Complex & Intelligent Systems (2026) 12:141
https://doi.org/10.1007/s40747-026-02249-9

ORIGINAL ARTICLE

AI-native cloud-edge orchestration for 6G metaverse networks: an LLM-guided multi-agent DRL approach

Daniel Ayepah-Mensah[1,2] · Amine Kidane Ghebreziabiher[1,2] · Gordon Owusu Boateng[3] · Rabeb Mizouni[1,4] · Azzam Mourad[1,2,5] · Hadi Otrok[1,4] · Jamal Bentahar[1,2,6]

Received: 8 August 2025 / Accepted: 18 January 2026 / Published online: 16 March 2026

Abstract

Emerging metaverse experiences, including interactive extended reality (XR) sessions and live holographic telepresence, necessitate motion-to-photon latencies of less than 10 ms. These applications must also manage the continuous streaming of multi-gigabit data volumes to thousands of mobile users. To meet these extreme requirements, an orchestration layer capable of instantly decomposing, placing, and adapting the dependency structures of microservices formally modeled as directed acyclic graphs (DAGs) underlying computationally intensive artificial intelligence (AI)-driven immersive applications is required. We propose an AI-native cloud-edge orchestration framework in which a Large Language Model (LLM) based cloud planner serves as a cognitive conductor. This planner uses Topology-Aware Retrieval-Augmented Generation (TopoRAG) to retrieve and interpret historical deployment traces to create latency-optimized orchestration plans. Trust-weighted logits, semantic cost estimates, and initial node bindings are output as soft priors and streamed to decentralized edge workers powered by deep reinforcement learning (DRL) with multiple agents. These DRL agents integrate global intentions with rapidly changing local conditions to enable real-time context-aware planning. In addition, we introduce a deviation-based reward mechanism that compares actual execution costs with estimates predicted by the LLM, providing dense and informative feedback that effectively halves the DRL convergence time. Simulations in urban-scale 6G networks with real-time volumetric video stitching and multiuser XR gaming workloads show a significant reduction in SLA violations and significantly lower end-to-end latency compared to baseline schedulers, while maintaining optimal motion-to-photon latency.

Keywords AI-native orchestration · Microservice placement · Edge computing · Large language models · Deep reinforcement learning · Retrieval-augmented generation (RAG)

Introduction

The convergence of sixth-generation (6G) networks with the Metaverse represents a significant paradigm shift in

✉ Azzam Mourad
azzam.mourad@ku.ac.ae

Daniel Ayepah-Mensah
daniel.mensah@ku.ac.ae

Amine Kidane Ghebreziabiher
amine.kidane@ku.ac.ae

Gordon Owusu Boateng
gordon.boateng@xjtlu.edu.cn

Rabeb Mizouni
rabeb.mizouni@ku.ac.ae

Hadi Otrok
hadi.otrok@ku.ac.ae

Jamal Bentahar
jamal.bentahar@ku.ac.ae

[1] Department of Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates

[2] KU 6G Research Center, Department of Computer Science, Khalifa University, 127788, Abu Dhabi, UAE

[3] Department of Communications and Networking, Xi'an Jiaotong-Liverpool University, Suzhou Industrial Park, Suzhou, Jiangsu, China

[4] Center for Cyber-Physical Systems (C2PS), Department of Computer Science, Khalifa University, Abu Dhabi, UAE

[5] Artificial Intelligence and Cyber Systems Research Center, Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

[6] Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada

telecommunications, driven by demanding applications such as extended reality (XR), holographic telepresence, and digital twins [1]. Beyond entertainment and collaborative XR, emerging Metaverse-ready infrastructures are expected to support critical services such as large-scale environmental monitoring and disaster management, where IoT and WSN deployments provide early-warning and situational awareness capabilities [2]. Supporting these immersive applications requires extraordinary network performance, characterized by stringent requirements including sub-10 ms motion-to-photon latency and continuous multigigabit-per-second volumetric data streaming for simultaneous delivery to thousands of users [3]. Achieving this requires a fundamental evolution from traditional connectivity-centric paradigms toward experience-centric network architectures [4]. Such advanced architectures must efficiently orchestrate complex microservice workflows in real time, ensuring seamless immersion and preserving the fidelity of interaction essential to the integrity and realism of Metaverse experiences [1].

Meeting the stringent demands of the Metaverse requires distributed microservice architectures in which containerized functions are dynamically provisioned on edge infrastructures [5]. Beyond conventional deployment, Metaverse requires complex replication strategies to ensure fault tolerance and ultra-low latency under high mobility [1, 4]. Orchestrating such interdependent services, organized as large-scale directed acyclic graphs (DAGs), presents a combinatorial placement problem where real-time optimization is often infeasible [6]. This challenge is compounded by the stochastic nature of 6G edge environments, characterized by variable wireless conditions and heterogeneous hardware. Thus, the challenge is to balance fast and heuristic-driven decisions with the computational demands of semantically informed and globally efficient resource allocation, a capability existing frameworks still lack [7].

Existing orchestration approaches are insufficient for the dynamic complexity of 6G-Metaverse environments. Traditional heuristic methods are overly static and reactive, lacking topology awareness and not adequately representing dependencies within DAG-based microservice workflows [6]. Despite their enhanced adaptability, classical deep reinforcement learning (DRL) methods suffer from limited local visibility and poor generalization, with performance deteriorating rapidly when faced with changes in infrastructure or service patterns [8]. Graph Neural Network (GNN)-based approaches effectively encode structural relationships but are critically sensitive to topology changes and typically assume centralized visibility, which contradicts the inherently distributed nature of the edge environment [9]. Furthermore, forecast-guided orchestration, while beneficial for proactive decision-making, remains dependent on prediction accuracy. Its reliance on offline training with static network snapshots makes it incompatible with the real-time adaptation required by rapidly evolving 6G-Metaverse scenarios [3].

Beyond these technical limitations, current orchestration frameworks display a fundamental semantic gap between high-level service intent and low-level resource allocation decisions, a limitation that existing Explainable AI (XAI) studies have only partially addressed. Traditional XAI approaches in network management focus mainly on post-hoc interpretability, providing explanations for decisions already made by opaque DRL or GNN models. While these methods improve transparency, they do not address the root cause: the models themselves lack inherent semantic reasoning capabilities and treat all workloads uniformly, without distinguishing diverse service requirements. Recent advances in semantic and contextual learning have enabled DRL agents to reason over complex multimodal constraints and align actions with abstract service objectives [10]. However, these approaches still function as black-box optimisers and provide limited insight into why specific orchestration decisions are made. This remains a critical shortcoming for safety-critical 6G–Metaverse deployments, where operators must be able to validate and audit automated decisions.

The emergence of Large Language Models (LLMs) presents a paradigm shift by bridging symbolic reasoning with subsymbolic learning through natural language understanding and structured output generation [11]. Unlike post-hoc XAI methods that retrofit explanations onto opaque models, LLMs offer inherently interpretable reasoning through chain-of-thought generation, allowing operators to inspect the rationale behind each orchestration decision [12]. However, operating in isolation, LLMs are prone to hallucinations that yield infeasible resource allocations, necessitating robust grounding mechanisms for safety. Although Retrieval-Augmented Generation (RAG) offers a solution incorporating historical deployment knowledge [13], conventional RAG frameworks rely on static retrieval and lack real-time telemetry integration. This leads to temporal misalignment, a critical flaw in mobile 6G-Metaverse environments where extreme latency sensitivity and dynamic user mobility demand highly responsive, context-aware orchestration.

Despite growing interest in explainable network automation, a critical gap persists as no existing framework unifies real-time grounded reasoning, inherent interpretability, and adaptive orchestration within a single architecture. Current XAI efforts tend to apply post-hoc explanations to black-box models without improving their underlying reasoning, rely on LLMs for network tasks without sufficient grounding to prevent hallucinations, or use RAG over static corpora that are disconnected from live network states. To address these limitations, we propose an AI-native orchestration framework that integrates domain-specific LLM planning with a dynamic, topology-aware Retrieval-Augmented Generation

(TopoRAG) module. A cloud-resident cognitive planner uses an LLM to reason over network states, historical data, and QoS constraints and then generates strategies that act as high-quality priors, reducing edge-side trial-and-error. TopoRAG grounds this reasoning by retrieving structurally relevant patterns from a continuously updated graph-based memory of service DAGs and topology. Within this hybrid architecture, the LLM provides long-horizon semantic guidance, while decentralized MAPPO agents perform fine-grained, real-time adaptations. A trust-weighted mechanism dynamically balances global foresight and local autonomy, ensuring robust, SLA-compliant deployment in latency-critical 6G–Metaverse environments. The main contribution of our work is summarized as follows:

1. We propose an AI-native cloud-edge orchestration framework for 6G Metaverse environments, where a partially frozen, fine-tuned LLM acts as a centrally hosted cognitive planner that generates deployment priors from live topology, telemetry, and service DAG inputs, while edge MAPPO agents refine these plans in real time to keep LLM inference off the per-request critical path and satisfy ultra-low-latency requirements.
2. We introduce TopoRAG, a topology-aware retrieval-augmented generation module that unifies real-time telemetry, network graphs, and service DAGs into a dynamic graph-based memory, using adaptive soft pruning, GNN-based re-ranking, and sharding policies to retrieve structurally relevant traces and enable grounded, topology-sensitive orchestration at MEC scale.
3. We propose an LLM-guided, trust-weighted reward shaping mechanism in which the LLM encodes constraint priorities, penalties, and cooperation biases, and edge agents modulate this guidance via a trust coefficient, steering policies toward globally consistent behaviors, accelerating DRL convergence, and reducing SLA violations under non-stationary, bursty traffic.
4. We perform extensive evaluations across diverse city-scale cloud-edge topologies and realistic Metaverse workloads including real-time volumetric video stitching, multiuser XR gaming, and digital twin services benchmarking against strong baselines, performing ablations and stress tests, and reporting both end-to-end latency/SLA gains and a deployment-oriented overhead budget for planner decoding, dispatch, and TopoRAG lookup.

The remainder of this paper is organized as follows. "Related works" section reviews related works. "System model" section details the system model. "Problem formulation" section presents the formulation of the problem. "LLM-driven cloud planning and multi-agent microservice deployment" section presents the proposed algorithm. "Performance evaluation" section describes the evaluation of the proposed algorithm, and "Conclusion" section concludes with future research directions.

Related works

Microservice placement and resource allocation in edge—cloud systems have been extensively explored using mixed-integer linear programming (MILP), heuristic algorithms, and DRL-based methods to optimize latency, resource utilization, and system reliability [7]. While DRL approaches offer superior adaptability to dynamic network conditions, most existing single-agent and multi-objective methods rely on centralized state assumptions and treat services as linear chains or independent tasks. Consequently, they fail to capture the complex interdependencies and precedence constraints inherent in DAG-based service workflows. Although frameworks such as [14, 15] demonstrate improvements in delay and energy efficiency, they are based on centralized observations and lack support for general DAG structures, thus limiting their applicability in distributed edge environments. Most DRL-based placement schemes suffer from structural awareness deficiencies and poor generalization capabilities. These approaches focus on specific traffic patterns and cannot adequately model the complex dependencies and parallelism characteristic of DAG-based workflows [15]. As a result, their learned policies struggle to adapt to dynamic topologies or previously unseen network configurations in heterogeneous edge–cloud environments.

To address scalability challenges, multiagent DRL enables decentralized decision-making at individual edge nodes. Distributed actor–critic frameworks reduce communication overhead by allowing agents to learn and update local policies independently while sharing only minimal coordination signals or value estimates, rather than complete system states [16]. In DRL-based microservice deployment, the environment is modeled as a Markov Decision Process (MDP), where agents learn placement policies through system state observations and reward feedback. However, flat vector representations fail to capture the inherent graph structure of both workflows and network topologies, which significantly limits the agents' ability to reason about dependencies and constraints effectively.

Graph Neural Networks (GNNs) have emerged as a promising approach to optimize service placement by learning complex network relationships. Some methods employ Graph Convolutional Networks (GCNs) to jointly embed service and infrastructure graphs to identify low-latency placements, while others use message passing to learn effective task–resource mappings [17]. These models typically encode the state of the system in a unified global graph and

assume complete visibility of both the service DAG and the network topology [9]. This assumption is impractical in edge environments, where nodes have only local views and must react to asynchronous changes, making purely global encodings fragile. Hybrid frameworks that pair centralized learners with distributed agents offer partial adaptability but often rely on offline training over static snapshots and thus struggle with real-time microservice deployment [18]. Effective solutions therefore require online planning that adapts to evolving service graphs and resource fluctuations while coordinating placement across the edge. From a proactive perspective, GH-Twin uses hierarchical digital twins and graph learning for self-healing, but targets fault diagnosis rather than latency-critical Metaverse orchestration [19].

Recent advances have integrated LLMs into edge–cloud orchestration frameworks, positioning them as high-level semantic planners for complex scheduling tasks. Early works, such as [20], demonstrate how LLMs can encode procedural knowledge and provide semantic abstractions that guide learning agents. In [21], LLMs are used to initialize DRL agents, improving both convergence speed and policy generalization capabilities. To improve adaptability to network-specific scenarios, Wu et al. [22] propose a structured input encoding method that transforms flow-level telemetry and routing policies into prompt templates, enabling context-aware decision-making through LLM. Using natural language interfaces, the authors of [23] employ LLMs to generate Software-Defined Networking (SDN) flow rules and virtual network function mappings. However, their approach does not account for the DAG structure of applications or the dependencies among service functions. In [24], the authors propose a token-level RAG framework for service placement that integrates tokenized deployment traces and performance metadata. Despite this innovation, the framework's text-based retrieval mechanism overlooks topology-aware relational dependencies, execution order constraints, and dynamic adaptability to evolving network and service topologies.

Beyond networking and orchestration, recent work increasingly uses pre-trained vision–language models (VLMs) for security analytics and semantic communications in cyber-physical systems. For example, [25] shows that standard VLMs can parse diagrammatic Internet of Medical Things architectures to detect cyber threats and recommend mitigations with minimal engineering effort, suggesting that multimodal foundation models can act as lightweight decision modules in safety-critical settings. In vehicular networks, [10] combines semantic feature extraction based on LLaVA, which reduces the data volume by over 90%, with GAE-PPO for robust transmission decisions in dynamic channels. Similarly, [26] proposes G-MSC, a multimodal generative semantic communication framework for V2X that optimizes semantic encoding, robustness, and decoding in both analog and digital modes. These approaches primarily use VLMs and generative AI for semantic encoding and data compression to enhance link-level robustness, rather than for system-level planning and network-wide orchestration of microservice placement, offloading, and scheduling in dynamic edge–cloud environments.

Previous work in microservice orchestration reveals three critical limitations that render existing approaches inadequate for 6G-Metaverse environments. First, traditional DRL and heuristic methods lack semantic awareness, relying exclusively on numerical optimization while ignoring service intent and historical deployment knowledge. Second, most frameworks fail to address fine-grained mobility scenarios by overlooking rapid user movement, dynamic handovers, and spatially varying QoS requirements, which are critical factors for mobile immersive applications. Third, many approaches have been evaluated only on synthetic workloads or small-scale testbeds, falling short of addressing the scale, heterogeneity, and ultralow latency demands of real-world XR, holographic telepresence, and digital twin services. To overcome these limitations, we introduce a cognitive multimodal planner that coordinates a network of edge agents for semantic-guided microservice placement and routing. By integrating the symbolic reasoning capabilities of LLMs with a TopoRAG system, our planner generates intent-aligned deployment plans. These plans serve as policy priors for decentralized MAPPO-based edge agents, enabling local adaptation to real-time conditions while maintaining global alignment with ultralow-latency and resource-efficient orchestration objectives in 6G-Metaverse systems.

System model

System architecture

Figure 1 illustrates the proposed hierarchical 6G-enabled Metaverse edge computing architecture, which comprises four interconnected layers: Cloud Core, Edge-Cloud, Network Edge, and Device Layers. This architecture is designed to support latency-critical services, such as XR content delivery, digital twin synchronization, and holographic communication, by providing ultralow latency and adaptive resource provisioning [27].

The Cloud Core Layer serves as the central orchestration hub, overseeing global connectivity and semantic coordination. It hosts service registries, network topology databases, digital twin repositories, and containerized application managers. A centralized cloud planner, powered by an LLM and TopoRAG, conducts intent-driven global planning. This planner translates high-level application intents, such as latency constraints or QoS targets, into executable

Table 1 Summary of notation

Symbol	Description	Symbol	Description
$\mathcal{G}_{\text{infra}} = (\mathcal{N}, \mathcal{L})$	Substrate (edge–cloud) graph	$\mathbf{C}_n$	Capacity vector of node n
$\boldsymbol{\ell}_n$	Latency tuple $(\ell^{\text{comp}}, \ell^{\text{radio}}, \ell^{\text{back}})$	$\mathcal{F}_n$	Multi-band radio profile of n
$\mathcal{G}_q = (\mathcal{M}_q, \mathcal{E}_q)$	Enhanced service DAG	Θ_q, Δ_q	Resource-demand and data-payload sets
$\mathcal{T}_q$	QoS profile of request q	$\mathbf{X}_t, \mathbf{Y}_t$	Placement/routing matrices (step t)
$\mathcal{P}_{ij}$	Candidate path set for edge (i, j)	$y^q_{ij,p,\sigma,t}$	Flow fraction on path p
R^r_n	Capacity of node n for resource r		
θ^r_m	Demand of microservice m on resource r	$r^{\text{g}}_t,\ r^{\text{l},k}_t,\ r^{\text{pen},k}_t$	Global, local, and constraint-penalty reward
$\mathbf{z}_t$	Multimodal fused state	$\mathbf{q}_{\text{ctx}}$	TopoRAG query vector
$\mathbf{w}^{\text{LLM}}, \mathbf{p}^{\text{LLM}}$	Reward weights and placement plan	$\boldsymbol{\psi}^{\text{LLM}}_t, \phi^{\text{LLM}}$	Penalty weights and coop. bias
$\mathbf{z}^{\text{plan}}$	Compact plan embedding	$\hat{R}^{(b)}_{T,k}$	Utility of plan b for agent k
λ_{KL}	KL-loss weight	λ_{ent}	Entropy-bonus weight
τ	Decay constant in $\xi_\tau(t)$	α	Trust-sensitivity constant
$\omega_i, \boldsymbol{\omega}^{\text{LLM}}$	Static and LLM-derived reward weights	$\rho_j, \boldsymbol{\psi}^{\text{LLM}}_t$	Static and LLM penalty weights
$\bar{\ell}_t, \bar{\ell}^k_t$	Global and local average latency	$\mathcal{LB}_t, \mathcal{LB}^k_t$	Global and local load imbalance
$\mathbb{R}_{\text{tot},t}$	Total normalized resource use	$\overline{P}_t$	Average packet-success probability
κ^k_t	Trust coefficient (agent k)	ϕ^k_t	Cooperation factor

deployment plans by analyzing historical deployment patterns, real-time telemetry, and service-level objectives.

The edge-cloud layer consists of geographically distributed Edge Controllers (ECs) that function as regional orchestrators. Each EC manages a cluster of Edge Servers (ESs) and refines the global plans into region-specific orchestration strategies. These controllers facilitate localized decision making by continuously monitoring infrastructure conditions and adjusting resource allocations using lightweight distributed DRL agents.

The Network Edge Layer is composed of densely deployed ESs (often embedded within small cells, roadside units, or urban infrastructure) that execute microservices in close proximity to end-users. This proximity is crucial for achieving the motion-to-photon latencies required for immersive Metaverse experiences. Finally, the Device Layer encompasses end-user equipment, including XR headsets, holographic terminals, autonomous vehicles, and IoT sensors. These devices continuously generate multimedia and telemetry data and issue service requests to nearby ESs to support real-time interaction within blended virtual–physical environments. Detailed notations used are summarized in Table 1.

