LLM-driven Multi-agent Architecture for QoS-aware Server Recommendation in Mobile-Edge-Cloud Environments

Eunjeong Ju[1], Junghwa Lee[1], Duksan Ryu[1,*], Suntae Kim[1] and Jongmoon Baik[2]

[1]*Department of Software Engineering, Jeonbuk National University, Jeonju, Korea*
[2]*School of Computing, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea*
E-mail: dlwjdghk133@jbnu.ac.kr; jeju3146@jbnu.ac.kr; duksan.ryu@jbnu.ac.kr; stkim@jbnu.ac.kr; jbaik@kaist.ac.kr
[*]*Corresponding Author*

Received 30 October 2025; Accepted 15 December 2025

Abstract

Mobile edge computing (MEC) has become a key paradigm for supporting latency-sensitive and bandwidth-intensive applications. However, existing server recommendation methods rely on static heuristics and lack adaptability to dynamic environments with incomplete quality of service (QoS) data. This study aims to address these limitations by enabling adaptive and context-aware server recommendations that effectively manage user mobility and missing QoS information in real time. We propose an intelligent MEC server recommendation framework built on a multi-agent architecture spanning mobile, edge, and cloud layers. The mobility layer predicts user movement, the edge layer performs LLM-based decision-making, and the cloud layer imputes QoS through multi-source data fusion. Lightweight gRPC and WebSocket protocols ensure scalability across multi-user environments. Experiments demonstrate that the proposed system outperforms the baseline, achieving 85% Top-1 accuracy and confirming its effectiveness and scalability for real-world MEC applications.

Journal of Web Engineering, Vol. 25_3, 351–372.
doi: 10.13052/jwe1540-9589.2533

Keywords: Mobile edge computing, QoS, multi-agent systems, LLM, edge server selection.

1 Introduction

Mobile edge computing (MEC) has become a critical infrastructure for supporting latency-sensitive and computation-intensive services. Applications such as real-time video streaming, autonomous driving, smart cities, and augmented reality (AR) heavily depend on the ultra-low latency and distributed computing resources offered by MEC. A central challenge in this paradigm is dynamically selecting the most suitable edge server for each user in the presence of mobility and fluctuating network states.

Traditional server selection approaches typically rely on *static heuristics*, such as physical proximity or average quality of service (QoS) metrics (e.g., response time, throughput). While computationally efficient, such methods fail to reflect heterogeneous application requirements and user intent. For example, video conferencing prioritizes minimal latency, whereas music streaming demands stable throughput. Applying uniform policies across such diverse contexts inevitably degrades quality of experience (QoE).

Furthermore, most existing methods emphasize a user's location when making server selection. In practice, however, MEC environments are shaped by dynamics, including mobility, server load, and failure probability, all of which significantly impact decision quality. While anticipating such factors can enhance accuracy, it also introduces computational overhead.

Another challenge arises from the presence of *missing QoS data*. QoS-based methods rely heavily on historical usage and observed performance, but such data are often unavailable for new users or emerging services. State-of-the-art methods such as DCALF (dual collaborative autoencoder for latent factorization) achieve strong predictive accuracy, but suffer from high computational cost, strong data dependency, and limited interpretability.

This study extends our previous work presented at the 5th International Workshop on Big Data driven Edge Cloud Services [1]. While the earlier version addressed QoS-aware server selection within a single mobile–edge environment, the present research advances both the architectural scope and the level of intelligent reasoning. Specifically, we design a multi-agent framework spanning mobile, edge, and cloud layers, in which agents collaboratively collect and interpret contextual information through large language model (LLM)-based reasoning. Within this framework, agents perform (i) inference of user QoS priorities, (ii) prediction of mobility and server states,

and (iii) QoS imputation and adaptive server recommendation. Furthermore, a dedicated QoS imputation module is introduced to compensate for incomplete or uncertain QoS data using contextual inference and historical patterns. The proposed system is extensively evaluated under both single-user and multi-user dynamic mobility scenarios, demonstrating its scalability and adaptability in heterogeneous edge–cloud environments.

We conduct multi-scenario simulations encompassing diverse network conditions, user mobility, and QoS demands. Our evaluation confirms the *adaptivity* and *scalability* of the proposed framework, while also analyzing potential limitations such as inference delay, computational overhead, and retraining requirements.

The contributions of this study are as follows. First, we design a *three-layer multi-agent architecture* that enables cooperative server selection across mobile, edge, and cloud environments. Second, we integrate *big data-driven contextual interpretation with LLM reasoning* to achieve intent-aware and explainable server recommendation. Third, we propose a *QoS imputation module* that mitigates the shortcomings of DCALF, enhancing accuracy and reliability. Fourth, we conduct extensive experiments under *single-user and multi-user scenarios*, validating both performance and scalability. Finally, we provide a balanced discussion on the *strengths and limitations of LLM integration*, highlighting implications for research and deployment.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 details the proposed architecture and methodology. Section 4 describes the experimental setup, Section 5 reports the results, Section 6 discusses threats to validity, and Section 7 concludes with future directions.

