HierCache-RAG: Fast Retrieval-Level Caching for GraphRAG

Jnana Durga Ravi Chand Mutthina[1,*], Sriharsha Doniparthi[2]

[1]Independent researcher, College Station, Texas, USA
[2]Electrical Engineering and Computer Engineering, University of Cincinnati, Cincinnati, USA

Abstract

Graph Retrieval-Augmented Generation (GraphRAG) improves multi-hop reasoning in Large Language Models (LLMs) by grounding generation in knowledge-graph structure, but most systems re-traverse the graph on every query, wasting computation whenever queries repeat or overlap semantically. We propose HierCache-RAG, a hierarchical retrieval-level caching framework that reuses graph-derived relational context through three cascading layers – an Exact Query Cache, a Semantic Query Cache, and a One-Hop Graph Cache – before falling back to graph traversal. Unlike prior work that caches at the generation or key-value stage, HierCache-RAG caches directly at the retrieval stage. By progressively reusing previously retrieved relational contexts, the framework avoids repeated graph traversal for both exact and semantically similar queries. On DBLP, it reaches 2.03 ms effective retrieval latency (2.32×/10.70× faster than NetworkX/Neo4j) with competitive answer quality; on the more semantically diverse arXiv dataset, it reaches 1.29 ms (up to 61.43× faster) while also achieving the highest answer quality of any system tested. These results indicate that retrieval-level caching can make interactive GraphRAG systems practical for larger knowledge graphs without sacrificing response quality. The gains persist as the graphs scale from 30K to 70K papers, although they narrow on arXiv (speedup over Neo4j falls from 61.43× to 4.00×), and the approach is complementary to existing generation-level and KV caching techniques.

Keywords

Retrieval-Augmented Generation, Knowledge Graphs, GraphRAG, Semantic Caching, Retrieval Latency, Large Language Models

CEUR-WS.org/Vol-4316/pd-short-5.pdf

1. Introduction

Large Language Models (LLMs) remain constrained by static parametric knowledge, making Retrieval-Augmented Generation (RAG) a dominant strategy for grounding responses in external evidence and reducing hallucination [1]. Because many applications require reasoning over structured relationships that plain text chunks cannot express, Graph Retrieval-Augmented Generation (GraphRAG) has emerged to combine knowledge graphs with LLM generation, enabling multi-hop and relationship-aware retrieval [2, 3, 4].

This structural benefit comes at a cost: GraphRAG systems typically re-traverse the graph for every query, so as query volume grows, the same or overlapping neighborhoods are explored redundantly and retrieval latency becomes the dominant bottleneck for interactive use. Prior work on semantic and key-value (KV) caching reduces redundant computation at the generation stage [5, 6, 8, 9], but little work targets reuse at the retrieval stage itself, before graph traversal occurs.

We address this gap with HierCache-RAG, a hierarchical retrieval-level caching framework that progressively attempts three reuse layers – an Exact Query Cache, a Semantic Query Cache, and a One-Hop Graph Cache – before falling back to graph traversal. By reusing graph-derived relational contexts extracted from one-hop neighborhoods, HierCache-RAG reduces redundant traversal while preserving the structured information LLMs need. We evaluate it on DBLP (structured, higher query overlap) and arXiv (semantically diverse, lower overlap) at graph sizes from 30K to 70K papers, against NetworkX and Neo4j baselines, and report retrieval latency, LLM-judged answer quality, scalability, and ablations that isolate the contribution of each cache layer.

IJCKG 2026 Posters, Demos and Workshops, November 19–21, 2026, Bangkok, Thailand
*Corresponding author.
ravimutthina@gmail.com (J. D. R. C. Mutthina); harshad09000@gmail.com (S. Doniparthi)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-06

Table 1
What existing caching approaches reuse, versus HierCache-RAG.

