality at the expense of scalability. More adaptive methods have incorporated metaheuristics such as particle swarm optimization (PSO) [13], though convergence speed can be an issue under fast changing topologies.

DRL has recently been applied to VEC to enable adaptive decision making under uncertainty [14]. Works employing Deep Q-Networks (DQN) [15] or Deep Deterministic Policy Gradient (DDPG) [16] focus mainly on independent tasks and ignore inter task dependencies. Multi-agent DRL frameworks [17] have explored coordination among RSUs but typically assume perfect knowledge of vehicle positions. Proximal Policy Optimization (PPO) has gained attention for its stability and sample efficiency [18], yet its integration with DAG-based scheduling in vehicular environments remains unexplored. Few studies have jointly optimized mobility prediction and task offloading. A representative example is [19], where an LSTM-based predictor informs a heuristic offloading scheme. However, this approach suffers from error propagation and lacks a unified learning paradigm. In [20], predicted positions are used for coarse-grained RSU selection, without integrating predictions into the DRL state space.

From the discussion above, three key gaps can be identified: (i) mobility prediction models are often separated from task offloading decisions, resulting in inefficient scheduling; (ii) dependent task structures are frequently simplified or overlooked, reducing performance for complex applications; and (iii) current DRL methods lacks leverage graph-based neural networks to capture task dependencies in vehicular edge computing (VEC). Our work addresses these gaps by proposing a unified trajectory-aware DAG offloading framework that combines an enhanced transformer predictor with a GAT-PPO policy, tested in a vehicular edge scenario.

## III. System Model

The system model, depicted in Fig. 1, describes a VEC environment along a multi-lane highway segment, comprising vehicles, RSUs, and a core network. The highway is a linear segment along the $y$-axis with vehicles moving in multiple lanes, divided into contiguous coverage zones of length $L_{\text{zone}}$ meters. The set of RSUs is $\mathcal{R} = \{1, 2, \ldots, N_{RSU}\}$, where $N_{RSU} \in \mathbb{N}$ is the number of RSUs, each positioned at intervals of $L_{\text{zone}}$ meters with coverage zone centered at $y_r = (r-1) \cdot L_{\text{zone}}$ for RSU $r \in \mathcal{R}$. Each RSU has a computing capacity of $f_r$ in GHz, a wireless bandwidth of $B_r$ in Mbps, and an instantaneous load of $u_r(t)$ representing resource utilization, connected to the core network via high-speed backhaul links with bandwidth $B_{bh}$ in Gbps. The set

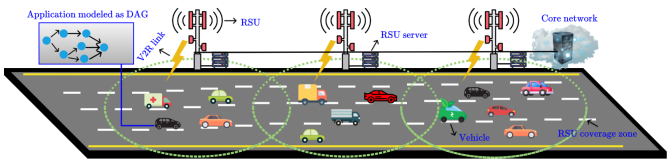


Fig. 1: Trajectory aware vehicular edge computing system model with DAG based task offloading.

of vehicles is $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$, where $|\mathcal{U}| \in \mathbb{N}$ is the number of vehicles. Each vehicle $u \in \mathcal{U}$ at time $t$ has a position $(x_u(t), y_u(t)) \in \mathbb{R}^2$, with $x_u(t)$ being the lane position and $y_u(t)$ being the longitudinal position, a local CPU frequency $f_u$, and a dynamic trajectory defined by speed $s_u(t)$, acceleration $\alpha_u(t)$, and lane changes. A vehicle $u$ at $(x_u(t), y_u(t))$ is associated with RSU $r \in \mathcal{R}$ if its longitudinal position $y_u(t)$ lies within the scalar range $(r-1)L_{\text{zone}} \leq y_u(t) < rL_{\text{zone}}$, which defines the coverage interval of RSU $r$ along the highway. Applications on vehicle $u$ are modeled as DAGs $G_u = (V_u, E_u)$, where $V_u = \{v_1, v_2, \ldots, v_{|V_u|}\}$ is the set of tasks with $|V_u| \in \mathbb{N}$, and $E_u \subseteq V_u \times V_u$ represents precedence constraints, with $(v_i, v_j) \in E_u$ indicating task $v_j$ depends on $v_i$'s completion and output. Each task $v_i \in V_u$ is defined by a tuple $(c_i, d_i, t_i^{dl})$, where $c_i$ is the required computation cycles, $d_i$ is the output data size, and $t_i^{dl}$ is the deadline in seconds.