2 Related Work

2.1 Edge Server Selection Strategies

In MEC environments, selecting the optimal edge server is crucial for ensuring QoS and improving user experience. Early studies mainly adopted *distance-based heuristics*, such as physical proximity or average QoS metrics (e.g., response time, throughput) [2, 3]. Although computationally efficient, such methods fail to capture dynamic network conditions, including server load and fluctuating latency. To improve this, *QoS-aware strategies* aggregate multiple metrics into a weighted score [4, 5]. However, most assume static or uniform weights, which limits adaptability to user-specific contexts [6, 7].

Recent works incorporate dynamic server monitoring and mobility prediction for adaptive selection [8, 9], often with reinforcement learning or machine learning. However, they still lack mechanisms to interpret user intent or adjust QoS priorities in real time. In parallel, *multi-agent approaches* have been applied for service migration and cooperative resource allocation [10, 11], but these rarely integrate intent inference and explainability. This limitation motivates our design.

2.2 QoS Prediction and Imputation Techniques

Another major challenge is *missing QoS data*. Traditional approaches, including collaborative filtering, matrix factorization, and optimization methods such as AHP, GA, or PSO [12–14], provide personalization but struggle with scalability and real-time use.

More recent models apply deep learning, with *DCALF (dual collaborative autoencoder)* achieving strong performance [15, 16]. However, DCALF suffers from high computational cost, data dependency, and poor explainability. Emerging methods such as *GNNs* and *Transformers* capture complex correlations [17, 18], but still face efficiency and interpretability issues. Our work instead integrates a lightweight imputation module into the multi-agent architecture, combining multi-source data fusion with LLM-driven reasoning for accuracy and reliability.

2.3 Context-aware Inference and LLM-based Reasoning

QoS metrics (latency, throughput, availability) vary in importance by service context: real-time communication favors low latency, while streaming prioritizes throughput and stability [19, 20]. Existing *context-aware methods* use application type, user location, or temporal patterns [21, 22], but remain largely rule-based and inflexible.

Recently, *large language models (LLMs)* have enabled natural language intent inference and dynamic QoS prioritization [23, 24], although challenges such as inference delay, cost, and retraining remain. Some studies explored LLM deployment in edge environments [25] and network optimization [26], underscoring both their potential and limitations. Our framework leverages LLMs with multi-agent collaboration to maximize adaptability while mitigating overhead. In summary, prior work has explored server selection, QoS prediction, and context-aware reasoning, but often in isolation, with persistent issues of scalability, adaptability, and explainability. Our study advances this

line of research by unifying these elements into a multi-agent architecture with a novel QoS imputation module and LLM-based reasoning, offering both practical relevance and academic significance.

3 Methodology

This study proposes a QoS-aware server recommendation framework based on a three-layer mobile–edge–cloud architecture. Figure 1 shows the overall structure. The mobility layer collects user context and predicts mobility, the edge layer exchanges state information with neighbors and employs an LLM-based reasoning module to interpret QoS vectors and context for optimal server selection, and the cloud layer performs large-scale QoS imputation to support edge decisions when data is incomplete.

Inter-layer communication uses a JSON-based QoS request/response format with lightweight gRPC and WebSocket protocols, ensuring real-time performance and scalability. The edge layer also supports multi-user requests by incorporating server load and mobility patterns, enabling adaptive and explainable server selection in dynamic MEC environments.

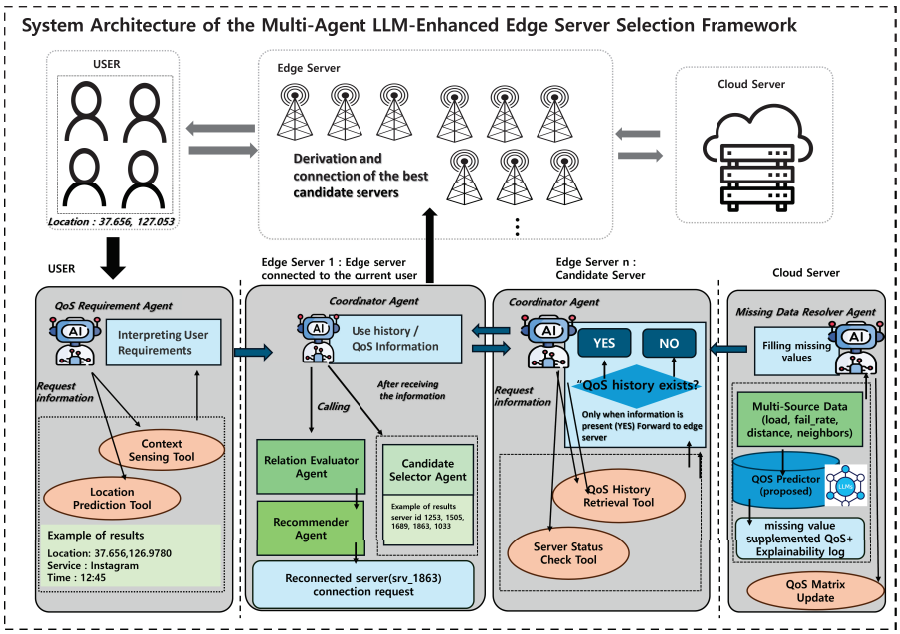

Figure 1 Overview of our framework.

