Cache-Enhanced RAG and Graph-RAG for O-RAN

Shehr Bano, *Student Member, IEEE,*, Zeinab Nezami, *Member, IEEE,*, Maryam Hafeez, *Member, IEEE,*, Syed A. R. Zaidi, *Member, IEEE,* and Qasim Ahmed, *Member, IEEE.*

Abstract—**Generative artificial intelligence (AI), particularly Large Language Models (LLMs), are envisioned to be a core enabler for Sixth Generation (6G) wireless networks, powering use cases such as autonomous troubleshooting, intelligent configuration, and real-time decision support. Open Radio Access Network (O-RAN), with its modular and disaggregated architecture, offers the most suitable platform for embedding these AI capabilities. However, the integration of LLMs into O-RAN is constrained by their high inference latency, lack of domain grounding, and resource-intensive retrieval workflows. To address these limitations, this paper proposes a cache-augmented Retrieval-Augmented Generation (RAG) and Graph-RAG framework specifically designed for O-RAN environments. The framework leverages semantic caching at the User Plane Function (UPF) and centralized model orchestration to support low-latency, high-throughput, and contextually accurate responses. The system is evaluated using the ORAN-Bench 13K benchmark, consisting of 13,952 domain-specific queries derived from official specification documents. Results show that cache-enhanced RAG reduces average latency by 35.8%, increases cache utilization to 76.38%, and achieves a cache hit rate of up to 42.5%, significantly outperforming baseline LLM and Graph-RAG configurations in responsiveness. Graph-RAG achieves the highest factual correctness at 71.2%, followed by RAG at 70.5% and LLM at 64.8%. These results demonstrate that semantic caching not only mitigates the performance bottlenecks of LLM-based reasoning in O-RAN, but also paves the way for scalable, AI-native telecommunication infrastructures aligned with the demands of next-generation networks.**

Index Terms—**Open Radio Access Network (O-RAN), Large Language Model (LLM), Retrieval-Augmented Generation (RAG), Graph-RAG, Resource utilization, Semantic Cache.**

I. Introduction

A. Motivation

O-RAN architecture provides a streamlined approach towards network disaggregation, offering modularity, interoperability, and vendor neutrality [1] for radio access networks (RANs). The ORAN architecture natively supports both running: i) Artificial Intelligence (AI)-for-RAN within components like the Near-Real time Radio Intelligence Controller (Near-RT RIC), Central Unit (CU), Distributed Unit (DU), or Service Manager and Orchestrator (SMO); and ii) AI-on-RAN within the distributed implementation of user-plane functions (UPFs) [2]. These components serve as key enablers for edge AI, allowing real-time decision making, adaptive policy enforcement, and localized intelligence closer to the data source (be it RAN performance data or user-application data), thereby reducing latency and improving responsiveness. Within this emerging AI-native architectural framework, generative AI, particularly Large Language Models (LLMs), presents a transformative capability for intent-based management of these networks. These models can facilitate sophisticated, context-aware reasoning, enabling functions such as real-time troubleshooting, knowledge-based query answering (for RAN co-pilots, for instance), and the generation of adaptive control logic [3]–[5]. In the specific context of O-RAN, LLMs can be leveraged to automate the interpretation of complex network specifications, enhance anomaly detection, and generate dynamic configuration recommendations. Furthermore, they can act as intelligent agents, responding to operational queries from both human and software entities.

However, general-purpose LLMs are limited by their lack of factual grounding and domain-specific knowledge, rendering them inadequate for the stringent accuracy and low-latency requirements of decision-making within an O-RAN. While techniques such as Retrieval-Augmented Generation (RAG) and Graph-RAG enhance LLM outputs by integrating structured data retrieval and semantic graph traversal [6], [7], they introduce substantial computational and latency overheads [8]. Fundamentally, publicly available implementations or systematic evaluation of RAG and Graph-RAG applied specifically to the O-RAN domain are still in their infancy, limiting their practical adoption [9]. Moreover, the high computational demands and strict latency requirements of these AI workloads pose major challenges, especially at the network edge, within Near-RT RIC and distributed UPFs [10]. This balance between performance and accuracy is further strained by the need to handle numerous simultaneous requests across geographically distributed edge locations and fluctuating network conditions in O-RAN's decentralized framework.