The data rate $R_{u,r}(t)$ between vehicle $u \in \mathcal{U}$ and RSU $r \in \mathcal{R}$ at time $t$ is given as:

$$R_{u,r}(t) = B_r \cdot \log_2 \left(1 + \frac{P_t G_{u,r}(t)}{N_0 + I_r(t)}\right), \tag{1}$$

where $P_t$ is the transmit power in W, $G_{u,r}(t)$ is the channel gain, $N_0$ is the noise power in W, and $I_r(t)$ is the inter-cell interference in W. The rate $R_{u,r}(t)$ in Mbps degrades with distance and load. Inter-RSU communication uses backhaul rate $R_{bh} = B_{bh}$ in Gbps. The transmission delay for data size $d$ is given as:

$$T_{tx}(d) = d / R_{u,r}(t) \tag{2}$$

For each task $v_i \in V_u$, the local and remote execution times are given as.

$$T_i^k = \begin{cases} \frac{c_i}{f_u}, & k = 0, \\ \frac{c_i}{f_k}, & k \in \mathcal{R}. \end{cases} \tag{3}$$

The start and finish times $S_i$ and $F_i$ are then defined as

$$S_i = \max_{v_j \in \text{pred}(i)} \{F_j + T_{tx}(d_j, \text{loc}(j), \text{loc}(i))\}, \tag{4}$$

$$F_i = S_i + \sum_{k \in \{0\} \cup \mathcal{R}} a_i^k \, T_i^k, \tag{5}$$

where $\text{pred}(i) \subseteq V_u$ is the set of predecessor tasks, and $\text{loc}(l)$ is the execution node of task $v_l$. The energy consumption for local execution of task $v_i \in V_u$ is $E^{\text{u}}(v_i) = c_i \cdot \kappa_{\text{cpu}}$, where $\kappa_{\text{cpu}}$ is the CPU energy per cycle. For offloaded tasks, energy consumption is mainly due to the transmission of output data

to dependent tasks at remote locations.. The total energy $E_{\text{total}}$ is:

$$E_{\text{total}} = \sum_{u\in\mathcal{U}} \sum_{v_i\in V_u} a_i^0 \cdot c_i \cdot \kappa_{\text{cpu}} + \sum_{u\in\mathcal{U}} \sum_{(v_j,v_i)\in E_u} \mathbb{1}_{\{a_j^0=1, a_i^k\neq 0\}} \cdot d_j \cdot \kappa_{\text{tx}}, \quad (6)$$

where $\kappa_{\text{tx}}$ is the energy consumed per transmitted bit, and $\mathbb{1}$ is the indicator function.

### A. Problem Formulation

For each vehicle $u \in \mathcal{U}$ and tasks $v_i$ the binary decision variable is:

$$a_i^k = \begin{cases} 1, & \text{if task } v_i \text{ is executed at RSU } k, \\ 0, & \text{if task } v_i \text{ is executed locally on the vehicle} \end{cases}$$

where $k \in \{0\}\cup\mathcal{R}$, with $k = 0$ for local execution on $u$. The objective is to minimize the weighted sum of total completion time and energy consumption:

$$\min_{\{a_i^k\}} \quad \lambda_t \sum_{u\in\mathcal{U}} T_{\text{tot},u} + \lambda_e \sum_{u\in\mathcal{U}} E_{\text{total},u} \quad (7)$$

$$\text{s.t.} \quad F_i \le t_i^{\text{dl}}, \forall v_i \in V_u, \forall u \in \mathcal{U}, \quad (C1)$$

$$\sum_{k\in\{0\}\cup\mathcal{R}} a_i^k = 1, \forall v_i \in V_u, \forall u \in \mathcal{U}, \quad (C2)$$

$$a_i^k \in \{0,1\}, \forall v_i \in V_u, \forall k \in \{0\}\cup\mathcal{R}. \quad (C3)$$

where $T_{\text{tot},u} = \max_{v_i\in V_u} F_{i,u}$ is the completion time for vehicle $u$, $E_{\text{total},u}$ is the energy consumption (Eq. 6), and $\lambda_t, \lambda_e$ are weighting coefficients that satisfy $\lambda_t + \lambda_e = 1$, with typical values between (0,1). This problem is NP-hard because it involves discrete offloading decisions under precedence and deadline constraints which can be solved using a reinforcement learning framework to sequentially determine $\{a_i^k\}$.

