Hybrid Multi-Source RAG for Context-Aware Conversational Data Access

Giorgia Di Nicola, Stefano Iannucci and Riccardo Torlone

Roma Tre University, Rome, Italy

Abstract

Retrieval-Augmented Generation (RAG) augments large language models (LLMs) with external knowledge, yet most RAG systems rely solely on vector-based semantic search, limiting their handling of structured or relational information. We propose a hybrid multi-source RAG architecture that unifies graph-based and vector-based retrieval within a single conversational framework. A prompt-driven orchestrator dynamically selects and fuses the appropriate source at runtime, leveraging graph databases for symbolic reasoning and vector stores for semantic similarity. Experimental evaluation with LLM-as-a-Judge scoring shows that the hybrid configuration consistently outperforms single-source baselines, achieving higher relevance, fluency, faithfulness, and interpretability without any additional training overhead.

1. Introduction

Large language models (LLMs) have revolutionized NLP, showing strong performance on many linguistic and reasoning tasks. Their knowledge, however, is limited by static, incomplete training data [1], leading to outdated or inaccurate answers on complex, domain-specific queries [2]. Retrieval-Augmented Generation (RAG) mitigates this issue by coupling generative models with external retrieval [3]. Conventional RAG relies on dense-vector retrieval, which excels at semantic similarity search over unstructured text [4] but lacks the explicit relational reasoning that graph databases provide [5].

Real-world knowledge, however, is rarely monolithic. It spans heterogeneous modalities, from unstructured to structured data, each providing distinct and complementary advantages for representing information. Within this landscape, vector databases are widely used today to capture semantic meaning and conceptual similarity among textual contents, enabling retrieval based on related ideas rather than exact wording. Conversely, graph databases are used to model explicit relationships and hierarchical connections between entities, supporting structured reasoning and interpretability. A truly context-aware conversational agent should be able to leverage both sources, integrating semantic and symbolic retrieval to deliver responses that are both fluent and interpretable.

To this end, we propose a hybrid multi-source RAG architecture that maintains a clear separation between retrieval modalities while allowing dynamic, transparent source selection at runtime. Unlike joint-embedding or reranking-based approaches, it keeps graph and vector pipelines independent, delegating their integration to a prompt-driven orchestration layer where the LLM acts as a reasoning controller. This late-fusion strategy preserves the complementary strengths of each modality and ensures modularity, adaptability, and interpretability, making the system easily extensible and auditable in heterogeneous production environments. The resulting architecture has been implemented in a system that dynamically routes queries through graph-based, vector-based, or hybrid retrieval paths and evaluates retrieved contexts through prompt-engineered selection logic. This allows the conversational agent to generate responses that are semantically rich, structurally consistent, and explainable simultaneously.

The contributions of this work are threefold: (i) the design and implementation of a modular RAG architecture integrating graph and vector databases under a unified agentic framework; (ii) a prompt-based source selection strategy enabling dynamic and transparent fusion of heterogeneous retrieval outputs without additional model training; (iii) a quantitative evaluation demonstrating that the hybrid

Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland

approach achieves higher response quality measured by relevance, fluency, and faithfulness, compared to graph-only and vector-only baselines.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the design of the hybrid multi-source RAG architecture. Section 4 presents the evaluation methodology, and Section 5 reports and discusses the experimental results. Section 6 concludes the paper and outlines future research directions.

2. Related Work

Retrieval-Augmented Generation (RAG) has become a principal technique for granting large language models access to external knowledge [3]. Conventional RAG pipelines rely on vector databases to perform semantic similarity search over unstructured text; this yields broad coverage but falls short on tasks that demand structural reasoning, precise entity resolution, or interpretability. To address these shortcomings, recent work has explored graph-based retrieval (GraphRAG), in which information is accessed through graph traversal or query execution rather than dense-embedding search [6]. GraphRAG provides advantages for multi-hop reasoning, disambiguation, and factual reliability, especially when coupled with graph-neural-network retrievers [7] or heterogeneous node designs such as NodeRAG [8] and multi-partite frameworks like Clue-RAG [9], which improve precision and cost-efficiency.