Network model

As illustrated in Fig. 1, we abstract the hierarchical cloud edge infrastructure into an undirected multitier graph $\mathcal{G}_{\text{infra}} = (\mathcal{N}, \mathcal{L})$, whose vertex set is partitioned as

$\mathcal{N} = \mathcal{N}_{\text{core}} \cup \mathcal{N}_{\text{EC}} \cup \mathcal{N}_{\text{ES}}$ is partitioned into *Cloud-Core nodes* ($\mathcal{N}_{\text{core}}$), *Edge Controllers* ($\mathcal{N}_{\text{EC}}$), and *Edge Servers* ($\mathcal{N}_{\text{ES}}$). Let $\mathcal{K}$ denote the set of ECs. Each controller $k \in \mathcal{K}$ governs a disjoint cluster of ESs, $\mathcal{N}_k \subset \mathcal{N}_{\text{ES}}$, plus its own control node, so that $\bigcup_k \mathcal{N}_k = \mathcal{N}_{\text{ES}}$ and $\mathcal{N}_k \cap \mathcal{N}_{k'} = \varnothing$ for $k \neq k'$.

Every ES $n \in \mathcal{N}_{\text{ES}}$ is annotated with (i) a *resource capacity vector* $\mathbf{C}_n = (C^{\text{cpu}}_n, C^{\text{mem}}_n, C^{\text{gpu}}_n, C^{\text{npu}}_n)$, (ii) a *radio profile* $\mathcal{F}_n \subset \{\text{sub-6}, \text{mmW}, \text{THz}\}$, and (iii) a latency tuple $\boldsymbol{\ell}_n = (\ell^{\text{compute}}_n, \ell^{\text{radio}}_n, \ell^{\text{backhaul}}_n)$. EC and Cloud Core nodes are modeled identically, with the Cloud Core offering higher backhaul capacity and practically unbounded compute. Each bidirectional edge $(u, v) \in \mathcal{L}$ stores the available bandwidth B_{uv}, the propagation delay d_{uv}, and an application class tag Σ_{uv}.

Service model

To support both massive and immersive communication, service requests are modeled as Metaverse-aware microservice chains. As illustrated in Fig. 1, each service request originates from different application domains such as Smart Home Integration, Digital Twin Services, Vehicular-to-Vehicular communication, and Industrial IoT–and is represented as a DAG with vertices denoting microservices and edges encoding invocation dependencies [28, 29]. For service-class differentiation, incoming requests are categorized into immersive services ($\mathcal{Q}_{\text{imm}}$), such as XR and haptics; massive connectivity requests ($\mathcal{Q}_{\text{mass}}$), such as IoT sensor fusion; and digital twin services ($\mathcal{Q}_{\text{twin}}$) [30]. Each service request q is defined by an enhanced service DAG $\mathcal{G}_q \triangleq (\mathcal{M}_q, \mathcal{E}_q, \Theta_q, \Delta_q, \mathcal{R}_q, \mathcal{T}_q)$, where $\mathcal{M}_q$ is the set of

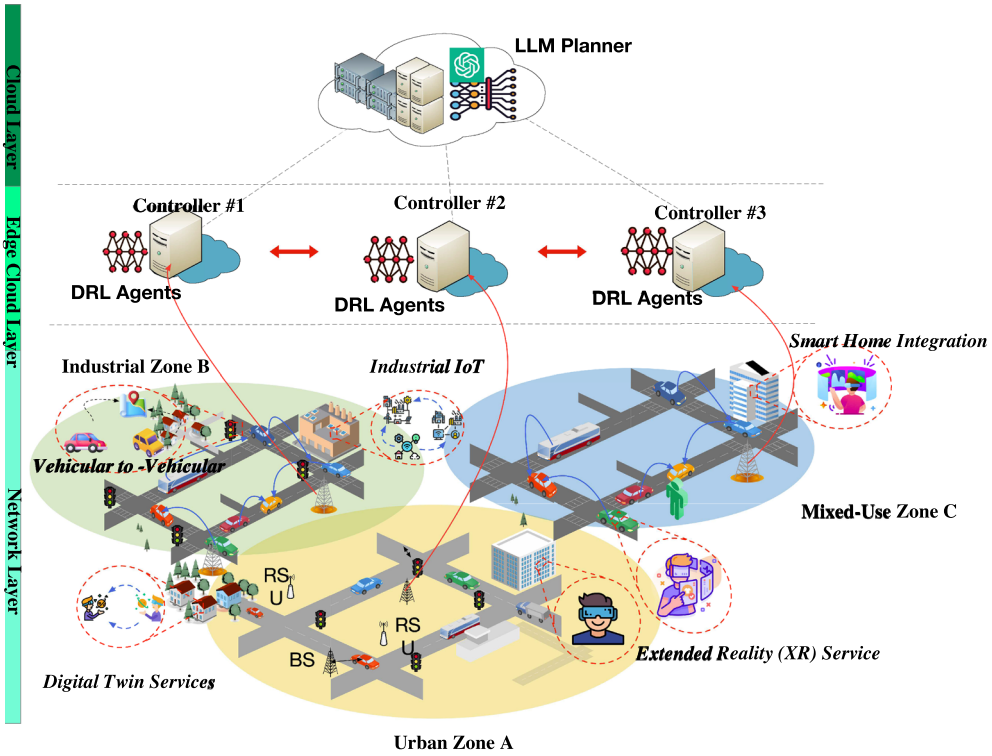


Fig. 1 System model of the AI-native cloud–edge orchestration framework for 6G-enabled Metaverse services

microservices, $\mathscr{E}_q$ represents directed edges encoding task dependencies, $\Theta_q = \{\boldsymbol{\theta}_i\}$ specifies resource demand vectors for each microservice with $\boldsymbol{\theta}_i = (r_i^{\text{cpu}}, r_i^{\text{mem}}, b_i^{\text{radio}})$, Δ_q denotes the data payload on each edge, $\mathscr{R}_q$ defines the replication limits per microservice and $\mathscr{T}_q$ captures QoS requirements. The QoS profile $\mathscr{T}_q$ imposes service-level constraints such as end-to-end latency $\mathscr{T}_q^{\text{e2e}} \leq \tau_{\max}$, jitter $\sigma_q^{\text{jitter}} \leq \sigma_{\max}$, synchronization tolerance $\Delta t_q^{\text{sync}} \leq \delta_{\text{sync}}$, and minimum success probability $P_q^{\text{success}} \geq P_{\min}$.

Cloud controller

Mobile Metaverse applications, including immersive XR services, massive IoT connectivity, and real-time digital twins, demand dynamic orchestration that traditional rule-based controllers cannot provide. These computationally intensive applications exhibit variable workloads, evolving topologies, and strict latency requirements that challenge conventional approaches. The 6G Metaverse expands this complexity with heterogeneous resource spaces spanning CPU, memory, GPU, NPU, storage, and radio bandwidth across diverse edge and cloud nodes [28]. Current reactive controllers lack semantic awareness of user intent and service interdependencies, resulting in suboptimal placement and routing decisions.

We propose a semantic cognitive controller that serves as the central orchestration planner, performing goal-conditioned reasoning over multi-modal input to generate adaptive, topology-aware deployment policies in real time. This controller leverages an RAG-enabled LLM to transform historical deployment knowledge, service intent, and system constraints into structured plans. These plans guide the edge level toward goal-aligned execution while meeting the stringent requirements of immersive Metaverse applications through intelligent resource allocation and semantic planning. The cloud tier hosts a policy-driven LLM orchestrator that converts global objectives into edge-executable directives while minimizing control-plane latency. The cognitive planner proposed for the deployment of intelligent microservices consists of the following stages, as shown in Fig. 2:

Modality-specific encoders

To enable semantic reasoning across heterogeneous observations in the 6G-enabled Metaverse environment, we design a set of modality-specific encoders aligned with the system model:

- *Visual–spatial data* Heat maps and camera feeds from ES nodes $n \in \mathcal{N}$ are encoded by a Vision Transformer (ViT) to capture the dynamics of the scene [31].
- *Telemetry* Throughput, delays $(\ell_n^{\text{compute}}, \ell_n^{\text{radio}}, \ell_n^{\text{backhaul}})$, and resource usage $\mathbf{C}_n$ are modeled by a dilated Tem-

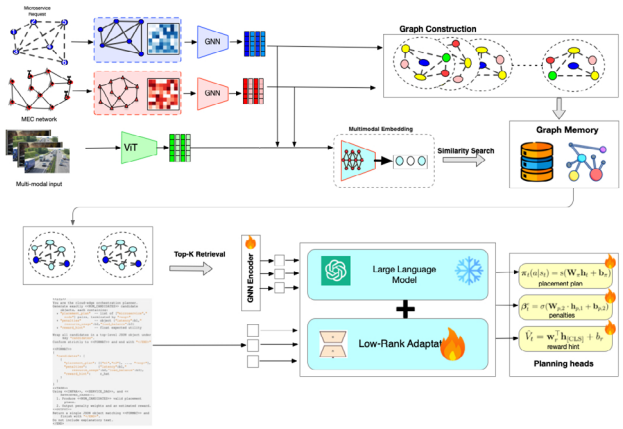


Fig. 2 TopoRAG—a topology-aware retrieval-augmented generation framework

poral Convolutional Network (TCN) for long-range patterns [32].

- *Graph inputs* Service DAGs $\mathcal{G}_q$ and network topology $\mathcal{G}_{\text{infra}}$ are embedded in GraphSAGE, aggregating the multi-hop context [33].

The output embeddings of all modalities are concatenated and projected into a unified latent vector $\mathbf{z}_t \in \mathbb{R}^d$, which serves as input for the retrieval and deployment planning stages of the LLM controller.

Topology-aware retrieval-augmented generation

To accelerate convergence and curb rollout inefficiency in latency-sensitive edge settings, we embed a topology-aware retrieval-augmented generation (TopoRAG) layer within the LLM controller as shown in 2. At every decision step, the LLM constructs a contextual query $\mathbf{q}_{\text{ctx}} = [\mathbf{z}_t \,\|\, \texttt{slice\_tag} \,\|\, \texttt{policy\_id}]$, which is matched against a persistent memory graph $\mathcal{G}_{\text{mem}}$ (stored in Memgraph [34]). Each memory graph node has a deployment trace $\tau_q = (\mathcal{G}_q,\ \mathcal{G}_q^{\text{map}},\ \mathbf{q}_{\text{ctx}},\ \text{trace}_q,\ \text{perf}_q)$, where $\mathcal{G}_q$ is the DAG service, $\mathcal{G}_q^{\text{map}}$ the associated placement-route graph, trace_q the rollout log and perf_q the QoS summary (e.g. latency $\mathcal{T}_q^{\text{e2e}}$, sync delay Δt_q^{sync} and success rate P_q^{success}).

Adaptive soft pruning

To remove stale or noisy patterns, each subgraph retrieved g_j is filtered through attention weights.

$$\alpha_v = \sigma\big(\text{MLP}_v\big(\|\mathbf{z}_v - \mathbf{q}_{\text{ctx}}\|_2,\ \mathbf{f}_v,\ \Delta t_v\big)\big), \tag{1a}$$

$$\alpha_e = \sigma\big(\text{MLP}_e\big(\|\mathbf{z}_e - \mathbf{r}_e\|_2,\ \mathbf{f}_e,\ \Delta t_e\big)\big), \tag{1b}$$

where $\mathbf{z}_v, \mathbf{z}_e$ are the embeddings of the nodes / edges, $\mathbf{f}_v, \mathbf{f}_e$ structural features, Δt age indicators, and $\mathbf{r}_e$ the edge prototype expected under the current topology.

GNN-based reranking

A lightweight GNN refines the pruned subgraphs via message passing

$$\mathbf{h}_v^{(l+1)} = U^{(l)}\Big(\mathbf{h}_v^{(l)},\ A^{(l)}\big(\{\alpha_{uv}\mathbf{h}_u^{(l)} : u \in \mathcal{N}(v)\}\big)\Big), \tag{2}$$

and computes the relevance at the node level $\text{relevance}(v) = \mathbf{q}_{\text{ctx}}^{\top}\mathbf{h}_v^{(L)}$.

The multitask decoder is supported by the top-k subgraphs, which are ranked based on collective relevance. This approach ensures that design plans, reward weights, and penalty coefficients are all derived from the most contextually significant past deployments.

Partially frozen LLM architecture

To translate the high-level service intents into infrastructure-aware placement plans, we employ a cloud-resident LLM controller that jointly reasons over the semantics of the service and the state of the infrastructure $\mathcal{G}_{\text{infra}}$ over time [22]. Fully fine-tuning a multibillion-parameter transformer would be prohibitive at run-time and risks catastrophic overfitting. Instead, we adopt a partially-frozen attention (PFA) design, keeping the general reasoning skills of the model intact while exposing a lightweight adaptation head for network-specific knowledge.

Layer freezing

Given an L layer transformer, the lower $L_{\text{base}} \approx 0.7L$ layers are frozen, preserving their broad linguistic priors; only the upper $L_{\text{adapt}} = L - L_{\text{base}}$ layers are trainable. This reduces the trainable parameters from $\mathcal{O}(10^9)$ to $\mathcal{O}(10^7)$, i.e. $\sim 0.1\%$ of a 7-B model [22].

LoRA-based adaptation

Each trainable attention or feedforward block is augmented with low-rank adapters $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ $(r \ll d)$:

$$\text{Proj}_{\text{adapted}}(x) = \underbrace{\text{Proj}_{\text{frozen}}(x)}_{\text{backbone}} + A\,B\,x. \tag{3}$$

Adapters are added to the $\{Q, K, V\}$ projections and the feedforward layers, introducing only a few million trainable parameters and allowing fast, lightweight fine-tuning.

Graph-aware attention

To blend token semantics with topological cues returned by TopoRAG, the attention calculation in an unfrozen layer l becomes:

$$\text{Attn}^{(l)} = \text{softmax}\left(\frac{Q^{(l)}K^{(l)\top} + Q_g^{(l)}K_g{}^{\top}}{\sqrt{d}}\right)\left(V^{(l)} + V_g\right), \tag{4}$$

where $\{Q^{(l)}, K^{(l)}, V^{(l)}\}$ are the standard token projections and $\{Q_g^{(l)}, K_g, V_g\}$ are the graph-derived projections computed using the learned matrices $\{W_Q^g, W_K^g, W_V^g\}$.

This hybrid attention allows the LLM to reason simultaneously about symbolic intent (what to deploy) and structural context (where to deploy). The PFA design preserves the zero-shot generalization of LLM for unseen topologies and eliminates the need for external fusion layers by directly integrating graph cues. This efficiency enables the controller to issue SLA-compliant placement plans in real time while operating within the resource constraints of the cloud tier.

Multi-task planning decoder

To enable structured orchestration from a frozen LLM, we adopt a two-stage multitask decoder that reuses encoder representations for semantic conditioning [35]. In stage 1, task-specific heads extract high-level deployment semantics (reward priorities, a placement plan, and contextual embeddings) directly from the [CLS] token and microservice tokens [36, 37]. In stage 2, these outputs are prepended as learnable tokens to guide autoregressive decoding by LoRA-adapted layers, which generate fine-grained orchestration variables such as penalty coefficients, cooperation bias, and utility estimates [22].

Stage 1: global planning

Given multimodal input, a frozen encoder produces hidden states $\mathbf{H}^L = \{\mathbf{h}_i\}$. The reward priorities $\mathbf{w}^{\text{LLM}} \in \mathbb{R}^{|\mathbb{O}|}$ are extracted using a classification head applied to the [CLS] token:

$$\mathbf{w}^{\text{LLM}} = \text{softmax}(W_w \mathbf{h}_{\texttt{[CLS]}}). \tag{5}$$

The placement plan $\mathbf{p}^{\text{LLM}} \in \mathbb{R}^{|\mathscr{M}_{\text{tokens}}|}$ is derived via a head pointer on service tokens:

$$\mathbf{p}_i^{\text{LLM}} = \text{softmax}(W_\Pi \mathbf{h}_i). \tag{6}$$

Stage 2: token-level refinement

The combined plan encoding $\{\mathbf{w}^{\text{LLM}}, \mathbf{p}^{\text{LLM}}\}$ is prepended for a second pass through LoRA-adapted transformer layers. The model then generates time-sensitive penalty coefficients $\boldsymbol{\psi}_t^{\text{LLM}} \in \mathbb{R}^{|\mathbb{C}|}$ using:

$$\boldsymbol{\psi}_t^{\text{LLM}} = \sigma(W_\psi \mathbf{h}_t), \tag{7}$$

and the scalar cooperation bias $\phi^{\text{LLM}} \in [0, 1]$ via:

$$\phi^{\text{LLM}} = \sigma(\mathbf{w}_\phi^\top \mathbf{h}_{\texttt{[CLS]}}). \tag{8}$$

All outputs are projected into a unified plan embedding:

$$\mathbf{z}^{\text{plan}} = \left[P_w \mathbf{w}^{\text{LLM}} \parallel P_\Pi \,\text{vec}(\mathbf{p}^{\text{LLM}}) \parallel P_\psi \boldsymbol{\psi}_t^{\text{LLM}} \parallel P_\phi \phi^{\text{LLM}} \right] \in \mathbb{R}^{d_z}, \tag{9}$$

where $P_\cdot$ are projection matrices that ensure alignment and compactness. This structured vector forms the backbone of downstream orchestration decisions.

Problem formulation

In this paper, we jointly optimize microservice replica placement and interservice traffic routing to minimize end-to-end latency, subject to resource constraints, slice isolation, and reliability guarantees. Each microservice $m \in \mathscr{M}_q$ can spawn up to R_m replicas, with binary placement variables $x_{m,n,\rho,t}^q \in \{0, 1\}$ indicating whether the replica ρ of the service m is placed at node $n \in \mathscr{N}$ at time t on request q. These variables populate a placement matrix $\mathbf{X}_t \in \{0, 1\}^{|\mathscr{M}_q^{\text{rep}}| \times |\mathscr{N} \times \Sigma|}$, where rows index replica IDs and columns represent node–slice pairs (n, σ) for slices $\sigma \in \Sigma$. For each service dependency edge $e_{ij} = (m_i, m_j) \in \mathscr{E}_q$, traffic is distributed across candidate paths $\mathscr{P}_{ij}$ using routing fractions $y_{ij,p,\sigma,t}^q \in [0, 1]$, constrained by $\sum_{p,\sigma} y_{ij,p,\sigma,t}^q = 1$. These populate the routing matrix $\mathbf{Y}_t \in [0, 1]^{|\mathscr{E}_q| \times |\mathscr{P} \times \Sigma|}$, mapping the logical edges to the path-slice pairs. Joint optimization of $\mathbf{X}_t$ and $\mathbf{Y}_t$ under non-linear latency functions is strongly NP-hard [8], especially in the presence of stochastic arrivals, time-varying channel conditions, and evolving slice topologies all within a decentralized, partially observable environment.

To address decentralized orchestration under uncertainty, we formulate the problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) defined by the tuple $\langle \mathscr{K}, \mathscr{S}, \{\mathscr{A}_k\}, \mathscr{T}, \mathbb{R}, \{\mathscr{O}_k\}, \gamma \rangle$, where $\mathscr{K}$ is the cooperative EC set, $\mathscr{S}$ is the global state space, $\mathscr{A}_k$ and $\mathscr{O}_k$ are the local action and observation spaces of the agent $k \in \mathscr{K}$, $\mathscr{T}$ is the global state transition function, $\mathbb{R}$ is the shared reward function and $\gamma \in (0, 1)$ is the temporal discount factor. Each CE, responsible for a subgraph $\mathscr{N}_k \subseteq \mathscr{N}$, makes decisions based solely on its local observation $\mathscr{O}_k$ without global state visibility. Coordination is achieved through the shared reward $\mathbb{R}$, which guides the distribution of policy learning between all agents. This formulation naturally captures the spatial decentralization, partial observability, and long-horizon optimization requirements of dynamic 6G edge–cloud environments.

