MAD-RAG: Multi-agent GraphRAG with Dual Pipelines for Structural Information Retrieval

Anonymous ACL submission

Abstract

The exponential growth of structured databases across enterprise systems has created an increasing demand for intelligent Information Retrieval (IR) frameworks. Traditional Retrieval-Augmented Generation (RAG) methods perform well in unstructured text but struggle to capture relational information in structured data sources. GraphRAG-based solutions fill this gap, yet overlook implicit relationships and require graph reconstruction upon database updates. Recent agentic RAG approaches have advanced IR through multi-step reasoning and coordinated query handling. Building on these strengths, we propose MAD-RAG – a **M**ulti-**A**gent **D**ual-pipeline framework for **RAG** that constructs semantic and structural graph representations separately from the database and coordinates specialized agents to ensure structural facts correctness while maintaining semantic comprehensiveness. By employing this dual-retrieval architecture, MAD-RAG surpasses 5 RAG baselines on structural information retrieval tasks by approximately 50% across 9 selected evaluation metrics. Extensive evaluation on 5 frontier models generalizes its consistent performance across diverse model capabilities and computational budgets.

1 Introduction

The rapid integration of AI across industries has led to an unprecedented surge in data volume and complexity. Modern enterprises must manage large amounts of information across knowledge bases, code repositories, and transactional systems (Raptis et al., 2019). While these data assets hold great potential for informed decision-making, their intricate structures pose significant challenges for traditional RAG systems (Arslan et al., 2024).

Existing GraphRAG-based approaches address these issues by explicitly modeling entity relationships (Peng et al., 2025). However, their rigid topological structures often fail to identify implicit or context-dependent relationships within the database (Peng et al., 2025), and most methods are typically static, requiring full graph reconstruction whenever the underlying data changes. Furthermore, as the graph database expands, query latency increases proportionally, making them less applicable for large-scale dynamic environments.

To address these limitations, we leverage LLM-based agents and propose MAD-RAG, a multi-agent dual-pipeline framework for structural information retrieval, transitioning the static retrieval procedure into a reasoning-driven agentic loop. As illustrated in Figure 1, MAD-RAG separates semantic retrieval and structural verification into two dedicated pipelines, and constructs dual knowledge graphs accordingly, where a semantic graph encodes contextual relationships and a structural graph captures factual dependencies. Specialized agents decompose complex queries into executable sub-tasks, retrieve information from both graphs through their respective pipelines, and aggregate cross-pipeline information for more accurate and coherent responses. Meanwhile, the framework supports incremental graph updates, improving the retrieval efficiency and mitigating computation overhead in dynamic settings. The main contribution of this work can be summarized as follows:

(i) We propose MAD-RAG – an agentic RAG system that constructs dual knowledge graphs to capture both structural and semantic information, each paired with a specialized multi-agent pipeline to retrieve information.

(ii) MAD-RAG routes user queries to the pipelines based on their types, and provides an enhancement mode to chain both pipelines, leveraging structural facts effectively alongside the semantic context.

(iii) We compare MAD-RAG with 5 representative baselines spanning RAG, GraphRAG, and

