A RAG-assisted DRL Framework for Microservices Deployment in 6G vehicular Networks

Daniel Ayepah-Mensah*, Amine Kidane Ghebreziabiher*, Gordon Owusu Boateng†, Rabeb Mizouni‡, Azzam Mourad*§, Hadi Otrok‡, Jamal Bentahar*¶, Sami Muhaidat‖,**
* KU 6G Research Center, Dept. of CS, Khalifa University, Abu Dhabi, UAE
† Dept. of Communications and Networking, Xi'an Jiaotong–Liverpool University, Suzhou, China
‡ Center for Cyber-Physical Systems (C2PS), Dept. of CS, Khalifa University, Abu Dhabi, UAE
§ Artificial Intelligence & Cyber Systems Research Center, Dept. of CSM, Lebanese American University, Beirut, Lebanon
¶ Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada
‖ KU 6G Research Center, Dept. of EECS, Khalifa University, Abu Dhabi, UAE
** Dept. of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada

***Abstract*—Modern edge cloud platforms must efficiently deploy and route containerized microservice DAGs under strict latency and cost constraints, while adapting to rapidly changing workloads and infrastructure states. Deep Reinforcement Learning (DRL) schedulers adapt well to dynamics but often lack semantic awareness of service intent and task dependencies, resulting in suboptimal decisions in unseen scenarios. To overcome these limitations, we introduce a Retrieval-Augmented Generation–assisted DRL (RAG-DRL) framework that integrates a lightweight DRL agent with a graph-based RAG module powered by a partially frozen LLM. A dynamic memory graph encodes contextual information such as node resources, network latencies, and SLA feedback. The LLM retrieves relevant historical deployments and current service intents to generate soft placement plans and reward estimates, which guide the DRL agent. These priors accelerate convergence, improve generalization across diverse conditions, and ensure real-time responsiveness. Evaluations on a realistic urban-scale edge cloud testbed confirm that RAG-DRL significantly reduces SLA violations, end-to-end latency, and resource imbalance, outperforming modern container-based schedulers. Our framework converges faster, maintains latency below 65 ms on scale, limits SLA violations to 12% under heavy load, and achieves 90% resource utilization with balanced distribution.**

***Index Terms*—Edge–Cloud Orchestration, Microservice Deployment, Deep Reinforcement Learning, Retrieval-Augmented Generation (RAG), Large Language Models (LLMs).**

I. Introduction

The introduction of fifth-generation networks (5G) has ushered in a new era of connected vehicles, enabling sensor sharing, cooperative driving, and immersive infotainment [1], [2]. Looking to the future, sixth generation (6G) networks extend these ambitions by aiming for ubiquitous coverage, 99. 999% reliability, submillisecond latency between movement and decision, and seamless mobility at speeds of more than 300 km/h [1]. To meet these stringent requirements, a fundamental shift in resource management is needed. Modern vehicular applications are increasingly designed as distributed microservices, where complex tasks such as cooperative perception and teleoperated driving are decomposed into directed acyclic graphs (DAGs) of interdependent components. The central orchestration challenge is deciding, in real time, where each microservice should be placed and how traffic should be routed between them to satisfy the performance demands of 6G networks.

Deep reinforcement learning (DRL) offers a promising approach to dynamic orchestration through adaptive allocation of bandwidth, computing resources, and microservice replicas [3]. However, current DRL-based schedulers in vehicular environments struggle due to three critical factors: (i) inference latency exceeding microsecond control intervals, (ii) poor generalization to rapidly evolving network topologies, and (iii) slow adaptation to sudden traffic fluctuations or roadside unit (RSU)) failures[2]. The design space for orchestration itself is huge. For example, if 10 microservices are deployed, each with 5 options for replica placement and 3 routing paths, there are more than 10^{16} possible configurations, making an exhaustive search impractical [3].

Although DRL is a powerful foundation for dynamic orchestration, traditional schedulers struggle with the extreme size and volatility of 6G vehicle networks and often exhibit slow adaptation and poor generalization. Even with modern techniques such as offline pretraining and predictive foresight, significant challenges remain. Key problems include state explosion, the difficulty of encoding large service graphs within the constraints of edge computation, prediction uncertainty, where prediction accuracy drops significantly with rapid network changes, and the heuristic policy trade-off, which involves balancing fixed rules against the flexibility required to discover new solutions [4]. Given the immense design space, a more agile and context-aware strategy is essential to meet the strict real-time performance requirements.