B. Research Gap and Contributions

A promising solution to this problem lies in the use of cache memory for faster access to frequently retrieved data [11], [12] and reduced computational load. However, a systematic framework outlining such cache-assisted retrieval in O-RAN is not well-researched in the current literature. Consequently, the central challenge is to engineer a cache-enhanced RAG and Graph-RAG framework for O-RAN capable of delivering accurate, low-latency, and context-specific responses for critical O-RAN use cases while operating efficiently within the decentralized and resource-constrained environment characteristic of next-generation telecommunication networks. In this context, the main contributions of this research can be summarized as follows.

1) We propose an O-RAN framework that integrates LLMs with RAG and Graph-RAG, enhanced through a semantic caching mechanism deployed at the edge. This framework supports low-latency, context-aware query response by reducing redundant retrieval operations and

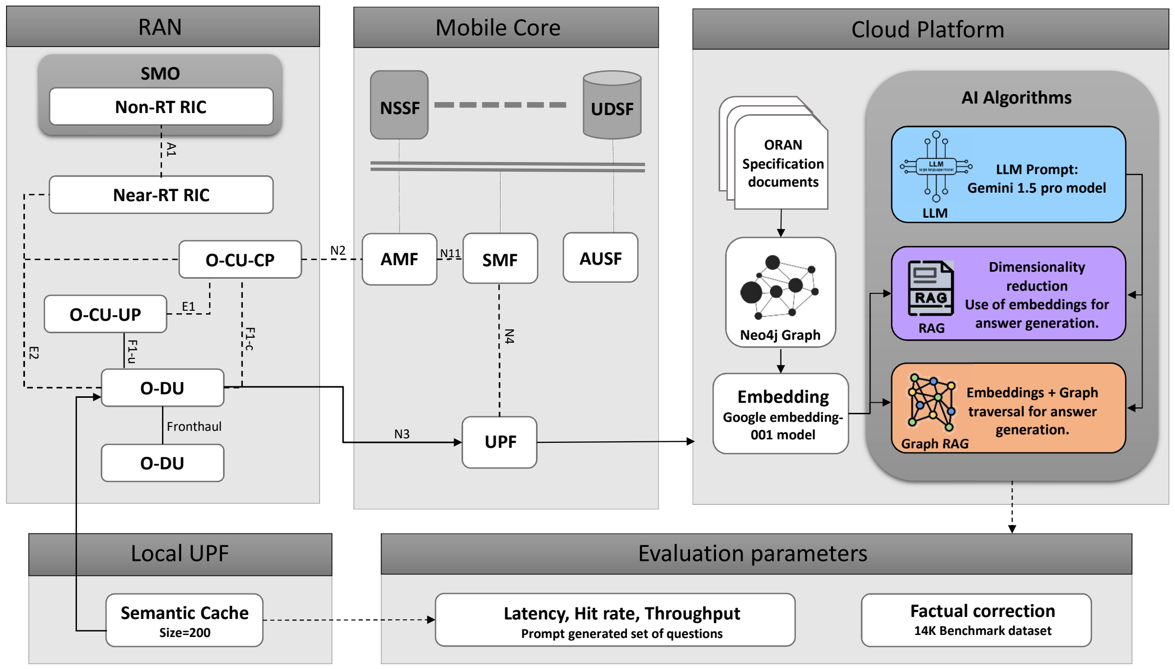


Fig. 1. System model illustrating the integration of the proposed framework in the O-RAN architecture. Notice that Generative AI can be deployed in UPF for user applications, but equally can also be deployed in the Near-RT RIC with caching when it is employed for optimization of RAN.

computational overhead, thereby aligning with the performance and scalability requirements of edge-deployed AI in O-RAN systems.

2) We conduct a comprehensive evaluation of the proposed framework across key performance metrics such as factual correctness, latency, throughput, and cache hit rate, by employing a purpose-built ORAN-Bench benchmark dataset. Our results demonstrate that cache-augmented RAG significantly outperforms standard LLM inference and Graph-RAG in latency and cache utilization, while Graph-RAG exhibits greater accuracy under high query volume conditions, validating the framework's effectiveness in different operational scenarios of AI-native telecommunication networks.