State representation ($\mathscr{S}$, $\mathscr{O}^k$)

Global state

The global state of the environment at the decision step t is defined as a tuple $s_t = (s_t^{\text{infra}}, s_t^{\text{svc}}, s_t^{\text{map}})$. The infrastructure state s_t^{infra} encodes the current substrate graph $\mathscr{G}_{\text{infra}} = (\mathscr{N}, \mathscr{L})$, where each node and link are annotated with dynamic attributes such as available compute, memory, and radio bandwidth. The service state s_t^{svc} captures the evolving structure of active service DAGs $\mathscr{G}_q = (\mathscr{M}_q, \mathscr{E}_q)$, which define microservice invocation dependencies across applications. Lastly, the mapping state s_t^{map} tracks the system-wide deployment configuration via a placement matrix $\mathbf{X}_t$ and a routing matrix $\mathbf{Y}_t$, reflecting which microservice replicas are hosted on which edge nodes and how traffic is routed along candidate paths, respectively.

Local observations

Each edge controller $k \in \mathscr{K}$, responsible for a regional subgraph $\mathscr{N}_k \subseteq \mathscr{N}$, receives a partial observation o_t^k of the environment. This observation includes the vector of residual capacity of the resources $\hat{\mathbf{c}}_{n,t}$ for each node $n \in \mathscr{N}_k$, and the available bandwidth $\hat{b}_{uv,\sigma,t}$ in the link (u, v) for each slice σ. It also contains recent performance metrics $\hat{\mathbf{h}}_{n,t}$, such as utilization and latency indicators; the local placement status $\mathbf{z}_{q,\sigma,t}^{(k)}$ for each service request q in the slice σ; and the traffic distribution fraction $y_{ij,p,\sigma,t}^{(k)}$ along each path $p \in \mathscr{P}_{ij}$, corresponding to the edge of the service $(m_i, m_j) \in \mathscr{E}_q$. Finally, the semantic context vector $\mathbf{z}_{T,k}^{\text{LLM}}$, received from the global LLM planner, encodes the intent-aligned deployment preferences to guide localized decision making.

Action space $\mathscr{A}^k$

At each time step t, each edge controller $k \in \mathscr{K}$ selects an action vector $a_t^k = (a_{m,n,\rho,t}^{(x)},\ a_{ij,p,t}^{(y)})$, where $a_{m,n,\rho,t}^{(x)} \in [0, 1]$ denotes the soft probability of placing the replica ρ of the microservice m on node $n \in \mathscr{N}_k$, subject to the constraint $\sum_{n\in\mathscr{N}_k,\rho} a_{m,n,\rho,t}^{(x)} = 1$. These probabilities are discretized via Gumbel-Softmax to support differentiable sampling during training. The routing decision $a_{ij,p,t}^{(y)} \in [0, 1]$ indicates the fraction of traffic allocated to the path $p \in \mathscr{P}_{ij}$ for the service edge $(m_i, m_j) \in \mathscr{E}_q$, constrained by $\sum_{p\in\mathscr{P}_{ij}} a_{ij,p,t}^{(y)} = 1$. The environment transition function is factored across nodes and agents as

$$P(s_{t+1} \mid s_t, \mathbf{a}_t) = P(s_{t+1}^{\text{sys}} \mid s_t^{\text{sys}}, \mathbf{a}_t) \prod_{k\in\mathscr{K}} \prod_{n\in\mathscr{N}_k} P(s_{n,t+1} \mid s_{n,t}, a_t^k) \tag{10}$$

where s_t^{sys} captures exogenous processes such as service arrivals, user mobility, and radio fluctuations, while $s_{n,t}$ reflects local queue lengths, link utilization, and remaining resources at node n.

Reward function design

The reward function is designed to jointly optimize four key objectives in the 6G Metaverse orchestration context. First, it aims to minimize end-to-end latency, ensuring responsiveness for immersive applications such as XR and holographic streaming. Second, it seeks to reduce total resource consumption across the infrastructure, including compute, memory, and GPU usage, to maintain energy efficiency and scalability. Third, the reward formulation promotes slice-aware load balancing by minimizing variance in resource usage between tenants, thus improving fairness and isolation. Lastly, it maximizes reliability, quantified by the probability of packet success across edge servers, to meet the stringent QoS guarantees required for mission-critical real-time services [38].

Global reward

At each decision step t, all edge controllers $k \in \mathscr{K}$ receive a shared coordination signal:

$$r_t^{\text{g}} = -\left(\alpha_1 \frac{\bar{\ell}_t}{\ell_{\max}} + \alpha_2 \frac{\mathbb{R}_{\text{tot},t}}{R_{\max}} + \alpha_3 \mathscr{LB}_t + \alpha_4 \frac{\sigma_{\mathbb{R}}(\mathscr{K})}{R_{\max}}\right), \tag{11}$$

where $\bar{\ell}_t$ denotes the average system-wide latency, $\mathbb{R}_{\text{tot},t}$ is the total resource utilization across all controllers, $\mathscr{LB}_t$ represents the global slice-sensitive load imbalance metric and $\sigma_{\mathbb{R}}(\mathscr{K})$ is the standard deviation of resource utilization across all edge controllers (measuring load balancing fairness). The scalar weights $\alpha_{1:4} \geq 0$ control the relative importance of each component in guiding the global coordination behavior.

Local reward

Each CE k computes a localized reward to guide its personalized policy adaptation.

$$r_t^{1,k} = -\left(\omega_{1,t}^k \frac{\bar{\ell}_t^k}{\ell_{\max}} + \omega_{2,t}^k \frac{\mathbb{R}_{\text{tot},t}^k}{R_{\max}} + \omega_{3,t}^k \mathscr{LB}_t^k + \omega_{4,t}^k \left[1 - \overline{P}_t^k\right]\right), \tag{12}$$

where $\bar{\ell}_t^k$ is the average latency within the local domain $\mathscr{N}_k$, $\mathbb{R}_{\text{tot},t}^k$ denotes the localized usage of resources, $\mathscr{LB}_t^k$ quantifies the local slice-sensitive load imbalance and $\overline{P}_t^k \in [0, 1]$

reflects the probability of packet success observed within the controller's subgraph. The dynamic weights $\omega_{j,t}^{k}$ enable the agent k to prioritize different optimization objectives in response to real-time slice-level service demands.

Reward

Each EC k also computes a localized reward to guide its personalized policy adaptation:

$$r_t^{1,k} = -\left(\omega_{1,t}^{k}\frac{\overline{\ell}_t^{k}}{\ell_{\max}} + \omega_{2,t}^{k}\frac{\mathbb{R}_{\text{tot},t}^{k}}{R_{\max}} + \omega_{3,t}^{k}\mathscr{LB}_t^{k} + \omega_{4,t}^{k}\left[1-\overline{P}_t^{k}\right]\right), \tag{13}$$

where $\overline{\ell}_t^{k}$ is the average latency within the local domain $\mathscr{N}_k$, $\mathbb{R}_{\text{tot},t}^{k}$ denotes the localized usage of resources (e.g., CPU, memory, GPU), $\mathscr{LB}_t^{k}$ quantifies the local slice-sensitive load imbalance, and $\overline{P}_t^{k} \in [0,1]$ reflects the probability of packet success observed within the controller's subgraph. Dynamic weights $\omega_{j,t}^{k}$ enable the agent k to prioritize different optimization objectives in response to real-time slice-level service demands. The detailed formulations for $\overline{\ell}_t$, $\mathbb{R}_{\text{tot},t}$, $\mathscr{LB}_t$, and $\overline{P}_t$ are as follows:

1. *End-to-End Latency* $\bar{\ell}_t$: For each Metaverse request $q \in \mathscr{Q}_t$, the total latency is:

$$\ell_q = \ell_q^{\text{acc}} + \ell_q^{\text{proc}} + \ell_q^{\text{sync}}, \tag{14}$$

where ℓ_q^{acc}, ℓ_q^{proc}, and ℓ_q^{sync} denote the access, processing, and synchronization delays, respectively [38]. To meet QoS guarantees, each request must satisfy $\ell_q \le \ell_q^{\max}$. The global average latency is computed as:

$$\bar{\ell}_t = \frac{1}{|\mathscr{Q}_t|}\sum_{q\in\mathscr{Q}_t}\ell_q. \tag{15}$$

2. *Resource Utilization* $\mathbb{R}_{\text{tot},t}$: Let $x_{m,n,\rho,t}^{q} \in \{0,1\}$ indicate whether the replica ρ of the microservice $m \in \mathscr{M}_q$ is deployed on node $n \in \mathscr{N}$. Each replica consumes θ_m^r units of resource type $r \in \mathscr{R}$, and $\mathscr{C}_n^r$ denotes the capacity of node n for the resource r [39]. The total normalized utilization is:

$$\mathbb{R}_{\text{tot},t} = \sum_{q}\sum_{m,\rho,n} x_{m,n,\rho,t}^{q}\sum_{r\in\mathscr{R}}\frac{\theta_m^r}{\mathscr{C}_n^r}. \tag{16}$$

3. *Load Balancing Cost* $\mathscr{LB}_t$: For each node $n \in \mathscr{N}$ and resource type $r \in \mathscr{R}$, the per-node utilization is:

$$u_{n,t}^{r} = \frac{1}{\mathscr{C}_n^r}\sum_{q}\sum_{m,\rho} x_{m,n,\rho,t}^{q}\cdot\theta_m^r, \quad \bar{u}_t^{r} = \frac{1}{|\mathscr{N}|}\sum_{n}u_{n,t}^{r}.$$

The variance across nodes defines the load imbalance:

$$\mathscr{LB}_t = \sum_{r\in\mathscr{R}}\frac{1}{|\mathscr{N}|}\sum_{n\in\mathscr{N}}\left(u_{n,t}^{r}-\bar{u}_t^{r}\right)^2. \tag{17}$$

4. *Packet Success Probability* $\overline{P}_t$: Let $\hat{P}_{q,t} \in [0,1]$ denote the success probability for service request q [40]. The global and local averages are:

$$\overline{P}_t = \frac{1}{|\mathscr{Q}_t|}\sum_{q\in\mathscr{Q}_t}\hat{P}_{q,t}, \quad \overline{P}_t^{k} = \frac{1}{|\mathscr{Q}_t^{(k)}|}\sum_{q\in\mathscr{Q}_t^{(k)}}\hat{P}_{q,t}. \tag{18}$$

The penalty term $1-\overline{P}_t$ in Eq. (13) captures network-level reliability degradation.

Constraint–violation penalties

To ensure reliability in decision making, especially under dynamic resource demands and strict SLA constraints, we introduce a penalty function that complements the reward formulation by discouraging infeasible or suboptimal deployments. We group all QoS violations into three penalty classes: resource overuse, temporal violations, and robustness degradation. Let $[x]_+ \triangleq \max(0,x)$. For agent k at time t, the penalty is:

$$\begin{aligned} r_t^{\text{pen},k} = -\Big(&\rho_1\sum_{n\in\mathscr{N}_k}\sum_{r\in\mathscr{R}}\left[\textstyle\sum_{q,m,\rho}x_{m,n,\rho,t}^{q}\theta_m^r - R_n^r\right]_+ \\ &+\rho_2\sum_{q\in\mathscr{Q}_t^{(k)}}\left[\frac{\ell_q}{\ell_q^{\max}}-1+\frac{\sigma_q}{\sigma_{\max}}-1+\frac{\Delta\ell_q}{\delta_{\text{sync}}}-1\right]_+ \\ &+\rho_3\sum_{q\in\mathscr{Q}_t^{(k)}}\left[\frac{\lambda_{m,n,t}^{q}}{\gamma_{\text{stab}}\mu_{n,m}}-1+\frac{P_{\min}}{\hat{P}_{q,t}}-1\right]_+\Big). \end{aligned} \tag{19}$$

here ℓ_q denotes the total end-to-end latency, $\Delta\ell_q$ the inter-modal synchronization delay, and σ_q the service jitter. The weights $\rho_{1:3}$ tune the influence of each penalty type. The set $\mathscr{R} = \{\text{cpu}, \text{mem}, \text{gpu}\}$ represents the dimensions of the tracked resources. SLA thresholds such as $\ell_q^{\max}$, $\sigma_{\max}$, δ_{sync}, γ_{stab}, and $P_{\min}$ are specified in the service profile $\mathscr{T}_q$. The success of the packet $\hat{P}_{q,t}$ is penalized when it falls below the reliability threshold $P_{\min}$, capturing instability and degradation in transmission quality.

Total reward and optimization objective

The total reward for agent $k \in \mathcal{K}$ at time t integrates global coordination, local performance, and the enforcement of constraints.

$$r_t^k = \phi \cdot r_t^{\mathrm{g}} + (1-\phi) \cdot r_t^{\mathrm{l},k} + r_t^{\mathrm{pen},k}, \tag{20}$$

where $\phi \in [0,1]$ is a cooperation parameter that balances global objectives r_t^{g} (e.g. system-wide latency $\bar{\ell}_t$, load balancing $\mathscr{LB}_t$, and semantic alignment) and local objectives $r_t^{\mathrm{l},k}$ (e.g. per agent efficiency and resource use). The penalty term $r_t^{\mathrm{pen},k}$ enforces feasibility with respect to physical constraints and SLA requirements (see Eq. (19)).

Each agent k follows a stochastic policy $\pi^k(a_t^k \mid \tau_t^k)$, where the trajectory $\tau_t^k = (o_0^k, a_0^k, \ldots, o_t^k)$ is summarized into a belief state b_t^k using a recurrent encoder. Actions are sampled as $a_t^k \sim \pi^k(\cdot \mid b_t^k)$. The joint policy $\boldsymbol{\pi} = (\pi^1, \ldots, \pi^{|\mathcal{K}|})$ aims to maximize the expected discounted return:

$$J(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi},\mathscr{T},\Omega}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r(s_t, \mathbf{a}_t)\right], \tag{21}$$

where $\mathbf{a}_t = (a_t^1, \ldots, a_t^{|\mathcal{K}|})$ is the joint action vector, $\mathscr{T}$ denotes the transition dynamics, and Ω models the observation process.

The optimization task involves a hybrid discrete-continuous-continuous action space with partial observability. The placement of replicas is governed by binary variables $x_{m,n,\rho,t}^q \in \{0,1\}$, which indicate whether the replica ρ of the microservice $m \in \mathscr{M}_q$ is deployed at node $n \in \mathscr{N}$ at time t. The routing of traffic is determined by flow fractions $y_{ij,p,t}^q \in [0,1]$, representing the proportion of traffic along the route $p \in \mathscr{P}_{ij}$ for the DAG edge of service $(m_i, m_j) \in \mathscr{E}_q$.

LLM-driven cloud planning and multi-agent microservice deployment

Jointly minimizing latency, utilization, and load imbalance in 6G systems is a complex nonconvex problem. Existing GNN-based approaches suffer from structural fragility under network changes [9], while DRL methods struggle with expanded state spaces and strict multimodal constraints [39]. To address this, we propose a *semantic intent–driven* framework combining a cloud-hosted LLM planner with edge-based MAPPO execution. The LLM generates high-level deployment plans based on global service DAGs and constraints. Edge controllers then refine these plans, adapting placement and routing to real-time conditions. This two-tier design merges long-horizon foresight with agile local control. As shown in Fig. 3, the workflow begins with a Topology-sensitive RAG (TopoRAG) retrieving historical contexts to guide the LLM in generating a soft deployment prior. Subsequently, edge agents adjust this proposal based on instantaneous states to produce final placement $\mathbf{X}_t$ and routing $\mathbf{Y}_t$ matrices. Performance metrics are fed back into the TopoRAG store, enabling the system to learn from successful patterns. Specific agent updates are detailed in "Semantic-guided policy execution for microservice placement and routing" section.

Global LLM planning

LLMPLANNER acts as a hybrid semantic planner, combining two complementary adaptation strategies: prompt-level reasoning, triggered at the start of each global planning window via TopoRAG, and periodic QLoRA fine-tuning, performed during low-traffic windows or after major system changes. This layered approach allows for responsive short-term adaptation while incrementally refreshing the model's long-term understanding of structural patterns, including evolving service DAGs, infrastructure dynamics, and SLA requirements.

Real-time prompting via TopoRAG

At the beginning of each global planning window, LLMPLANNER constructs a fused query graph $\mathscr{G}_{\mathrm{qry}}$ by integrating the current state of the infrastructure, the DAGs of the service, and the recent telemetry. This involves three stages: (i) multimodal fusion, where surveillance streams, time series telemetry, and graph structures are encoded using ViT, TCN, and GraphSAGE, respectively; (ii) global state aggregation, which consolidates rollout statistics from all edge agents; and (iii) query graph construction, which merges service requirements with relevant patterns retrieved from the memory graph.

The resulting query graph supports graph-of-thought (GoT) reasoning by representing deployment challenges across functional, topological, and historical dimensions. Using a composite similarity metric, which combines DAG edit distance, placement overlap, and latent vector similarity, the planner retrieves the top-κ subgraphs from $\mathscr{G}_{\mathrm{mem}}$, encoding them into a global context vector $\mathbf{z}^{\mathrm{LLM}}$ that captures structural and temporal patterns from prior deployments.

This context is then passed into a structured prompt that enables the LLM to perform semantic planning. The prompt construction process builds a schema-based instruction set that requests exactly ⟨NUM_CANDIDATES⟩ candidate deployment plans. Each plan includes a list of assignments from microservice to node, denoted as `["microservice", "node"]` pairs and terminated by `"<eop>"`.

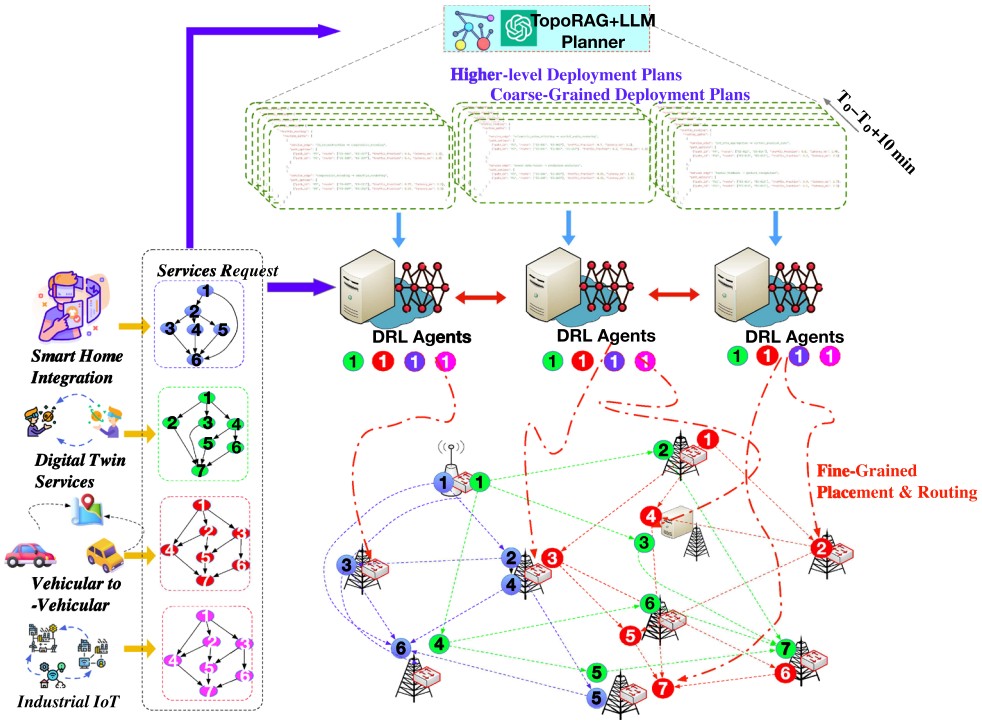


Fig. 3 Semantic planning and agent coordination in the AI-native cloud–edge orchestration framework

Plan decoding and updates

At the start of each planning window T, the cloud LLM receives a query graph $\mathcal{G}_{\text{qry}}^{(k)}$ for every edge agent k, obtained by combining the current service DAG with the most relevant subgraphs from $\mathcal{G}_{\text{mem}}$. This graph is encoded as a context vector $\mathbf{z}_{T,k}^{\text{LLM}}$ and passed to the two-stage decoder described in "Multi-task planning decoder" section. Stage 1 extracts reward-priority vector $\boldsymbol{\omega}_{T,k}$ and placement plan $\mathbf{p}_{T,k}$ from [CLS] and microservice tokens. Stage 2 refines this intent with penalty coefficients $\boldsymbol{\psi}_{T,k}$ and a cooperation bias $\beta_{T,k}$, then packs all four signals into the plan embedding $\mathbf{z}_{T,k}^{\text{plan}}$ (Eq. 9). By sampling B candidate plans around $\mathbf{z}_{T,k}^{\text{plan}}$, the planner scores their expected utility $\hat{R}_{T,k}^{(b)}$ and sends the best plan $\texttt{plan}_k^\star$ to agent k. The plan, its reward weights, and the performance achieved are recorded in the memory graph, triggering a LoRA adapter update whenever the priority drift exceeds the threshold δ_{mis}.