Large Language Models (LLMs) are promising for the orchestration of vehicle networks. They provide semantic state representation, accelerate DRL convergence, and enable adaptation to unknown scenarios without manually created rules [5]. However, their use in 6G real-time networks is hampered by critical challenges. The primary risk is hallucination, as dynamic conditions such as rapid topology and mobility changes can make LLMs produce seemingly valid but incorrect instructions, resulting in wrong decisions [6]. This risk is

979-8-3503-9281-4/25/$31.00 ©2025 IEEE

exacerbated by the sparse training data for edge networks. Moreover, LLMs are slower than lightweight DRL agents in adapting to sudden events such as RSU failures, limiting their usefulness in practise.

In this paper, we address the above challenge by proposing a Retrieval-Augmented Generation-assisted Deep Reinforcement Learning (RAG-DRL) framework for deploying containerized microservices in 6G vehicular networks. Our approach decouples high-level semantic reasoning from low-level real-time control. The LLM functions as a semantic controller that uses domain-specific knowledge rather than generic conversational patterns. Based on historical traces of microservice placement and long-term observations of changes in network topology, it generates context-dependent priorities and adaptive reward signals. These signals guide a lightweight DRL agent that can react in milliseconds. By combining the high situational awareness of the LLM with the efficient control of the DRL agent, the hybrid framework provides a scalable, low-latency, and interpretable orchestration suitable for the volatility of 6G edge cloud environments. The main contributions of this work are summarized as follows:

- We design a hybrid orchestration framework that integrates RAG. An LLM-based semantic controller retrieves historical deployment knowledge from a structured graph and provides context-aware priorities that accelerate DRL convergence while maintaining submillisecond decision latency.
- We develop a metalearning mechanism that, in conjunction with RAG, enables rapid policy transfer and cross-domain knowledge reuse, and supports rapid adaptation to changing network topologies and traffic patterns without costly retraining.
- Extensive simulations under realistic 6G vehicular edge–cloud conditions demonstrate that our RAG–DRL framework outperforms state-of-the-art baselines in end-to-end latency and resource utilization.

The rest of this article is organized as follows. Section II describes the system model and formulates the problem, Section III presents the proposed algorithm, and Section IV presents its evaluation. Section V concludes the paper.

II. System Model and Problem Formulation

A. System Architecture Overview

Fig. 1 illustrates a three-tier architecture for urban 6G vehicle networks. At the top level, the Cloud Core hosts a centralized controller with an LLM-based scheduler that creates coarse-grained deployment plans and distributes them to regional Edge Cloud (EC) nodes. ECs are distributed throughout the city, each running a lightweight DRL scheduler. Each EC maintains a soft-real-time gRPC channel with the Cloud Core to receive updated blueprints, stream local telemetry, and refine micro-level placement for the Edge Service Providers (ESPs) under their control. The network layer consists of RSUs and micro base stations connected in a multihop mesh.

B. Network and Service Model

We consider a hierarchical edge cloud infrastructure designed for latency-sensitive applications such as real-time video analytics and vehicular intelligence [7]. The network is modeled as an undirected graph $\mathcal{G}_{\mathrm{S}} = (\mathcal{N}, \mathcal{L}, \mathcal{R}, \mathcal{B})$, where $\mathcal{N} = \mathcal{N}^{\mathrm{edge}} \cup \mathcal{N}^{\mathrm{cloud}}$ represents the set of nodes, comprising edge ESPs and ECs. Here, $\mathcal{L}$ denotes the set of communication links, $\mathcal{R}$ the available computing resources at each node, and $\mathcal{B}$ the link bandwidth capacities. . Each node $n \in \mathcal{N}$ has static resources $\mathbf{R}_n = (R_n^{\mathrm{cpu}}, R_n^{\mathrm{mem}}, R_n^{\mathrm{disk}})$ and time-varying availability $\mathbf{r}_n(t)$. Links $\mathcal{L} \subseteq \mathcal{N} \times \mathcal{N}$ are characterized by capacity B_{uv}, utilization $b_{uv}(t)$, and delay D_{uv}. The cloud controller handles global orchestration, while ESPs execute localized placement. Service requests are modeled as DAGs $\mathcal{G}_q = (\mathcal{M}_q, \mathcal{E}_q)$, where nodes represent microservices and edges encode invocation dependencies [3]. Each request $q \in \mathcal{Q}$ specifies resource demands Θ_q, traffic profiles Δ_q, and replica limits $\mathcal{R}_q$. A microservice $m_i \in \mathcal{M}_q$ can instantiate up to R_i replicas $\{m_i^r\}$, each requiring $\boldsymbol{\theta}_i = (r_i^{\mathrm{cpu}}, r_i^{\mathrm{mem}})$. Communication between microservices m_i and m_j incurs traffic $\boldsymbol{\delta}_{ij} \in \Delta_q$. This model captures both vertical scaling via resource allocation and horizontal scaling via replica deployment across ESPs.