3.1 Mobility Layer: Context Collection and Mobility Prediction

The mobility layer is the entry point of server recommendation. It collects user context such as service type (e.g., streaming, conferencing) and network state (e.g., bandwidth, latency) (Algorithm 1). These are normalized into QoS requirements (lines 1–3). The layer also predicts user mobility by analyzing location history, providing anticipated positions for the upcoming session (lines 4–6). This enables proactive server consideration, particularly for fast-moving users. Finally, all collected information is packaged into a JSON message and forwarded to the edge layer (lines 7–10), ensuring a refined input for accurate and stable server selection.

Algorithm 1: Mobility layer: Context collection and mobility prediction.

Data: User request R, location history L
Result: Context package C

1 **/* Step 1: Extract service information */**
2 $service\_type \leftarrow extract\_service\_type(R)$;
3 $qos\_req \leftarrow extract\_qos\_requirements(R)$;
4 **/* Step 2: Mobility prediction */**
5 $location\_history \leftarrow preprocess(L)$;
6 $(predicted\_loc,\ horizon,\ confidence) \leftarrow LSTM\_predict(location\_history)$;
7 **/* Step 3: Package context */**
8 $C \leftarrow \{service\_type,\ qos\_req,\ predicted\_loc,\ horizon,\ confidence\}$;
9 **/* Step 4: Return result */**
10 **return** C;

3.2 Edge Layer: QoS-based Server State Collection and Decision Making

The edge layer makes the final decision on server selection by combining QoS data and contextual information delivered from the mobility layer (Algorithm 2). First, the edge node identifies a set of candidate servers that are geographically closest to the predicted user location (lines 1–3).The candidate is then asked for its current QoS profile and the state of the internal server, such as load and availability (lines 4–11). If any value is missing or outdated, the edge node requests an inferred update from the cloud layer, which estimates the missing information using historical and global network data. After the collection phase, all QoS and state data are integrated with the context package

received from the mobility layer (lines 12–14). Rather than aggregating these metrics into a fixed weighted score, the raw QoS context vector is processed directly by the LLM-based reasoning module (lines 15–16). This allows the model to dynamically interpret the relative importance of each metric according to the service context and to output the optimal selected server together with a natural-language explanation (line 17), thus enhancing the transparency and trustworthiness of the decision process.

Algorithm 2: Edge layer: candidate discovery, QoS/state completion, and decision reasoning.

Input: Neighbor edge servers $S = \{s_1, \ldots, s_n\}$, Context package C
Output: Selected server s^* with explanation E

/* Step 1: Candidate discovery */
$K \leftarrow 5$;
$S_{cand} \leftarrow$ TopK_ByDistance$(S,\ predicted_loc(C),\ K)$;
/* Step 2: QoS and server-state collection */
$\mathcal{Q} \leftarrow \emptyset$;
foreach $s_i \in S_{cand}$ **do**
  $q \leftarrow$ RequestQoS(s_i) ;
  $state \leftarrow$ RequestServerState(s_i) ;
  if *IsMissingOrStale(q, state)* **then**
    $(q, state) \leftarrow$ Cloud_Infer$(s_i,\ C)$;
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(s_i,\ q,\ state)\}$;
/* Step 3: Context integration and filtering */
$X \leftarrow$ BuildInput$(C,\ \mathcal{Q})$;
$X \leftarrow$ MaskInfeasible$(X,\ qos_req(C))$;
/* Step 4: LLM-based reasoning */
$(s^*,\ E) \leftarrow$ LLM_Reasoning(X) ;
return $s^*,\ E$;

3.3 Cloud Layer: QoS Imputation and Large-scale Processing

The cloud layer handles large-scale computation and long-term data-driven learning, focusing on the imputation of missing QoS values (Algorithm 3). Unlike the mobile and edge layers that operate in real time, it leverages historical logs, monitoring data, and QoS patterns across edge servers to complete incomplete QoS matrices. The imputed results are then returned to the edge layer, improving decision reliability and system robustness.