Fine-tuning cycle

To complement real-time prompting, LLMPLANNER periodically refreshes its planning policy by fine-tuning the QLoRA-based adapter. This cycle is triggered during off-peak intervals or after critical changes in service patterns or infrastructure topology. Training data is constructed from deployment traces and deviation metrics computed as:

$$\delta_k = \left| \bar{\omega}_{T,k} - \hat{R}_k^{(b^\star)} \right|, \tag{22}$$

where $\bar{\omega}_{T,k}$ denotes the average priority weights assigned by the planner and $\hat{R}_k^{(b^\star)}$ is the observed utility of the selected plan. The adapter refinement process involves extracting successful deployment patterns from recent traces, followed by early stopped training on the QLoRA parameters. Model validation ensures compliance with SLA and latency constraints using holdout traces. Upon successful validation, the updated adapter is seamlessly implemented without interruption of service.

Algorithm 1 summarizes the training and inference procedure for the LLMPlanner. First, multimodal fusion (lines 1–2) embeds visual, telemetry and topological inputs using ViT, TCN, and GraphSAGE, respectively, to construct the global snapshot $\mathbf{z}_t$. Next, context retrieval (lines 3–4) builds a query graph for each edge agent k and retrieves the top κ most similar traces from the memory graph $\mathcal{G}_{\text{mem}}$. Prune & rerank (lines 5–11) applies adaptive scoring weights from Eq. (1) and refines the set of candidates using a lightweight GNN through Eq. (2), producing the agent-specific context vector $\mathbf{z}_{T,k}^{\text{LLM}}$. In the two-stage decoding phase (lines 12–19), the model first generates reward weights $\boldsymbol{\omega}_{T,k}$ and a semantic deployment plan $\mathbf{p}_{T,k}$, then refines these results using penalty coefficients $\boldsymbol{\rho}_{T,k}$ and a cooperation bias term $\beta_{T,k}$. The resulting embedding scores all candidate plans in B, from which the optimal plan $\texttt{plan}_k^\star$ is selected and sent.

Algorithm 1: LLMPLANNER

Input: Multimodal snapshot $\mathbf{z}_t$, service DAGs $\{\mathscr{D}_q\}$, memory graph $\mathscr{G}_{\text{mem}}$, edge-agent set $\mathscr{K}$
Output: Selected plans $\{\texttt{plan}_k^\star\}_{k\in\mathscr{K}}$, context vectors $\{\mathbf{z}_{T,k}^{\text{LLM}}\}$

1 Fuse visual, telemetry, and topology data into global embedding $\mathbf{z}_t$
2 **foreach** $k \in \mathscr{K}$ **do**
3 Construct query graph $\mathscr{G}_{\text{qry}}^{(k)}$ from $\mathscr{D}_q$ and current state
4 Retrieve κ nearest deployment traces from $\mathscr{G}_{\text{mem}}$
5 **foreach** *retrieved subgraph* g_j **do**
6 **foreach** *entity* (v, e) *in* g_j **do**
7 Compute pruning weights (α_v, α_e) via Eq. (1)
8 **end**
9 Perform GNN message passing on g_j using Eq. (2)
10 **end**
11 Aggregate reranked subgraphs into context vector $\mathbf{z}_{T,k}^{\text{LLM}}$
12 Stage 1 decode: obtain reward weights $\boldsymbol{\omega}_{T,k}$ and plan $\mathbf{p}_{T,k}$
13 Stage 2 decode: obtain penalties $\boldsymbol{\rho}_{T,k}$ and bias $\beta_{T,k}$
14 Assemble plan embedding $\mathbf{z}_{T,k}^{\text{plan}}$
15 Sample B candidate plans based on $\mathbf{z}_{T,k}^{\text{plan}}$
16 **foreach** $b = 1$ **to** B **do**
17 Evaluate utility $\hat{R}_{T,k}^{(b)}$
18 **end**
19 Select $\texttt{plan}_k^\star = \arg\max_b \hat{R}_{T,k}^{(b)}$; dispatch to agent k; log trace
20 **if** $\|\boldsymbol{\omega}_{T,k} - \boldsymbol{\omega}_{T,k}^{eval}\| > \delta_{mis}$ **then**
21 Queue LoRA adapter update for agent k
22 **end**
23 **end**
24 **if** *traffic is off-peak* ***or*** *topology changed* **then**
25 Fine-tune queued LoRA adapters on recent traces; validate; redeploy safely
26 **end**
27 **return** $\{\texttt{plan}_k^\star\}$, $\{\mathbf{z}_{T,k}^{\text{LLM}}\}$

Finally, logarithm & adaptation (lines 20–27) record the results of the execution in $\mathscr{G}_{\text{mem}}$. If the observed priority drift exceeds the threshold δ_{mis}, a LoRA-based update is queued for fine-tuning and safely redeployed during the next off-peak interval or upon topological reconfiguration.

Semantic-guided policy execution for microservice placement and routing

At the beginning of each local control cycle t, each edge controller $k \in \mathscr{K}$ queries the message bus for its most recent ranked deployment plan $\texttt{plan}_k^{(b_k^\star)}$, generated during the latest global planning interval T. Upon receipt, the edge controller merges this high-level plan with its local observations in real time o_t^k to condition the execution of the policy. Specifically, the semantic-guided MAPPO loop enables rapid and constraint-aware adaptation under partial observability: edge agents align with the intent of the LLM planner while optimizing local actions based on instantaneous load, link state, and resource availability. This tight semantic-to-action coupling ensures low-latency, SLA-compliant service delivery across the distributed 6G Metaverse infrastructure.

Local belief update and action sampling

In partial observability, each edge agent $k \in \mathscr{K}$ fuses its local observation o_t^k with the semantic context vector $\mathbf{z}_{T,k}^{\text{LLM}}$ provided by the LLM planner. This is achieved through a lightweight cross-attention mechanism:

$$\mathbf{x}_t^k = \left[o_t^k \parallel \mathbf{W}_k \mathbf{z}_{T,k}^{\text{LLM}} \right], \tag{23}$$

where $\mathbf{W}_k \in \mathbb{R}^{d\times d_z}$ is a learned projection matrix that aligns the LLM semantic vector with the agent's local embedding space.

The fused input $\mathbf{x}_t^k$ is encoded in a belief state b_t^k using a gated recurrent unit (GRU), capturing both historical dependencies and semantic priors:

$$b_t^k = \text{GRU}(b_{t-1}^k, \mathbf{x}_t^k). \tag{24}$$

Actions are sampled from the agent's local policy $\pi_{\theta_k}(a_t^k \mid b_t^k)$ in the action space $\mathscr{A}_k$, integrating real-time feedback and guidance from the global planner.

LLM-guided reward shaping

The LLM-generated deployment plan serves as a semantic scaffold for adaptive execution at the edge. Upon receipt, each edge controller k calculates a trust coefficient $\kappa_t^k \in [0, 1]$, reflecting the alignment between the LLM's predicted reward $\hat{R}_{T,k}^{\text{LLM}}$ and the previously observed return R_{t-1}^k:

$$\kappa_t^k = \exp\left(-\alpha \left| \hat{R}_{T,k}^{\text{LLM}} - R_{t-1}^k \right| \right), \tag{25}$$

where $\alpha > 0$ is a tunable sensitivity parameter. The trust coefficient κ_t^k dynamically balances static priors and LLM-inferred semantic guidance. The reward weights $\boldsymbol{\omega}_t^k = (\omega_{1,t}^k, \ldots, \omega_{4,t}^k)$, the penalty coefficients $\boldsymbol{\rho}_t^k = (\rho_{1,t}^k, \ldots, \rho_{3,t}^k)$, and the cooperation factor ϕ_t^k are mixed as:

$$\omega_{i,t}^k = (1-\kappa_t^k)\cdot\omega_i + \kappa_t^k \cdot \frac{\omega_{i,t}^{\text{LLM}}}{\|\boldsymbol{\omega}_t^{\text{LLM}}\|_1}, \quad i = 1, \ldots, 4, \tag{26}$$

$$\rho_{j,t}^k = (1-\kappa_t^k)\cdot\rho_j + \kappa_t^k \cdot \rho_{j,t}^{\text{LLM}}, \quad j = 1, \ldots, 3, \tag{27}$$

$$\phi_t^k = (1-\kappa_t^k)\cdot\phi_0 + \kappa_t^k \cdot \sigma\left(\beta_t^{\text{LLM}}\right), \tag{28}$$

where ω_i and ρ_j are the static weight and penalty coefficients, ϕ_0 is the baseline cooperation factor, $\sigma(\cdot)$ is the sigmoid function and β_t^{LLM} is the LLM-suggested cooperation bias

Algorithm 2: EdgeAgent: Semantic-Guided MAPPO Execution

1 **Input:** LLM plan $\Pi_k^\star$, local graph $\mathscr{G}_k$, replay buffer $\mathscr{B}_k$, hyper-params $(\delta, \lambda_{\mathrm{KL}}, \tau, \delta_{\mathrm{mis}})$
2 **Output:** updated policy π_{θ_k}, placement $\mathbf{X}_t$, routing $\mathbf{Y}_t$, rollout stats $\Sigma_{T,k}$
3 Prompt LLM planner for plan $\Pi_k^\star$ /* $\Pi_k^\star \leftarrow$ `QueryLLMPlanner`$(\mathscr{G}_k)$ */
4 Extract plan embedding $\mathbf{z}_{T,k}^{\mathrm{LLM}}$ from $\Pi_k^\star$; initialise belief b_0^k and trust κ_0^k
5 **foreach** *local step* $t = 1 \,..\, T_{local}$ **do**
6 Observe local state o_t^k; fuse with plan to get x_t^k
7 Update GRU belief b_t^k with x_t^k
8 Evaluate trust κ_t^k using Eq. 25; blend weights via Eqs. 26–28
9 Compute shaped reward $\hat{r}_t^k$ by Eq. 31; store $(b_t^k, a_t^k, \hat{r}_t^k)$ in $\mathscr{B}_k$
10 Derive LLMPlanner policy $\pi_{\mathrm{LLM},k}$ from plan; sample $a_t^k \sim \pi_{\theta_k}$; apply action; update $\mathbf{X}_t$, $\mathbf{Y}_t$
11 **end**
12 Compute advantages $\widehat{A}_t^k$ and statistics $\Sigma_{T,k}$
13 **foreach** *training epoch* **do**
14 Optimise actor–critic loss with PPO clipping and KL regularisation (34)
15 **end**
16 Log rollout to memory; if semantic deviation $> \delta_{\mathrm{mis}}$ then queue LoRA update request
17 **return** $\pi_{\theta_k}, \mathbf{X}_t, \mathbf{Y}_t, \Sigma_{T,k}$

scalar. These adaptive parameters shape three distinct reward components:

$$\hat{r}_t^{\mathrm{l},k} = -\sum_{i=1}^{4} \omega_{i,t}^k \cdot r_{t,i}^{\mathrm{l},k}, \tag{29}$$

$$\hat{r}_t^{\mathrm{pen},k} = -\sum_{j=1}^{3} \rho_{j,t}^k \cdot c_{j,t}^k, \tag{30}$$

where $r_{t,i}^{\mathrm{l},k}$ represent local rewards, $c_{j,t}^k$ denotes the penalty incurred by violating the constraint j and $\hat{r}_t^{\cdot}$ are the shapes of rewards used in optimization. The final trust-aware reward signal that guides policy updates becomes:

$$\hat{r}_t^k = \phi_t^k \cdot \hat{r}_t^{\mathrm{g}} + (1 - \phi_t^k) \cdot \hat{r}_t^{\mathrm{l},k} + \hat{r}_t^{\mathrm{pen},k}. \tag{31}$$

MAPPO update with semantic priors

We extend the standard MAPPO actor–critic framework by conditioning each agent's policy not only on its internal belief state, but also on the projected semantic plan from LLMPlanner. Specifically, the policy receives an augmented input that fuses the agent's encoded state with the semantic context vector:

$$\pi_{\theta_k}(a_t^k \mid b_t^k) = \mathrm{Softmax}\Big(\mathbf{f}_{\theta_k}(b_t^k) + \kappa_t^k \, \mathbf{W}_k \, \mathbf{z}_{T,k}^{\mathrm{LLM}}\Big), \tag{32}$$

where $\mathbf{z}_{T,k}^{\mathrm{LLM}}$ is the LLM-derived plan embedding for agent k, $\mathbf{W}_k$ is a trainable projection matrix, and κ_t^k is the trust coefficient defined in Eq. (25). To maintain semantic alignment, the derivation of the LLMPlanner distribution $\pi_{\mathrm{LLM},k}(a)$ is conducted from the normalized plan $\mathbf{p}_{T,k}^{\mathrm{LLM}}$, constrained by a trust-weighted feasible action set:

$$\pi_{\mathrm{LLM},k}(a) = \frac{p_{T,k}^{\mathrm{LLM}}(a)}{\sum_{a' \in \mathscr{A}_k} p_{T,k}^{\mathrm{LLM}}(a')}, \quad a \in \mathscr{A}_k, \tag{33}$$

where $\mathscr{A}_k = \left\{ a \;\middle|\; p_{T,k}^{\mathrm{LLM}}(a) \geq \delta \right\}$, and $\delta > 0$ is a soft activation threshold to prune low-confidence actions. The loss of MAPPO policy integrates three terms: clipped surrogate objective [41], KL regularization toward $\pi_{\mathrm{LLM},k}$, and entropy for exploration:

$$\mathscr{L}_{\mathrm{policy}}^{(k)} = \mathbb{E}_t\left[\min\left(r_t^{(k)} \widehat{A}_t^k, \; \mathrm{clip}(r_t^{(k)}, 1 \pm \varepsilon)\widehat{A}_t^k\right)\right] + \lambda_{\mathrm{KL}} \, \xi_\tau(t) \, \mathbb{E}_t\left[\log \frac{\pi_{\theta_k}(a_t^k \mid b_t^k)}{\pi_{\mathrm{LLM},k}(a_t^k)}\right], \tag{34}$$

where $r_t^{(k)} = \frac{\pi_{\theta_k}(a_t^k | b_t^k)}{\pi_{\theta_k}^{\mathrm{old}}(a_t^k | b_t^k)}$ is the importance sampling ratio and $\xi_\tau(t) = e^{-t/\tau}$ decays the influence of semantic guidance over time.

The total loss across agents combines policy loss, entropy regularization, and critic value loss:

$$\mathscr{L} = \sum_{k \in \mathscr{K}} \left[\mathscr{L}_{\mathrm{policy}}^{(k)} - \lambda_{\mathrm{ent}} \, \mathscr{H}(\pi_{\theta_k})\right] + c_V \, \mathscr{L}_{\mathrm{critic}}, \tag{35}$$

where $\mathscr{H}(\cdot)$ denotes the policy entropy and $\mathscr{L}_{\mathrm{critic}}$ is the loss of temporal difference between the predicted and target state values. This semantically grounded MAPPO formulation preserves decentralized execution while encouraging agents to remain aligned with high-level LLM intent, even in non-stationary, partially observable edge environments.

A summary of the algorithm is given in Algorithm 2. The `EdgeAgent` first prompts the LLM planner with the current context to obtain an updated plan (line 2), extracts the plan embedding, and initializes the belief and trust states (line 3). During execution (lines 4–10), at each local time step it observes the local state and fuses it with the plan (line 5), updates the GRU belief state (line 6), and evaluates trust (line 7). It then computes a shaped reward that reflects both local and global goals (line 8), derives the LLMPlanner policy from the plan, and selects actions while updating placement and routing (line 9). After collecting rollout data, the agent computes benefits and statistics (line 11) and enters the training phase (lines 12–14), where it optimizes the actor–critic loss using PPO with clipping and KL regularization (line 13). Finally, it logs the rollout, queues LoRA updates based on

semantic deviation (line 15), and returns the updated policy, placement, routing, and rollout statistics (line 16).

Theoretical analysis

Convergence properties

The proposed framework optimizes a regularized objective that combines the shaped reward signal $\hat{r}_t^k$ in Eq. (29) with a KL divergence term that keeps each local policy π_{θ_k} close to the LLM-induced prior $\pi_{\text{LLM},k}$ in Eqs. (32)–(34). For agent $k \in \mathcal{K}$, define the regularized return

$$\mathcal{J}_k(\theta_k) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \hat{r}_t^k - \lambda_{\text{KL}}\, \xi_\tau(t)\, D_{\text{KL}}\big(\pi_{\theta_k}(\cdot|b_t^k) \,\|\, \pi_{\text{LLM},k}(\cdot)\big)\Big], \tag{36}$$

where $D_{\text{KL}}(\cdot\|\cdot)$ is the KL divergence, $\xi_\tau(t) = e^{-t/\tau}$ is the decaying guidance factor, and $\hat{r}_t^k$ is the trust-weighted reward in Eq. (30).

Assumptions

We require the following conditions:

(A1) Boundedness: Rewards satisfy $|r_t| \le r_{\max}$ and advantages $|\widehat{A}_t| \le A_{\max}$ almost surely; gradients are Lipschitz on the compact policy set.

(A2) Stable trust: The trust coefficient $\kappa_{k,t}$ remains within the interval $[0, \kappa_{\max}]$ with $\kappa_{\max} < 1$, and it either decays or stays bounded whenever semantic mismatch persists, as governed by Eq. (33) and the δ_{mis} gate in Algorithm 2.

(A3) KL control: The per-update KL is constrained via clipping and penalty with schedule $\lambda_{\text{KL}}\xi_\tau(t)$ (Eq. (35)), yielding $D_{\text{KL}}(\pi_{\theta_k}^{\text{new}} \| \pi_{\theta_k}^{\text{old}}) \le \delta$.

(A4) Planner stationarity: $\pi_{\text{LLM},k}$ is held fixed within a planning window T and updated asynchronously across windows (Alg. 1).

(A5) Vanishing regularization: The product $\lambda_{\text{KL}}\xi_\tau(t)$ decays to zero as training progresses, that is, $\lim_{t\to\infty} \lambda_{\text{KL}}\xi_\tau(t) = 0$.