C. Problem Formulation

We address the joint problem of *microservice placement* and *routing* between services in 6G vehicular edge cloud networks. The state of the system is defined by two decision variables: a binary variable $x_{i,n,r}^q \in \{0,1\}$, which indicates whether the r-th replica of the microservice $m_i \in \mathcal{M}_q$ is placed at node $n \in \mathcal{N}$; and a continuous variable, $y_{ij,p}^q \in [0,1]$, which denotes the traffic fraction for the dependency $e_{ij} \in \mathcal{E}_q$ that is routed through the path $p \in \mathcal{P}_{ij}$. The goal is to jointly optimize conflicting objectives, including minimizing overall service latency, $\overline{T}_t$, and overall resource utilization, $\mathbb{R}_{\mathrm{tot},t}$, to improve CPU and memory efficiency [8]. This optimization is subject to the important constraint that the end-to-end delay for

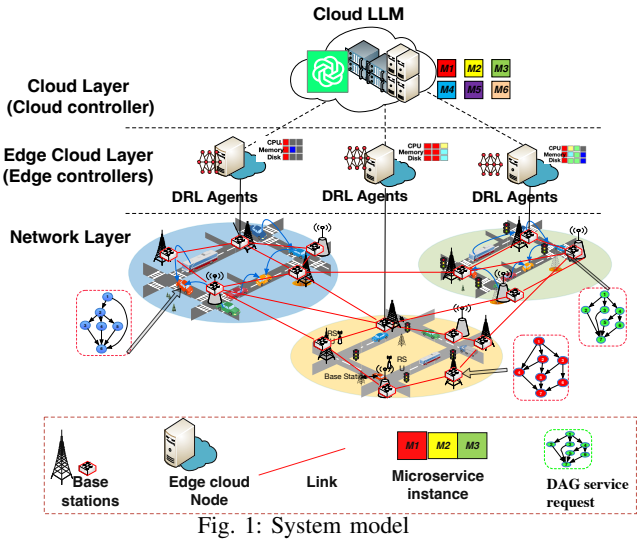

Fig. 1: System model

each request, T_q (consisting of access, processing, queueing and transmission times), must comply with the Service Level Agreement (SLA), so that $T_q \le d_q^{\max}$.

a) Optimization Problem.: The joint placement–routing problem is formulated as follows:

$$\min_{\mathbf{X}_t, \mathbf{Y}_t} \quad J_t = \alpha_1 \overline{T}_t + \alpha_2 \mathbb{R}_{\text{tot},t} \tag{1a}$$

$$\text{s.t.} \quad \sum_{n,r} x^q_{i,n,r} = R_i, \quad \forall i, q, \qquad \sum_{p \in \mathcal{P}_{ij}} y^q_{ij,p} = 1, \quad \forall i, j, q, \tag{1b}$$

$$\sum_{q,m,\rho} x^q_{m,n,\rho} r^r_m \le R^r_n, \quad \forall n, r, \tag{1c}$$

$$\sum_{q,i,j} \sum_{p \ni \ell} y^q_{ij,p} \, \delta_{ij} \le B_\ell, \quad \forall \ell. \tag{1d}$$

$$T_q \le d_q^{\max}, \quad \forall q, \qquad x^q_{i,n,r} \in \{0,1\}, \quad y^q_{ij,p} \in [0,1]. \tag{1e}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2) \succ 0$ is a vector that balances the trade-off between delay and resource consumption. The problem is formulated as a constrained mixed-integer nonlinear program (MINLP), which is computationally intractable due to the combinatorial nature of replica placement, continuous routing decisions, and nonlinear delay constraints. Constraint (1b) ensures that each microservice i in the query q has exactly R_i replicas and that all call paths are valid. Constraint (1c) enforces resource and bandwidth restrictions, while constraint (1e) guarantees compliance with the latency requirements.

III. Proposed Algorithm

In this section, we propose a hybrid RAG-DRL framework (Fig. 2) that decouples high-level semantic planning from low-level real-time control at the edge. By integrating the semantic reasoning and historical context of a cloud-based LLM with the low latency responsiveness of a DRL agent, our approach achieves agile and context-aware microservice provisioning in dynamic 6G environments. To accelerate convergence and improve adaptability to new network conditions, we use a Proximal Policy Optimization (PPO) -based metalearning strategy that boots the DRL agent with initial deployment plans generated by the RAG module.