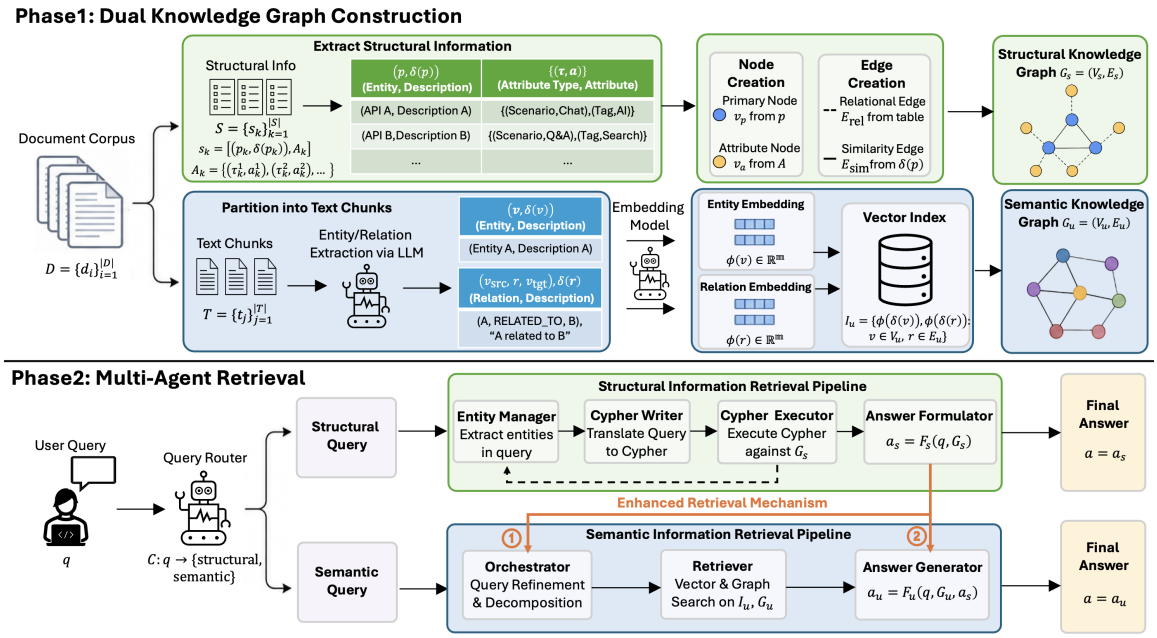

Figure 1: Overview of the MAD-RAG framework.

agentic RAG approaches, achieving approximately 50% improvement in structural query-answering tasks across 9 evaluation metrics for both generated content quality and retrieval effectiveness, while maintaining on-par performance on general semantic queries.

(iv) We evaluate MAD-RAG on 5 frontier LLMs to study and verify its effectiveness across diverse model capabilities and computational budgets. Detailed analysis of resource consumption has also been provided.

2 Related Work

RAG enhances LLMs by dynamically retrieving relevant information from external knowledge bases (Lewis et al., 2020), widely used in domains with frequent information changes to reduce hallucinations and improve accuracy. Building on this, HyDE (Gao et al., 2023) advances dense retrieval by learning from automatically constructed training examples rather than human-annotated query-document pairs. Many prior studies (Chan et al., 2024; Ma et al., 2023; Mao et al., 2024) improve retrieval performance by rewriting and refining user queries to better handle complex or ambiguous questions, while others explore re-ranking retrieved factual information in context (Sun et al., 2026; Tang et al., 2024, 2025).

GraphRAG enhances retrieval by leveraging knowledge graph structures (Edge et al., 2024). Follow-up work, such as LightRAG (Guo et al., 2025), employs a dual-level retrieval strategy over graph representations to enhance both contextual reasoning and retrieval efficiency. KET-RAG (Huang et al., 2025) constructs a knowledge graph skeleton from key chunks and another bipartite graph from all chunks, reducing indexing costs while maintaining retrieval quality. More recent works have advanced GraphRAG from different perspectives (Hu et al., 2025; Sarmah et al., 2024; Luo et al., 2026).

Agentic RAG introduces autonomous agents to enhance RAG, which iteratively plan, retrieve, and synthesize information resources (Singh et al., 2025). Given a user query, the agentic system usually decomposes retrieval tasks, decides proper strategies, and queries multiple knowledge sources, such as text corpora, graphs, or APIs. Based on the retrieved evidence, the agent aggregates all relevant information for final answer generation. Well-known examples include A-RAG (Du et al., 2026) and Search-o1 (Li et al., 2025).