C. *Organization*

This paper is organized as follows. Section II discusses the related work, followed by the discussion of the system architecture and workflow of the proposed framework in Section III. The evaluation parameters and their settings are elaborated on in Sections IV and V, respectively. The results are discussed in Section VI, and the paper is concluded in Section VI.

II. Related Work

RAG and Graph-RAG augment LLMs with external knowledge during inference, thereby improving factual accuracy and contextual relevance [13]. Graph-RAG extends this paradigm by leveraging semantic graphs to enhance reasoning over structured relationships, with applications in technical and domain-specific NLP [14]. Yuan et al. [10] proposed a framework that combines knowledge graphs with RAG to improve context-specific reasoning for telecommunication data. Ahmad et al. [9] provided the first systematic benchmark of vector, graph, and hybrid RAG pipelines for O-RAN, demonstrating improvements in factual grounding and retrieval quality. While these methods demonstrate enhanced accuracy and controllability, they often incur high latency due to additional graph traversal or document lookup steps—posing a challenge for real-time O-RAN integration. Furthermore, their evaluation in telecom-specific contexts remains limited. Moreover, none of the approaches incorporates caching strategies that are critical for latency-sensitive inference in edge-based network environments.

Recent advances in caching mechanisms have shown significant improvements in LLM response time and resource efficiency. Jin et al. [11] introduced RAGCache, a system that caches retrieved knowledge snippets to reduce redundant retrieval and improve throughput. Li et al. [15] developed SCALM, a semantic caching framework for chatbot applications, which enhances hit rate using embedding similarity. Ye et al. Despite these innovations, caching has yet to be explored in the context of O-RAN-specific RAG pipelines. In [16], Mohandoss introduces an LLM that leverages user identity and contextual information to enhance cache efficiency, achieving an 80% reduction in average response time and improving quality for both context-sensitive and context-free queries. Semantic-based caching technique proposed by Li et al. [15] improves cache hit rates by 63% and reduces token usage by 7%. SGLang, developed by Zheng et al. [17], further boosts throughput (up to 6.4x) by employing

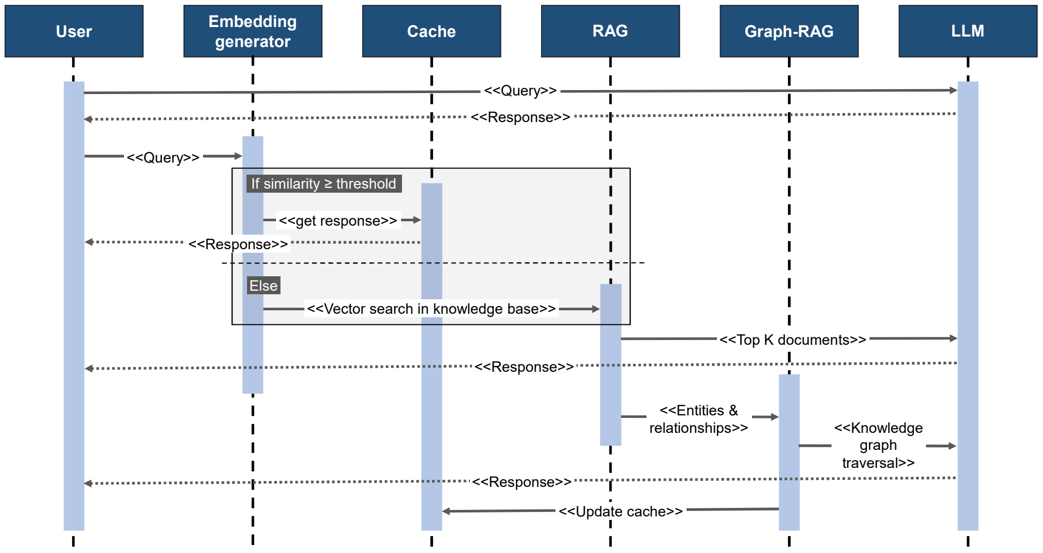


Fig. 2. Workflow of the proposed framework

a Key-Value (KV) cache. Similarly, Ye et al. [18] proposed a prefix-aware caching system utilizing a tree structure to lower latency, outperforming current state-of-the-art methods. Retrieval-augmented Language Models [19] integrate local cache systems to reduce the frequency of retrieval steps, resulting in faster and more accurate outcomes. On the other hand, Chan et al. [20] proposed a Cache-Augmented Generation (CAG) for faster response time using a simplistic architecture, and deduced that the performance of CAG was better in comparison to the dense RAG.