Hybrid retrieval systems aim to combine the semantic richness of vector stores with the explicit structure of graph stores, but in doing so they must balance two objectives that are complementary yet potentially in conflict. PruningRAG [10] embeds a static knowledge-filtering step into the retrieval phase to reduce hallucinations and enhance factual consistency, but its fixed heuristics limit adaptability to the specific informational demands of a query. Multi-Source Prompt Retrieval (MSPR) [11] learns a dynamic selection strategy that decides, on a per-query basis, which sources to consult, achieving strong empirical performance at the cost of substantial training and maintenance overhead and reduced transparency. Joint-embedding approaches attempt to fuse modalities into a shared latent space, enabling multimodal retrieval but inevitably blurring semantic precision and compromising interpretability [12]. KG-infused RAG augments conventional text-based retrieval with knowledge-graph facts to curb hallucination, yet it relies on static integration rules that hinder flexibility and obscure source-priority decisions [13]. The more elaborate multi-partite design of Clue-RAG [9] explicitly models cross-source dependencies and retrieves compact, semantically coherent sub-graphs for combination with textual passages, preserving graph semantics but introducing significant computational overhead and orchestration complexity that limit runtime efficiency and modular extensibility.

These collective efforts illustrate the core dilemma in hybrid retrieval: the trade-off between maximizing semantic breadth and maintaining structural clarity. While some methods improve recall, they can reduce interpretability. Conversely, those that prioritize graph semantics often sacrifice flexibility and efficiency. To address these challenges and unlike previous work, our solution maintains a clear separation between retrieval modalities while allowing dynamic, transparent source selection at runtime. The design ensures modularity and adaptability across heterogeneous domains by deferring integration to a prompt-driven orchestration layer rather than relying on joint embeddings or reranking mechanisms. This late-fusion strategy preserves the complementary strengths of graph and vector pipelines and supports extensible, auditable operation in real-world production settings.

3. Design of the Hybrid Multi-Source RAG Agent

3.1. Architecture Overview

The agent is designed around a modular architecture that separates retrieval modalities while enabling dynamic coordination at runtime. Integration is achieved through a late-fusion orchestration layer, enabling dynamic selection and combination of results at runtime while maintaining interpretability, modularity, and adaptability. The modular design facilitates extension to additional sources or

modalities with minimal modification to existing components. The architecture is composed of three complementary components: (1) a *Vector RAG pipeline* for semantic retrieval over unstructured documents, (2) a *Graph RAG pipeline* for schema-aware reasoning over structured knowledge, and (3) an *Entity Resolution subroutine*, an auxiliary chain that resolves raw identifiers into canonical labels through a Graph Lookup process.

Vector pipeline. The vector pipeline embeds the input query using a transformer-based model and retrieves top-k chunks from a vector store through approximate nearest neighbor search. The retrieved documents are concatenated with the original query and forwarded to an LLM for answer generation. The pipeline leverages dense embeddings to identify relevant content beyond strict term matching, making it effective for lexically distant queries from the source corpus. This approach enables retrieval across heterogeneous documents and varying linguistic formulations, making it well-suited for exploratory or underspecified information needs. In summary, the Vector RAG pipeline provides schema-indepentent retrieval, ensuring that when the knowledge graph yields limited or no relevant results, the system can still rely on vector-based evidence to construct the final response.

Graph pipeline. The graph pipeline leverages a knowledge graph to perform structured retrieval and multi-hop reasoning over domain knowledge. It begins by translating the input query into a Cypher query via prompt-guided LLM reasoning (see Section 3.2), which is then executed against the knowledge graph to retrieve relevant entities and relationships. Finally, the retrieved data are reformulated in natural language using a dedicated prompt, producing a human-readable answer. By operating over explicitly structured data, the Graph RAG pipeline ensures that answers are grounded in version-controlled, schema-compliant information. This separation between symbolic query execution and natural-language rendering improves transparency, facilitates debugging, and grounds the answers in version-controlled structured data. As a result, the Graph RAG component provides precise, schema-aware retrieval that complements the Vector RAG pipeline.

Entity Resolution (Graph Lookup). A distinctive feature of the proposed design is the Graph Lookup subroutine, which addresses the frequent occurrence of identifier-based outputs in graph databases (e.g., `ns/m.07ssc` in the WikiGraphs [14] dataset). Upon detection of unresolved identifiers in the Graph RAG output, the subroutine is triggered to execute a narrowly scoped Cypher query to retrieve the corresponding canonical names or human-readable values for the returned nodes. The subroutine then produces these results as key-value mappings, which are applied by the main pipeline to replace the original identifiers in the graph-derived output, producing fully interpretable results. This refinement step ensures that all structured information retrieved from the knowledge graph is interpretable and unambiguous, improving downstream usability and clarity in production contexts.