## IV. Trajectory Aware DAG Offloading Framework

Our framework integrates a Transformer-based mobility prediction module and a GAT-based DAG aware task offloading policy trained with PPO. Transformer uses multi head self-attention mechanism enables direct modeling of long range temporal dependencies in high frequency data, allowing the model to accurately correlate distant historical triggers with future non-linear maneuvers that sequential models often fail to capture. By incorporating predicted mobility into offloading decisions, our framework operates in two sequential stages: (i) a Transformer-based mobility prediction stage that forecasts short-horizon trajectories and RSU coverage windows, and (ii) a GAT-PPO offloading stage that uses these predictions to make task-assignment decisions. These stages are executed sequentially for each scheduling interval, with the prediction stage feeding updated mobility information to the offloading stage.

**Stage 1 Trajectory Prediction:** This stage forecasts short horizon vehicular mobility to estimate RSU coverage windows, critical for avoiding misaligned task scheduling that potentially increases latency and energy consumption. For vehicle $u \in \mathcal{U}$ at time $t$, the state is defined as

$$\mathbf{s}_u(t) = [x_u(t), y_u(t), s_u(t), \theta_u(t), a_u(t)], \quad (8)$$

where $x_u(t), y_u(t)$ denote the vehicle's position coordinates, $s_u(t)$ is its speed, $\theta_u(t)$ is the heading angle that is direction of motion relative to the $x$-axis, measured in radians, and $\alpha_u(t)$ is the acceleration. The input to the Transformer-based mobility predictor is a sequence of $w$ historical states:

$$X_{u,t} = [\mathbf{s}_u(t-w+1), \ldots, \mathbf{s}_u(t)] \in \mathbb{R}^{w\times 5}. \quad (9)$$

The goal is to predict the next $H$ positions:

$$\hat{Y}_{u,t} = \{(\hat{x}_{u,t+1}, \hat{y}_{u,t+1}), \ldots, (\hat{x}_{u,t+H}, \hat{y}_{u,t+H})\}, \quad (10)$$

mapped to RSU $r \in \mathcal{R}$ if $\hat{y}_{u,t+h} \in [(r-1)\cdot L_{\text{zone}}, r\cdot L_{\text{zone}})$. The sequence $X_{u,t}$ is projected into a $d$-dimensional embedding space:

$$X'_{u,t} = W_p X_{u,t} + b_p, \quad (11)$$

where $W_p \in \mathbb{R}^{d\times 5}$, $b_p \in \mathbb{R}^d$ are learnable parameters. Sinusoidal positional encodings with learnable scaling and bias preserve temporal ordering. The encoded sequence is processed through $N$ Transformer encoder blocks, each with multi-head self-attention and a two-layer feedforward network (FFN) with residual connections and layer normalization. For input $X'_{u,t} \in \mathbb{R}^{w\times d}$, attention computes queries $Q \in \mathbb{R}^{w\times d_k}$, keys $K \in \mathbb{R}^{w\times d_k}$ and values $V \in \mathbb{R}^{w\times d_v}$, with output:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \quad (12)$$

The FFN uses the Gaussian Error Linear Unit (GELU):

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \quad (13)$$

where $W_1, W_2, b_1, b_2$ are learnable parameters. Outputs are aggregated via adaptive average pooling, and a two-layer FFN maps to $H$ coordinate pairs $(\hat{x}_{u,t+h}, \hat{y}_{u,t+h})$. The predicted $\hat{Y}_{u,t}$ yields RSU coverage intervals for Stage 2.

**Stage 2 Task Offloading via Reinforcement Learning:** The task offloading problem is formulated as a sequential DRL process for each vehicle $u \in \mathcal{U}$ with DAG $G_u$. The DRL agent sequentially assigns each task $v_i \in V_u$ to either local execution on the vehicle or to an RSU, respecting the DAG's dependencies and the system model's constraints ($F_i \le t_i^{\text{dl}}$, $\sum_k a_i^k = 1$). The agent leverages mobility predictions from Stage 1 to anticipate RSU availability, optimizing for the system model's objective. In the next section we will define the problem as a Markov Decision Process (MDP) and then solve it using DRL.