3 Preliminaries

To elaborate on our MAD-RAG approach better, we formalize the process of information retrieval with pre-defined notations in Table 1. Given a document corpus $D = \{d_i\}_{i=1}^{|D|}$, traditional RAG partitions each document d_i into smaller text chunks, forming a chunk set $T = \{t_j\}_{j=1}^{|T|}$. Each chunk t_j, together with the incoming user query q, are

Notation	Description
D, d_i	Document corpus; individual document
T, t_j	Text chunk set; each chunk
$\phi(\cdot)$	Text embedding function, $\phi(t) \in \mathbb{R}^m$
q	Queries from users
C	LLM-based query router
$G = (V, E)$	Knowledge graph G with vertices V and edges E
$G_s = (V_s, E_s)$	Structural knowledge graph G_s with vertices V_s and edges E_s
$G_u = (V_u, E_u)$	Semantic knowledge graph G_u with vertices V_u and edges E_u
$S, r_k = (p_k, A_k)$	Structural set S with primary entity p_k and type-attribute pairs A_k
$A_k = \{(\tau_k^1, a_k^1), ...\}$	Type-attribute pairs A_k; type labels τ_k^i with attribute values a_k^i
F_s, F_u	Answer formulator of the structural and semantic pipeline

Table 1: Frequently used notations for MAD-RAG.

mapped into dense vector representations using an embedding function $\phi(\cdot)$. The retriever then computes the semantic similarity between two embeddings $\phi(q)$ and $\phi(t_j)$, returning the top-k most relevant chunks as context for LLMs:

$$T_k(q) = \text{Top}_{t_j \in T} \ \text{sim}(q, t_j) \tag{1}$$

GraphRAG (Edge et al., 2024) replaces the flat document corpus with a knowledge graph $G = (V, E)$, where $V = \{v_i\}_{i=1}^{|V|}$ is the set of entity nodes (vertices) and $E = \{e_{ij}\}_{i,j}$ the set of edges. Similar to RAG, each node v_i is mapped into a dense vector using $\phi(\cdot)$, and the retriever computes the semantic similarity between $\phi(q)$ and $\phi(v_i)$ to retrieve the top-k most relevant nodes. For each retrieved node v_i, its neighbors $\mathcal{N}(v_i)$ and connecting edges E_i can be optionally included to form the context for generation:

$$G_k(q) = \{(v_i, \mathcal{N}(v_i), E_i) \mid v_i \in V_k(q)\} \tag{2}$$

4 MAD-RAG

Given these preliminary knowledge, MAD-RAG operates in two phases, as illustrated in Figure 1. In the **graph construction phase**, a structural knowledge graph $G_s = (V_s, E_s)$ is built from structural information to support precise fact retrieval over structured metadata, while a semantic (unstructured) knowledge graph $G_u = (V_u, E_u)$ is constructed from text chunks T to enable contextual understanding. In the **multi-agent retrieval phase**, an LLM-based router $C : q \rightarrow \{\text{“structural”}, \text{“semantic”}\}$ classifies each query and dispatches it to a specialized pipeline: 1) Structural queries are answered via Cypher queries against G_s; 2) Semantic queries undergo vector-and-graph retrieval over G_u. An enhanced retrieval mechanism chains both pipelines when a query demands both structural precision and broader contextual understanding.

4.1 Structural Information

In real-world environments, documents often carry implicit structural information beyond unstructured text. For instance, API catalogs organize information using structured fields such as domain, tags, and usage scenarios, while biographical articles contain categorical attributes like nationality, occupation, and affiliation. We formalize such structural information as a set $S = \{s_k\}$, where each record $s_k = [(p_k, \delta(p_k)), A_k]$ comprises a primary entity name p_k with its textual description $\delta(p_k)$, and a set of type-attribute pairs $A_k = \{(\tau_k^1, a_k^1), (\tau_k^2, a_k^2), \ldots\}$, in which τ_k^i represents a type label and a_k^i its corresponding attribute value. Figure 2 illustrates the structural information of an API catalog, where the primary entity name p_k is labeled in blue and its description $\delta(p_k)$ in purple. For each type-attribute pair A_k in gray, type labels τ_k^i are tags, product lines, vertical domains, while the corresponding values a_k^i include eMobility, Sustainability, etc.