3.2. Automatic Cypher Query Generation

A central component of the Graph RAG pipeline is the automatic generation of Cypher queries from natural language questions. To achieve this, a large language model with an extended context window is leveraged to handle detailed prompts that describe the full knowledge graph schema, entity types, and relationships.

The methodology is structured as follows:

1. Expert Role Declaration. The prompt begins by instructing the LLM to act as an expert Cypher query generator for Freebase-like knowledge graphs. This sets the reasoning context and ensures the model produces valid, schema-compliant queries.

2. Schema Inclusion. The complete knowledge graph schema is provided to the LLM, defining the structure of nodes, their properties, and allowed relationships. The schema is presented in a simplified form in Listing 1. However, the full schema conceptually includes:

- **Nodes:** `Entity` with properties (`center`: STRING, identifier; `title`: STRING, name).
- **Relationships:** `REL` between `Entity` nodes with property (`type`: STRING).
- **Allowed edges:** `Entity --[REL]--> Entity`.

In addition, all relations are described in natural language and grouped by category, such as:

- `ns/type.object.name`: primary human-readable name of an entity.
- `ns/common.topic.description`: textual description providing context for a topic.
- `ns/people.person.date_of_birth`: birth date of an individual (YYYY-MM-DD).
- `ns/people.person.profession`: occupation of a person.

3. Natural Language Question. The user's query is inserted into the prompt.

4. Formatting and Term Transformation Rules. Instructions specify basic text formatting and term adjustments, such as replacing spaces with underscores, quoting string literals, and simplifying expressions.

5. Cypher Pattern Instructions. The prompt enforces a `MATCH-WHERE-RETURN` construct and limits `r.type` to the set of relations present in the provided schema.

6. Identifier Handling and Output Formatting. If retrieved entities contain machine-readable identifiers, the above described entity resolution mechanism is engaged to map these to human-readable labels before the final output is returned to the user.

Gaining from a large-context LLM, the pipeline can encode the full schema without truncation, producing accurate Cypher queries that form the basis for structured graph retrieval. Listing 1 presents a simplified prompt omitting relation semantics for conciseness.

Role. You are an expert Cypher query generator for Freebase-like knowledge graphs.

Input schema.

- Node: `Entity {center: STRING, title: STRING}`
- Relationship: `REL` connecting `Entity` nodes `{type: STRING}`
- Allowed edges: `Entity -[REL]-> Entity`
- Input question: `{question}`

Instructions.

- Generate a valid Cypher query using `MATCH--WHERE--RETURN`.
- Use only the relationships listed below.

Allowed relationships.

- *General topic information:* `ns/common.topic.alias`, `key/wikipedia.en`
- *People and biographical information:* `ns/people.person.date_of_birth`, `ns/people.person.profession`

Output. Return only the Cypher query.

Listing 1: Prompt used to generate Cypher queries constrained by a predefined schema and relationship whitelist.

3.3. Coordination and Flow Activation

The coordination mechanism constitutes a central aspect of the system architecture, governing the sequential execution of multiple retrieval pipelines in response to user queries. Each pipeline is executed in turn, beginning with the Vector RAG pipeline, followed by the Graph RAG pipeline. The Graph Lookup subroutine is conditionally triggered when unresolved identifiers are detected in the Graph RAG output, refining the results with canonical labels.

Intermediate outputs from all pipelines are aggregated by the final LLM node, which orchestrates the integration of retrieved information based on contextual relevance and informativeness. In cases

where one source provides limited or non-informative content—-i.e., when the retrieved context does not add relevant or precise information to the user's question as defined in the prompt instructions (see Listing 2)–the system relies on the alternative source; when both sources provide complementary information, their contents are fused to generate a coherent response that is both contextually grounded and comprehensive.

This adaptive orchestration preserves modularity, transparency, and interpretability, while balancing semantic coverage and structural precision to handle ambiguous or underspecified queries effectively.