**State Space:** At decision step $t$ for task $v_i$, the reinforcement learning state $\mathbf{s}_{u,t}^{\text{RL}} \in \mathcal{S}$ for vehicle $u$ encapsulates task, mobility, and infrastructure information:

$$\mathbf{s}_{u,t}^{\text{RL}} = (\mathbf{h}_i, \mathbf{c}_r, \mathbf{z}_u, u_r(t)), \quad (14)$$

where $\mathbf{h}_i \in \mathbb{R}^d$ is the embedding of task $v_i$ from a GAT applied to $G_u$, capturing dependencies, $\mathbf{c}_r = [r, r_{\text{next}}, t_{\text{cov}}, \bar{u}_r]$ is the RSU context, with $r \in \mathcal{R}$ as the current RSU, $r_{\text{next}} \in \mathcal{R}$ as the predicted next RSU based on $\hat{Y}_{u,t}$, $t_{\text{cov}}$ as the estimated

coverage duration for RSU $r$, and $\bar{u}_r$ as the time-averaged load of RSU $r$; $\mathbf{z}_u$ is the trajectory embedding derived from $\hat{Y}_{u,t}$ via Stage 1's pooling layer; and $u_r(t)$ is the instantaneous load of RSU $r$, reflecting current resource utilization.

**Action Space:** For task $v_i$, the action $a_i \in \mathcal{A}$ selects the execution node:

$$a_i \in \{0\} \cup \mathcal{R}, \tag{15}$$

where $a_i = 0$ denotes local execution on vehicle $u$ with execution time $T_i^{\text{loc}} = \frac{c_i}{f_u}$, and $a_i = r \in \mathcal{R}$ denotes offloading to RSU $r$ with execution time $T_i^r = \frac{c_i}{f_r}$. This corresponds to setting $a_i^k = 1$ for the chosen $k \in \{0\} \cup \mathcal{R}$ in the system model, ensuring $\sum_k a_i^k = 1$. The agent respects precedence constraints by only selecting $a_i$ for task $v_i$ after all predecessor tasks $v_j \in \text{pred}(i)$ have been assigned and completed. After selecting an action $a_i$ for task $v_i$, the decision is stored in the vector $\mathbf{a}$, which records the execution node of every task in the DAG and is later used to evaluate delay and energy metrics.

**Reward Function:** After executing the DAG $G_u$ under action sequence $\{a_i\}$, the reward for vehicle $u$ is:

$$r_u = \lambda_t \cdot \frac{1}{1 + T_{\text{tot},u}} + \lambda_e \cdot \frac{1}{1 + E_{\text{total},u}}, \tag{16}$$

where $T_{\text{tot},u} = \max_{v_i \in V_u} F_{i,u}$ is the total completion time of vehicle $u$, $E_{\text{total},u}$ is the corresponding energy consumption, and $\lambda_t$ and $\lambda_e$ are weighting coefficients, as defined in the System Model (Eq. (4)–(5) for $S_i, F_i$ and Eq. (3) for $E_{\text{total},u}$). The weights $\lambda_t, \lambda_e$ balance latency and energy, aligning with the system model's objective. The inverse form promotes minimization of $T_{\text{tot},u}$ and $E_{\text{total},u}$.