A. MDP Transformation

In fast-changing vehicular and IoT environments, where decisions must be made within hundreds of microseconds, traditional solvers (e.g., MILP, heuristics) and retrieval-only methods fall short. We model the placement–routing task as a finite-horizon Markov Decision Process (MDP) [2], enabling the DRL agent to make rapid and sequential decisions and adapt to real-time network dynamics. $(\mathcal{S}, \mathcal{A}, P, R, \gamma, T_{\max})$ that evolves in synchronized scheduling slots $t = 0, 1, \ldots, T_{\max}$:

1) State $s_t \in \mathcal{S}$: At the decision step t, the state of the system is represented as:

$$s_t = \big(\mathbf{r}_t,\ \mathbf{b}_t,\ \mathcal{Q}_t,\ \pi_t\big), \tag{2}$$

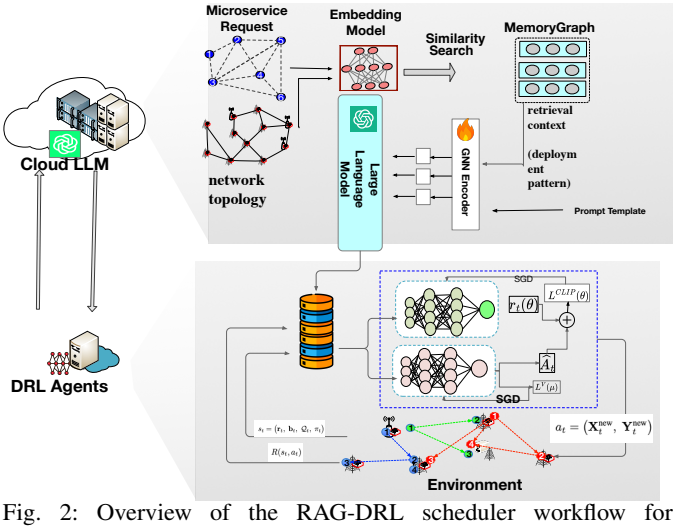

Fig. 2: Overview of the RAG-DRL scheduler workflow for microservice deployment

where $\mathbf{r}_t \in [0,1]^{2|\mathcal{N}|}$ denotes normalized residual CPU and memory resources across all nodes, and $\mathbf{b}_t \in [0,1]^{|\mathcal{L}|}$ encodes the available bandwidth on each network link. The set $\mathcal{Q}_t$ contains DAGs for pending service in slot t, while $\pi_t = (\mathbf{X}_t, \mathbf{Y}_t)$ captures the current placement and routing state.

2) Action $a_t \in \mathcal{A}$: The agent's action at time t comprises a new placement and routing decision for tasks in $\mathcal{Q}_t$:

$$a_t = \big(\mathbf{X}_t^{\text{new}},\ \mathbf{Y}_t^{\text{new}}\big), \tag{3}$$

where $\mathbf{X}_t^{\text{new}}$ assigns replicas to nodes and $\mathbf{Y}_t^{\text{new}}$ defines routing paths and bandwidth allocation. The action must satisfy capacity, affinity, and latency constraints.

3) Transition Kernel $P(s_{t+1} \mid s_t, a_t)$: State transitions are driven by deterministic resource updates and stochastic service arrivals. Node and link resources are updated as $\mathbf{r}_{t+1} = \mathbf{r}_t - \Delta\mathbf{r}(a_t) + \mathbf{r}_t^{\text{release}}$ and $\mathbf{b}_{t+1} = \mathbf{b}_t - \Delta\mathbf{b}(a_t) + \mathbf{b}_t^{\text{release}}$. New service DAGs $\mathcal{Q}_{t+1}$ arrive via a Poisson process with rate λ, and the placement–routing map is updated as $\pi_{t+1} = \pi_t \cup a_t \setminus$ completed DAGs.

4) Reward $R(s_t, a_t)$: The reward is the negative of the system-wide cost:

$$R(s_t, a_t) = -\left[\alpha_1 \overline{T}_t + \alpha_2 \mathbb{R}_{\text{tot},t}\right], \tag{4}$$

with weights $\boldsymbol{\alpha} = (\alpha_1, \alpha_2 \succ 0$ balancing latency and resource utilization.

5) Planning Horizon $T_{\max}$: The decision-making process unfolds over a finite horizon $T_{\max}$, reflecting the global planning interval. For instance, with 100 scheduling slots and sub-10ms resolution, $T_{\max} \approx 1$ second of real-time execution. The scheduler seeks a policy $\pi^\star : \mathcal{S} \to \mathcal{A}$ that maximises the expected discounted return $\mathbb{E}_P\big[\sum_{t=0}^{T_{\max}} \gamma^t R(s_t, a_t)\big]$.