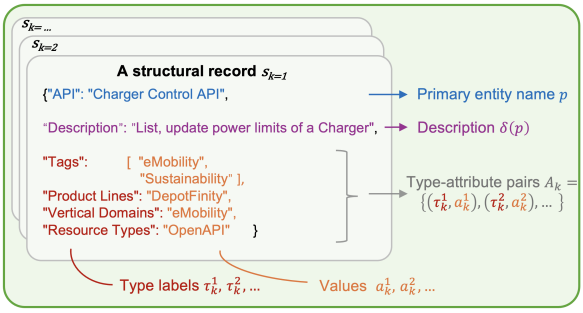

Figure 2: Structural API information with notations.

4.2 Dual Knowledge Graph Construction

4.2.1 Structural Knowledge Graph

The structural knowledge graph $G_s = (V_s, E_s)$ supports precise querying of type-attribute pairs and entity relationships. As shown in Figure 3, it is built from the structural record set S via an LLM-free two-step process. **1) Nodes Creation**:

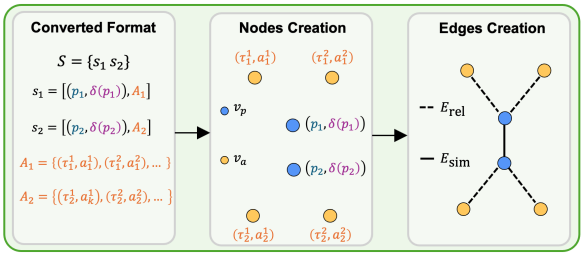

Figure 3: Structural graph construction.

For each structural record $s_k = [(p_k, \delta(p_k)), A_k]$, a primary node v_p is created from the entity name p_k with its description $\delta(p_k)$. An attribute node v_a is created for each $(\tau_k^i, a_k^i) \in A_k$. Notably, duplicate attribute nodes are avoided, ensuring that two primary nodes that share the same attribute node are indirectly connected through this shared node. **2) Edge Creation**: Two types of edges are constructed. Relational edges E_{rel} link each primary node to its attribute nodes:

$$E_{\text{rel}} = \bigcup_{s_k \in S} \{(v_p, v_a) \mid (\tau_k^i, a_k^i) \in A_k,\ v_a = \text{node}(\tau_k^i, a_k^i)\} \quad (3)$$

Similarity edges E_{sim} connect primary nodes whose description embeddings exceed a cosine similarity threshold θ:

$$E_{\text{sim}} = \{(v_i, v_j, w_{ij}) \mid w_{ij} = \cos(\phi(\delta(v_i)),\ \phi(\delta(v_j))) \geq \theta\} \quad (4)$$

where $\theta = 0.85$ by default. The complete structural edge set is $E_s = E_{\text{rel}} \cup E_{\text{sim}}$. Notably, the graph supports incremental updates: new records are merged into existing nodes by name, and similarity edges are recomputed only for affected nodes.

4.2.2 Semantic Knowledge Graph

The semantic knowledge graph $G_u = (V_u, E_u)$ captures deep semantic and contextual relationships in textual content. For each text chunk $t_j \in T$, an LLM extracts entities and relation triplets of the form $(v_{\text{src}},\ r,\ v_{\text{tgt}})$, where $v_{\text{src}}, v_{\text{tgt}} \in V_u$ are entity nodes and $r \in E_u$ the relation edge in-between. Each entity v and each relation r carries a textual description $\delta(\cdot)$ summarized by the LLM from t_j. Entity descriptions and relation descriptions are embedded via $\phi(\cdot)$ and indexed as follows:

$$I_u = \{\phi(\delta(v)),\ \phi(\delta(r)) : v \in V_u,\ r \in E_u\} \quad (5)$$

Entities that share the same name across different chunks are merged into a single node, with their descriptions aggregated and re-summarized. This process enables cross-document reasoning. Similar to the structural graph, new documents extend G_u incrementally without full graph reconstruction.