**Policy Network:** The agent employs an actor-critic policy $\pi_\theta(a_i|s_{u,t})$ parameterized by $\theta$, implemented using a GAT to model the DAG $G_u$. For each task $v_i$, the GAT computes $\mathbf{h}_i$ by aggregating features from neighboring tasks $v_j \in \text{pred}(i)$ and successors, incorporating $(c_i, d_i, t_i^{dl})$ and edge weights based on $d_j$ for $(v_j, v_i) \in E_u$. The state $s_{u,t}$ is formed by concatenating $\mathbf{h}_i$, $\mathbf{c}_r$, $\mathbf{z}_u$, and $u_r(t)$, enabling the agent to account for task dependencies, RSU availability, and predicted mobility. The actor head outputs logits $\boldsymbol{\ell}_i$ over the action space $\{0\} \cup \mathcal{R}$ for $a_i$, which are masked to ensure feasibility based on RSU availability in $\mathcal{C}$ and completion of predecessors $v_j \in \text{pred}(i)$ ($F_j \leq t$). The critic head estimates the value function $V(s_{u,t})$ to reduce variance in policy updates. PPO maximizes the clipped surrogate objective with entropy regularization:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\Big[\min\left(\rho_t(\theta)A_t,\ \text{clip}(\rho_t(\theta),\, 1-\epsilon,\, 1+\epsilon)\, A_t\right) + \beta\, \mathcal{H}[\pi_\theta(\cdot \mid s_t)]\Big]. \tag{17}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_i|s_{u,t})}{\pi_{\theta_{\text{old}}}(a_i|s_{u,t})}$ is the probability ratio, $A_t$ is the advantage computed via generalized advantage estimation using reward $r_u$, $H[\pi_\theta]$ is the policy entropy, and $\epsilon, \beta$ are hyperparameters. This architecture enables the agent to make informed offloading decisions by integrating task dependencies via $\mathbf{h}_i$, RSU availability via $\mathbf{c}_r$ and $u_r(t)$, and mobility forecasts via $\mathbf{z}_u$, ensuring robust scheduling that aligns with the system model's objective of minimizing $\lambda_t \sum_u T_{\text{tot},u} + \lambda_e \sum_u E_{\text{total},u}$. The overall procedure of the proposed trajectory aware DAG offloading framework is summarized in Algorithm 1.

**Algorithm 1:** Trajectory Aware DAG Offloading with PPO

**Input:** $H$, DAG $G_u = (V_u, E_u)$, RSU set $\mathcal{R}$
**Output:** Offloading decisions $\{a_i^k\}$ for all tasks $v_i \in V_u$
`// Stage 1: Trajectory Prediction`
Predict future positions $\{(\hat{x}_{u,t+h}, \hat{y}_{u,t+h})\}_{h=1}^{H}$ using Transformer $\Rightarrow \hat{Y}_{u,t}$
Compute trajectory embedding $\mathbf{z}_u \leftarrow \text{PoolingLayer}(\hat{Y}_{u,t})$
Estimate RSU coverage intervals $\mathcal{C} \leftarrow \text{RSUCoverageWindows}(\hat{Y}_{u,t}, \mathcal{R})$
`// Stage 2: GAT-PPO Policy Decision`
Normalize task features $\mathbf{X} \leftarrow \text{NormalizeFeatures}(G_u)$
Obtain RSU states $\mathbf{S} \leftarrow \text{GetRSUStates}(\mathcal{R})$
Extract RSU context $\mathbf{c}_r \leftarrow \text{ExtractContext}(\mathcal{C}, \mathbf{S})$
Initialize empty decision vector $\mathbf{a}$
**for** *each task* $v_i \in V_u$ *in topological order* **do**
  Compute task embedding $\mathbf{h}_i \leftarrow \text{GAT}(\mathbf{X}, E_u)[i]$
  Form complete state $\mathbf{s}_{u,t} \leftarrow \text{Concat}(\mathbf{h}_i, \mathbf{c}_r, \mathbf{z}_u, u_r(t))$
  Obtain action logits $\boldsymbol{\ell} \leftarrow \text{Actor}(\mathbf{s}_{u,t})$
  Mask infeasible actions $\boldsymbol{\ell}^m \leftarrow \text{ActionMask}(\boldsymbol{\ell}, \mathcal{C}, \mathbf{S}, \text{pred}(i))$
  Sample action $a_i \sim \text{Categorical}(\text{Softmax}(\boldsymbol{\ell}^m))$
  Store decision $\mathbf{a}[i] \leftarrow a_i$
**if** *training mode* **then**
  `// Training update using PPO`
  Execute decisions $\mathbf{a}$ in environment
  Compute total delay $T_{\text{tot},u}$ and total energy $E_{\text{total},u}$ using $\text{ComputeMetrics}(\mathbf{a}, G_u)$
  Compute reward $r_u \leftarrow \lambda_t \cdot \frac{1}{1+T_{\text{tot},u}} + \lambda_e \cdot \frac{1}{1+E_{\text{total},u}}$
  Store experience $(\mathbf{s}_{u,t}, \mathbf{a}, r_u)$ in replay buffer
  **if** *replay buffer full* **then**
    Estimate advantage $\hat{A}_t \leftarrow \text{ComputeGAE}(\text{buffer})$
    Update policy parameters $\theta$ using PPO objective with clipping parameter $\epsilon$
**Output:** $\mathbf{a}$: final offloading decisions for tasks $v_i \in V_u$