Currently, there exists no optimized framework that simultaneously meets the low-latency, high-throughput, and factual correctness requirements of AI-assisted reasoning in O-RAN deployments. The lack of intelligent caching mechanisms in existing RAG and Graph-RAG implementations leads to redundant retrieval and processing, degrading system responsiveness under high query volumes. Therefore, the central problem is to design and evaluate a cache-augmented RAG and Graph-RAG framework capable of supporting efficient, context-aware, and factually correct response generation for O-RAN domain-specific tasks, while operating within the stringent latency and resource constraints of edge-based network infrastructure. This requires solving trade-offs between inference latency, cache hit rate, and response accuracy in a dynamically changing network environment.

III. Cache-Enhanced Retrieval in O-RAN

The proposed system architecture, illustrated in Fig. 1, integrates a semantic caching framework with RAG and Graph-RAG pipelines to enable low-latency, context-aware reasoning over O-RAN specification documents. The architecture is deeply embedded into the O-RAN system stack, interacting with components such as the Near-RT RIC, Open-Central Unit-User Plane (O-CU-UP), and SMO through standardized interfaces (A1, E2, N2/N3, etc.). This integration allows the AI reasoning pipeline to dynamically adapt to current network conditions, user demand, and policy requirements.

A. System Architecture

The system consists of two main components: the centralized cloud platform and the edge-based UPF. The centralized cloud platforms house AI models that can be used to generate the response for document-based question answering, including the LLM, the embedding model for dense vector representations, and a knowledge graph for Graph-RAG processing. These models are preloaded with domain-specific knowledge derived from O-RAN specification documents. Dimensionality reduction techniques are applied to embeddings to enable efficient semantic comparisons and graph traversal.

The integration with 5G core network functions like Access and mobility Management Function (AMF), Session Management Function (SMF), and UPF facilitates smart traffic routing and efficient resource allocation by considering cache hit rates and AI workload features. Meanwhile, the connection via the RAN's Near-RT RIC supports real-time adjustments to caching strategies in response to current network conditions and user demand trends, which primarily include latency and resource utility optimization.

The cache memory is deployed at the local UPF, enabling edge-based caching to further reduce the latency to fetch data from the cloud platform. The cache can either store semantically aware embeddings from the AI models, or key-value pairs, based on the type of cache being deployed. In this research, a semantic cache is deployed on the edge server. By embedding caching logic at the UPF and executing high-compute tasks in the cloud, this system achieves a balance between low-latency edge inference and high-accuracy, graph-augmented reasoning, addressing a critical gap in deploying generative AI within O-RAN environments.

B. Workflow

Fig. 2 depicts how the response for any query is generated in the O-RAN pipeline. When a user query is received—originating from a human operator or network function—the system follows a multi-stage process. First, embeddings for the query are generated. If the cosine similarity score between the new query and existing cached entries at the local UPF exceeds a threshold (e.g., 0.85), a relevant cached response is retrieved from the semantic cache (size: 200 entries). Otherwise, the query is routed to the cloud for processing via RAG or Graph-RAG. The RAG module performs vector-based retrieval over pre-embedded documents, while the Graph-RAG module leverages Neo4j traversal paths to retrieve structured, semantically connected information. In both cases, the retrieved context is appended to the LLM prompt to generate a grounded and context-aware response.

IV. Evaluation

The proposed framework is evaluated on the basis of the following parameters: factual correctness, latency, throughput, and hit rate of the cache. Factual correctness is used to determine the quality of the answers, where the questions asked have a concrete answer. Contextually similar questions are used to determine the performance of the framework using cache, for the parameters of latency, throughput, and hit rate. Throughput is calculated as the total number of tokens, which includes the input and output tokens, divided by the latency of the query. Hit rate is calculated as the ratio of token hits found in the cache and the total number of requested tokens.