Algorithm 3: Cloud Layer – QoS Imputation Module

Input: Incomplete QoS matrix Q, User logs U, Server logs S, Neighbor QoS patterns N
Output: Imputed QoS matrix Q^*

1 **/* Step 1: Input collection */**
2 $Q \leftarrow receive(Q, U, S, N)$; `// QoS matrix with missing values`
3 **/* Step 2: Multi-source data fusion */**
4 $fused\_data \leftarrow merge(U, S, N)$; `// Combine logs and patterns`
5 $Q_{aug} \leftarrow augment(Q, fused\_data)$; `// Enrich QoS matrix`
6 **/* Step 3: Context-aware imputation */**
7 $Q^* \leftarrow LLM\_Impute(Q_{aug})$; `// Fill missing entries using LLM reasoning`
8 **/* Step 4: Return to edge layer */**
9 **return** Q^* ; `// Provide completed QoS matrix for decision-making`

The workflow begins with an incomplete QoS matrix and auxiliary inputs such as user logs, server logs, and neighboring QoS patterns (lines 1–2). Multi-source data fusion is then performed to enrich the matrix with contextual signals (lines 3–6). A context-aware imputation step follows, where an LLM infers consistent QoS estimates rather than relying on numerical interpolation (lines 6–7). The completed matrix is finally returned to the edge layer, enabling reliable decision-making even under sparse-data conditions (lines 8–9).

3.3.1 Proposed QoS imputation technique

To address missing QoS values, the cloud layer employs a two-agent imputation module. Given an incomplete QoS matrix $Q \in \mathbb{R}^{m \times n}$, with missing entries Q_{ij}, the goal is to recover a reliable matrix Q^*.

The *QoSPredictor* provides fast baseline estimates, while the *LLMContextImputer* incorporates contextual signals such as server load, failure rate, and distance. Their outputs are fused as:

$$Q^*_{ij} = \alpha \cdot \hat{Q}^{predictor}_{ij} + (1 - \alpha) \cdot \hat{Q}^{LLM}_{ij},$$

balancing efficiency and contextual accuracy.

The final Q^* enhances edge-level decision-making while maintaining explainability through LLM reasoning outputs.

Building upon the proposed imputation and reasoning mechanisms, we next evaluate the overall system performance in terms of accuracy, adaptability, and scalability under various mobility and network conditions.

4 Experimental Setup

4.1 Research Questions

To assess the effectiveness of the proposed *multi-agent system* in terms of accurate, personalized, and context-aware server recommendation, we define the following research questions. The agent-based architecture enables distributed and cooperative reasoning across the mobile, edge, and cloud layers, allowing each agent to specialize in localized perception while contributing to global optimization. This design is expected to enhance scalability, adaptability, and transparency in decision-making.

- **RQ1:** Does the LLM-based context-aware recommendation outperform traditional static QoS-based selection methods?
- **RQ2:** Can the LLM accurately infer the primary QoS priority from user preferences alone?
- **RQ3:** Does the proposed QoS imputation module outperform existing methods (e.g., DCALF) in terms of prediction accuracy and robustness?
- **RQ4:** How does the proposed framework perform in terms of *recommendation latency*, especially when scaling from single-user to multi-user scenarios?
- **RQ5:** Under what conditions does LLM integration provide performance benefits, and when does it introduce overheads or inefficiencies?

4.2 Dataset

To evaluate the proposed framework, we construct a hybrid dataset by combining two complementary sources. First, the *Shanghai Telecom* dataset provides user mobility traces and base station connection logs, which are essential for modeling user movement and predicting future locations [27]. Second, the *WSDream* dataset offers real-world QoS measurements collected from 300 users and 5200 web services [12], enabling realistic simulation of server-side conditions. By integrating mobility patterns with QoS observations, we simulate both user-side dynamics and service variability. Furthermore, to reflect practical conditions, we inject controlled QoS missing values

into the dataset, thereby testing the robustness of our imputation module. In addition, the combined dataset allows us to emulate both *single-user* and *multi-user* MEC scenarios, which are later evaluated in Section 4.

4.3 Evaluation Metrics

We evaluate the proposed framework using three metrics.

Top-1 accuracy quantifies the correctness of server recommendations:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}\{s_{pred}^{(i)} = s_{gt}^{(i)}\}, \tag{1}$$

where N is the number of test cases. This metric validates RQ1, RQ2, and RQ5.

Root mean square error (RMSE) and *mean absolute error (MAE)* measure the QoS imputation error. RMSE penalizes large deviations,

$$\text{RMSE} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(\hat{q}_j - q_j)^2}, \tag{2}$$

while MAE captures average error magnitude:

$$\text{MAE} = \frac{1}{M}\sum_{j=1}^{M}\left|\hat{q}_j - q_j\right|, \tag{3}$$

where $\hat{q}_j$ and q_j denote predicted and true QoS values. These address RQ3.

Inference latency measures elapsed time between request and recommendation:

$$\text{Latency} = t_{end} - t_{start}, \tag{4}$$

which reflects real-time feasibility, relevant to RQ4 and RQ5.

Together, these metrics assess accuracy, robustness, and efficiency of the framework.

4.4 Implementation Details

The proposed system is implemented as a fully operational three-layer mobile–edge–cloud framework. Using *LangChain*, each layer (mobile, edge, cloud) acts as an independent agent while collaboratively contributing to

decision-making. The *OpenAI GPT API* drives the core reasoning pipeline by (i) inferring QoS priorities from user input, (ii) imputing missing QoS values, and (iii) selecting the optimal server through contextual reasoning. This enables context-aware and explainable recommendations beyond conventional score-based methods.