Approach	What it caches	Stage
RAGCache, Cache-Craft [5, 6]	retrieved chunks / outputs	generation
Semantic cache [10], telecom RAG [7]	responses / embeddings	generation
SubGCache, DepCache [8, 9]	KV states (subgraph-aware)	generation
HierCache-RAG	**graph-derived relational context**	**retrieval**

Contributions. (1) HierCache-RAG is, to our knowledge, the first caching framework among the GraphRAG caching approaches reviewed in Sect. 2 to operate at the graph-retrieval stage rather than the generation or KV stage, reusing graph-derived relational context through three cascading layers instead of caching model inputs/outputs. (2) It cuts effective retrieval latency by up to 61× over indexed graph-database retrieval while matching or exceeding baseline answer quality, evaluated on two datasets and three graph scales (30K–70K papers) against NetworkX and Neo4j. (3) An ablation isolates each layer's contribution, showing query-level caching drives most of the latency reduction while the One-Hop Graph Cache preserves answer quality on the queries query-level caching misses.

2. Related Work

RAG systems ground LLMs in externally retrieved evidence, reducing hallucination without retraining [1]; recent surveys identify retrieval efficiency as a persistent scaling challenge [11]. GraphRAG extends this by retrieving over structured knowledge graphs rather than isolated text chunks: Microsoft's GraphRAG builds hierarchical graphs via community detection [2], HybridRAG combines graph traversal with vector retrieval [3], and KG-Retriever focuses on efficient graph-aware indexing [4]. These systems improve retrieval quality but still traverse the graph on every query.

A separate line of work reduces redundant computation through caching, but none of it caches and reuses graph-derived relational context ahead of repeated graph traversal (Table 1): RAGCache and Cache-Craft cache retrieved chunks or intermediate retrieval outputs [5, 6] rather than the graph-derived context itself; a knowledge-graph-enhanced semantic cache improves accuracy while cutting inference cost [10], and domain-specific systems combine semantic caching with GraphRAG for telecom applications [7]; and the most recent GraphRAG-specific caches, SubGCache and DepCache, reuse subgraph-level or dependency-aware KV caches to cut generation-stage computation [8, 9]. None of these retain the relational context that graph traversal itself produces, so none avoids a fresh traversal when a query repeats or overlaps semantically. HierCache-RAG targets that gap directly – caching the graph-derived relational context itself – and is complementary to, not a replacement for, the methods above.

3. Methodology

3.1. Datasets and Query Construction

We use two scientific metadata graphs built from pre-converted JSON sources [12, 13]: DBLP (paper, author, venue, citation relations; higher structural query overlap) and arXiv (paper, author, venue, category relations, with abstracts as node attributes; richer text, lower query overlap). Graphs were built at three scales (30K, 50K, 70K papers) for both datasets. 500 training queries populate the caches; evaluation uses 101 (DBLP) and 100 (arXiv) held-out queries split into exact, semantic, and unseen categories to respectively stress exact-cache reuse, semantic-cache reuse, and graph-fallback behavior.

3.2. Cache Architecture

HierCache-RAG applies three reuse layers, in order, before falling back to graph traversal (Fig. 1):