A. Experimental Setup

A graph is generated using Neo4j Graph Builder using the O-RAN Alliance Specification documents [21], with the configuration settings being 200 tokens per chunk, chunk overlap of 20, and 3 chunks to combine, with these settings being more optimal for large text based datasets [22]. Context is extracted from the graph data, and used to retrieve answers from it using RAG and Graph-RAG. Google's embedding model, "embedding-001" is used by RAG and Graph-RAG, for entity extraction. "Gemini 1.5 pro" LLM model is used for the generation of a comprehensive response. The number of output tokens is fixed to be 300 to ensure the comparability of results and mitigate inconsistencies, that may occur by the number of output tokens varying by a huge margin for each query. The whole pipeline is executed 3 times using the same datasets to validate the consistency of the performance.

B. Datasets

ORAN-Bench 13K is the first in-depth benchmark specifically created to assess how well LLMs perform in the context of O-RAN. It includes 13,952 carefully constructed multiple-choice questions, generated through a novel three-stage LLM process, and derived from 116 O-RAN specification documents. The questions are divided into three difficulty levels- Easy, Medium, and Difficult, to ensure broad coverage of O-RAN-related topics, such as network analytics, anomaly detection, and code generation.

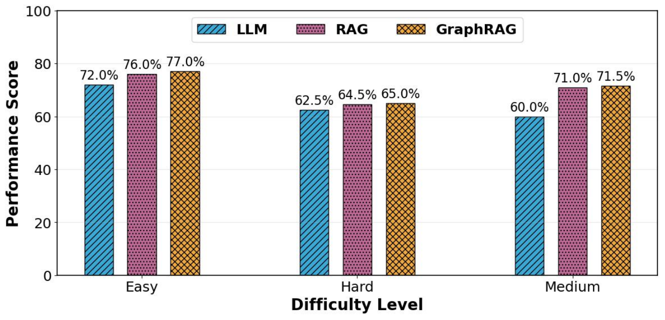


Fig. 3. Factual correctness with respect to different difficulty levels

For latency, throughput, and cache hit-rate evaluation, O-RAN Alliance-specific questions are generated using Google's Gemini. The questions are generated in sets of 25, 50, 75, and 100, with 60% of the questions being unique and the rest varied in context, but the answer being similar. This dataset is generated so that the measure of throughput and hit rate can be calculated correctly, as the framework is expected to retrieve the answers from cache if the same documents are requested under different contextual queries. A semantic cache of size 200 is used with the embeddings model, since the framework is using external LLM of Gemini, and it can not work with local cache memory.

V. Results and Discussion

The experimental results present a comparative evaluation of LLM, RAG, and Graph-RAG pipelines, both with and without semantic caching, across key performance metrics: factual correctness, latency, throughput, and cache hit rate.

A. Factual correctness

The factual correctness is determined on the basis of how accurately the LLM answers any question. Fig. 3 shows the comparison of factual correctness between LLM, RAG, and Graph-RAG. All three pipelines show more than 70% accuracy for the easy questions, but the performance declines slightly when the difficulty level of the questions increases, due to the limitation of any LLM to answer complex domain specific questions. These results show that the proposed framework outperforms the existing context-based models with an average accuracy of 64.8% for LLM, 70.5% for RAG, and 71.2% for the Graph-RAG, and that is caused due to the optimized configuration of Neo4j graph builder as mentioned in the experimental setup.

B. Latency

The results show that the latency is significantly lower for LLM and RAG, for all the sets of questions, in comparison to the Graph-RAG, as it traverses through the graph nodes and also the embeddings to generate the answer. Cache reduces the latency for all the frameworks; the most consistent improvement is seen to be in the case of Graph-RAG, as seen in the Fig. 4, for all the sets of questions. The latency is lesser