3.4. Hybrid Integration Strategy

The integration of heterogeneous retrieval modalities is deferred to a late-fusion stage. Instead of relying on static merging rules, the outputs of the retrieval pipelines (optionally refined through identifier resolution) are provided to the final LLM. An excerpt of the prompt guiding the final LLM is shown in Listing 2.

Prompt used for hybrid orchestration

You are given two independent context sources: GraphRAG and VectorRAG, and the user's question. Your task is to generate the most accurate, complete, and fluent answer based strictly on the provided contexts and the user's question.

Instructions:

- If both sources provide relevant and compatible information, merge them.
- If only one source provides useful content, use that source.
- If the sources conflict, select the more specific or factual one.
- If neither source is relevant, reply: "The information is not available in the provided context."
- Output only one unified answer.

Listing 2: Prompt guiding the final LLM in the hybrid RAG architecture.

This prompt concretely operationalizes the integration strategy, ensuring context-aware, accurate, and coherent answers without hardcoded merging rules. Through prompt-driven orchestration, the model dynamically assesses the GraphRAG and VectorRAG contexts, integrating them when both contribute relevant information, favoring the more precise or specific source in case of conflict, and filtering out irrelevant or uninformative content.

This strategy preserves the strengths of each modality while avoiding the semantic overlap observed in joint embedding approaches and the rigidity of pruning- or reranking-based methods.

4. Evaluation Methodology

The evaluation systematically compared Vector, Graph, and Hybrid RAG. The methodology comprised three components: a manual curated dataset, standardized evaluation metrics, and a semi-automated annotation protocol, each described in the following subsections.

4.1. Dataset Construction

The evaluation dataset builds upon WikiGraphs [14], which pairs full Wikipedia articles from the WikiText-103 corpus with automatically extracted subgraphs of Freebase. The dataset is released in three structured variants—-`max256`, `max512`, and `max1024`—-that differ in the maximum number of nodes per subgraph (256, 512, and 1024 respectively). These variants provide progressively larger and more connected graph representations of the same underlying knowledge, enabling systematic analysis of retrieval and reasoning performance across increasing structural complexity. In all configurations,

each graph remains aligned to its corresponding Wikipedia article, preserving semantic consistency and factual grounding across scales.

A set of approximately 100 query–answer pairs, applicable to all three dataset variants, was manually created to expose complementary strengths and weaknesses of the retrieval modalities. They include general knowledge questions, fine-grained factual queries, relation-centric queries, and adversarial or underspecified cases. Ground-truth answers were consolidated from both textual and graph sources to ensure benchmark reliability and prevent biases.

To further extend the evaluation to large-scale conditions, an additional dataset was specifically constructed for the `max1024` configuration. This supplementary set comprises approximately 80 query–answer pairs associated specifically with the largest and most structurally complex graphs. The questions related to the `max1024` variant were generated with the assistance of ChatGPT and subsequently verified manually to ensure factual correctness and coverage.

4.2. Evaluation Metrics

The evaluation of system outputs was conducted along three principal dimensions, each capturing a distinct aspect of answer quality:

1. **Relevance**: the extent to which the generated answer addresses the user query and effectively incorporates the retrieved context. High relevance indicates strong alignment with the information need and meaningful integration of supporting content, reflecting the effectiveness of both retrieval and grounding.
2. **Faithfulness**: the degree to which the generated answer is factually consistent with the retrieved knowledge. A faithful response does not introduce information that is unsupported or misrepresents the source material. This dimension is critical in retrieval-augmented generation, where the trustworthiness of the output depends on its grounding in the retrieved context.
3. **Fluency**: the naturalness, grammatical correctness, and structural coherence of the response. Evaluated independently from factual accuracy, this dimension ensures linguistic quality, acknowledging that a fluent output may still be factually incorrect or irrelevant.

4.3. Annotation Protocol

Evaluation was conducted using a semi-automated protocol following the *LLM-as-a-Judge* paradigm, in which a large language model (Llama-4-Scout-17B-16E-Instruct-FP8a) was employed as the annotator. Each instance of the dataset, comprising query, generated answer, retrieved context, ground-truth reference, and retrieval method, was processed automatically to generate scores for all evaluation dimensions. The workflow involved structured input preparation, automated evaluation, and parsing and aggregation of outputs. Queries, answers, contexts, and ground-truth references were serialized into structured records to ensure reproducibility. The LLM evaluated each instance according to the defined criteria, returning both numeric scores (on a 1–-10 scale, subsequently normalized to [0,1]) and textual justifications. Outputs were normalized and stored alongside the original dataset.