4.3 Multi-Agent Retrieval Architecture

4.3.1 Query Routing

In the multi-agent retrieval phase of Figure 1, the router C is an LLM-based agent that classifies each incoming query based on the optimal retrieval strategy. Structural queries like "Which APIs support the Chat scenario?", are answered by traversing over G_s, which covers operations such as entity filtering, attribute lookups, and similarity rankings. In comparison, semantic queries, e.g. "How does API A differ from API B in terms of design philosophy?", require understanding the source document text, perform multi-hop reasoning to generate explanations, and support comparative analysis.

4.3.2 Structural Retrieval Pipeline

If the router C classifies a query as structural, the retrieval process follows a four-agent pipeline that operates sequentially: **Entity Manager** extracts key entities from the natural-language query and maps each to a verified node in G_s, ensuring all downstream operations are grounded in existing entities of G_s. **Cypher Writer** translates the query into formal Cypher statements to retrieve relevant nodes V_s and edges E_s from graph databases, such as Neo4j. **Cypher Executor** executes the generated Cypher queries on G_s, and returns any errors to the Cypher Writer, forming a closed-loop cycle for automatic query correction. **Answer Formulator** synthesizes the retrieved entities, relations, and their associated descriptions into a human-readable answer $a_s = F_s(q, G_s)$.

4.3.3 Semantic Retrieval Pipeline

If the router C classifies a query as semantic, another three-agent pipeline will be triggered: **Orchestrator** breaks down the original query into properly structured sub-queries by resolving pronouns, expanding abbreviations, and splitting multi-part questions into separate retrieval targets. **Retriever** extracts key entities k_e and topics k_t from each sub-query using an LLM. Key entities are matched against node descriptions in G_u via embedding similarity $\cos(\phi(k_e), \phi(\delta(v)))$, while topics are matched against relation descriptions via $\cos(\phi(k_t), \phi(\delta(r)))$. Matched nodes and relations are further expanded by traversing their neighbors in G_u to enrich the retrieved context. This process accumulates across all sub-queries, with duplicates removed to form the final context set. **Answer Generator** synthesizes the final answer $a_u = F_u(q, \text{context})$ from the retrieved context.

4.3.4 Cross-Pipeline Enhancement

MAD-RAG provides an enhancement mode to chain both pipelines, combining structural facts with semantic context. The enhancement is realized through two complementary mechanisms:
1) Query conditioning: The structural answer $a_s = F_s(q, G_s)$ is forwarded to the Orchestra-

tor, decomposed into sub-queries of the semantic pipeline. Consequently, the sub-queries contain verified structural facts beyond the raw query, leading to more focused retrieval against the indexed embeddings I_u.

2) Answer grounding: To preserve verified facts during multi-step generation, a_s is directly supplied to the Answer Generator alongside the semantic context set. The final answer $a = F_u(q, G_u, a_s)$ inherits structural precision from a_s while gaining semantic richness from G_u.

5 Experimental Setup

All experiments are conducted on a single Apple M4 Pro machine equipped with a 12-core CPU, 16-core GPU (Metal 4), and 24 GB unified RAM shared across CPU and GPU. The software environment consists of Python 3.12.13 and PyTorch 2.12.0 running on macOS 26.3 (Build 25D125). LLMs and embedding models are accessed via our organization-internal API gateways.

5.1 Datasets

We include three datasets with complementary features and domains: MultiHop-RAG, MuSiQue, and a new API-catalog dataset curated by us. We process the original questions from these datasets, categorizing them into structural or semantic queries to investigate different retrieval patterns.

1) MultiHop-RAG (Tang and Yang, 2024) is designed for multi-hop cross-document reasoning, consisting of news articles paired with structured metadata like title, author, source, and publication date. This dataset can be used to evaluate MAD-RAG's ability to answer queries that require both filtering of structured data (e.g., finding articles by a specific author) and understanding semantic content (e.g., analyzing authors' writing style).