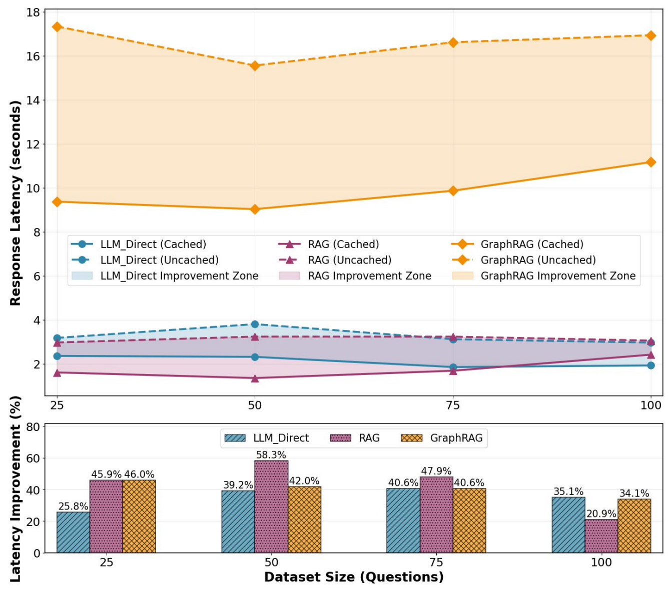


Fig. 4. Latency performance with respect to the number of prompts.

in the case of RAG, with the cache further reducing it for the set of 25, 50, and 75 questions, with an average improvement of 50%, but the performance declines when more prompts are to be processed, making the average latency improvement using RAG to be 35.8%. On the other hand, Graph-RAG has more latency but it's performance is more consistent with the changes in the number of questions, with an average of 40.68%. This shows that for a larger number of prompts, Graph-RAG is prone to perform better than RAG and vice versa, as for a smaller number of prompts, RAG can retrieve the context efficiently, while Graph-RAG's ability to retrieve additional entities and relationships allows it to better handle a larger number of prompts.

C. Throughput and Hit Rate

The average value of throughput has the least difference in terms of performance for all the frameworks, as the number of output tokens is set to be the same, and the same prompt is used for the output generation. In case of uncached performance, the throughput performance is not dependent on the number of questions, but when cache memory is used, the throughput significantly increases and also shows a decrease with the increase in the number of questions. This pattern is expected as the increase in the number of questions require more tokens to be generated, but due to the cache memory already containing those tokens, they are not generated again. While in the case of uncached performance, all tokens are to be generated for each query, regardless of whether the questions are similar in context.

As observed in the Fig. 6, hit rate is the highest for the least number of questions, and then decreases with the increase in the number of questions for RAG and Graph-RAG. LLM does not use embeddings to generate answers, hence the hit rate in that case is zero. Results show that RAG uses the cache memory more effectively than Graph-RAG.

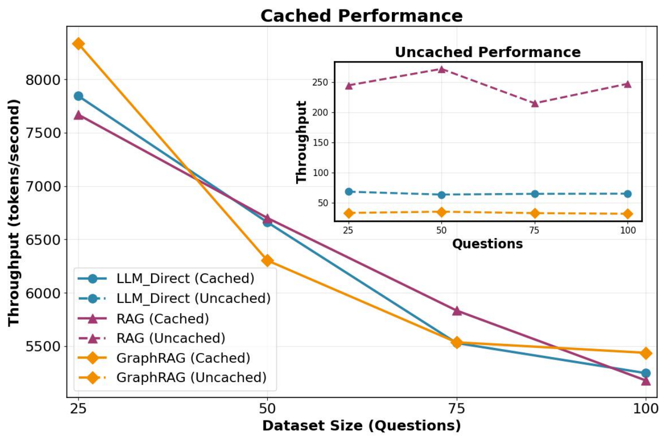


Fig. 5. Throughput performance for end-to-end tokens generation.