A minimal input record is shown below:


```
"question": "What gameplay modes are available
             in Maniac Mansion?",
"method": "Graph RAG",
"context": [
  "The gameplay modes available
  in Maniac_Mansion are ns/m.02hjn4",
  {"id":"ns/m.02hjn4","name":"Single-player"}
],
"answer": "The gameplay modes available are
           Single-player",
"ground_truth": "The gameplay modes available
are Single-player video game."
```

These records form the basis for LLM-based scoring and enable stratified analyses across retrieval strategies, supporting both aggregate comparisons and the identification of method-specific strengths and weaknesses.

5. Experimental Results

We conducted experiments aimed at evaluating both, the effectiveness and the performance of the proposed approach. All of them have been run on a machine equipped with an Intel Xeon Gold 5512U@2.1 Ghz and 128GB RAM. The graph database employed is Neo4j 5.19.0 (with APOC 5.19.0), while the vector store is Qdrant 1.11.0 (running in a Docker container). Flowise 3.0.0, also running in a Docker container, was used for pipeline orchestration and execution, and LangSmith SDK 0.4.31 for trace logging and debugging. Ollama 0.12.2, running in a Docker container, was employed for embedding query vectors within the Vector RAG pipeline. The Groq Python SDK version 0.29.0 was utilized for integrating the inference capabilities of Groq's Language Processing Unit (LPU).

5.1. Effectiveness Evaluation

The empirical evaluation provides a comparative analysis of Vector RAG, Graph RAG, and the proposed Hybrid RAG agent along the three assessment dimensions defined in Section 4.2: relevance, faithfulness, and fluency. Table 1 reports the average normalized scores (0–1 scale) for each retrieval strategy.

The Cypher queries used by the Graph RAG pipeline were generated automatically by an LLM (Llama-4-Scout-17B-16E-Instruct-FP8a) with a large context window to accommodate the full graph schema. This allowed prompt-driven, step-by-step query construction covering all entity types and relationships.

Method	Relevance	var	Faithfulness	var	Fluency	var
Vector RAG	0.71	0.091	0.75	0.103	0.68	0.031
Graph RAG	0.68	0.087	0.88	0.059	0.57	0.041
Hybrid RAG	**0.92**	**0.039**	**0.90**	**0.035**	**0.67**	**0.024**

Table 1
Average normalized scores (0–1 scale) across evaluation dimensions for all retrieval strategies.

The results show that each retrieval modality exhibits a characteristic trade-off among the different dimensions. Vector RAG achieves relatively balanced outcomes, with moderate relevance and faithfulness and the highest fluency score among the baselines. Its reliance on semantic similarity search allows it to capture diverse contexts, but at the cost of occasional factual imprecision. Graph RAG demonstrates the strongest faithfulness, confirming the reliability of schema-compliant retrieval, yet its relevance and fluency are lower, reflecting difficulties in handling underspecified queries and the rigidity of graph-derived outputs. The Hybrid RAG agent clearly outperforms both baselines, achieving the highest scores in relevance and faithfulness while maintaining fluency comparable to the best-performing method.

By coupling graph and vector pipelines, the Hybrid approach mitigates each method's weaknesses: structured graph evidence limits hallucinations, while dense-vector retrieval fills coverage gaps. Automatic Cypher generation guarantees correct, reproducible graph queries, and a Graph-Lookup subroutine replaces raw IDs with canonical labels, boosting interpretability and perceived answer quality. Performance differences stem from the underlying retrieval mechanisms: Graph RAG exploits explicit KG structure, Vector RAG leverages semantic embeddings for broad context, and Hybrid RAG uses prompt-driven late-fusion to pick and merge the most pertinent information from both sources. Consequently, Vector RAG shows larger relevance/faithfulness fluctuations, whereas Hybrid RAG attains the highest average scores and the lowest variance, indicating more stable behavior across query types. These findings validate the central claim: a modular, multi-source RAG architecture that preserves