2) MuSiQue (Trivedi et al., 2022) provides a different evaluation context. Each sample contains 20 candidate paragraphs, of which only 2 support the answer. This sparse retrieval setting makes it primarily suitable for benchmarking MAD-RAG's semantic retrieval performance. While the original dataset lacks metadata, we manually build structural queries from existing QA pairs through counting, aggregation, and metadata filtering.

3) API catalog is curated based on a publicly accessible developer portal (to portal). The dataset contains 138 API entries, spanning 32 product lines and 13 domains. Each API entry contains structured fields, such as product line, domain, resource type, and tags, as well as text descriptions covering product overviews and API documentation. We create this dataset to reflect real-world engineering scenarios, where precise retrieval from heterogeneous technical documentation is required.

5.2 Metrics

We select an extensive set of evaluation metrics: 1) LLM-as-a-Judge to assess the generated content quality; 2) RAGAs to measure the effectiveness of retrieval methods; 3) human evaluation as a reference; 4) and efficiency metrics to estimate resource consumption. To be more specific:

1) LLM-as-a-Judge (Gu et al., 2026) employs prompt-based evaluation, where LLMs assess the generated answers on a 1-10 scale across four critical dimensions: **Comprehensiveness** quantifies how well an answer addresses all aspects of the given query; **Diversity** measures the richness and range of perspectives covered within the answer; **Empowerment** reflects how the answer supports in reader's understanding and decision-making; **Overall** evaluates the general quality and utility of the answer. To ensure reproducible outcomes and fair comparison among frontier models, we use GPT-4.1-mini as the judge with temparature set to 0.

2) RAGAs (Es et al., 2024) is an evaluation framework for RAG systems, which assesses retrieval effectiveness on a 0-1 scale across four dimensions: **Faithfulness** measures whether the claims in an answer are supported by the retrieved context; **Answer Relevance** quantifies how accurately the generated answers address the intent of the query; **Context Recall** reflects whether the retrieved context covers all information required to answer the question, while **Context Precision** shows whether each retrieved context chunk is useful to answer the question.

3) Human Evaluation is applied to the newly released API catalog data. Three software engineers are invited to evaluate the final answers on a 1-10 scale across three dimensions: **Readability**, **Usefulness**, and **Detail Coverage**. Each of them has at least two years of code-review experience in industry. In addition to their own experience and judgment criteria, we also provided the reference answer, the prompt and grading schema used in LLM-as-a-judge to ensure consistent evaluation.

4) Efficiency metrics are included to evaluate resource consumption, covering **Output Length**, **Token Usage**, and **Inference Time**.

5.3 Selected Language Models

We strategically select five LLMs that span different performance tiers. GPT-4.1-mini serves as an essential model for baseline comparison. We further deploy three frontier LLMs to investigate the full potential of MAD-RAG: GPT-5.2 (OpenAI, 2025) from OpenAI, Gemini-3.1-Pro (DeepMind, 2026) from Google, and Claude-Sonnet-4.6 (Anthropic, 2026) from Anthropic. The open-srouce Qwen-3.6-27B (Alibaba, 2026) is included to represent the intermediate scale, and Qwen3-Embedding-8B (Zhang et al., 2025) is used as the embedding model for data insertion and graph construction.

5.4 Baseline Selection

We compare MAD-RAG against five representative RAG baselines. HyDE (Gao et al., 2023) serves as a pure text retrieval baseline without structural information extraction, establishing a lower bound for structure-aware methods. Beyond this text-only method, we incorporate three graph-based variants to exploit structural information: GraphRAG (Edge et al., 2024), the foundational work from Microsoft, derives implicit structure from document text; LightRAG (Guo et al., 2025) balances efficiency and retrieval quality through dual-level indexing with incremental document insertion; and KET-RAG (Huang et al., 2025) combines graph skeletons with bipartite graphs for cost-effective indexing. Furthermore, we include one newly-proposed agentic approach A-RAG (Du et al., 2026) for comparison. Search-o1 is excluded, considering that the online search feature is not suitable for confidential data like API catalogs. All baselines are evaluated on both structural and semantic queries across the selected datasets, metrics, and LLMs in Section 6.