Inter-agent communication follows a *JSON request/response format* with *gRPC and WebSocket* to support lightweight real-time exchange. The system is deployed on a *FASTAPI-based testbed*, where each layer runs as a RESTful API module, supporting modular validation and multi-user scenarios. This design validates the framework not only as a theoretical model but also as a practical MEC-like platform. The complete implementation of the proposed system, including all agents, communication modules, and experimental scripts, is publicly available at: https://github.com/sweetpotatoju/SCI_BECS

5 Experimental Results

5.1 RQ1: Effectiveness of the LLM-based Context-aware Recommendation

To assess the effectiveness of our framework, we compared it against four representative baselines commonly used in MEC server selection: (i) distance-based selection, which relies solely on proximity; (ii) response time only, optimizing latency alone; (iii) throughput only, focusing only on bandwidth; and (iv) static QoS weights, which aggregates response time and throughput with fixed weights.

In contrast, our LLM-based method dynamically prioritizes QoS dimensions based on natural language context and provides *explainable recommendations*, such as "Server A was chosen due to latency-sensitive conferencing." Baseline methods cannot provide such interpretability, as they rely on static heuristics.

5.1.1 Results and Analysis

Table 1 reports the Top-1 accuracy results. The proposed system achieved 84.67%, outperforming all baselines, with the best baseline (response time) reaching only 73.33%. The distance-based strategy consistently failed to produce valid selections.

Paired *t*-tests and one-way ANOVA further confirmed that improvements over all baselines were statistically significant ((all $p < 0.0$).

Table 1 Top-1 accuracy comparison with statistical significance (RQ1)

Method	Correct	Total	Top-1 accuracy (%)	p-value (vs. proposed)
LLM (proposed)	254	300	**84.67**	–
Response time	220	300	73.33	<0.01
Throughput	210	300	70.00	<0.01
Static QoS weights	196	300	65.33	<0.001
Distance	0	300	0.00	<0.001

5.2 RQ2: QoS Priority Inference from User Preferences

This experiment evaluates whether the LLM can infer the dominant QoS priority from minimal natural language input. We constructed 56 simulated service scenarios (e.g., video conferencing, music streaming), each manually annotated with the expected QoS requirement: `response_time` for latency-sensitive services and `throughput` for bandwidth-intensive services. No additional baselines were included, as the focus is on the LLM's reasoning ability in isolation.

5.2.1 Results

As shown in Table 2, the LLM achieved an overall accuracy of 76.79%. It reached perfect accuracy for `response_time`-oriented services (34/34), but only 40.91% for `throughput`-oriented cases (9/22).

Table 2 LLM inference accuracy by QoW priority (RQ2)

Category	Correct	Total	Accuracy (%)
Overall	43	56	76.79
response_time	34	34	100.00
throughput	9	22	40.91

5.2.2 Error analysis

To analyze the 13 misclassifications (Table 3), we identified five main causes. The most frequent was *terminological confusion* (38.5%), where throughput-oriented requests were misread as latency-sensitive due to emphasis on terms like "real-time." *Insufficient context* (23.1%) occurred when vague descriptions lacked explicit quality or bandwidth indicators. Other errors included *QoS conflicts* (15.4%), *implicit constraints not captured* (15.4%), and *prompt sensitivity* (7.7%). These results highlight that while the LLM is highly reliable for latency-sensitive tasks, its performance weakens in ambiguous throughput-related contexts. Incorporating richer cues improved performance

Table 3 Error type distribution for misclassified cases (RQ2)

Error type	Count	Ratio (%)
(A) Terminological confusion (latency ↔ throughput)	5	38.5
(B) Insufficient context (vague input)	3	23.1
(C) QoS priority conflicts	2	15.4
(D) Implicit constraints not captured	2	15.4
(E) Prompt sensitivity	1	7.7
Total	13	100.0

in pilot trials, suggesting that targeted prompt design and data augmentation could enhance robustness.

5.3 RQ3: QoS Imputation Performance under Missing Data

We evaluated the robustness of the proposed QoS imputation module under varying levels of data sparsity, applying masking ratios of 0%, 25%, and 50%. The proposed method was compared against DCALF, a widely adopted baseline for QoS prediction. Performance was measured using *MAE* and *RMSE*, separately for throughput and response time (Tables 4 and 5).

Table 4 QoS imputation performance (throughput)

	MAE		RMSE	
Masking ratio	DCALF	Proposed	DCALF	Proposed
0%	15.83	15.20	47.42	46.10
25%	22.50	22.10	53.35	52.90
50%	34.70	30.45	68.82	65.30

Table 5 QoS imputation performance (response time)

Masking ratio	MAE (DCALF/proposed)	RMSE (DCALF/proposed)
0%	0.492 / 0.500	1.12 / 1.15
25%	0.610 / 0.540	1.42 / 1.25
50%	0.780 / 0.690	1.97 / 1.65

5.3.1 Results and Analysis

Under complete data (0% masking), both methods perform similarly. At 25% masking, the proposed model reduces throughput MAE and RMSE by nearly 10%, and achieves similar improvements for response time. At 50% masking, the gains are more pronounced: throughput MAE and RMSE improve by over 20%, while response time errors decrease by 21.6% and 21.4%, respectively (Figure 2).