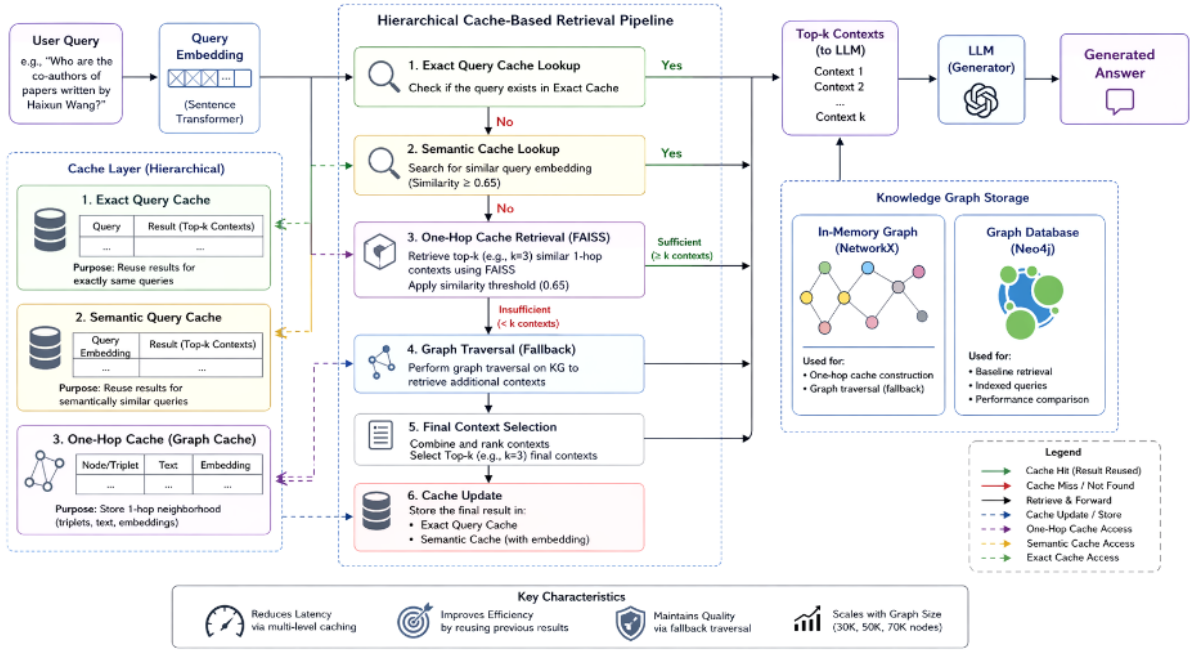


Figure 1: HierCache-RAG retrieval pipeline: a query attempts the three cache layers in order; a hit at any layer returns context directly, while a miss cascades to the next layer and ultimately to graph traversal, whose result is written back into the query-level caches.

Exact Query Cache. Stores: the raw query string as implemented, together with its final retrieval output. Hits: when an incoming query exactly matches the stored string. Avoids: all further computation – the lowest-latency path for repeated queries.

Semantic Query Cache. Stores: query embeddings alongside their retrieval outputs. Hits: when FAISS similarity search [14] against a cached embedding exceeds threshold T ($T = 0.65$ on DBLP, $T = 0.60$ on arXiv, selected empirically). Lower T permits more reuse but may admit weaker matches, whereas higher T is stricter and increases graph fallback; thus, the selected threshold is dataset-dependent.

One-Hop Graph Cache. Stores: one-hop graph neighborhoods serialized as flattened triplet text and embedded for FAISS retrieval. Motivated by prior KG-LLM alignment work on serialized relational structures [22], it attempts to retrieve K graph-derived contexts above T; if sufficient qualifying contexts are unavailable, the request proceeds to Graph Fallback.

If no layer hits, HierCache-RAG falls back to a one-hop traversal over NetworkX [15] (an indexed Neo4j/Cypher deployment [16, 17] serves as an alternative graph-database baseline), and writes the result back into the Exact and Semantic caches so coverage grows over the course of execution.

3.3. Experimental Setup

Implementation used Python, NetworkX, Neo4j, and FAISS, with sentence-transformers/all-MiniLM-L6-v2 embeddings (384-d) [21]; all runs were CPU-based and repeated three times (mean ± std reported). Graph-cache construction, graph-context embedding, and FAISS index construction occur before online query evaluation and are excluded from reported online retrieval latency; the study evaluates steady-state retrieval, not reconstruction under dynamic updates. Effective Latency = (1 – Query-Cache Hit Rate) × Average Retrieval Latency, where Average Retrieval Latency is measured only on cache-miss queries reaching the One-Hop Graph Cache or Graph Fallback. We also report the average retrieval latency on cache-miss queries (Table 2) and the speedup over NetworkX and Neo4j. GPT-4 [19] judged GPT-4o-mini [18] responses generated only from each system's retrieved context. As specified in the evaluation prompt, the judge scored factual support, coverage of requested information, and alignment with the query and context as correctness, completeness, and relevance, and recorded pairwise win/tie/loss comparisons [20].