## V. Experimental Setup

For the experimentation we utilize the US101 highway dataset from the Next Generation Simulation (NGSIM) program [21], containing high-resolution vehicle trajectory data at 10 Hz sampling frequency over a 2000-meter highway segment. We have included 10 RSUs providing 200-meter coverage zones with 20% overlap. Each RSU $i$ has compute capacity $4.0 + 0.4i$ GHz and bandwidth $100 + 10i$ Mbps. The channel model incorporates free-space path loss with shadowing, co-channel interference, and SINR-based data rates following Shannon capacity. The core network connects the RSUs via 10 Gbps fiber backhaul links. The pre processing of the dataset includes: (i) filtering stationary vehicles (max speed $< 5$ m/s) and short trajectories ($< 60$ time steps) and (ii) applying 4-sigma clipping to remove outliers in speed, acceleration, and heading, and z-score normalization per feature. The mobility predictor uses a 50-step history window to predict 50 future $(x, y)$ coordinates.

Task workloads are synthetically generated using the *DAGGEN* tool [22], which produces realistic DAGs modeling three vehicular computing scenarios: *Light tasks* (8–15 nodes, 0.5–3.0 MB, 0.1–1.0 GCycles) representing mobile applica-

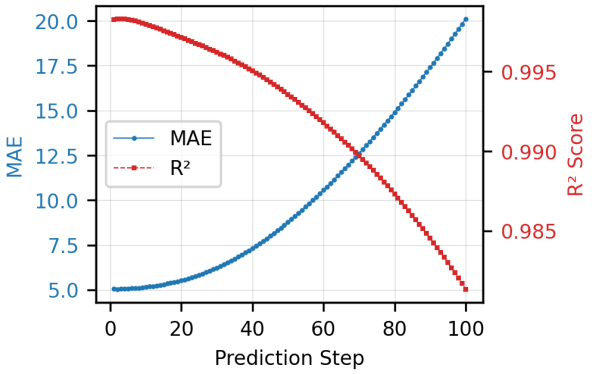


Fig. 2: Step-wise MAE and $R^2$ over the prediction horizon.

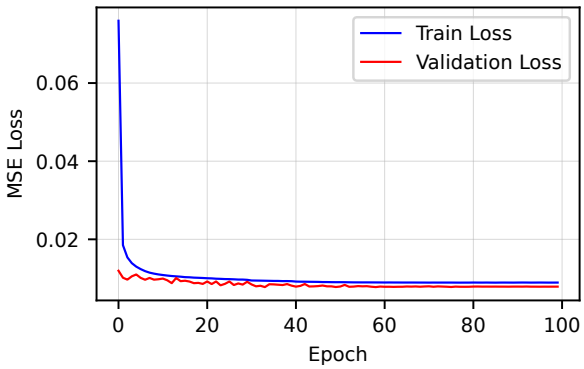


Fig. 3: Mean Squared Error (MSE) loss progression during transformer training over 100 epochs.

tions like object detection; *Medium tasks* (10–20 nodes, 1.0–5.0 MB, 0.3–2.0 GCycles) for IoT sensor processing; and *Heavy tasks* (15–25 nodes, 2.0–8.0 MB, 0.5–3.0 GCycles) for AR/VR applications. The complete dataset comprises 50,000 DAGs with realistic precedence structures: 15,000 light, 15,000 medium, and 20,000 heavy tasks.