B. RAG–Driven Scheduling Framework

Fig. 2 illustrates our proposed RAG-driven scheduling framework, where a semantic controller combines structured memory retrieval with generative reasoning to enable real-time

orchestration. The controller leverages a cloud-based LLM as a semantic planner and is composed of four key components: (i) a GNN encoder, (ii) a graph-based memory representation, (iii) context-aware query construction, and (iv) an LLM-driven semantic reasoning module.

a) GNN Encoder.: The first stage encodes service DAGs into compact vector sketches using a 3-layer Graph Attention Network (GAT). Each layer applies multi-head attention with 8 heads and 64-dimensional embeddings:

$$h_i^{(l+1)} = \sigma \sum_{j \in \mathcal{N}(i)} \alpha_{ij} \, W^{(l)} h_j^{(l)}, \tag{5}$$

where α_{ij} are attention weights derived from node features such as CPU/memory requirements and service type. The final layer applies global mean pooling to obtain a fixed-size DAG sketch $\phi(G_t) \in \mathbb{R}^{512}$, which serves as the entry representation for memory storage.

b) Graph Memory Representation.: Encoded DAG sketches are stored in a dynamic graph memory $\mathcal{G}_{\text{mem}} = (V, E, \mathcal{F}_v, \mathcal{F}_e)$, implemented in Memgraph. Each node v_i maintains a telemetry–intent embedding $\mathbf{q}_i$, the original service DAG $\mathcal{G}_i$, its placement $\mathcal{G}_i^{\text{map}}$, and a performance tuple $\text{perf}_i = (\text{lat}, \text{util}, \text{SLA})$. Edges capture structural or contextual similarity between deployments, while temporal decay down-weights stale examples to emphasize recent, high-fidelity traces.

c) Context-Aware Query Embedding.: At the decision step t, the encoder produces latent vectors.

$$\mathbf{z}_t^{\text{obs}} \in \mathbb{R}^d, \quad \mathbf{z}_t^{\text{task}} \in \mathbb{R}^{d_T}, \tag{6}$$

encoding network observations and service intent, respectively. These are fused into a unified query:

$$\mathbf{q}_t = \mathbf{W}_o \, \mathbf{z}_t^{\text{obs}} \parallel \mathbf{W}_s \, \mathbf{z}_t^{\text{task}} \parallel \phi(\mathcal{G}_t), \tag{7}$$

where $\mathbf{W}_o, \mathbf{W}_s$ are learnable projections and $\phi(\cdot)$ summarizes the service DAG via GNN pooling [3].

The query $\mathbf{q}_t$ and top-K contexts C_t are composed into an LLM prompt as:

```
Given the current network state and
service requirements, analyze K=50
similar deployments from C_t and recommend
an optimal placement plan with estimated
reward.
```

d) Similarity Metric.: We adopt a composite similarity measure combining (i) cosine similarity between telemetry–intent embeddings, (ii) Jaccard similarity over service DAGs, and (iii) an exponential decay term to down-weight stale traces [9]. The weights are normalized to sum to 1, and the top-K matches form a weighted candidate set:

$$\mathcal{C}_t = \{(\mathbf{p}_j, \mathbf{w}_j)\}_{j=1}^{|\mathcal{C}_t|}, \tag{8}$$

where $\mathbf{p}_j$ is a placement plan from exemplar graph $\mathcal{G}_i^{\text{map}}$ and $\mathbf{w}_j \propto \mathcal{S}(\mathbf{q}_t, \mathbf{q}_i)$.

The top-50 candidates are passed to the LLM, which refines them into a semantic prior:

$$(\Pi_t^\star, \hat{R}_t) \leftarrow \text{LLM}(\mathcal{C}_t, \mathbf{q}_t), \tag{9}$$

where $\Pi_t^\star$ is the recommended placement and $\hat{R}_t$ its estimated utility. During decoding, the LLM generates multiple candidate plans $\{\Pi_t^{(k)}\}$, each with a confidence score computed as

$$s_k = \frac{\exp(\ell_k)}{\sum_j \exp(\ell_j)}, \tag{10}$$

where ℓ_k is the log-likelihood of plan $\Pi_t^{(k)}$. The scores $\{s_k\}$ are used to rank the candidates, and the highest-confidence plan $\Pi_t^* = \arg\max_k s_k$ is selected as the primary output, while lower-confidence alternatives are retained only for uncertainty estimation. Finally, only Π_t^* is forwarded to guide the DRL agent's action sampling.

C. DRL Scheduler Bootstrapping at the Edge

To enable fast and adaptive scheduling under dynamic 6G conditions, we adopt a two-phase learning pipeline comprising an offline warm-start phase and an online decision phase.

a) Offline Bootstrapping and Meta-Learning: The cloud-based LLM creates several deployment samples $(\Pi_t^\star, \hat{R}_t)$ from simulated arrival data, labeling each with the corresponding edge state s_t, forming training samples $(s_t, \Pi_t^\star, \hat{R}_t)$ stored in the replay buffer. These initialize the DRL scheduler $\pi_\theta(a_t \mid s_t)$ via supervised behavior cloning, which reduces online convergence time by over $8\times$ compared to random starts. For robustness against network issues (e.g., RSU failures, bandwidth changes, node churn), the agent is fine-tuned across M perturbed environments. In each m, the agent updates its parameters over K steps and uses the Reptile meta-gradient for aggregation.