Table 2
Main performance comparison, DBLP and arXiv (30K papers). Retr. Lat.: average retrieval latency on cache-miss queries; Eff. Lat.: effective latency as defined in Sect. 3.3.

System	Dataset	Retr. Lat. (ms)	Eff. Lat. (ms)	Avg. Score	Fail. Rate
HierCache-RAG	DBLP	7.32	2.03	3.80 ± 0.02	0.21
NetworkX	DBLP	4.72	4.72	3.51 ± 0.05	0.29
Neo4j	DBLP	21.71	21.71	3.84 ± 0.02	0.15
HierCache-RAG	arXiv	9.95	1.29	4.08 ± 0.11	0.16
NetworkX	arXiv	29.10	29.10	2.35 ± 0.03	0.51
Neo4j	arXiv	79.45	79.45	2.22 ± 0.07	0.56

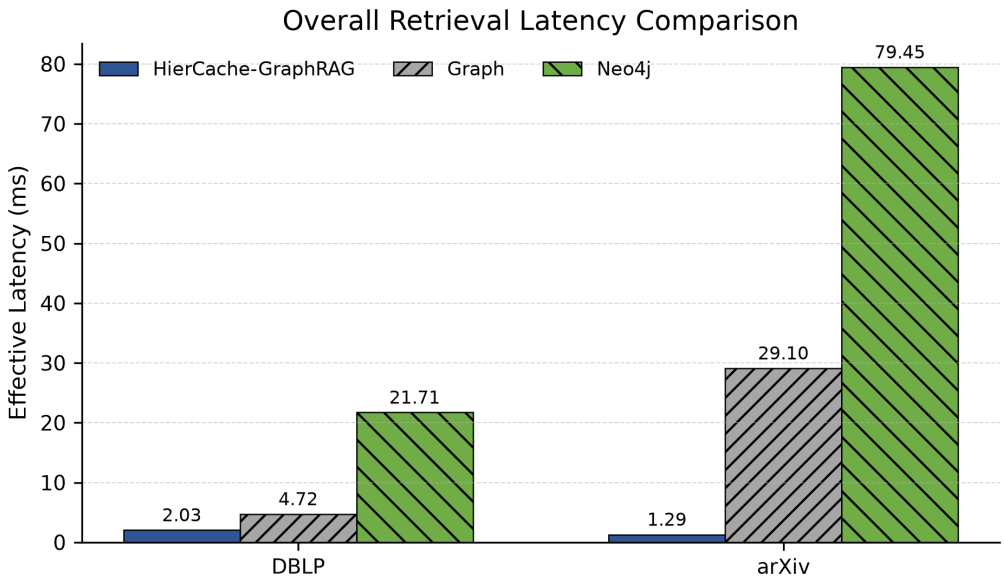


Figure 2: Effective retrieval latency: HierCache-RAG vs. NetworkX and Neo4j baselines, DBLP and arXiv (30K papers).

4. Results

Table 2 and Fig. 2 summarize retrieval efficiency and answer quality on both datasets at the 30K scale. Two findings matter most. First, HierCache-RAG cuts effective latency by roughly an order of magnitude or more relative to Neo4j (2.03 ms vs. 21.71 ms on DBLP; 1.29 ms vs. 79.45 ms on arXiv, up to 61.43× faster than Neo4j; the gain relative to NetworkX ranges from 2.32× on DBLP to over 22× on arXiv) – the gain comes almost entirely from query-level cache hits, which return context with no graph access at all, as the ablation below confirms. On cache-miss queries, HierCache-RAG pays a lookup overhead (7.32 ms vs. 4.72 ms for NetworkX on DBLP, Table 2), so its advantage over NetworkX depends on the query-cache hit rate. Second, this speed does not cost accuracy: on DBLP, HierCache-RAG matches Neo4j's answer quality to within 0.04 points and exceeds NetworkX's by 0.29 (3.80 ± 0.02 vs. 3.51 and 3.84); on the more semantically diverse arXiv dataset it achieves the highest answer quality of any system tested (4.08 ± 0.11 vs. 2.35 and 2.22) alongside the lowest failure rate (0.16), because the One-Hop Graph Cache still supplies useful relational context on the queries the query-level caches miss. Pairwise evaluation shows HierCache-RAG preferred over NetworkX in 16% of DBLP queries and over Neo4j in 30% (remainder tied), and preferred in 40% and 49% of arXiv queries respectively.