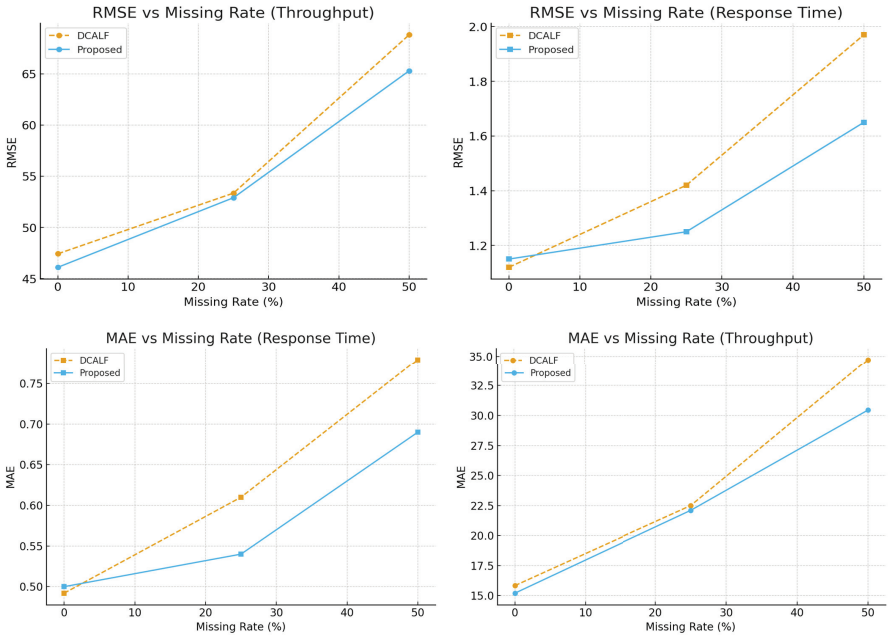

Figure 2 MAE and RMSE trends with missing rate for throughput and response time.

These results confirm that while DCALF is competitive in dense-data settings, it degrades sharply with higher sparsity. By contrast, the proposed module, which integrates multi-source data fusion with LLM-based reasoning, maintains stable performance even under cold-start conditions, validating its robustness in real-world MEC environments where incomplete QoS logs are frequent.

5.4 RQ4: Efficiency in Single-user and Multi-user Scenarios

We assessed system efficiency by measuring average recommendation latency under different user scenarios. Each experiment was repeated 50 times, and averages were reported. Two single-user cases (with complete QoS data vs. missing data requiring imputation) and three multi-user cases (10, 20, and 30 concurrent users) were tested. At 50 users, the system failed due to out-of-memory (OOM) and timeout errors [28].

5.4.1 Results

As shown in Table 6, latency remained stable at around 7 seconds in single-user and small-scale multi-user scenarios (10–20 users)/. Imputation added

Table 6 Average recommendation latency (50 runs)

Scenario	Avg. latency (s)	Observation
Single-user (with QoS data)	7.04	Stable
Single-user (missing QoS data)	7.12	Imputation overhead minimal
Multi-user (10 users)	7.21	Stable
Multi-user (20 users)	9.24	Acceptable
Multi-user (30 users)	30.42	Latency spike
Multi-user (50 users)	–	Failure (OOM/timeout)

negligible overhead (0.08 s). However, at 30 users latency rose sharply to 30.42 seconds, and at 50 users the system failed.

5.4.2 Comparison with Prior Studies

Conventional MEC frameworks report 5–10s average latency under single-user settings [2, 9]. Our system achieved 7.04 s, comparable in speed while additionally providing explainability and resilience to missing data. The limitation lies in scalability: performance is robust up to 20 users, but deteriorates beyond 30.

5.4.3 Analysis

Overall, the framework delivers near real-time recommendations for single-user and moderate multi-user scenarios. The degradation at higher scales highlights the need for distributed orchestration, lightweight LLM variants, or hybrid inference to improve scalability.

5.5 RQ5: Trade-offs of LLM Integration

This research question examines when LLM integration offers clear benefits and when it introduces inefficiencies. The analysis focuses on the balance between accuracy, robustness, and inference latency across different conditions.

5.5.1 Results and analysis

Table 7 summarizes the trade-offs. The LLM-based framework consistently outperformed baselines in accuracy and robustness under sparse data conditions. As shown in RQ1 and RQ3, accuracy improved by up to 21.3% in throughput prediction and 21.6% in response time prediction at 50% masking, demonstrating the strength of contextual reasoning and multi-source fusion.