$$\theta_{\text{meta}} \leftarrow \theta - \alpha \sum_{m=1}^{M} \nabla_\theta \mathcal{L}_m, \tag{11}$$

where $\mathcal{L}_m$ is the PPO loss and α is the meta-learning rate. The resulting meta-parameters θ_{meta} encode topology-agnostic

Algorithm 1: RAG-DRL Scheduler

Input: Memory graph $\mathcal{G}_{\text{mem}}$, LLM $\mathcal{L}$, PPO agent π_θ
Output: Optimized policy π_θ^*, placement $\mathbf{X}_t$, routing $\mathbf{Y}_t$

for *planning cycle* $t = 1, \ldots, T_{\max}$ **do**
  Observe state $s_t = (\mathbf{r}_t, \mathbf{b}_t, \mathcal{Q}_t, \pi_t)$;
  foreach *DAG* $q \in \mathcal{Q}_t$ **do**
    Encode context $\mathbf{q}_t$; // Eq. (7)
    Retrieve top-K exemplars $\mathcal{C}_t$; // Eq. (8)
    $(\Pi_t^*, \hat{R}_t) \leftarrow \text{LLM}(\mathcal{C}_t, \mathbf{q}_t)$; // Eq. (9)
  Sample $a_t = (\mathbf{X}_t, \mathbf{Y}_t)$ **guided by** Π_t^*;
  Compute reward $r_t = R(s_t, a_t)$; // Eq. (4)
  Store transition (s_t, a_t, r_t, s_{t+1}) in buffer;
  if *buffer ready* **then**
    Estimate advantages $\hat{A}_t$; // Eq. (12)
    Update PPO: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{PPO}}(\theta)$; // Eq. (13)
  if *meta-learning phase* **then**
    for $m = 1, \ldots, M$ **do**
      Fine-tune on perturbed env. $\mathcal{E}_m$ for K steps;
    Apply Reptile meta-update to θ ; // Eq. (11)
  Update deployment map π_{t+1} and memory graph;

priors that enable rapid generalization during live deployment.

b) Online Deployment Phase: At each decision step t, the edge scheduler observes the current system state $\mathbf{s}_t = (\text{node load, mobility cues, task queue})$ and selects a placement-and-routing action $\mathbf{a}_t$. The environment responds with a scalar reward r_t, capturing the impact of the action on end-to-end latency, resource utilisation, and load balance. To estimate the policy advantage, we apply generalised advantage estimation (GAE) [10]:

$$\hat{A}_t = \sum_{\ell=0}^{T-t} (\gamma\lambda)^{\ell}\, \delta_{t+\ell}, \qquad \delta_t = r_t + \gamma V_\phi(\mathbf{s}_{t+1}) - V_\phi(\mathbf{s}_t), \tag{12}$$

where V_ϕ is the value network, γ is the discount factor, and λ controls the bias–variance trade-off. The policy π_θ is then updated using the clipped surrogate loss of PPO [10]:

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_t\Big[\min\big(\rho_t \hat{A}_t,\ \text{clip}(\rho_t, 1-\varepsilon, 1+\varepsilon)\hat{A}_t\big) - c_v\big(V_\phi(\mathbf{s}_t) - \hat{R}_t\big)^2 + c_e\, \mathcal{H}\big[\pi_\theta(\cdot \mid \mathbf{s}_t)\big]\Big], \tag{13}$$

where $\rho_t = \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_{\text{old}}}(\mathbf{a}_t|\mathbf{s}_t)}$ represents the importance sampling ratio, $\hat{R}_t = \sum_{k\geq 0} \gamma^k r_{t+k}$ is the reward, and $\mathcal{H}[\cdot]$ denotes the policy entropy. This lightweight training loop facilitates sub-millisecond scheduling and rapid policy adaptation, which are critical for latency-sensitive 6G applications. Algorithm 1 outlines the RAG-DRL meta-training process designed to promote generalization and robust performance across diverse and dynamic network conditions.

IV. Performance Evaluation

We evaluate our LLM-enhanced DRL scheduler against two key baselines under simulated urban 6G conditions. Our method is compared with: 1) Distributed Redundant Placement (DRP) [11], a distributed redundancy-aware scheme for microservice deployment, and 2) DQN Scheduler [2], which uses a multi-objective deep Q network to optimize container placement. We evaluated performance on latency, resource utilization, load imbalance, and deployment cost across varying mobility and traffic densities.