Category	Method	Relevance	Faithfulness	Fluency
General Knowledge	Vector RAG	0.643	0.703	0.668
	Graph RAG	0.730	**0.971**	0.605
	Hybrid RAG	**0.889**	0.908	**0.697**
Fine-grained Factual	Vector RAG	0.729	0.765	0.671
	Graph RAG	0.676	**0.900**	0.506
	Hybrid RAG	**0.894**	0.841	**0.671**
Relation-centric	Vector RAG	0.725	0.735	**0.755**
	Graph RAG	0.560	0.764	0.520
	Hybrid RAG	**0.955**	**0.880**	0.660
Adversarial / Underspecified	Vector RAG	0.660	0.640	0.620
	Graph RAG	0.593	–	0.527
	Hybrid RAG	**0.887**	**0.960**	**0.653**

Table 2
Average normalized scores by query category.

modality separation yet integrates them via late-fusion delivers superior reliability, adaptability, and interpretability compared to single-modality systems. The Hybrid RAG agent thus emerges as a promising design for real-world deployment scenarios where both accuracy and transparency are essential, while its modular implementation in a low-code framework (e.g., Flowise) ensures seamless integration without reliance on hardcoded logic.

Performance by Query Type. To further examine the behavior of the retrieval pipelines, the evaluation was stratified by question category-general knowledge, fine-grained factual, relation-centric, and adversarial or underspecified cases, as defined in Section 4.1.

Table 2 reports average normalized scores (0–1) per method and category. The results highlight the complementary strengths and limitations of each retrieval strategy. Hybrid RAG consistently achieves the highest relevance and maintains strong faithfulness, effectively combining graph precision with vector flexibility. Graph RAG delivers exceptional faithfulness on schema-anchored queries (general and fine-grained) but suffers in relevance and fluency when broader or less structured context is needed. Vector RAG ensures fluent, semantically rich responses, particularly for relation-centric queries, yet its moderate faithfulness exposes a risk of factual errors. Overall, the hybrid approach balances these trade-offs, leveraging structured grounding and semantic coverage to achieve the strongest overall performance.

5.2. Performance Evaluation

For scalability evaluation, all graphs from the three datastore configurations (`max256`, `max512`, and `max1024`) were included in the experimental pool. To enable controlled cross-scale comparison, we identified the subset of graphs that are structurally identical across variants—-hereafter denoted as G_{common}. These matched graphs share the same number of nodes and relations within Neo4j, and identical semantic content in the corresponding Qdrant chunks. By issuing identical query sets over G_{common} across configurations, any observed performance differences can be attributed exclusively to datastore growth rather than graph-specific structural variations. For each configuration, the same queries were executed on the corresponding $|G_{\text{common}}|$ graphs using identical prompting and LLM configurations, in order to isolate the contribution of datastore size and composition.

Dataset expansion between configurations is quantified as follows. From `max256` to `max512`, nodes increased by approximately 1.17× (407,218 / 347,949), relations by 1.16× (714,582 / 613,705), and vector chunks by 1.02× (969,993 / 946,423), yielding an average structural increase of ≈ 1.12×. From `max512` to `max1024`, nodes increased by 1.05× (426,344 / 407,218), relations by 1.04× (742,265 / 714,582), and

vector chunks by 1.00× (973,719 / 969,993), resulting in an average structural increase of ≈ 1.03×.

To verify Groq's stability as an execution environment, each query was run three times per configuration, measuring latency mean and standard deviation. The resulting low variability confirmed Groq's consistent inference times, allowing its use for the scalability evaluation.

Figure 3 presents a stacked visualization of the latency contributions from the main pipeline components: Vector RAG, Graph RAG, Graph Lookup, selection and final answer generation (including coordination and flow activation), and Groq execution.

Notably, the selection and generation stage remains largely stable across scales, measuring values of 1105 ms (`max256`), 1065 ms (`max512`), and 1097 ms (`max1024`), despite growth in dataset size and the latencies of upstream components. This stability reflects the effectiveness of the dynamic orchestration and prompt-driven source selection , which integrates only the most informative outputs and filters uninformative content.

Groq execution is also highly stable, with standard deviations ranging from $\sigma \approx 0.13$ s to $\sigma \approx 0.30$ s. Vector RAG is consistently the fastest component, followed by Graph Lookup and selection/generation, while Graph RAG exhibits the highest latencies due to Cypher generation, traversal, and reformulation. Overall, standard deviations remain moderate ($\sigma \approx 25$–30 ms for Vector and Graph RAG, 15–20 ms for Lookup and selection), confirming reproducible performance and sub-linear scalability across dataset scales.