Bias bound from semantic shaping

Let A_t be the true advantage and write $\widehat{A}_t = A_t + b_t$. Under (A1)–(A2), the semantic bias satisfies:

$$|b_t| \le \kappa_{\max}\, \Delta_{\text{mis}}, \qquad \Delta_{\text{mis}} := \big|\widehat{R}_{T,k}^{\text{LLM}} - R_{t-1,k}^{\text{env}}\big|, \tag{37}$$

so that shaping perturbs the policy gradient by a bounded term that vanishes as alignment improves, with $\Delta_{\text{mis}} \downarrow$ due to LoRA refresh and gating, or as trust decays, with $\kappa_{k,t} \downarrow$.

Proposition 1 (Convergence to Stationary Point) *Under Assumptions (A1)–(A5), the update rule in Eqs. (32)–(35) performs a stochastic gradient ascent on the regularized objective (36). Consequently, each local policy π_{θ_k} converges in probability to a stationary point of $\mathcal{J}_k(\theta_k)$. When $\lambda_{\text{KL}}\xi_\tau(t) \to 0$, the limit points coincide with stationary points of the MAPPO objective with reward $\hat{r}_t^k$.*

Proof sketch The clipped PPO surrogate in Eq. (35) is an unbiased estimator of the policy gradient of $\mathcal{J}_k$ up to the clipping term, which bounds the importance ratio $r_t^{(k)}$ and ensures stable updates. Under (A1)–(A3) and small enough step size, the PPO-style update with KL control admits the lower bound

$$J(\pi^{\text{new}}) \;\ge\; J(\pi^{\text{old}}) \;+\; \mathbb{E}\big[\min(r_t, \text{clip}(r_t, 1\pm\epsilon))\,\widehat{A}_t\big] \;-\; C\, D_{\text{KL}}(\pi^{\text{new}} \| \pi^{\text{old}}), \tag{38}$$

for some $C > 0$. Combining Eqs. (37) and (38) yields an improvement up to semantic bias:

$$J(\pi^{\text{new}}) \;\ge\; J(\pi^{\text{old}}) \;+\; \mathbb{E}\big[\min(r_t, \text{clip}(r_t, 1\pm\epsilon))\,\widehat{A}_t\big] \;-\; C\delta \;-\; \epsilon\,\kappa_{\max}\,\Delta_{\text{mis}}. \tag{39}$$

The additional KL term acts as a time-varying regularizer with gradient $\nabla_{\theta_k} D_{\text{KL}}(\pi_{\theta_k} \| \pi_{\text{LLM},k})$. Since $\lambda_{\text{KL}}\xi_\tau(t)$ is bounded and decays to zero under Assumption A5, this term introduces only a transient perturbation, and for fixed δ and ϵ the bound reduces to the standard PPO case as Δ_{mis} and $\kappa_{\max}$ become negligible. Within each window T, the sequence $\pi_\theta^{(u)}{}_u$ converges to a stationary point of the shaped objective, while across windows the objective smoothly tracks planner updates due to KL control and trust gating; then bounded rewards and gradients imply that θ_k converges to a stationary point of (36). □

Scalability

We denote the number of infrastructure nodes by $|\mathcal{N}|$, the total active microservices by $|\mathcal{M}| = \sum_{q\in Q} |M_q|$, and the number of edge controllers by $K = |\mathcal{K}|$. The framework exhibits scalability across three key dimensions.

Linear scaling in agents

With K edge controllers, planner decoding scales linearly with respect to K for dispatch and logarithmically with respect to memory size:

$$\mathcal{C}_{\text{dispatch}} = \mathcal{O}\big(K \log |G_{\text{mem}}|\big) + \mathcal{O}\big(K \cdot |\texttt{plan}|\big). \tag{40}$$

Regional sharding combined with top-κ pruning and reranking, as illustrated in Fig. 2, reduces the effective lookup complexity to $\log(|G_{\text{mem}}|/S)$ given S shards. The total loss defined in Eq. (35) decomposes across agents, where each

agent k updates π_{θ_k} using only local trajectories and the shared global signal r_t^g. Consequently, the training iteration scales as

$$\mathscr{C}_{\text{training}} = \mathscr{O}\big(K\,\bar{B}\,C_{\text{local}}\big), \tag{41}$$

where $\bar{B}$ represents the average batch size and C_{local} denotes the per-sample actor-critic cost.

Sublinear scaling per agent

Each controller k encodes strictly its local subgraph $G_k = (N_k, L_k)$ and the mapped service DAGs. Given that $|N_k|$ is significantly smaller than $|\mathscr{N}|$ and the node degree is bounded, GraphSAGE encoders and GRU updates incur $\mathscr{O}(|N_k|)$ memory usage. The computational cost ranges from $\mathscr{O}(|N_k|)$ to $\mathscr{O}(|N_k|^2)$. No global state reconstruction is required, enabling near-linear scaling in K while each agent operates on a fixed-size subgraph.

Service/DAG size

The construction of the query graph scales according to

$$\mathscr{C}_{\text{query}} = \mathscr{O}(|\mathscr{N}| + |\mathscr{L}| + |M_q| + |E_q|), \tag{42}$$

where $|M_q|$ and $|E_q|$ denote the count of microservices and DAG edges per request, respectively. GNN reranking requires $\mathscr{O}(\sum_{\ell=1}^{L} |E^{(\ell)}|)$ operations distributed across L layers. The top-κ cutoff bounds the subgraph size, which renders the retrieval cost effectively independent of the global memory size $|G_{\text{mem}}|$.

Throughput and workers

Given planner calls arriving at a rate λ_{plan} with an average service time $\bar{s}$ distributed across C workers, Kingman's approximation provides the bound

$$\mathbb{E}[W_q] \lesssim \frac{\rho}{1-\rho}\,\frac{c_a^2 + c_s^2}{2}\,\bar{s}, \quad \text{where } \rho = \frac{\lambda_{\text{plan}}\bar{s}}{C}, \tag{43}$$

and c_a, c_s denote the coefficients of variation. Maintaining a utilization factor of $\rho \le 0.6$ ensures that planning remains amortized and outside the critical path, as ensured by the asynchronous planning cadence and worker scaling detailed in Algorithm 1 (lines 24–26).

Overhead and MEC/6G deployment feasibility

We analyze the system-level overhead of the LLM planner and TopoRAG, focusing on computational cost and energy constraints.

Cloud-side overhead

The LLM planner performs a TopoRAG query, a forward pass, and a plan decoding per window. With embedding size d_z, L_{GNN} reranking layers, and L_{adapt} unfrozen layers, the cloud cost is

$$\mathscr{C}_{\text{cloud}} = \mathscr{O}\big(K L_{\text{GNN}} |\tilde{V}| d_z^2 + L_{\text{adapt}}\, T_{\text{tok}} d_z^2 + B K d_z^2\big), \tag{44}$$

where $|\tilde{V}|$ denotes the nodes in the retrieved subgraphs and T_{tok} is the prompt length. Planning asynchronously every $T = 600$s amortizes this overhead over many requests.

Edge-side overhead

Agents receive compact plan embeddings and perform GRU updates, cross-attention fusion, and actor-critic passes. These operations add constant overhead over standard MAPPO, while the cost of the message bus scales as $\mathscr{O}(K|M_q|)$ per window.

Compute and time budget

We decompose the planner call into retrieval $\mathscr{O}(\log |G_{\text{mem}}|)$, decoding $\Theta(B L_{\text{adapt}} d^2)$, and dispatch components:

$$t_{\text{plan}} = t_{\text{retrieval}} + t_{\text{decode}} + t_{\text{dispatch}}. \tag{45}$$

Since planning occurs every $T \gg t_{\text{plan}}$, the per-request overhead vanishes. The Worst-case staleness is bounded by Eq. (43).

Energy budget

The energy per plan aggregates retrieval, decoding, and network transmission costs:

$$E_{\text{plan}} = P_{\text{DB}}\, t_{\text{retrieval}} + P_{\text{LLM}}\, t_{\text{decode}} + P_{\text{net}}\, t_{\text{dispatch}}. \tag{46}$$

This decouples planner energy from millisecond-scale control. Furthermore, partial freezing with LoRA restricts the trainable parameters to approximately 0.1% of the model.

Memory-graph footprint

For N_{tr} traces, the memory footprint scales as

$$\mathscr{M}_{\text{store}} = \mathscr{O}(N_{\text{tr}}\,(d_{\text{meta}} + d_z)). \tag{47}$$

Techniques such as TTL aging and top-κ pruning bound lookup tails as shown in Fig. 2. Capping N_{tr} ensures practical memory usage in commodity instances.

Dimensioning rules

The feasibility of deployment requires: (i) choosing T such that $t_{\text{plan}} \ll T$; (ii) maintaining utilization $\rho \le 0.6$ via worker scaling; (iii) bounding N_{tr} via TTL to meet latency budgets; and (iv) minimizing $|\texttt{plan}|$ to reduce control-plane costs.

Computational complexity

We summarize the asymptotic time and space complexity of one global planning window followed by T_{local} local control steps.

Planner complexity

Let $|\mathscr{E}_{\text{mem}}|$ denote the number of edges in the memory graph G_{mem}. TopoRAG utilizes an index that supports subgraph retrieval in $\tilde{\mathscr{O}}(\log |\mathscr{E}_{\text{mem}}|)$ time per query, where the notation $\tilde{\mathscr{O}}$ suppresses logarithmic factors. The planner cost comprises three components: TopoRAG retrieval with

GNN reranking, LLM decoding, and plan dispatch. These are expressed as

$$\mathscr{C}_{\text{planner}} = \mathscr{O}\big(K \log |G_{\text{mem}}| + \kappa\, |E_{\text{avg}}| + BL_{\text{adapt}} d_z^2 + K \cdot |\texttt{plan}|\big),$$
$$\mathscr{T}_{\text{planner}} = \tilde{\mathscr{O}}\big(\log |G_{\text{mem}}|\big) + \Theta(BL_{\text{adapt}} d_z^2) + \mathscr{O}(K). \quad (48)$$

here κ represents the number of subgraphs retained after adaptive pruning and $|E_{\text{avg}}|$ denotes the mean edge count after pruning as defined in Eq. (2). The construction of the query graph scales linearly according to Eq. (42).

Edge controller complexity

Per agent, the computational cost includes belief updates and PPO training. This is formulated as

$$\mathscr{C}_{\text{ctrl}} = \mathscr{O}\big(T_{\text{local}}\,(d_{\text{GRU}} + |\mathscr{A}_k|) + E_{\text{PPO}} \cdot \tfrac{T_{\text{local}}|\mathscr{A}_k|}{M}\big), \quad (49)$$

where the complexity of the steps $\mathscr{T}_{\text{step}} = \mathscr{O}(d_{\text{GRU}} + |\mathscr{A}_k|)$ supports millisecond-scale control cycles. Summing over all agents yields the epoch cost

$$\mathscr{C}_{\text{epoch}} = \mathscr{O}\big(BL_{\text{adapt}} d_z^2 + K \log |G_{\text{mem}}| + KT_{\text{local}}(d_{\text{GRU}} + |\mathscr{A}_k|)\big). \quad (50)$$

The dominant terms scale as

$$\mathscr{C}_{\text{total}} = \mathscr{O}\big(KL_{\text{GNN}}|\tilde{V}| + L_{\text{adapt}} T_{\text{tok}} d_z^2 + KT_{\text{local}} C_{\text{local}}\big), \quad (51)$$

which is linear in the number of controllers K and local steps T_{local}, and quasi-linear in $|G_{\text{mem}}|$ due to efficient indexing.

Space complexity

The primary contributors to memory usage include: (i) the infrastructure graph $G_{\text{infra}} = (\mathscr{N}, \mathscr{L})$ requiring $\mathscr{O}(|\mathscr{N}| + |\mathscr{L}|)$ storage; (ii) the active service DAGs $\{G_q\}$ with size $\mathscr{O}(|\mathscr{M}| + \sum_q |E_q|)$; and (iii) the semantic memory G_{mem} which maintains a bounded history via regional sharding and TTL. Additionally, the learned models contribute a fixed parameter budget independent of the workload scale.

Communication complexity

Control-plane messages from the planner to controllers scale as $\mathscr{O}(K \cdot |\texttt{plan}|)$ per window, while telemetry from controllers is compressed into a multimodal snapshot z_t for TopoRAG. Combining scalability and overhead analysis, the planner cost is *amortized* over the asynchronous cadence T, and each agent step remains *constant-time* per cycle. Memory lookup and plan dispatch scale sublinearly or linearly with practical parameters such as $|G_{\text{mem}}|$ and K. Together with partial freezing, LoRA, and a bounded memory store, these properties make the approach feasible for commercial MEC/6G deployments on commodity cloud and edge hardware. Under reasonable sharding and index maintenance, the framework preserves tractable time and space complexity while achieving accelerated convergence and strong QoS guarantees.

Performance evaluation

In this section, we evaluate the proposed AI-native orchestration framework within a city-scale Metaverse-ready 6G edge-cloud environment, emphasizing immersive communication scenarios characterized by ultra-low latency requirements. We use realistic data sets adapted to emulate Metaverse-specific workloads, including XR content delivery, digital twin synchronization, and holographic communications.

Simulation setup

Dataset

To evaluate the proposed framework under realistic 6G-Metaverse conditions, we construct a city-scale edge–cloud testbed by combining real-world datasets with synthetically extended workloads. The OpenCelliD base station dataset [42] provides spatial deployment data for a dense metropolitan region, which we cluster into 200 edge servers (ESs) managed by 8–16 edge cloud orchestrators (ECO), with node-level resource capacities reflecting urban infrastructure and user demand hotspots. The WSDOT highway video dataset [43] includes labeled traffic footage captured from Seattle's I-5 corridor, which we synthetically convert into semantically annotated video streams and distribute to edge nodes using mobility-informed OpenCelliD traces, allowing spatially and temporally realistic service requests. Finally, Alibaba cluster trace [44] is adapted to emulate Metaverse-specific microservice invocation behaviors, including heterogeneous graph structures and compute-intensive job profiles characteristic of XR, digital twin and holographic applications.

Preprocessing for reproducibility We filter OpenCelliD in the target region, remove invalid/duplicate towers, and cluster sites into 200 ES catchments; cluster centroids define ES coordinates, radio profiles F_n, and propagation delays d_{uv} via great-circle distance, while backhaul classes yield each node's latency tuple ℓ_n. WSDOT clips are converted into semantically annotated streams, mapped to ESs using inferred mobility, and assigned per-request payloads drawn from a log-normal distribution ($\mu = 1.5$ MB, $\sigma = 1.3$ MB) to emulate XR/holographic traffic.

Microservice workloads

We consider three representative microservice classes aligned with major Metaverse verticals. XR content delivery DAGs (40%) consist of 4–6 microservices with strict motion-to-photon latency requirements of 10–20 ms. Typical stages include volumetric video stitching, spatial audio rendering, and object localization in real time, with resource demands ranging from medium (0.5 to 1.5 vCPU, 512 MB to 2 GB

RAM) to heavy (1.5 to 4 vCPU, 2 to 8 GB RAM). Digital twin synchronization DAGs (35%) involve 8–15 stages with sub-30 ms synchronization constraints, including sensor fusion, predictive analytics, and virtual–physical model alignment. These workloads range from light (0.1–0.5 vCPU, 128–512 MB RAM) to medium intensity. Holographic communication DAGs (25%) are large graphs with compute intensiveness of 12 to 20 microservices and latency limits below 15 ms.

Request generation

User requests are generated according to a spatially modulated Poisson process augmented with bursty dynamics typical of immersive engagement cycles, with arrival rates ranging from 150–400 requests per second in urban centers and event venues, 80–200 requests per second in medium-density areas, and 10–50 requests per second in suburban or peripheral regions. Collaborative cross-region interactions, such as shared XR sessions, contribute 20–35% of total traffic volume. Each request contains a multimodal input payload sampled from a log-normal distribution ($\mu = 1.5$ MB, $\sigma = 1.3$ MB), representative of immersive content including annotated video streams and 3D scene descriptors. The corresponding CPU workloads range from 50 to 800 million instructions, capturing the heterogeneous computational demands of various Metaverse services.

Environment setup

Our simulation leverages a 6G enabled Metaverse infrastructure, emulated with Docker containers orchestrated through AWS, Docker Compose, and Calico [45]. A global cloud-resident semantic planner runs on a `c6a.4xlarge` EC2 instance (16 vCPUs, 64 GB RAM), providing semantic orchestration updates every 10 min, supplemented by rapid recomputation triggered by significant user mobility shifts or immersive events. CSP controllers (8–16 regional entities) manage edge nodes with 32–128 vCPUs, 64–256 GB RAM, and 20–100 Gbps backhaul connectivity to support immersive Metaverse services. Each of the 200 ESP nodes is provisioned with 6–10 vCPUs, 8–16 GB RAM, and 1–2 Gbps backhaul, scaling to immersive service demand intensity.

Requests are dynamically routed within 250 m (line of sight) or 100 m (nonline of sight), achieving end-to-end latencies of 1–3 ms typical of 6G URLLC. Mobility patterns from 4,328 taxis are replayed at 10-second intervals to simulate user mobility at multiple scales (static, walking speed, real-time driving speed, accelerated). Traffic exhibits pronounced burstiness, modeled using Pareto ON/OFF processes, emulating user behavior in immersive collaborative sessions. Evaluations include 500 Monte Carlo replications to ensure statistical rigor, varying traffic intensity, mobility patterns, and network disruptions.

We utilize the *Qwen2-VL-7B-Instruct* model (13.6 GB, FP16) fine-tuned specifically for semantic orchestration tasks of Metaverse microservices. 70% of the transformer layers remain frozen, with 30% fine-tuned through LoRA Three specialized decoder heads support placement (sigmoid-activated, 512 units), semantic guidance (logits distribution), and context-aware reward estimation (tanh-bounded scalar). TopoRAG maintains a memory of 100K–200K historical deployment cases within Memgraph, processed using GAT for effective real-time retrieval.

DRL agents use the PPO algorithm with a learning rate of 3×10^{-4} (actor) and 1×10^{-3} (critic), optimized by Adam [38] ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$). Each policy update processes 2,048 rollout steps, split into mini-batches of 256, iterated over 10 epochs. GAE parameters ($\gamma = 0.99$, $\lambda = 0.95$) and a clipped PPO threshold (0.2) are applied. The value loss weighting is set at 0.5, with entropy coefficients decreasing linearly from 0.01 to 0.001 over training. The actor and critic networks use three-layer MLP configurations ([128, 256, 128] and [256, 512, 256], respectively) [38]. Simulations use Python 3.11, PyTorch 2.3, and HuggingFace Transformers 4.20 for efficient training and evaluation [46]. Table 2 summarizes key parameters.

Baselines

To evaluate our proposed orchestration framework, we compare it with several state-of-the-art baselines adapted from centralized, graph-based, and distributed DRL methods. RSPPO [7] uses reward shaping and heuristic scaling to optimize placement and routing under dynamic edge conditions. GNN-DRL [9] combines Graph Neural Networks with DRL to learn context-sensitive policies from service and network graphs. μ-DDRL [47] adopts a decentralized approach with local agents using partial observations and neighbor communication. CROMADQN [8] integrates coral reef optimization with multi-agent DQNs for server placement, focusing on energy, delay, and load balancing. All baselines are implemented using available open-source code or re-created from their published methods.