A. Dataset

Our evaluation utilizes three public datasets enhanced by preprocessing and synthetic enhancements to model a realistic 6G vehicle edge cloud environment. The Shanghai Telecom Base Station [12] dataset includes 3,233 cell tower locations grouped into 200 edge controllers (EC) to simulate a density of urban deployment. The Shanghai Taxi Trajectory dataset [12] provides minute-level GPS traces of 4,328 cabs, enabling accurate modeling of mobility patterns and dynamic spatial traffic distributions. Finally, Alibaba Cluster Trace [13] is used to generate containerized microservice workloads and service dependency graphs that reflect real-world resource usage and scheduling behavior.

B. Simulation Setup

1) Environment: We simulate a hierarchical urban network on AWS using a Docker-based testbed. A cloud core (`c6a.2xlarge`) with a MongoDB knowledge graph computes global plans every 30 minutes or on anomalies, which are refined by 16 edge controllers (`c6a.xlarge`) and pushed every 500 ms to 3,233 containerized ESPs mapped to Shanghai Telecom base stations. Calico enforces policies by routing requests to the nearest ESP, achieving 1–5 ms one-way latency. Mobility and traffic are modeled with 4,328 real taxi traces (0×–2× speed) and a nonstationary Poisson process (5–300 requests/s) with Pareto peaks. The simulation, implemented in Python with Ray and TensorFlow, was validated by 1,000 Monte Carlo runs to ensure 95% confidence.

2) LLM Backbone and Fine-Tuning: We use the *Qwen2-7B-Instruct* model (13.6 GB, FP16), fine-tuning the top 30% of its layers using LoRA (rank 16, $\alpha = 32$) and freezing the rest [14]. The model features three decoder heads for placement, utility, and reward predictions. Its RAG module manages 75K–150K subgraphs in Memgraph, encoded via a GAT. Training involves supervised fine-tuning of placement-utility pairs from Alibaba cluster dataset, followed by reinforcement-style optimization to align with scheduling goals.

3) Deep-RL Hyperparameters: Our PPO agent uses Adam optimizer with learning rates of 3×10^{-4} for the actor and 1×10^{-3} for the critic, together with the GAE parameters $\gamma = 0.99$ and $\lambda = 0.95$ [10]. Each training update consists of $2{,}048$ rollout steps, which are processed in mini-batches of 256 over 10 epochs. Key PPO settings include a clipping value of 0.2, a loss weight of 0.5 and an entropy coefficient that decreases from 0.01 to 0.001. All simulations are implemented in Python 3.11 using PyTorch 2.3 and HuggingFace Transformers [6].

C. Results and Analysis

In this paper, we evaluate performance based on four key metrics: cumulative reward, average end-to-end latency, SLA violation rate, and resource utilization. Scalability is evaluated using four deployment configurations that vary the number of microservices and replicas: from a basic version with 26 microservices and 3 replicas (26μS/3R) to a high-density configuration with 37 microservices and 8 replicas (37μS/8R).

Fig. 3 shows the convergence and learning efficiency of the training between different schedulers over 4,000 episodes. RAG-DRL achieves a reward of 0.6 within 400 episodes, significantly faster than DRP, which requires 800 episodes. Although the DRL scheduler initially shows rapid convergence to 1.0 reward by episode 600, it stagnates early on, indicating a trade-off between exploration and exploitation. DRP starts at 0.2 Reward and converges slowly, indicating high sample complexity.

Fig. 4 illustrates the scalability of latency with increasing network size. RAG-DRL maintains the most stable latency, which increases from 27 to 65 ms (2.4 times the increase). In contrast, DRL Scheduler's latency increases from 30ms to 95ms (3.2×), while DRP exhibits poor scalability, jumping from 25ms to 145ms (5.8×). As the size of the deployment

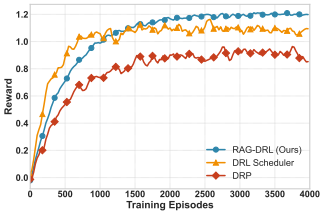

Fig. 3: Reward.

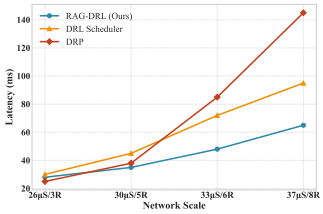

Fig. 4: Mean end-to-end latency.

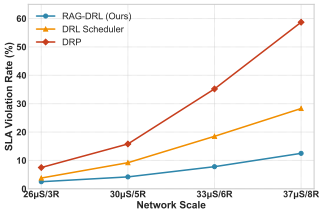

Fig. 5: SLA violation rate.

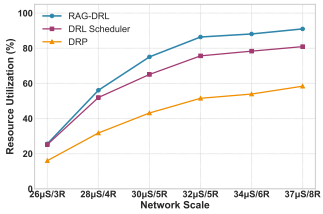

Fig. 6: Connection pool utilization.

increases, RAG-DRL retains a significant latency advantage of 30 ms over DRL Scheduler and 80 ms over DRP at the largest scale.