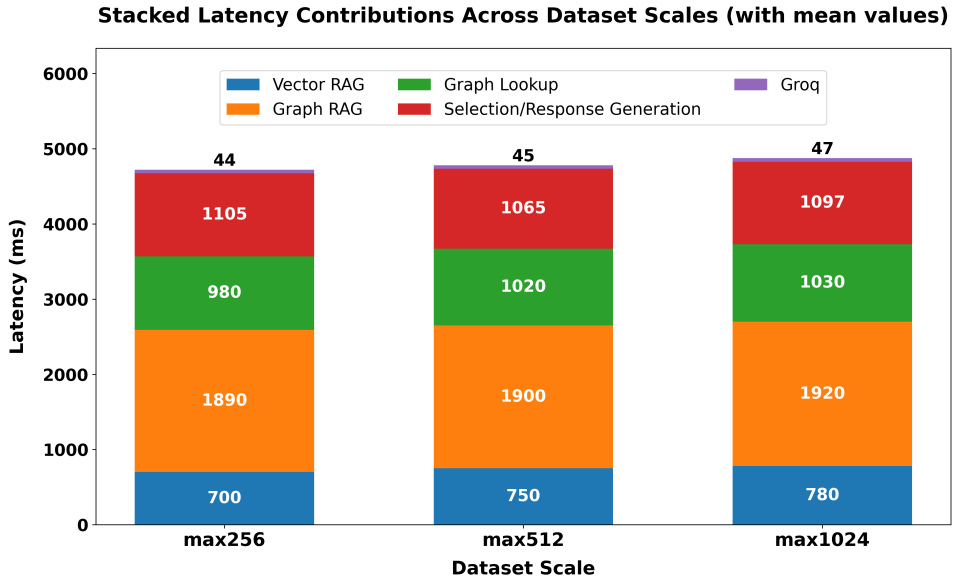

Figure 3: Latency contributions across dataset scales.

5.2.1. Full-scale descriptive runs (max1024)

Whereas the matched-subset analysis provided a controlled comparison by holding per-graph structure constant, the full-scale descriptive runs aim to characterize system behaviour under high-load conditions. This evaluation employs the largest graphs available in the `max1024` variant. The results confirm that all pipelines remain within practical latency ranges even under these demanding conditions.

Vector RAG exhibits the lowest mean latency (≈0.8s), followed by Graph RAG (≈1.9s). Hybrid RAG, as expected, shows higher latency (≈4.9s) due to the additional orchestration, selection, and generation step via the LLM; however, this overhead remains stable across scales and does not compound with datastore size. The full-scale evaluation indicates that all pipelines sustain efficient latency even when processing the most structurally complex graphs. Notably, the Hybrid RAG—-despite incorporating an LLM-driven orchestration layer—-retains stable and robust scalability, demonstrating the system's ability to handle high-load conditions effectively without critical bottlenecks.

6. Conclusions and future work

This work presented a hybrid multi-source RAG architecture that integrates graph-based and vector-based retrieval through a prompt-orchestrated late-fusion strategy. By preserving modality separation and enabling dynamic source selection, the proposed system achieves a balanced trade-off between semantic coverage and structural precision. Experimental results confirm that the hybrid approach improves relevance and faithfulness over single-modality baselines while maintaining low latency and high scalability.

Future work will explore adaptive weighting of retrieval sources, integration of additional modalities, and automated evaluation frameworks to further enhance transparency and robustness in data-centric conversational AI.

Artifacts

The source code is available at:
https://github.com/Giorgia-DNicola/rag-multisource-agent

Declaration on Generative AI

During the preparation of this manuscript, the authors used ChatGPT-5.2 for grammar and spelling review. After using this service, they carefully reviewed and edited the manuscript as needed and take full responsibility for the content of the publication.

Acknowledgements

This work was partially supported by the PRIN-MUR project 2022XERWK9 S-PIC4CHU.