Quantitative comparison of LLMPLANNER variants across load conditions

We evaluate the proposed cloud-based LLMPLANNER using mean squared error (MSE) as the primary metric for prediction accuracy and resource provisioning efficiency under varying network loads. Table 3 presents the orchestration precision of Qwen2-VL under different training configurations and load conditions. Without TopoRAG, performance gradually improves from no pretraining to LoRA-based fine-

Table 2 Key simulation parameters and microservice workload requirements

Parameter	Value/range	Parameter	Value/range
Edge servers (ES)	200	Algorithm	PPO
Edge orchestrators (ECOs)	8–16	Learning rate (A/C)	$3\times10^{-4}/1\times10^{-3}$
Number of BS	4328	GAE/PPO clip	$\gamma = 0.99, \lambda = 0.95, 0.2$
Urban arrival rate	150–400 req/s	Rollout/batch	2048/256, 10 epochs
Medium-density rate	80–200 req/s	MLP architectures	[128, 256, 128], [256, 512, 256]
Suburban rate	10–50 req/s	Entropy coef	$0.01 \rightarrow 0.001$
Cross-region traffic	20–35%	Value loss weight	0.5
Semantic memory	100K–200K cases, memgraph	CPU workload	50–800 M instr
	Metaverse microservice workload classes		
Workload	Microservices	Latency target	Resource (vCPU, RAM)
XR content delivery	4–6	10–20 ms	0.5–4, 0.5–8 GB
Digital twin sync	8–15	< 30 ms	0.1–1.5, 0.1–2 GB
Holographic comm	12–20	< 15 ms	1.5–4, 2–8 GB

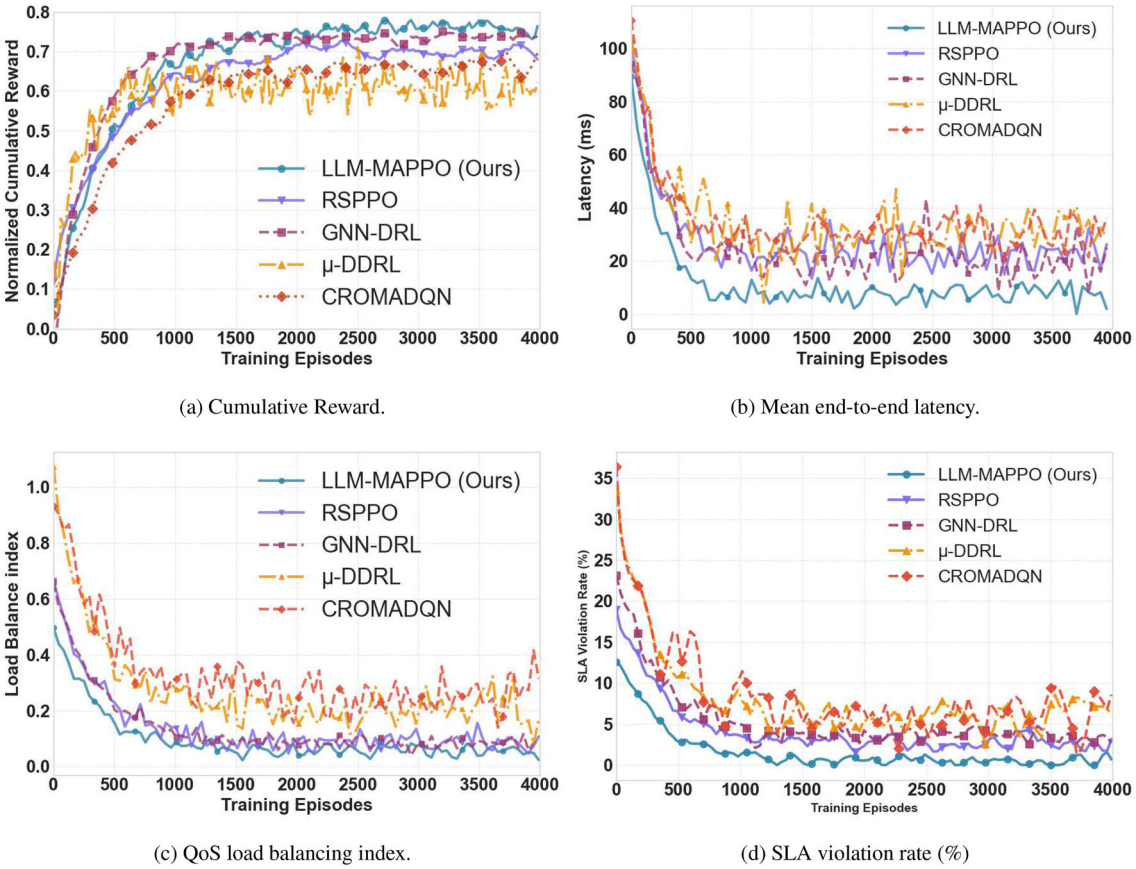
Table 3 Performance comparison of Qwen2-VL for 6G metaverse orchestration across training configurations and network load scenarios

Training method	Low load (≤30%)			Medium load (30–60%)			High load (60–85%)		
	MSE	±	*p*-value	MSE	±	*p*-value	MSE	±	*p*-value
				Baseline (without TopoRAG)					
No pretraining	0.385	0.062	0.004	0.428	0.063	0.003	0.512	0.066	0.002
LoRA adapters	0.234	0.044	0.032	0.267	0.047	0.029	0.328	0.053	0.021
Frozen backbone + LoRA	0.198	0.037	0.045	0.225	0.042	0.041	0.278	0.046	0.039
PFA (ours)	0.215	0.036	0.040	0.243	0.042	0.037	0.295	0.047	0.034
				Enhanced (with TopoRAG)					
No pretraining	0.312	0.056	0.005	0.354	0.055	0.003	0.431	0.058	0.002
LoRA adapters	0.167	0.037	0.030	0.194	0.041	0.027	0.243	0.046	0.023
Frozen backbone + LoRA	0.135	0.034	0.043	0.158	0.036	0.039	0.201	0.040	0.035
PFA + TopoRAG (Ours)	0.087	0.025	–	0.108	0.027	–	0.142	0.032	–

tuning with a frozen backbone, which achieves an MSE of 0.198 ± 0.037 under low load and 0.278 ± 0.046 under high load. With TopoRAG enabled, all models show substantial error reductions due to enhanced semantic retrieval and topology-aware context propagation. The proposed PFA + TopoRAG configuration achieves the best performance across all load levels, with MSEs of 0.087 ± 0.025 under low load and 0.142 ± 0.032 under high load, corresponding to up to 56% improvement (under low load) relative to the best baseline not tuned with TopoRAG. This improvement results from two factors: partial fine-tuning preserves the generalisation strength of the pretrained LLM while efficiently adapting it to orchestration tasks, and TopoRAG augments planning with deployment-aware retrieval from a topology-indexed memory. Paired t-tests confirm the statistical significance of these improvements with $p < 0.05$, demonstrating the robustness of our method under realistic and fluctuating network conditions.

Convergence and learning analysis

In Fig. 4, we analyze the convergence and training performance of our proposed LLM-MAPPO framework using a 24-hour microservice workload trace. This trace simulates diurnal request patterns, with urban peaks of 150–400 req/s and off-peak loads of 10–15 req/s, covering a full day of activity. Figure 4a shows the total normalized cumulative reward over training episodes. LLM-MAPPO consistently outperforms all baselines, achieving a peak reward of approximately 0.78 and converging smoothly with negligible oscillations after episode 1500. This stable ascent highlights the effectiveness of the LLM-driven central planner, which uses TopoRAG to provide semantically rich priors that accelerate policy improvement. In contrast, RSPPO achieves a respectable reward of around 0.75, confirming that reward-shaping alone can produce strong multiobjective performance. GNN-DRL levels drop to near 0.72, showing that while graph-based state representations capture some topo-



(a) Cumulative Reward.

(b) Mean end-to-end latency.

(c) QoS load balancing index.

(d) SLA violation rate (%)

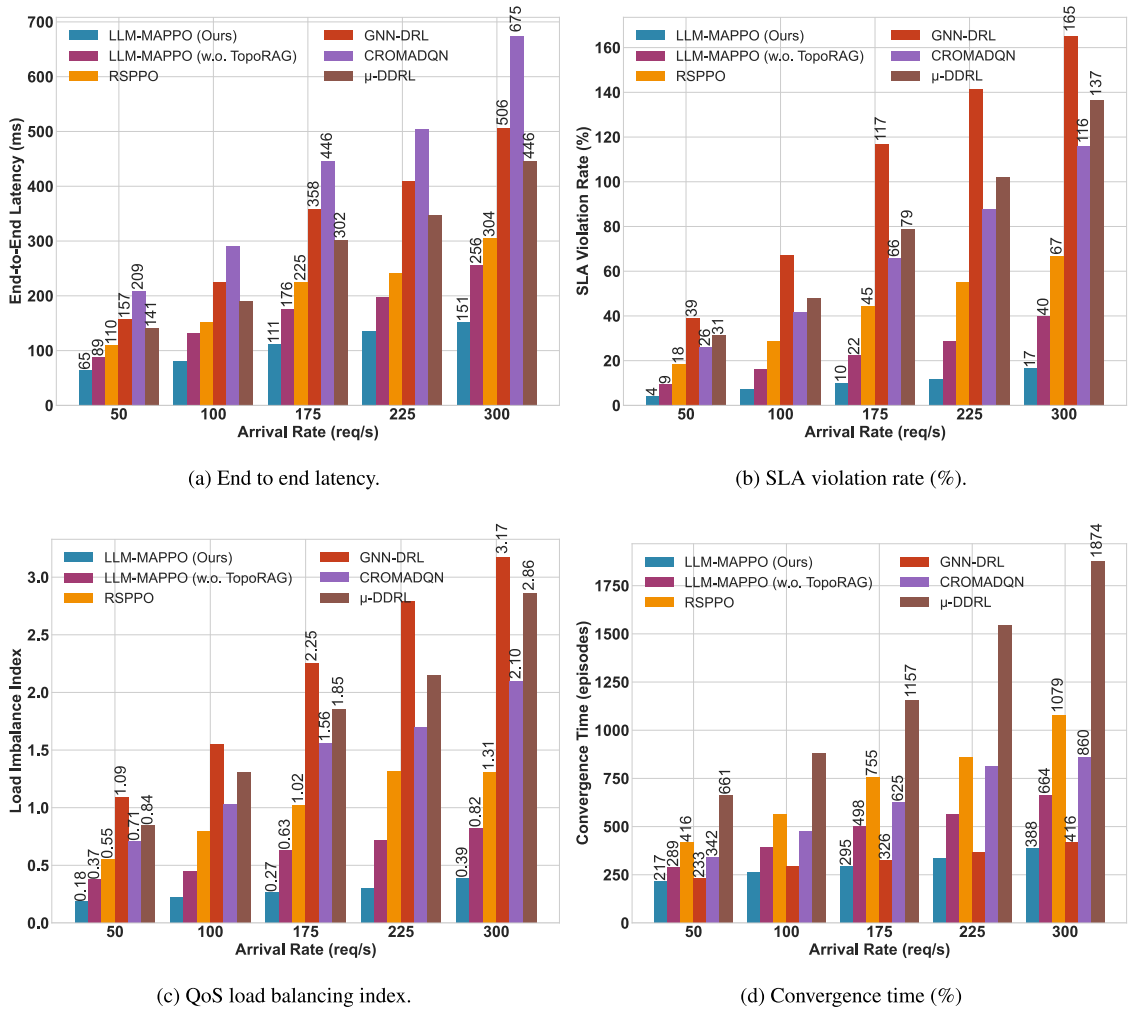
Fig. 4 Training convergence comparison across four algorithms

logical structure, they may struggle with rapidly changing loads. The μ-DDRL and CROMADQN agents both plateau between 0.60 and 0.65 with pronounced variance, suggesting that fully decentralized schemes and multiagent Q-learning are less effective in the high-dimensional, non-stationary decision space of the microservice deployment. This superiority is also reflected in the mean end-to-end latency shown in Fig. 4b. LLM-MAPPO rapidly converges to a stable 8–10 ms, significantly lower than RSPPO (20 ms), GNN-DRL (25 ms) and inconsistent μ-DDRL and CROMADQN (30–35 ms). The fast convergence of LLM-MAPPO and its sustained low latency underscore its suitability for real-time applications under dynamic conditions.

To verify QoS and the algorithm's ability to meet SLA during convergence, we analyze the load balance index (Fig. 4c) and the SLA Violation Rate (Fig. 4d). In Fig. 4c, LLM-MAPPO achieves a near-optimal load distribution, converging to approximately 0.05, reflecting uniform resource use across edge nodes. This outperforms RSPPO (0.08) and GNN-DRL (0.12), whose performance indicates that graph models capture local but not global trends. In contrast, μ-DDRL and CROMADQN remain at 0.25–0.35 with high variance, underscoring the instability of fully decentralized schemes. The smooth convergence of LLM-MAPPO to a low index highlights the power of centralized, LLM-guided planning for hotspot avoidance and system-wide fairness. Figure 4d shows that LLM-MAPPO maintains an SLA violation rate below 2% even before full convergence, a reduction of 63% compared to baselines. This is crucial for latency-sensitive Metaverse services. RSPPO maintains violations at 3–4% and GNN-DRL at 4–5%, illustrating solid but weaker QoS control. Meanwhile, μ-DDRL and CROMADQN incur 6–10% violations with erratic spikes, which shows that they are unreliable for production environments.

Performance evaluation under varying arrival rates

In Fig. 5, we analyze scalability by testing each method with increasing request intensities. First, Fig. 5a plots the mean end-to-end latency against varying arrival rates. At a low load of 50 req/s, LLM-MAPPO sustains a latency of just 65 ms, significantly outperforming all baselines. As the load intensifies to 300 req/s, its latency increases smoothly to 151 ms, which is the smallest degradation observed among all the methods tested. RSPPO performs well at lower loads, but experiences a sharp spike to 304 ms at peak traffic. In



(a) End to end latency.

(b) SLA violation rate (%).

(c) QoS load balancing index.

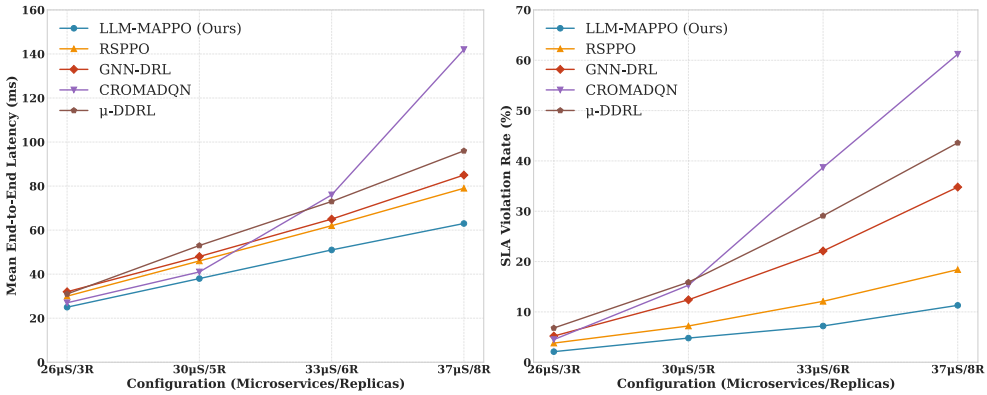
(d) Convergence time (%)

Fig. 5 System performance under varying arrival rates

contrast, the latency for GNN-DRL, CROMADQN, and μ-DDRL exceeds 400–600 ms under heavy demand, reflecting their limited ability to adapt to dynamic workloads. The near-linear latency growth of LLM-MAPPO highlights its robust architectural design for stress-resilient edge computing.

Next, the analysis of the SLA violation rate in Fig. 5b reveals the reliability of the service under stress. LLM-MAPPO successfully keeps violations below 20% even at a maximum load of 300 req/s, having started from only 4% at 50 req/s. RSPPO scales moderately, maintaining acceptable rates up to 175 req/s before violations spike to 55% at maximum load. Traditional methods perform poorly. GNN-DRL records 165% violations at maximum load, which indicates systemic overload. CROMADQN and μ-DDRL are similarly ineffective, exceeding 100–135% violations. In Fig. 5c, the QoS load balance index highlights the superior resource distribution of LLM-MAPPO under varying loads. At 50 req/s, it achieves an index of 0.18, well below all baselines. This index only increases to 0.39 at 300 req/s, demonstrating stable resource management even under peak demand. Although RSPPO starts competitively, its index degrades to 1.31 at maximum load. GNN-DRL peaks at 3.17, and CROMADQN and μ-DDRL exceed 2.0–2.8, indicating severe imbalances and hotspot formation.

Finally, the convergence time analysis in Fig. 5d reveals the adaptation speed of each algorithm under dynamic loads. LLM-MAPPO converges the fastest, requiring just 274 episodes at 50 req/s and 386 episodes at 300 req/s, which demonstrates its superior learning efficiency. RSPPO follows, taking between 289 and 1,075 episodes. In contrast, GNN-DRL is sluggish, requiring 661 to 1874 episodes. CROMADQN and μ-DDRL perform the worst, needing 453–825 and 295–880 episodes, respectively, with high variance that reflects unstable learning. The steep increase in the convergence time for baselines under heavy load underscores their unsuitability for dynamic edge environments.



(a) Mean end-to-end latency.

(b) SLA violation rate.

Fig. 6 System performance under varying system complexity

Impact of microservices complexity

In Fig. 6, we evaluate algorithm performance under increasing complexity of Metaverse service. To achieve this, we vary the number of microservices from 26 to 37 and the replica count from 3 to 8. This setup reflects a range of scenarios, from basic XR interactions to demanding applications like high-fidelity volumetric video stitching and real-time physics simulation. In Fig. 6a, we analyze latency. Under the baseline service graph of 26 services and 3 replicas, LLM-MAPPO achieves a 25 ms latency, which is sufficient to ensure sub-100 ms motion-to-photon responsiveness for seamless holographic streams. As complexity grows to 37 services and 8 replicas, LLM-MAPPO's latency increases only to 62 ms (a 2.5x rise), demonstrating graceful degradation. Other methods struggle to orchestrate large Metaverse workloads, with latency rising from 30 to 78 ms for RSPPO, 85 ms for GNN-DRL, 96 ms for μ-DDRL, and 143 ms for CROMADQN.

In Fig. 6b, we examine the SLA violation rate. LLM-MAPPO maintains violations below 2% at low complexity and only 11% at maximum complexity. This is possible because TopoRAG's topology-aware priors preempt bottlenecks in expansive service graphs. In contrast, RSPPO violations increase from 3% to 18%, while GNN-DRL, μ-DDRL, and CROMADQN deliver unacceptable performance for Metaverse production deployments, with violations increasing to 35%, 43%, and 61%, respectively.