Fig. 5 presents the SLA violation rates under different deployment scales, highlighting the reliability of each planning approach. RAG-DRL achieves the strongest compliance, with violations increasing only from 2% at small scale (26µS/3R) to 12% at large scale (37µS/8R), demonstrating enterprise-grade reliability. In contrast, the DRL scheduler shows moderate performance, ranging from 3% to 28%, while the DRP performs poorly, increasing from 7% to 58%. Thus, at the largest scale, RAG-DRL maintains a clear advantage of 16% over DRL and 46% over DRP.

Fig. 6 analyzes resource utilization in network sizes. RAG-DRL consistently delivers the most efficient allocation, improving from 25% to 90% utilization as the network scales. DRL follows closely, increasing from 25% to 81%, whereas DRP lags behind, reaching only 58%. On a maximum scale, RAG-DRL achieves 9% higher utilization than DRL and 32% higher than DRP, reflecting significantly reduced resource waste. RAG-DRL excels in key metrics, converging faster, scaling effectively, maintaining SLA violations under 12% even under heavy load, and ensuring near-optimal infrastructure utilization.

V. Conclusion

In this paper, we presented a hybrid RAG-DRL scheduler that integrates a RAG-enhanced LLM with a lightweight DRL agent to address dynamic microservice placement and routing in 6G vehicular edge networks. Our approach leverages semantic cues from historical data to accelerate policy adaptation and improve scalability in rapidly changing environments. Extensive evaluations show that our scheduler significantly outperforms the current state-of-the-art by achieving faster convergence, maintaining low end-to-end latency under high traffic density, and ensuring high SLA compliance with optimal resource utilization. By effectively combining semantic reasoning with real-time control, the RAG-DRL framework delivers a robust and efficient solution for dynamic network management, improving both service reliability and system efficiency. For future work, we aim to enhance the framework for multi-LLM collaboration in cross-domain orchestration, explore online continual learning for adapting to workload changes, and add energy-awareness to optimize sustainability and performance.

References

[1] S. Chou, J. Hribarl, I. Dusparic, and C. Fortuna, "Towards 6g for connected autonomous vehicles: A trial facility analysis," in *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2024, pp. 01–06.

[2] J. B. Ssemakula, J.-L. Gorricho, G. Kibalya, and J. Serrat-Fernandez, "An artificial intelligence strategy for the deployment of future microservice-based applications in 6g networks," *Neural Computing and Applications*, vol. 36, no. 18, pp. 10 971–10 997, 2024. [Online]. Available: https://doi.org/10.1007/s00521-024-09643-9

[3] S. Chen, Q. Yuan, J. Li, H. He, S. Li, X. Jiang, and J. Yang, "Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3742–3757, 2024.

[4] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4794–4806, 2022.

[5] G. O. Boateng, H. Sami, A. Alagha, H. Elmekki, A. Hammoud, R. Mizouni, A. Mourad, H. Otrok, J. Bentahar, S. Muhaidat, C. Talhi, Z. Dziong, and M. Guizani, "A survey on large language models for communication, network, and service management: Application insights, challenges, and future directions," *IEEE Communications Surveys& Tutorials*, pp. 1–1, 2025.

[6] G. Sun, Y. Wang, D. Niyato, J. Wang, X. Wang, H. V. Poor, and K. B. Letaief, "Large language model (llm)-enabled graphs in dynamic networking," *IEEE Network*, pp. 1–1, 2024.

[7] L. Wang, X. Deng, J. Gui, X. Chen, and S. Wan, "Microservice-oriented service placement for mobile edge computing in sustainable internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 9, pp. 10 012–10 026, 2023.

[8] Y. Hu, H. Wang, L. Wang, M. Hu, K. Peng, and B. Veeravalli, "Joint deployment and request routing for microservice call graphs in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 11, pp. 2994–3011, 2023.

[9] I. Radeva, I. Popchev, and M. Dimitrova, "Similarity thresholds in retrieval-augmented generation," in *2024 IEEE 12th International Conference on Intelligent Systems (IS)*, 2024, pp. 1–7.

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

[11] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundant placement for microservice-based applications at the edge," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1732–1745, 2022.

[12] K. Cheng, S. Zhang, M. Liu, Y. Gu, L. Wei, H. Cheng, K. Liu, Y. Song, X. Shi, A. Zhu, and L. Tang, "Geoscale: Microservice autoscaling with cost budget in geo-distributed edge clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 4, pp. 646–662, 2024.

[13] S. Luo, H. Wang, and J. Li, "Prediction-based resource allocation for microservice deployment in edge computing," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9876–9889, 2022.

[14] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang *et al.*, "Qwen2. 5-vl technical report," *arXiv preprint arXiv:2502.13923*, 2025.