References

[1] F. Petroni, A. Piktus, A. Fan, P. Lewis, M. Yazdani, N. De Cao, J. Thorne, Y. Jernite, V. Karpukhin, J. Maillard, V. Plachouras, T. Rocktäschel, S. Riedel, KILT: a benchmark for knowledge intensive language tasks, in: K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, Y. Zhou (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 2523–2544. URL: https://aclanthology.org/2021.naacl-main.200/. doi:10.18653/v1/2021.naacl-main.200.

[2] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, ACM Comput. Surv. 55 (2023). URL: https://doi.org/10.1145/3571730. doi:10.1145/3571730.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive nlp tasks, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.

[4] G. Izacard, E. Grave, Leveraging passage retrieval with generative models for open domain question answering, in: P. Merlo, J. Tiedemann, R. Tsarfaty (Eds.), Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, Association for Computational Linguistics, Online, 2021, pp. 874–880. URL: https://aclanthology.org/2021.eacl-main.74/. doi:10.18653/v1/2021.eacl-main.74.

[5] C. Peng, F. Xia, M. Naseriparsa, F. Osborne, Knowledge graphs: Opportunities and challenges, Artif. Intell. Rev. 56 (2023) 13071–13102. URL: https://doi.org/10.1007/s10462-023-10465-9. doi:10.1007/s10462-023-10465-9.

[6] H. H. et al.and Yu Wang, H. Shomer, K. Guo, J. Ding, Y. Lei, M. Halappanavar, R. A. Rossi, S. Mukherjee, X. Tang, Q. He, Z. Hua, B. Long, T. Zhao, N. Shah, A. Javari, Y. Xia, J. Tang, Retrieval-augmented generation with graphs (graphrag), 2025. URL: https://arxiv.org/abs/2501.00309. arXiv:2501.00309.

[7] C. Mavromatis, G. Karypis, GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs, in: W. Che, J. Nabende, E. Shutova, M. T. Pilehvar (Eds.), Findings of the Association for Computational Linguistics: ACL 2025, Association for Computational Linguistics, Vienna, Austria, 2025, pp. 16682–16699. URL: https://aclanthology.org/2025.findings-acl.856/. doi:10.18653/v1/2025.findings-acl.856.

[8] T. Xu, H. Zheng, C. Li, H. Chen, Y. Liu, R. Chen, L. Sun, Noderag: Structuring graph-based rag with heterogeneous nodes, arXiv preprint arXiv:2504.11544 (2025). URL: https://arxiv.org/abs/2504.11544. doi:10.48550/arXiv.2504.11544, submitted 15 Apr 2025.

[9] Y. Su, Y. Fang, Y. Zhou, Q. Xu, C. Yang, Clue-rag: Towards accurate and cost-efficient graph-based rag via multi-partite graph and query-driven iterative retrieval, arXiv preprint arXiv:2507.08445 (2025). URL: https://arxiv.org/abs/2507.08445. doi:10.48550/arXiv.2507.08445, versions v1-v3.

[10] S. Yu, M. Cheng, J. Yang, J. Ouyang, Y. Luo, C. Lei, Q. Liu, E. Chen, Multi-source knowledge pruning for retrieval-augmented generation: A benchmark and empirical study, arXiv preprint arXiv:2409.13694 (2024). URL: https://arxiv.org/abs/2409.13694.

[11] Q. Zhao, R. Wang, X. Wang, D. Zha, N. Mu, Prefrag: Preference-driven multi-source retrieval augmented generation, arXiv preprint arXiv:2411.00689 (2024). URL: https://arxiv.org/abs/2411.00689.

[12] X. Yin, X. Li, Z. Zhu, W. Wang, J. Zhou, H. Guan, Tri-modal motion retrieval by learning a joint embedding space, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 1234–1243. URL: https://arxiv.org/pdf/2403.00691.

[13] D. Wu, Y. Yan, Z. Liu, Z. Liu, M. Sun, Kg-infused rag: Augmenting corpus-based rag with external knowledge graphs, arXiv preprint arXiv:2506.09542 (2025). URL: https://arxiv.org/abs/2506.09542.

[14] L. Wang, Y. Li, O. Aslan, O. Vinyals, WikiGraphs: A Wikipedia text - knowledge graph paired dataset, in: Proceedings of the Fifteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-15), 2021, pp. 67–82. URL: https://arxiv.org/pdf/2107.09556. doi:10.18653/v1/2021.textgraphs-1.7.