We compare against four representative baseline strategies: (i) "all local" executes all tasks on the vehicle's processor; (ii) "all remote" offloads all tasks to the first available RSU; (iii) "random" assigns each task uniformly between local and remote execution; and (iv) "compute-aware" uses a threshold-based heuristic that offloads tasks exceeding 1.0 GCycles to the best available RSU, while smaller tasks execute locally. The Transformer mobility predictor employs 4 encoder blocks with 8 attention heads and 128-dimensional embeddings. Training uses the Adam optimizer with learning rate $1 \times 10^{-4}$, weight decay $1 \times 10^{-5}$, and early stopping based on validation MSE. The GAT-based PPO policy features 2 GAT layers with 4 attention heads each, processing task features, RSU states, and predicted trajectories. Key hyperparameters are summarized in Table I.

TABLE I: Training Hyperparameters

| **Parameter** | **Transformer** | **PPO Policy** |
|---|---|---|
| Learning Rate | $1 \times 10^{-4}$ | $3 \times 10^{-4}$ (Actor)<br>$1 \times 10^{-3}$ (Critic) |
| Batch Size | 1024 | 64 episodes |
| Hidden Dimension | 128 | 128 |
| Attention Heads | 8 | 4 |
| Layers/Blocks | 4 | 2 (GAT) |
| Dropout | 0.2 | – |
| Weight Decay | $1 \times 10^{-5}$ | – |
| PPO Epochs | – | 100 |
| Clip Parameter | – | 0.2 |
| GAE Lambda | – | 0.95 |
| Discount Factor | – | 0.99 |

## VI. Results and Discussion

We evaluated the proposed trajectory-aware DAG offloading framework through experiments on real vehicular trajectories and synthetic workloads. The experimental assessment focuses on three aspects: the accuracy of trajectory prediction, the performance of task offloading, and scalability.

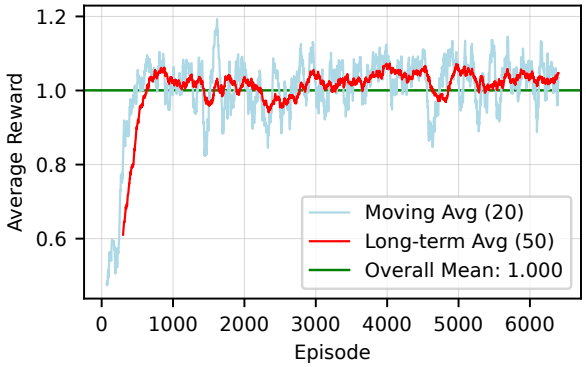


Fig. 4: Training reward progression for the PPO-based offloading policy.

Fig. 2 shows the step-wise deterioration of prediction accuracy over the 100-step prediction interval. The MAE gradually rises from 5.0 meters at the initial steps to 20.0 meters at the final step, following a quadratic growth trend typical of position prediction tasks. Meanwhile, the $R^2$ value decreases from 0.998 to 0.980, indicating that the model still maintains strong correlation with the actual positions even over extended prediction horizons.

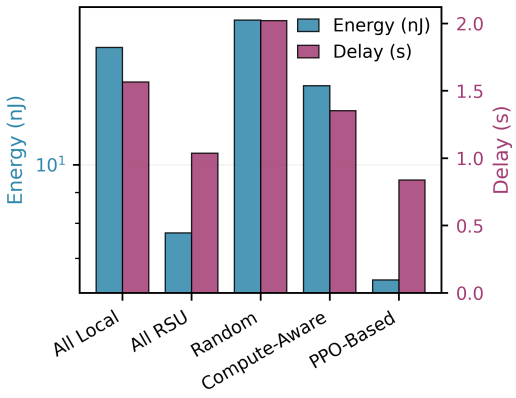


Fig. 5: Average energy and delay comparison.

Fig. 3 shows the Mean Squared Error (MSE) loss progression during transformer training over 100 epochs. Both training and validation losses decrease rapidly in the first 20 epochs, reaching approximately 0.02 MSE. The validation loss stabilizes around epoch 40, while training loss continues to decrease slightly, indicating appropriate model capacity without severe overfitting. The final training-validation gap remains minimal, suggesting good generalization to unseen trajectory data. The combination of these mechanisms enables the proposed framework to achieve superior performance across multiple evaluation criteria while maintaining computational efficiency suitable for real-time vehicular applications.