Aggregate performance statistics for varying workload complexity

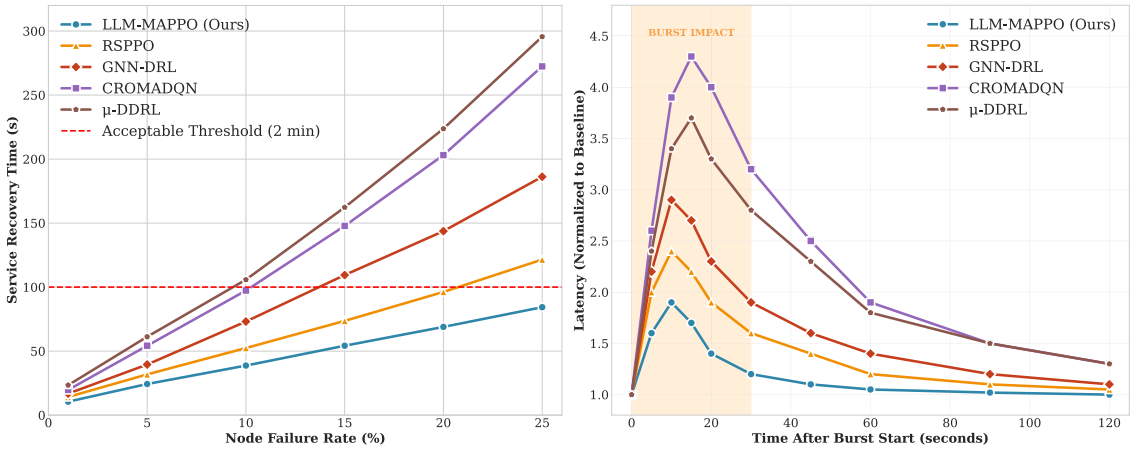
To ensure statistical robustness and assess generalization capabilities, we conduct evaluations over five independent random seeds and four distinct workload configurations. Specifically, we consider (i) a baseline XR scenario comprising 26 microservices and 3 replicas, (ii) a multiuser gaming scenario with 30 microservices and 5 replicas, (iii) a digital twin scenario with 33 microservices and 6 replicas, and (iv) a volumetric video scenario with 37 microservices and 8 replicas. As shown in Table 4, LLM-MAPPO achieves the lowest mean latency of 44.3 ± 15.8 ms, improving RSPPO by 17% and GNN-DRL by 26%. Crucially, tail latencies remain well below the SLA threshold, with a p99 of 71.5 ms, whereas the baselines exhibit p95 tails ranging from 78 to 151 ms. This responsiveness results in superior reliability, maintaining a violation rate of 6.5% with a narrow confidence interval. This represents a statistically significant improvement over RSPPO at 10.3% and dramatic reductions compared to GNN-DRL at 23.8% and μ-DDRL at 31.0%, confirming the efficacy of TopoRAG in bottleneck preemption. Regarding training dynamics, LLM-MAPPO converges in a median of 820 episodes, 24% faster than RSPPO. The low interquartile range of 85 episodes indicates stable learning from semantic guidance, whereas μ-DDRL and CROMADQN, despite converging earlier, show high variance and poor final performance, suggesting premature convergence to suboptimal policies.

Reliability and adaptation to dynamic conditions

In Fig. 7, we assess the reliability of the proposed algorithm in the event of sudden infrastructure failures and traffic surges. In these simulations, we introduce random edge-node outages and bursty request loads to evaluate fault tolerance and adaptation speed. Figure 7a illustrates the resilience of LLM-MAPPO to random edge node outages. With up to 5% nodes failing, it restores full service in less than 40 s, outpacing all baselines. Even with a 25% failure rate, recovery remains below 85 s, well within the 2-min SLA target. This robustness comes from proactive replanning of the LLM planner, which uses TopoRAG's insights to quickly identify

Table 4 Aggregate performance statistics for varying workload complexity

Method	Mean latency $\mu \pm \sigma$	p95 / p99 latency (ms)	SLA (%) violation	CI_{95} violations	Convergence median [IQR]
LLM-MAPPO	44.3 ± 15.8	62.1/71.5	6.5	[5.8, 7.2]	820 [780, 865]
RSPPO	53.4 ± 19.2	78.3/93.8	10.3	[9.4, 11.4]	1075 [1020, 1145]
GNN-DRL	59.7 ± 21.6	89.5/108.2	23.8	[22.1, 25.8]	1650 [1580, 1740]
μ-DDRL	71.5 ± 28.3	112.4/138.7	31.0	[28.9, 33.5]	650 [590, 725]
CROMADQN	92.8 ± 36.7	151.3/184.9	39.0	[36.8, 41.6]	680 [625, 760]



(a) Service recovery time after edge node failures.

(b) Adaptation speed to sudden traffic bursts.

Fig. 7 System reliability and adaptation under dynamic arrival rates

alternative placements. RSPPO also meets the SLA for up to 20% failures. In contrast, GNN-DRL crosses the 120 s threshold at 15% failures, while μ-DDRL and CROMADQN fail at just 10% outages, taking nearly 300 s to recover under severe conditions. The LLM's diverse contingency plans allow the DRL agents to switch to precomputed fallbacks, minimizing downtime.

Figure 7b shows rapid adaptation to sudden traffic bursts. During an instantaneous request spike (0–20 s), LLM-MAPPO limits the latency increase to 1.9x over baseline and restores normal performance in 40 s, the fastest of all methods. RSPPO follows with a 2.4x peak and a 60-second recovery. In contrast, CROMADQN and μ-DDRL suffer 4.3x and 3.7x latency surges and require 80–120 s to stabilize. Their sluggish response highlights the limitations of decentralized management for bursty workloads. The TopoRAG module provides topology-aware priors, enabling immediate resource reallocation without waiting for full policy retraining and ensuring consistent QoS.

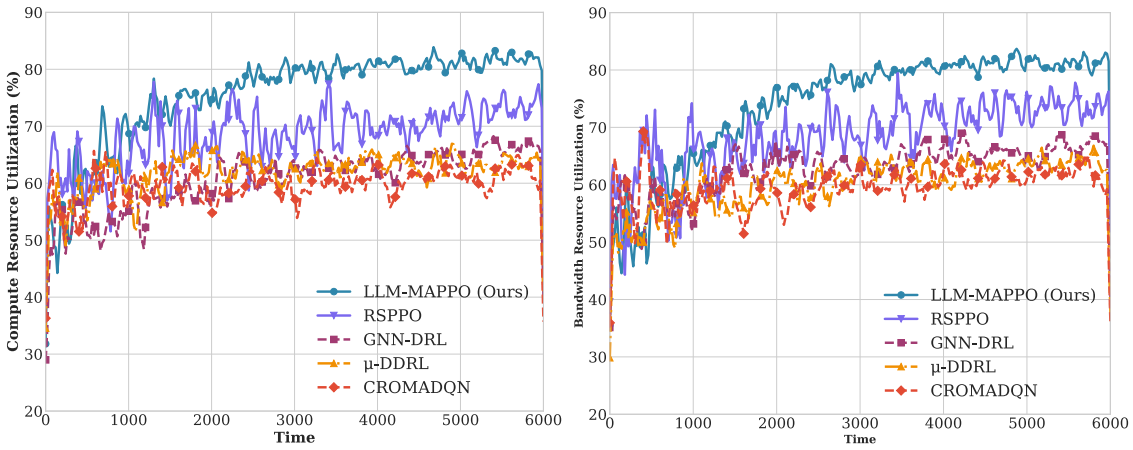
Resource utilization and spatial load balancing

In Fig. 8, we analyze the utilization of compute and bandwidth during a 24-h Metaverse service run and demonstrate how the proposed LLM-MAPPO algorithm maximizes resource efficiency while maintaining stability.

As depicted in Fig. 8a, LLM-MAPPO exhibits higher compute utilization, reaching a peak of approximately 82% and consistently maintaining a level above 78%, thus illustrating its ability to efficiently allocate workloads without exceeding the capacity of any individual node. This performance is guided by TopoRAG's topology-aware priors, which help exploit available CPU cycles fully. For comparison, RSPPO averages 74% utilization with dips to 68%, indicating a less consistent balance. GNN-DRL and μ-DDRL plateau at 65% and 63%, respectively, leaving substantial capacity idle, while CROMADQN reflects unstable allocation by fluctuating between 58 and 65%.

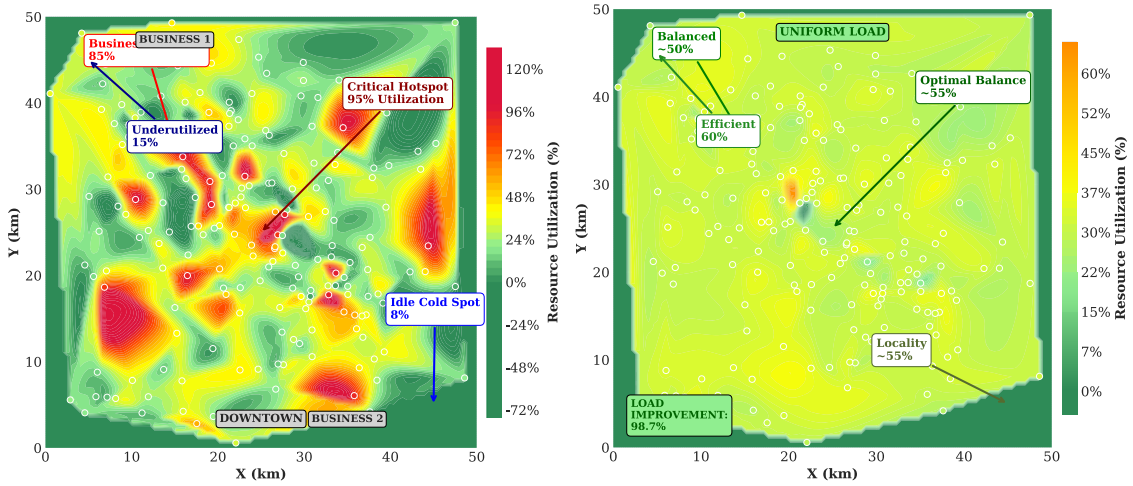
Figure 8b shows a similar pattern for the bandwidth. LLM-MAPPO maintains around 81% utilization with less than 5% % variance, ensuring high throughput. This is achieved by jointly optimizing placement and routing to prevent congestion. RSPPO averages 73%, while GNN-DRL, μ-DDRL, and CROMADQN only achieve 61–66%, which is prone to bottlenecks. The sustained and substantial utilization rate of LLM-MAPPO highlights its sophisticated network-aware orchestration capabilities.

Figure 8c presents the distribution of resource utilization before convergence, while Fig. 8d illustrates this distribution



(a) Resource utilization of compute over time.

(b) Resource utilization of bandwidth over time.



(c) Load balance before orchestration.

(d) Load balance after orchestration.

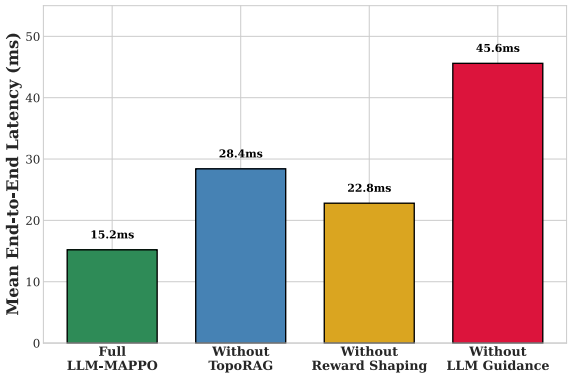
Fig. 8 Effectiveness of resource utilization and spatial load balancing

after convergence. In Fig. 8c, we observe a severe spatial imbalance where the central nodes, corresponding to the dense business and entertainment districts, reach a utilization of 95% utilization while the peripheral nodes idle at 8%. This disparity arises from a naïve, demand-agnostic placement that overloads hotspots and underuses resources on the fringes of the network. After convergence (Fig. 8d), utilization is uniformly distributed between 50 and 60% at all nodes. TopoRAG's priors correct for these biases by factoring in network distances and real-time load, enabling intelligent service migrations that eliminate hotspots.

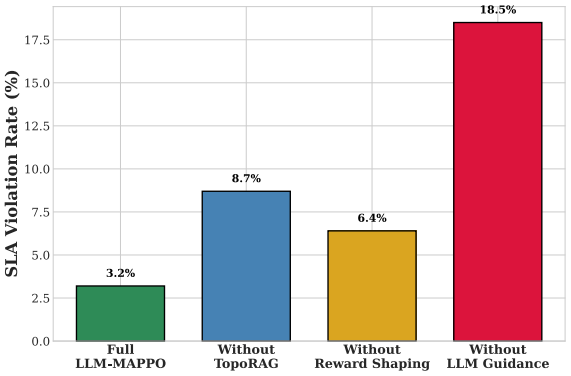
Ablation and hyperparameter sensitivity analysis

The ablation study in Fig. 9a–c confirms that each architectural component is critical to performance. The complete LLM-MAPPO framework sets a strong baseline with 15.2 ms latency, 3.2% SLA violations, and 820-episode convergence. Removal of the LLM guidance causes the most severe degradation, increasing latency by 200%, violations by 5.8x, and convergence time by 3.9x. The absence of TopoRAG is also highly detrimental, increasing latency by 87%, violations by 2.7x, and more than double the training time. Finally, removing reward shaping increases latency by 50% and violations by 2x.

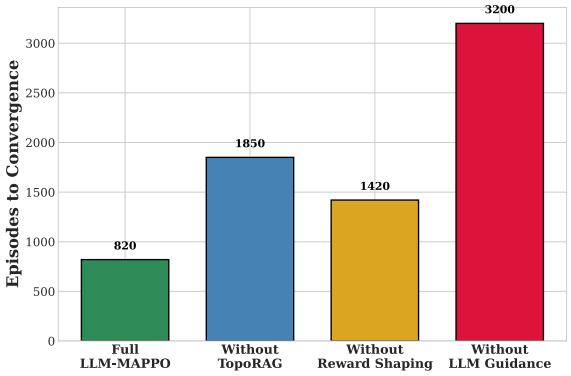
A hyperparameter sensitivity analysis (Fig. 9d) reveals a clear tuning hierarchy. The frequency of planning is the dominant setting, mainly driving SLA violations (63.8% sensitivity) and overhead (39.9%). The frozen layer ratio follows, creating a crucial trade-off between adaptation and stability, with high impacts on SLA compliance (39.1%) and convergence (24.8%).



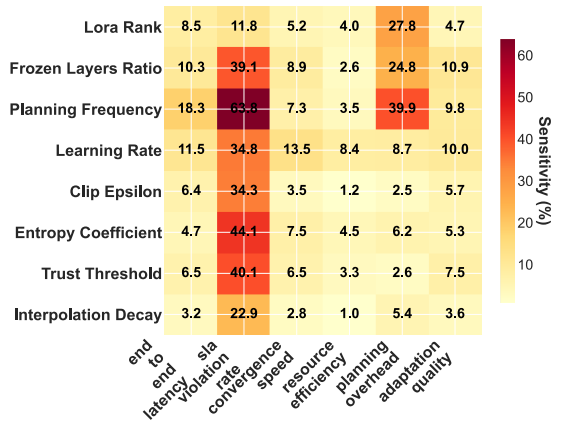
(a) Latency Impact.



(b) Service Quality Impact.

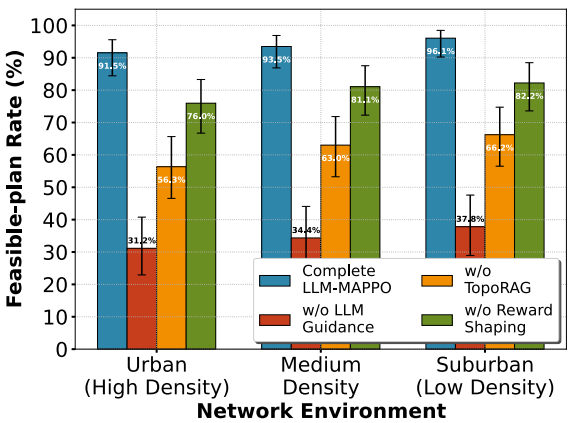


(c) Training Efficiency Impact.

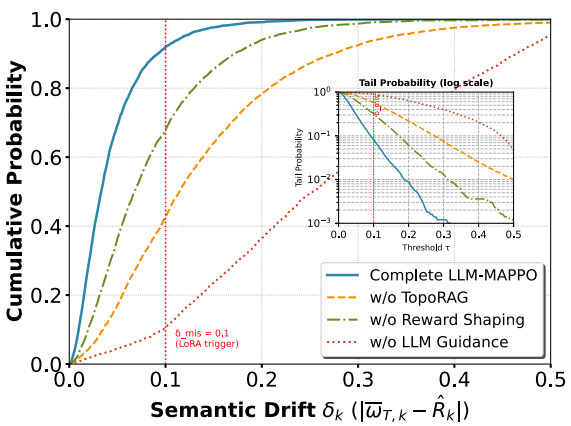


(d) Hyperparameter Sensitivity Analysis.

Fig. 9 Ablation and hyperparameter sensitivity analysis



(a) Feasible-plan rate.



(b) Semantic drift distribution δ_k.

Fig. 10 Performance of safety, semantic drift, and pre-agent plan feasibility

Performance of safety, semantic drift, and pre-agent plan feasibility

In Fig. 10, we evaluate planner reliability using two metrics: pre-agent feasibility and semantic alignment. Pre-agent feasibility measures the proportion of generated plans that satisfy hard constraints (capacity, connectivity, and slice isolation) at dispatch time, prior to any corrective actions by edge agents. The semantic drift, denoted δ_k, quantifies the deviation between the intended utility of the planner and the actual utility achieved. Together, these metrics capture whether the planner produces safe and well-grounded decisions off the critical path and how much correction the agents must subsequently apply.

Across all network regimes, the Complete Framework consistently exhibits high pre-agent feasibility, ranging from 91.5% in dense urban environments to 96.1% in suburban settings, as shown in Fig. 10a, based on Wilson 95% confidence intervals. Ablation studies highlight the contribution of each component. Removing LLM guidance reduces feasibility to 31–38%, an average drop of 63.2% relative to the full system. Excluding TopoRAG reduces feasibility to 56–66%, while removing reward shaping causes moderate degradation, with feasibility between 76% and 82%. Urban settings remain the most challenging due to tighter resource headroom and greater contention.

For semantic alignment, as illustrated in Fig. 10b, the Complete Framework maintains a low drift, with a median δ_k of 0.033 and an interquartile range of [0.018, 0.057]. The drift tails remain light, with the 95th percentile at 0.119 and fewer than 0.88% of cases exceeding 0.2. The removal of TopoRAG increases the mean drift to 0.114 and raises the 95th percentile to 0.335, causing deviations greater than 0.1 in more than 57% of cases; this confirms that topology-aware retrieval is the primary mechanism for reducing semantic divergence. Removing reward shaping produces moderate drift growth with a median of 0.067, indicating its influence on accelerating post-dispatch alignment. Without LLM guidance, drift becomes severe, with a median of 0.253 and nearly 90% of cases showing deviations above 0.1. These findings underscore the effectiveness of semantic guidance and trust-gating.

The combination of high dispatch feasibility and low drift tails demonstrates that the planner can operate asynchronously and off the per-request critical path. Edge agents are left to perform only bounded, rapid corrections, enabling an operational profile that aligns with the strict latency and reliability requirements of commercial 6G MEC deployments.

Table 5 Commercial MEC benchmarks and 3GPP/ETSI specifications

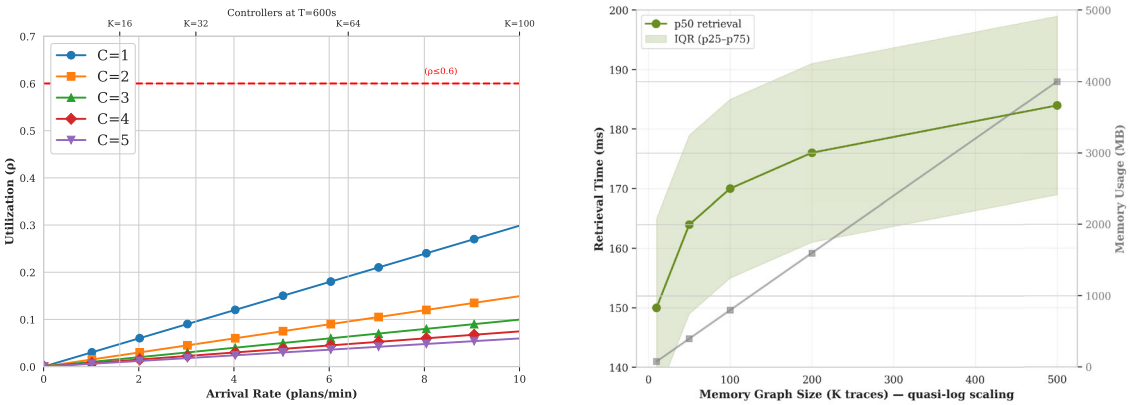
Parameter	Value
Edge node compute resources	
CPU cores per node	24–48 vCPUs
CPU frequency	2.2–3.5 GHz
GPU model	NVIDIA T4
GPU memory/compute	16 GB/65 TFLOPS
RAM capacity	64–128 GB

Real-world deployment feasibility analysis

We analyze the practical feasibility of our AI-native orchestration through a containerized deployment on an HPC cluster. The planner (TopoRAG+LLM) and controller tier are encapsulated in Docker containers and configured with real-world 6G edge–cloud parameters derived from commercial MEC benchmarks, as summarized in Table 5. Edge nodes are provisioned with compute profiles that match Azure Stack Edge Pro deployments [48] (24–48 vCPUs, 64–128 GB RAM and NVIDIA T4 GPUs), while network latencies and V2X constraints follow 3GPP TR 38.913 [49] and TS 22.186 specifications [50].