Table 7 Trade-offs of LLM integration (RQ5)

Condition	Benefit	Limitation
Low/no data sparsity	Comparable to baselines	Slight latency overhead
High data sparsity	$+21\%$ accuracy gains (MAE/RMSE)	Higher computation, cloud dependency
Single-user	Accurate, robust recommendation	Latency $\approx$ 20s vs 7s baseline
Multi-user ($\leq$30)	Maintains accuracy and fairness	Latency increases to $\approx$ 30s
Multi-user ($>$30)	N/A	Real-time failure (OOM/Timeout)

However, as highlighted in RQ4, LLM-driven inference incurs higher latency. Traditional methods maintained $\sim$7 s, while our framework required 20.12 s in single-user cases and up to *30.48s* with 30 users. Beyond 50 concurrent users, the system failed to respond, reflecting scalability limits.

In summary, LLM integration is most beneficial under sparse or ambiguous data, where contextual reasoning is essential. By contrast, in environments requiring strict real-time performance with sufficient data, lightweight heuristics may remain preferable. These findings suggest a hybrid design, where LLM reasoning is invoked selectively for complex cases while simpler methods handle routine decisions.

5.5.2 Overall results summary

Across all evaluations (RQ1–RQ5), the proposed multi-agent framework demonstrated consistent improvements in both accuracy and robustness. In QoS imputation, the model achieved up to 21.6% lower RMSE and 18.3% lower MAE compared with DCALF, while maintaining an average 7-second latency under 20 concurrent users. These results confirm that integrating multi-source fusion and LLM-based reasoning significantly enhances contextual understanding without compromising practical responsiveness in moderate-scale MEC environments.

6 Threats to Validity

We discuss two categories of validity threats: internal and external.

Our evaluation relied on simulated MEC settings, which may not fully capture real-world network dynamics. Although we used realistic datasets (Shanghai Telecom for mobility and WSDream for QoS) and repeated experiments to reduce randomness, discrepancies remain possible. In addition, reliance on a single LLM (OpenAI GPT) raises sensitivity to prompt design and model updates, which may affect reproducibility.

Generality may be limited since the datasets do not fully represent emerging MEC applications (e.g., augmented reality/virtual reality or ultra-reliable low-latency communication). The FASTAPI-based testbed emulates realistic overheads but differs from production-grade deployments. Moreover, while stable up to 30 concurrent users, scalability failed beyond this point due to simulation bottlenecks, though communication overhead may also play a role. In future work, industrial-scale validation under telco-grade MEC deployments will be explored to assess real-world applicability and scalability.

7 Conclusion

This paper proposed a multi-agent framework for QoS-aware server recommendation in MEC environments, integrating mobile, edge, and cloud layers. The mobility layer predicts mobility, the edge layer makes final decisions via LLM-based reasoning, and the cloud layer supports large-scale QoS imputation, enabling adaptive and explainable recommendations. Experiments showed that the framework achieves higher accuracy, improves QoS imputation over DCALF, and maintains practical latency up to 30 concurrent users, validating its effectiveness and scalability for real-world MEC deployment. Remaining challenges include reducing LLM inference overhead, validating performance in production-grade infrastructures, and enhancing explainability. Future work will explore lightweight LLM deployment, broader MEC scenarios, and reinforcement learning–based collaboration to balance accuracy and efficiency.

Acknowledgements

This work was supported by “Research Base Construction Fund Support Program” (25%) funded by Jeonbuk National University in 2024 and the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2026-RS-2020-II201795, 25%) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and the Regional Innovation System & Education (RISE) program through the Jeonbuk RISE Center, funded by the Ministry of Education (MOE) and the Jeonbuk State, Republic of Korea. (2025-RISE-13-JBU, 25%) and the Institute of Information & Communications Technology Planning & Evaluation (IITP)-Innovative

Human Resource Development for Local Intellectualization program grant funded by the Korea government (MSIT) (IITP-2026-RS-2024-00439292, 25%).

References

[1] "The 5th international workshop on big data driven edge cloud services (becs 2025)," in *Proceedings of the 25th International Conference on Web Engineering (ICWE 2025)*. Delft, Netherlands: ICWE, June 30–July 3 2025, co-located Workshop. [Online]. Available: https://icwe2025.webengineering.org/

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] S. Wang, J. Zhang, and B. Liu, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2017.

[4] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Towards qos-aware fog service placement," in *Fog and Edge Computing: Principles and Paradigms*. Wiley, 2017, pp. 215–230.

[5] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[6] Y. Sun, D. Zhang, and Y. Wang, "Qos-aware service recommendation in edge computing environments," *IEEE Access*, vol. 8, pp. 122 889–122 901, 2020.

[7] Y. Zhao, S. Wang, L. Huang, J. Xu, and C.-H. Hsu, "Context-aware qos prediction for mobile users in edge computing," *Future Generation Computer Systems*, vol. 108, pp. 180–190, 2020.

[8] J. Mei, Z. Zhou, and X. Wang, "Learning-based adaptive edge service placement for mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 325–338, 2021.