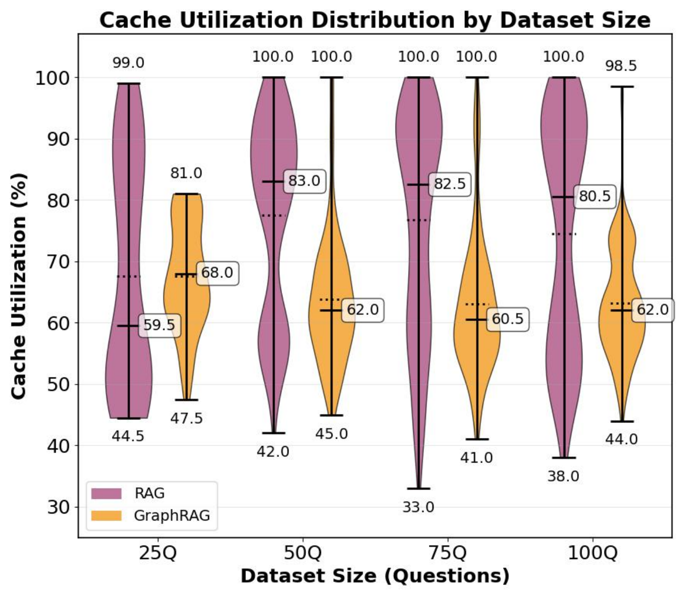
D. All Together

Fig. 6 shows an overall comparison of the algorithms, in terms of cache utilization and hit rate for each set of questions. RAG demonstrates better cache performance with both higher utilization (76.38%) and better hit rate distribution (42.5%), leading to more consistent response times. Graph-RAG shows lower cache effectiveness across all metrics with greater performance variability, but a more consistent cache hit utilization median across all the sets of questions. Cache Hit Rate Mean and Variance graph shows that the cache hit rate decreases for both RAG (46.7% to 31.2%) and Graph-RAG (45.4% to 32.1%) as the dataset size increases. Although RAG generally has a slightly higher cache hit rate than Graph-RAG, both methods show high variability and a decreasing trend in hit rate as the dataset size grows.

These results depict slightly improved results in the case of RAG, since there is a higher chance of data variability and more time consumption in the case of Graph-RAG, as it traverses through the embeddings and the graph nodes. LLM does perform better in case of latency but it has lesser factual correctness, and overall performance also lags behind. These results depict that the performance of RAG and Graph-RAG with semantics cache is very similar, with RAG outperforming the Graph-RAG in terms of cache utilization and latency improvement when lesser number of prompts are to be processed, and Graph-RAG shows more consistent performance even with a larger number of prompts. In case of factual correctness, the Graph-RAG outperforms RAG with a slight margin. The variance in cache utility depends on the cache size and the size of output tokens, which in this case, is more stable in the case of RAG. The stability of results in this case is caused by the use of semantic cache, as it has a higher chance of having similar embeddings, while in the case of Graph-RAG, the probability of new semantics being used is higher, since it also traverses the graph.

VI. Conclusion

These results show that there is a certain tradeoff between the different evaluation parameters, and depending on the user's requirements, both RAG and Graph-RAG can be



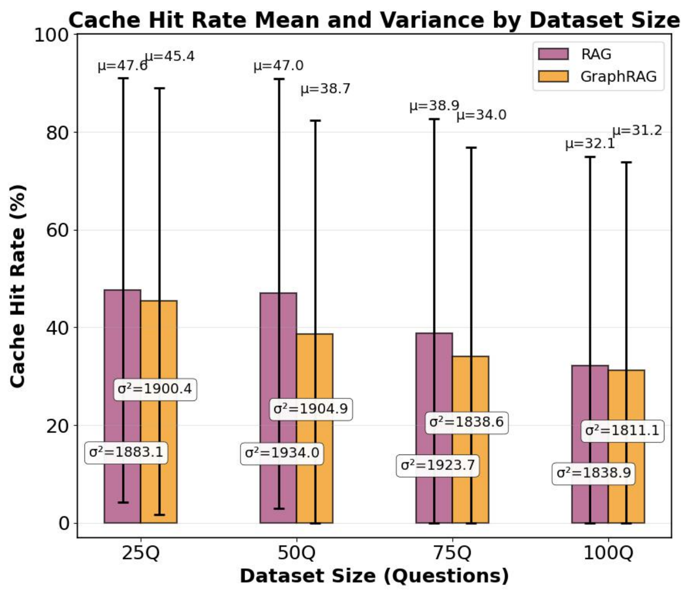


Fig. 6. Performance improvement. Left: Comparison of cache utilization & Right: Comparison of Cache Hit rate, between RAG and Graph-RAG, respectively.

deployed in the O-RAN architecture. Graph-RAG provides a better quality of data, i.e., has better accuracy, with a slight decline in the case of RAG. RAG provides a better middle-ground, with the quality improvement over LLM, and latency improvement over LLM and Graph-RAG both. Cache utilization shows more stability in the case of RAG than Graph-RAG, but the effectiveness of the cache is higher in the case of Graph-RAG, which is also scalable and dependent on the size of the cache. This proves that the Graph-RAG-based cache framework, in comparison homes more reusable cache entries. This research demonstrates that intelligent caching can successfully balance AI capabilities with the network performance, enabling scalable and intelligent O-RAN deployments from edge-based inference to cloud-level reasoning. It also highlights the trade-off between latency, answer quality, and resource utilization, emphasizing the need to optimize caching strategies to achieve the right balance among these factors.