Scalability. From 30K to 70K papers, DBLP query-cache hit rate stays in the 72–84% range, with effective latency between 1.17 ms and 2.03 ms and speedup over Neo4j between 10.70× and 16.43×. On arXiv, effective latency rises from 1.29 ms to 10.37 ms and the speedup over Neo4j falls from 61.43× (30K) to 9.48× (50K) and 4.00× (70K), so the benefit persists across the evaluated sizes but narrows as

the graph grows.

Ablation. Removing the Exact and Semantic Query Caches eliminates query-level reuse, increasing arXiv effective latency from 0.00129 s to 0.03605 s and Graph Fallback from 8% to 58%, showing that query-level caching drives most latency reduction. Removing the One-Hop Graph Cache increases reliance on direct Graph Fallback and reduces answer quality (4.08 to 3.28 on arXiv), supporting the usefulness of graph-derived reuse. Two-hop retrieval increases latency without improving quality, so one-hop is the better practical trade-off. The T sweep (0.60, 0.65, 0.70) supports the dataset-specific operating points.

5. Limitations and Future Work

The framework assumes largely static graphs and workloads with repeated or semantically overlapping queries; dynamic invalidation and offline cache/index reconstruction costs under graph updates were not evaluated. Cache growth was unbounded, and memory footprint and memory-latency trade-offs were not quantified; bounded-memory eviction remains future work. T was tuned per dataset, and the flattened triplet representation, although motivated by serialized KG-LLM alignment [22], may omit higher-order graph topology. Downstream answer-quality evaluation measures the usefulness of reused semantic contexts but does not separately annotate semantic-cache false hits across thresholds. Finally, LLM-as-judge evaluation lacks human agreement validation, and results use sampled DBLP/arXiv subsets, which may limit generalization beyond the evaluated workloads and domains.

6. Conclusion

HierCache-RAG shows that caching does not have to stop at the generation stage: reusing graph-derived relational context at retrieval time cuts effective latency by up to 61× without sacrificing answer quality, and the gain persists as graphs scale from 30K to 70K papers, although it narrows on arXiv (61.43× to 4.00× over Neo4j). The main lesson is that most of this benefit comes from the cheapest layer – query-level reuse – while the graph-derived One-Hop Cache is what preserves answer quality on the queries that layer misses; a practical deployment should invest first in fast, well-tuned query caching before turning to more expensive graph-level reuse. The next step is evaluating under conditions closer to production: dynamic graphs that require cache invalidation, bounded cache budgets, and workloads beyond the citation-graph domain studied here.

Disclosure of Interests

The authors have no competing interests to declare that are relevant to the content of this article.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT for language refinement, formatting assistance, and code-related support. All research design, experiments, analysis, interpretation, and final manuscript content were reviewed and validated by the authors, who take full responsibility for the publication's content.