We first validate the scalability of the LLM-based planner to ensure it supports dense multi-controller deployments without inducing control-plane bottlenecks. Figure 11a analyzes planner worker utilization ρ under an M/G/C queuing model, defined as $\rho = \lambda s/C$. Here, a pool of C workers serves K edge controllers with an arrival rate $\lambda = K/T$, based on a planning cadence of $T = 600$ s and a measured service time $s \approx 1.792$ s. The figure maps these arrival rates to controller counts ($K \in \{16, \ldots, 100\}$) against a conservative stability threshold of $\rho \leq 0.6$. Configuring the pool with $\mathbf{C = 3}$ workers yields a negligible load: even aggressive deployments of $K = 100$ controllers maintain $\boldsymbol{\rho \approx 0.10}$, while moderate setups ($K = 32$) operate at $\boldsymbol{\rho \approx 0.032}$.

A critical enabler of the above utilization profile is TopoRAG's sub-linear retrieval scaling. Figure 11b quantifies the retrieval performance as the memory graph expands. The green curve (left axis) shows that the median retrieval latency (p50) increases from 150 ms at 10 k traces to only 184 ms at 500 k traces, a modest 23% increase despite a 50× growth in corpus size. The shaded interquartile range (IQR) band (p25–p75) remains tight, widening only slightly from ±10 ms to ±12 ms, with the 75th-percentile latency consistently below 200 ms. The gray curve (right axis) confirms a strictly linear growth in the memory footprint at approximately 8 KB per trace, resulting in approximately 4 GB at 500 k traces, which is well within the limits of commodity server DRAM.



(a) **Planner worker utilization vs. arrival rate.** (b) **TopoRAG scaling with memory size.**

Fig. 11 Commercial deployment feasibility

Table 6 Inference cost and energy overhead per planning cycle

Metric	Per cycle	Daily (4 cycles)
Input tokens	∼1200	∼4800
Output tokens	∼180	∼720
Decode time	1.32 s	5.28 s
Inference cost (API)	$0.15	$ 0.60-$ 1.80[†]
Energy consumption	0.42 kWh	1.68 kWh
Edge baseline overhead	–	<5%

[†]Range reflects provider pricing variation and candidate count $B \in \{3, 5\}$

Furthermore, we show that the inference cost and energy overhead introduced by LLM-based planning are minimal due to three design choices: asynchronous invocation, token-budgeted prompts and parameter-efficient fine-tuning. Table 6 summarizes the operational costs of the LLM planner. With a partially frozen LoRA-adapted decoder, typical planning calls consume $\mathcal{O}(10^3)$ input tokens and generate concise output plans. At four planning cycles per day, the daily LLM cost remains $0.60 to $1.80, which is negligible compared to always-on DRL baselines that require continuous GPU allocation. The energy overhead remains modest at less than 5% relative to the edge-based baseline, driven primarily by the combined efficiencies of asynchronous window planning, token-budgeted prompt construction and LoRA-based parameter adaptation.

Conclusion

In this paper, we introduced an AI-native orchestration framework to address the ultra-low-latency demands of Metaverse applications in 6G cloud–edge environments. Our approach integrates a fine-tuned LLM as a cognitive planner with TopoRAG, a module that ingests live telemetry and network graphs to enable topology-sensitive decision-making. In addition, an LLM-guided reward-shaping mechanism accelerates the convergence of distributed edge agents by encoding global constraint priorities. Extensive evaluations on city-scale topologies with realistic workloads, including volumetric video and multiuser XR gaming, demonstrate that our framework achieves 17–35% lower latency, 37–63% fewer SLA violations, and 24–50% faster convergence compared to state-of-the-art baselines. Based on our simulations, we recommend running the cloud planner asynchronously off the per-request critical path with a conservative planning cadence, and scaling regional controllers elastically. TopoRAG memory should be sharded by region with TTL-based and priority-based pruning, while compact, partially frozen LLMs with lightweight adapters are preferred to control cost and energy. Finally, pre-agent plan-feasibility and semantic-drift checks should be instrumented so that edge agents can down-weight stale guidance without impacting real-time responsiveness.

Limitations

First, the memory graph remains vulnerable to long-term trace aging and adversarial poisoning, as current pruning methods cannot fully mitigate non-stationary or malicious inputs. Second, the framework prioritizes aggregate SLA compliance over multi-tenant fairness and privacy, currently lacking specific mechanisms to ensure equitable resource allocation or protect sensitive deployment metadata in shared infrastructure.

Future directions

Future research will focus on hierarchical planning with Small Language Models (SLMs) to enhance scalability and privacy. Lightweight, domain-specific SLMs at regional controllers will handle sub-second microservice placement, while a cloud-resident LLM provides strategic guidance. This decomposition enables federated learning, allowing

operators to fine-tune local adapters on proprietary traces without exposing raw data. Additionally, we aim to integrate fairness and security constraints by extending the shaped reward with per-tenant fairness terms (e.g., Jain's index) and developing real-time trace validation modules. Integrating differential privacy guaranties will further protect deployment traces from inference attacks while maintaining TopoRAG retrieval quality.

This work represents a significant step toward autonomous 6G networks, showcasing the transformative potential of combining symbolic reasoning with subsymbolic learning to solve complex distributed system optimization problems at scale.

Author Contributions DA-M: Conceptualization, Methodology, Software, Data Curation, Formal Analysis, Writing—Original Draft. AKG: Methodology, Software, Validation, Writing—Original Draft. GOB: Investigation, Literature Review, Data Curation, Visualization. RM: Conceptualization, Methodology, Supervision, Writing—Review and Editing. AM: Conceptualization, Project Administration, Supervision, Writing—Review and Editing. HO: Conceptualization, Theoretical Framework, Supervision, Writing—Review and Editing. JB: Conceptualization, Theoretical Framework, Supervision, Writing—Review and Editing. All authors have read and agreed to the published version of the manuscript.

Funding The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability The data that support the findings of this study are available on request from the corresponding author, Prof. Azzam Mourad. The data are not publicly available due to ongoing project requirements. Upon completion of the project, the full code and data will be made publicly available at: https://github.com/amew0/o.git.

Code Availability Code is available at: https://github.com/amew0/o.git.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no Conflict of interest.

References

1. Karunarathna S, Wijethilaka S, Ranaweera P, Hemachandra KT, Samarasinghe T, Liyanage M (2023) The role of network slicing and edge computing in the metaverse realization. IEEE Access 11:25502–25530. https://doi.org/10.1109/ACCESS.2023.3255510
2. Adeel A, Gogate M, Farooq S, Ieracitano C, Dashtipour K, Larijani H, Hussain A (2019) In: A survey on the role of wireless sensor networks and IoT in disaster management. Durrani TS, Wang W, Forbes SM (eds), pp 57–66. Springer, Singapore. https://doi.org/10.1007/978-981-13-0992-2_5
3. Aloudat MZ, Aboumadi A, Soliman A, Al-Mohammed HA, Al-Ali M, Mahgoub A, Barhamgi M, Yaacoub E (2025) Metaverse unbound: a survey on synergistic integration between semantic communication, 6g, and edge learning. IEEE Access 13:58302–58350. https://doi.org/10.1109/ACCESS.2025.3555753
4. Mao B, Zhou X, Liu J, Kato N (2024) On a cooperative deep reinforcement learning-based multi-objective routing strategy for diversified 6g metaverse services. IEEE Trans Veh Technol 73(9):14092–14096. https://doi.org/10.1109/TVT.2024.3397707
5. Blinowski G, Ojdowska A, Przybyłek A (2022) Monolithic vs. microservice architecture: a performance and scalability evaluation. IEEE Access 10:20357–20374. https://doi.org/10.1109/ACCESS.2022.3152803
6. Li G, Tan H, Liu L, Zhou H, Jiang SH-C, Han Z, Li X-Y, Chen G (2023) Dag scheduling in mobile edge computing. ACM Trans Sen Netw 20(1):1–28. https://doi.org/10.1145/3616374
7. Hu M, Wang H, Xu X, He J, Hu Y, Deng T, Peng K (2024) Joint optimization of microservice deployment and routing in edge via multi-objective deep reinforcement learning. IEEE Trans Netw Serv Manage 21(6):6364–6381. https://doi.org/10.1109/TNSM.2024.3443872
8. Asghari A, Sohrabi MK (2022) Multiobjective edge server placement in mobile-edge computing using a combination of multiagent deep q-network and coral reefs optimization. IEEE Int Things J 9(18):17503–17512. https://doi.org/10.1109/JIOT.2022.3161950
9. Chen S, Yuan Q, Li J, He H, Li S, Jiang X, Yang J (2024) Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing. IEEE Trans Serv Comput 17(6):3742–3757. https://doi.org/10.1109/TSC.2024.3417241
10. Zhang R, Zhao C, Du H, Niyato D, Wang J, Sawadsitang S, Shen X, Kim DI (2025) Embodied ai-enhanced vehicular networks: An integrated vision language models and reinforcement learning method. IEEE Trans Mob Comput 24(11):1–16. https://doi.org/10.1109/TMC.2025.3582864
11. Zhou H, Hu C, Yuan D, Yuan Y, Wu D, Chen X, Tabassum H, Liu X (2025) Large language models for wireless networks: an overview from the prompt engineering perspective. IEEE Wirel Commun 32(4):1–9. https://doi.org/10.1109/MWC.001.2400384
12. Guo W (2020) Explainable artificial intelligence for 6g: improving trust between human and machine. IEEE Commun Mag 58(6):39–45. https://doi.org/10.1109/MCOM.001.2000050
13. Wang J, Guo L, Wu J, Yan C, Sun H, Zhang L, Zhuang Z, Qi Q, Liao J (2025) Hierarchical index retrieval-driven wireless network intent translation with llm. IEEE Trans Mob Comput 24(10):1–15. https://doi.org/10.1109/TMC.2025.3564937
14. Zhao Y, Chai Z, Li Y, Huang H, Kang H (2024) Multi-objective computation offloading based on decentralized deep reinforcement learning in industrial internet of things. IEEE Trans Cogn Commun Netw. https://doi.org/10.1109/TCCN.2024.3466889
15. Sami H, Mourad A, Otrok H, Bentahar J (2022) Demand-driven deep reinforcement learning for scalable fog and service placement. IEEE Trans Serv Comput 15(5):2671–2684. https://doi.org/10.1109/TSC.2021.3075988

16. Zare M, Elmi Sola Y, Hasanpour H (2023) Towards distributed and autonomous iot service placement in fog computing using asynchronous advantage actor-critic algorithm. J King Saud Univ Comput Inf Sci 35(1):368–381. https://doi.org/10.1016/j.jksuci.2022.12.006
17. Hu Y, Wang H, Wang L, Hu M, Peng K, Veeravalli B (2023) Joint deployment and request routing for microservice call graphs in data centers. IEEE Trans Parallel Distrib Syst 34(11):2994–3011. https://doi.org/10.1109/TPDS.2023.3311767
18. Zhang X, Zhao H, Xiong J, Liu X, Yin H, Zhou X, Wei J (2024) Decentralized routing and radio resource allocation in wireless ad hoc networks via graph reinforcement learning. IEEE Trans Cogn Commun Netw 10(3):1146–1159. https://doi.org/10.1109/TCCN.2024.3360517
19. Moustafa N (2024) Gh-twin: graph learning empowered hierarchical digital twin for optimizing self-healing networks. Sustain Mach Intell J 8(3):35–45
20. Qu G, Chen Q, Wei W, Lin Z, Chen X, Huang K (2025) Mobile edge intelligence for large language models: a contemporary survey. IEEE Commun Surv Tutor. https://doi.org/10.1109/COMST.2025.3527641
21. Jiang F, Peng Y, Dong L, Wang K, Yang K, Pan C, Niyato D, Dobre OA (2024) Large language model enhanced multi-agent systems for 6g communications. IEEE Wirel Commun 31(6):48–55. https://doi.org/10.1109/MWC.016.2300600
22. Wu D, Wang X, Qiao Y, Wang Z, Jiang J, Cui S, Wang F (2024) Netllm: adapting large language models for networking. In: Proceedings of the ACM SIGCOMM 2024 conference. ACM SIGCOMM '24, pp 661–678. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/3651890.3672268
23. Rao K, Coviello G, Benedetti P, Giuseppe De Vita C, Mellone G, Chakradhar S (2024) Eco-llm: llm-based edge cloud optimization. In: Proceedings of the 2024 workshop on AI For Systems. AI4Sys '24, pp 7–12. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/3660605.3660941
24. Feng K, Luo L, Xia Y, Luo B, He X, Li K, Zha Z, Xu B, Peng K (2024) Optimizing microservice deployment in edge computing with large language models: integrating retrieval augmented generation and chain of thought techniques. Symmetry 16(11):1470. https://doi.org/10.3390/sym16111470
25. Abbasi SF, Bilal M, Mukherjee T, Pournik O, Moukafih N, Epiphaniou G, Maple C, Arvanitis TN (2025) Preliminary exploration of pre-trained vision-language models for cyber-threat modelling in internet of medical things (iomt). In: Intelligent health systems—from technology to data and knowledge: proceedings of MIE 2025. Studies in health technology and informatics, vol 327, pp 1195–1199. IOS Press. https://doi.org/10.3233/SHTI250579
26. Lu J, Yang W, Xiong Z, Xing C, Tafazolli R, Quek TQS, Debbah M (2025) Generative artificial intelligence-enhanced multimodal semantic communication in internet of vehicles: system design and methodologies. IEEE Veh Technol Mag 20(2):71–82. https://doi.org/10.1109/MVT.2025.3545399
27. Qiao X, Huang Y, Dustdar S, Chen J (2020) 6g vision: an ai-driven decentralized network and service architecture. IEEE Int Comput 24(4):33–40. https://doi.org/10.1109/MIC.2020.2987738
28. Xu C, Chen Z, Tao M, Zhang W (2023) Edge-device collaborative rendering for wireless multi-user interactive virtual reality in metaverse. In: GLOBECOM 2023–2023 IEEE global communications conference, pp 3542–3547. https://doi.org/10.1109/GLOBECOM54140.2023.10437783
29. Luo S, Xu H, Lu C, Ye K, Xu G, Zhang L, Ding Y, He J, Xu C (2021) Characterizing microservice dependency and performance: Alibaba trace analysis. In: Proceedings of the ACM symposium on cloud computing. SoCC '21, pp. 412–426. Association for computing machinery, New York, NY, USA. https://doi.org/10.1145/3472883.3487003
30. Theodoropoulos T, Rosa L, Boudi A, Benmerar TZ, Makris A, Taleb T, Cordeiro L, Tserpes K, Song J (2025) Cross-cluster networking to support extended reality services. IEEE Netw 39(2):250–258. https://doi.org/10.1109/MNET.2024.3453301
31. Bai S, Chen K, Liu X, Wang J, Ge W, Song S, Dang K, Wang P, Wang S, Tang J, et al (2025) Qwen2. 5-vl technical report. arXiv preprint arXiv:2502.13923
32. Su C, Wen J, Kang J, Wang Y, Su Y, Pan H, Zhong Z, Shamim Hossain M (2025) Hybrid rag-empowered multimodal llm for secure data management in internet of medical things: a diffusion-based contract approach. IEEE Int Things J 12(10):13428–13440. https://doi.org/10.1109/JIOT.2024.3521425
33. Huang Y, Du H, Zhang X, Niyato D, Kang J, Xiong Z, Wang S, Huang T (2025) Large language models for networking: applications, enabling techniques, and challenges. Netw Mag Global Int 39(1):235–242. https://doi.org/10.1109/MNET.2024.3435752
34. Ltd. M (2021) Memgraph: high-performance in-memory transactional and analytical graph database. https://memgraph.com
35. Song C, Ye Z, Lin Q, Peng Q, Wang J (2024) A framework to implement 1+N multi-task fine-tuning pattern in LLMs using the CGC-LORA algorithm. https://arxiv.org/abs/2402.01684
36. Zou Y, Yi S, Li Y, Li R (2024) A closer look at the cls token for cross-domain few-shot learning. Adv Neural Inf Process Syst 37:85523–85545
37. Zhan L-M, Liu B, Lu Z, Xie C, Cao J, Wu X-M (2025) Deal: disentangling transformer head activations for llm steering. arXiv preprint arXiv:2506.08359
38. Kang J, Chen J, Xu M, Xiong Z, Jiao Y, Han L, Niyato D, Tong Y, Xie S (2024) Uav-assisted dynamic avatar task migration for vehicular metaverse services: a multi-agent deep reinforcement learning approach. IEEE/CAA J Autom Sin 11(2):430–445. https://doi.org/10.1109/JAS.2023.123993
39. Tabatabaei F, Khalili H, Requena M, Kahvazadeh S, Mangues-Bafalluy J (2023) Dynamic service placement in 6g multi-cloud scenarios. In: 2023 23rd international conference on transparent optical networks (ICTON), pp 1–4. https://doi.org/10.1109/ICTON59386.2023.10207547
40. Pallewatta S, Kostakos V, Buyya R (2024) Reliability-aware proactive placement of microservices-based iot applications in fog computing environments. IEEE Trans Mob Comput 23(12):11326–11341. https://doi.org/10.1109/TMC.2024.3394486
41. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
42. OpenCelliD: The world's largest open database of cell towers. https://www.opencellid.org. Accessed: 2025–06-30 (2023)
43. Sun Y, Lin C, Hu J (2016) Real-time traffic flow classification from urban surveillance video. In: Proceedings of IEEE ITS, pp 1244–1249
44. Luo S, Wang H, Li J (2022) Prediction-based resource allocation for microservice deployment in edge computing. IEEE Int Things J 9(12):9876–9889
45. Zhang L, Chen X, Wang F (2023) Edge computing network emulation: a comprehensive framework for realistic testing. IEEE Netw 37(4):112–119
46. Zhu D, Chen J, Shen X, Li X, Elhoseiny M (2023) Minigpt-4: enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592
47. Goudarzi M, Rodriguez MA, Sarvi M, Buyya R (2024) $\mu\mu$-ddrl: a qos-aware distributed deep reinforcement learning technique for service offloading in fog computing environments. IEEE Trans Serv Comput 17(1):47–59. https://doi.org/10.1109/TSC.2023.3332308
48. Microsoft: azure stack edge Pro with GPU technical specifications and compliance. Microsoft learn. Accessed: Nov. 2024

(2024). https://learn.microsoft.com/en-us/azure/databox-online/azure-stack-edge-gpu-technical-specifications-compliance
49. 3GPP: study on scenarios and requirements for next generation access technologies. Technical report TR 38.913 V14.2.0, 3rd generation partnership project (2017). https://www.3gpp.org/ftp/Specs/archive/38_series/38.913/
50. 3GPP: service requirements for enhanced V2X scenarios. Technical specification TS 22.186 V15.3.0, 3rd generation partnership project (2018). https://www.3gpp.org/ftp/Specs/archive/22_series/22.186/