VII. Acknowledgment

This research was funded by EP/X040518/1 EPSRC CHEDDAR, UKRI Grant EP/X039161/1, ORAN-TWIN Project Grant EP/Y037421/1 and EP/X040518/1, MSCA Horizon EU Grant 101086218, and partially by the UKRI Funding Service under Award UKRI851.

References

[1] O-RAN Alliance, "O-RAN Resources," 2024. [Online]. Available: https://www.o-ran.org/o-ran-resources
[2] ETSI, "O-RAN Use Cases Analysis Report," ETSI, Tech. Rep. TR 104 037 V12.0.0, apr 2025, version 12.0.0.
[3] X. Zhou *et al.*, "LLM-enhanced data management," *arXiv:2402.02643*, 2024.
[4] Y. Sui *et al.*, "Table meets LLM: Can large language models understand structured table data? a benchmark and empirical study," in *17th ACM WSDM*, 2024, pp. 645–654.
[5] S. Hong *et al.*, "Data interpreter: An LLM agent for data science," *arXiv:2402.18679*, 2024.
[6] J. Yuan *et al.*, "RAG-Driver: Generalisable driving explanations with retrieval-augmented in-context learning in multi-modal large language model," *arXiv:2402.10828*, 2024.
[7] J. C. dos Santos Junior *et al.*, "Domain-driven LLM development: insights into RAG and fine-tuning practices," in *Proceedings of the 30th ACM SIGKDD*, 2024.
[8] Q. Zhang *et al.*, "A survey of graph retrieval-augmented generation for customized large language models," *arXiv:2501.13958*, 2025.
[9] S. Ahmad *et al.*, "Benchmarking vector, graph and hybrid retrieval-augmented generation (rag) pipelines for open radio access networks (oran)," *in PIMRC*, 2025.
[10] D. Yuan *et al.*, "Enhancing large language models (llms) for telecommunications using knowledge graphs and retrieval-augmented generation," *arXiv preprint arXiv:2503.24245*, 2025.
[11] C. Jin *et al.*, "Ragcache: Efficient knowledge caching for retrieval-augmented generation," *arXiv:2404.12457*, 2024.
[12] J. Yao *et al.*, "CacheBlend: Fast large language model serving for RAG with cached knowledge fusion," in *Proceedings of the Twentieth European Conference on Computer Systems*, 2025, pp. 94–109.
[13] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474.
[14] Y. Xiao *et al.*, "GraphRAG-Bench: Challenging domain-specific reasoning for evaluating graph retrieval-augmented generation," *arXiv:2506.02404*, 2025.
[15] J. Li *et al.*, "SCALM: Towards semantic caching for automated chat services with large language models," in *2024 IEEE/ACM 32nd IWQoS*. Guangzhou, China: IEEE/ACM, 2024, pp. 1–10.
[16] R. Mohandoss, "Context-based semantic caching for LLM applications," in *2024 IEEE CAIW*. Singapore, Singapore: IEEE, 2024, pp. 371–376.
[17] L. Zheng *et al.*, "SGLang: Efficient execution of structured language model programs," arXiv.org, 2023.
[18] L. Ye *et al.*, "Chunkattention: Efficient self-attention with prefix-aware kv cache and two-phase partition," *arXiv:2402.15220*, 2024.
[19] Z. Zhang *et al.*, "Accelerating retrieval-augmented language model serving with speculation," *arXiv:2401.14021*, 2024.
[20] B. J. Chan *et al.*, "Don't do rag: When cache-augmented generation is all you need for knowledge tasks," in *Proceedings of WWW*, 2025, pp. 893–897.
[21] "ORAN specifications," Online. [Online]. Available: https://orandownloadsweb.azurewebsites.net/specifications
[22] K. Joshi *et al.*, "Llm knowledge graph builder front-end architecture and integration," 2025, neo4j Developer Blog, Medium. [Online]. Available: https://tinyurl.com/5n96radc