References

[1] Lewis, P., et al.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: NeurIPS (2020)

[2] Microsoft Research: GraphRAG: hierarchical knowledge graphs with community detection for local and global reasoning (2024)

[3] Sarmah, B., Hall, B., Rao, R., Patel, S., Pasquali, S., Mehta, D.: HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction. In: Proc. 5th ACM Int. Conf. on AI in Finance (ICAIF'24), pp. 608–616 (2024)
[4] Chen, W., Bai, T., Su, J., Luan, J., Liu, W., Shi, C.: KG-Retriever: Efficient Knowledge Indexing for Retrieval-Augmented Large Language Models. arXiv:2412.05547 (2024)
[5] Jin, C., et al.: RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. arXiv:2404.12457 (2024)
[6] Agarwal, S., Sundaresan, S., Mitra, S., Mahapatra, D., Gupta, A., Sharma, R., Kapu, N.J., Yu, T., Saini, S.: Cache-Craft: Managing Chunk-Caches for Efficient Retrieval-Augmented Generation. Proc. ACM Manag. Data 3(3), Article 136, 1–28 (2025)
[7] Bano, S., Nezami, Z., Hafeez, M., Zaidi, S.A.R., Ahmed, Q.: Cache-Enhanced RAG and Graph-RAG for O-RAN. In: Proc. 2025 IEEE Globecom Workshops (GC Wkshps), IEEE (in press)
[8] Zhu, Q., Zhang, L., Xu, Q., Long, C., Zhang, J.: SubGCache: Accelerating Graph-based RAG with Subgraph-level KV Cache. Proc. AAAI Conf. Artif. Intell. 40(41), 35204–35212 (2026)
[9] Yuan, H., Ai, X., Wang, Q., Li, P., Yu, J., Chen, C., Yang, X., Zhang, Y., Fu, Z., Wen, Y., Yu, G.: DepCache: A KV Cache Management Framework for GraphRAG with Dependency Attention. Proc. ACM Manag. Data 3(6), 1–29 (2025)
[10] Dominic, N., Pardamean, B.: Knowledge Graph-Enhanced Semantic Cache for Low-Latency and Cost-Effective Inference in Large Language Models. In: 2024 Int. Conf. on Information Management and Technology (ICIMTech), pp. 340–344. IEEE (2024)
[11] Şakar, T., Emekci, H.: Maximizing RAG efficiency: a comparative analysis of RAG methods. Nat. Lang. Process. 31(1), 1–25 (2025)
[12] FROSch Dataset Repository: DBLP: Pre-converted JSON dataset. https://frosch.cosy.sbg.ac.at/datasets/json/dblp (2021), accessed 5 July 2026
[13] FROSch Dataset Repository: arXiv: Pre-converted JSON dataset. https://frosch.cosy.sbg.ac.at/datasets/json/arxiv (2021), accessed 5 July 2026
[14] Douze, M., Guzhva, A., Deng, C., et al.: The Faiss library. arXiv:2401.08281 (2024)
[15] Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: Proc. 7th Python in Science Conf. (SciPy2008), pp. 11–15 (2008)
[16] Francis, N., Green, A., Guagliardo, P., et al.: Cypher: An Evolving Query Language for Property Graphs. In: Proc. 2018 Int. Conf. on Management of Data (SIGMOD), pp. 1433–1445 (2018)
[17] Neo4j: Cypher Manual. https://neo4j.com/docs/cypher-manual/current/, accessed 5 July 2026
[18] OpenAI: GPT-4o mini: advancing cost-efficient intelligence (2024)
[19] OpenAI: GPT-4 Technical Report. arXiv:2303.08774 (2023)
[20] Zheng, L., Chiang, W.-L., Sheng, Y., et al.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: NeurIPS 2023 Datasets and Benchmarks Track. arXiv:2306.05685 (2023)
[21] Reimers, N., Gurevych, I.: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proc. 2019 Conf. on Empirical Methods in Natural Language Processing and 9th Int. Joint Conf. on Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992. ACL (2019)
[22] Tian, S., Xing, S., Li, X., Luo, Y., Yuan, C., Chen, W., Jiang, H., Wang, X.: A Systematic Exploration of Knowledge Graph Alignment with Large Language Models in Retrieval Augmented Generation. Proc. AAAI Conf. Artif. Intell. 39(24), 25291–25299 (2025)