Fig. 4 illustrates the training process of the PPO-based policy over 6000 episodes. The training progresses through three distinct stages: an exploration phase (episodes 0-1000) with high variance in rewards, a convergence phase (episodes 1000-3000) where rewards rise from 0.8 to 1.1, and a stabilization phase (episodes 3000-6000) with rewards consistently around the average of 1.000. The moving averages (20-episode and 50-episode windows) indicate stable learning without oscillations or performance drops, reflecting robust policy convergence. Incorporating trajectory prediction and action masking further enhances stability by structuring exploration and limiting the action space to feasible options.

Fig. 5 compares the performance of five offloading strategies in terms of average delay and energy consumption. The proposed PPO-based approach achieves a notably lower average makespan of $0.7$ seconds, representing reductions of $70.8\%$ and $58.8\%$ compared to the all-local and compute-aware strategies, respectively. Regarding energy consumption, our method averages $4.8$ nJ, corresponding to decreases of $80.3\%$ and $71.4\%$ relative to the all-local and compute-aware baselines. By contrast, the all-remote strategy consumes $7.6$ nJ on average. These results highlight the superior energy efficiency of our approach, which smartly balances local execution with RSU offloading to reduce transmissions while taking advantage of RSU computing resources.

## VII. Conclusion

We introduced a trajectory-aware DAG-based task offloading framework for vehicular edge computing (VEC), combining a Transformer-based mobility predictor with a GAT-PPO scheduling agent. Extensive experiments using real vehicular trajectories and synthetic workloads showed that our approach consistently outperforms baseline methods in terms of delay and energy efficiency.

Future work will expand this framework to multi-agent scenarios and incorporate federated reinforcement learning, enabling collaborative coordination among RSUs and improving overall scalability.

## References


[1] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[2] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, vol. 26, no. 3, pp. 1145–1168, 2021.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

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

[5] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[6] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, vol. 25, no. 5, pp. 1999–2017, 2022.

[7] J. Liu, X. Mao, Y. Fang, D. Zhu, and M. Q.-H. Meng, "A survey on deep-learning approaches for vehicle trajectory prediction in autonomous driving," *arXiv preprint arXiv:2110.10436*, 2021.

[8] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1672–1678.

[9] N. Nikhil and B. T. Morris, "Convolutional neural network for trajectory prediction," *arXiv preprint arXiv:1809.00696*, 2018.

[10] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, 2014.

[11] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506–521, 1996.

[12] G. F. C. de Queiroz, J. F. de Rezende, and V. C. Barbosa, "A flexible algorithm to offload dag applications for edge computing," *Journal of Network and Computer Applications*, 2024, early version available as arXiv:2306.09458.

[13] Q. You and B. Tang, "Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things," *Journal of Cloud Computing*, vol. 10, no. 1, p. 41, 2021.

[14] D. Hortelano, I. de Miguel, R. J. D. Barroso, J. C. Aguado, N. Merayo, L. Ruiz, A. Asensio, X. Masip-Bruin, P. Fernández, R. M. Lorenzo, and E. J. Abril, "A comprehensive survey on reinforcement-learning-based computation offloading techniques in edge computing systems," *Journal of Network and Computer Applications*, vol. 216, p. 103669, 2023.

[15] F. Dai, G. Liu, Q. Mo, W. Xu, and B. Huang, "Task offloading for vehicular edge computing with edge-cloud cooperation," *World Wide Web*, vol. 25, no. 5, pp. 1999–2017, 2022.

[16] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, p. 188, 2020.

[17] S.-H. Moon and Y. Lim, "Federated deep reinforcement learning based task offloading with power control in vehicular edge computing," *Sensors*, vol. 22, no. 24, p. 9595, 2022.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[19] S. Lv, W. Liu, D. Chen, and Z. Qin, "Task offloading and serving handover of vehicular edge computing networks based on trajectory prediction," *IEEE Access*, vol. 9, pp. 130 793–130 804, 2021.

[20] R. Yan, Y. Gu, Z. Zhang, and S. Jiao, "Vehicle trajectory prediction method for task offloading in vehicular edge computing," *Sensors*, vol. 23, no. 18, p. 7954, 2023.

[21] U.S. Department of Transportation, "Next Generation Simulation (NGSIM) Program US-101 Videos," https://doi.org/10.21949/1504477, 2016.

[22] F. Suter, "DAGGEN: A synthetic task graph generator," 2013, accessed: 2025-10-09. [Online]. Available: https://github.com/frs69wq/daggen