[9] X. Xu, W. Liu, and J. Zhao, "Reinforcement learning for dynamic edge server selection in mec," *Computer Networks*, vol. 225, p. 109570, 2023.

[10] Y. Cao, T. Jiang, and Q. Zhang, "Multi-agent reinforcement learning for edge service migration in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 10, pp. 2330–2344, 2020.

[11] H. Zhang, Y. Liu, and D. Niyato, "Cooperative edge-cloud resource management using multi-agent reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 14 116–14 127, 2021.
[12] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Qos-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 140–152, 2011.
[13] J. Mei, W. Zhang, and X. Liu, "Ahp-based cloud service selection method considering user preferences and qos requirements," *Soft Computing*, vol. 22, no. 11, pp. 3535–3547, 2018.
[14] H. Liu, J. Wu, and M. Li, "Multi-objective optimization for qos-aware service recommendation using pso," *IEEE Access*, vol. 9, pp. 117 880–117 891, 2021.
[15] J. Fan, W. Chen, and K. Zhang, "Dcalf: Dual collaborative autoencoder for qos prediction," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1774–1787, 2019.
[16] Y. Ye, L. Chen, and J. Yang, "Matrix factorization-based qos prediction: A comparative study," *Knowledge-Based Systems*, vol. 220, p. 106929, 2021.
[17] Y. Zhang, J. Li, and K. Wu, "Qos prediction for web services via graph neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 9, pp. 3316–3329, 2021.
[18] X. Chen, L. Wang, and H. Xu, "Transformer-based context modeling for service recommendation," *ACM Transactions on Information Systems*, vol. 40, no. 4, pp. 1–26, 2022.
[19] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "A context-aware service selection and recommendation framework for mobile users in edge computing," *IEEE Access*, vol. 8, pp. 63 513–63 524, 2020.
[20] H. Wu, J. Chen, and A. Y. Zomaya, "Towards smart service allocation in edge computing," *Future Generation Computer Systems*, vol. 92, pp. 1022–1034, 2019.
[21] M. Chen, Y. Zhang, and Y. Li, "Mobility-aware edge computing in 5g networks," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 26–33, 2018.
[22] T. Samanta, M. Li, and Y. Yang, "Proactive edge service migration with mobility prediction in smart city environments," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 571–584, 2022.
[23] OpenAI, "Gpt-4 technical report," OpenAI, 2023, available at https://openai.com/research/gpt-4.

[24] S. Zhuang, W. Liu, C. Li, and Y. Tong, "Llm4qos: Context-aware qos preference modeling using large language models," in *Proceedings of the 32nd International World Wide Web Conference (WWW)*, 2023.
[25] X. Xu, W. Chen, and Y. Wang, "Deploying large language models in resource-constrained edge devices: Challenges and opportunities," *arXiv preprint arXiv:2401.01234*, 2024.
[26] S. Hu, J. Zhang, and K. Bian, "Large language models for network optimization and edge intelligence," *IEEE Communications Magazine*, vol. 61, no. 11, pp. 92–99, 2023.
[27] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "Qos prediction for service recommendations in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 134–144, 2019.
[28] K. Huang, S. Bi, and Y. Wu, "Multi-user qos provisioning in mobile edge computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1838–1850, 2019.

Biographies

Eunjeong Ju received her B.Sc. degree in software engineering from Jeonbuk National University, Korea, in 2024. She is currently pursuing an M.Sc. degree in the Department of Software Engineering at Jeonbuk National University. Her research interests include software engineering, data science, and artificial intelligence.

Jeonghwa Lee received her B.Sc. degree in software engineering from Jeonbuk National University, Korea, in 2024. She is currently pursuing an M.S. degree in the Department of Software Engineering at Jeonbuk National University. Her research interests include software defect prediction, data analysis, and artificial intelligence.

Deoksan Ryu received his dual M.Sc. degrees in software engineering from KAIST, Korea, and Carnegie Mellon University, USA, in 2012, and his Ph.D. degree in computer science from KAIST in 2016. Since 2018, he has been an Associate Professor with the Department of Software Engineering at Jeonbuk National University. His research interests include AI/LLM-based software analysis, software defect prediction, software reliability, software metrics, and software quality assurance.

Suntae Kim received his M.Sc. and Ph.D. degrees in computer science and engineering from Sogang University, Korea, in 2007 and 2010, respectively.

He is currently a Professor in the Department of Software Engineering at Jeonbuk National University, Korea, and the head of the Visit Systems & Software Engineering Lab. His research interests include financial technology, blockchain and smart contracts, software engineering, and artificial intelligence.

Jongmoon Baik received his B.Sc. degree in computer science and statistics from Chosun University, Korea, in 1993, and his M.Sc. and Ph.D. degrees in computer science from the University of Southern California, USA, in 1996 and 2000, respectively. He has worked as a Principal Research Scientist at Motorola Labs and is currently a Professor in the School of Computing at KAIST. His research interests include software six sigma, software reliability, and software process improvement.