6 Experiments

In this session, we evaluate the MAD-RAG framework from four distinct perspectives, each guided by a specific research question:

RQ-1 (Performance): How does MAD-RAG perform compared to existing RAG baselines when handling structural and semantic queries?

RQ-2 (Efficiency): Given such superior performance, how efficient is MAD-RAG in terms of retrieval latency and token consumption compared to other RAG baselines?

RQ-3 (Ablation): What is the contribution of each retrieval pipeline to overall performance, and how does the cross-pipeline enhancement balance the final answer?

RQ-4 (Model Consistency): How does MAD-RAG's performance vary across different LLM backends, and can we generalize the conclusion?

6.1 SOTA Comparison with Baselines (RQ-1)

Table 2 compares MAD-RAG against five baseline approaches on three datasets, with their average scores weighted by the dataset size. For structural queries, MAD-RAG achieves dominant performance, surpassing all baselines in weighted average by over 50% across almost all metrics. Examining individual datasets, MAD-RAG excels in MultiHop-RAG dataset, significantly outperforming other approaches in retrieving implicit structural information. On MuSiQue dataset, despite the lack of structural metadata in corpus, MAD-RAG still secures the second place after LightRAG while surpassing the others. And human expert evaluation on API catalog dataset further confirms MAD-RAG's effectiveness. To explain the performance gain explicitly, we provide an ablation study for each pipeline component in Section 6.3.

While MAD-RAG is primarily designed for structural retrieval, it remains competitive for semantic queries, achieving the best weighted average on all metrics of LLM-as-Judge as well as context precision from RAGAs. GraphRAG seems to perform better on API-catalog and MultiHop-RAG datasets, but these overrated scores are attributed to its verbose responses from exhaustive community-level retrieval, for which we further clarify in Section 6.2 and Appendix H.

6.2 Resource Consumption (RQ-2)

We study the efficiency of MAD-RAG considering three different aspects. As illustrated in Figure 4, MAD-RAG produces output with a moderate length. In comparison, GraphRAG and lightRAG tend to generate redundant answers by involving as much retrieved information as possible. This pattern explains their higher diversity and faithfulness scores in Table 2, particularly when handling semantic queries that require thorough analysis. With respect to token usage, MAD-RAG and A-RAG are more expensive, an intuitive outcome given their agent-based workflows and iterative processes. However, the increased token usage does not lead to an exponential growth in inference time. Notably, even in its dual-pipeline mode, MAD-RAG responses faster than GraphRAG and A-RAG for

Query Type	Dataset	Approach	LLM-as-a-Judge (content quality)				RAGAs (retrieval effectiveness)				Human
			Comprehensiveness	Diversity	Empowerment	Overall	Faithfulness	Answer Relevance	Context Recall	Context Precision	
Structural Queries	**Weighted average across datasets**	**MAD-RAG**	**8.00**	**6.86**	**8.31**	**7.33**	**0.9860**	**0.6635**	**0.7059**	**0.6107**	-
		KET-RAG	3.24	2.82	5.15	3.10	0.6002	0.4000	0.4320	0.3444	-
		LightRAG	5.55	5.51	7.28	5.53	0.8727	0.5302	0.4278	0.1387	-
		GraphRAG	3.71	5.20	6.37	3.89	0.7393	0.5069	0.1858	0.1131	-
		HyDE	2.13	3.09	4.62	2.38	0.9307	0.2335	0.0268	0.0288	-
		A-RAG	3.28	4.15	6.62	4.12	0.8402	0.5123	0.2468	0.3013	-
	Structural-API	**MAD-RAG**	**8.72**	7.30	**9.05**	**7.87**	**1.0000**	**0.8799**	**0.8511**	**0.5612**	**8.63**
		KET-RAG	3.43	3.28	7.02	3.45	0.5168	0.7167	0.7012	0.5343	4.52
		LightRAG	6.45	6.27	7.85	6.30	0.7383	0.7357	0.4335	0.0687	6.30
		GraphRAG	5.15	**7.58**	8.67	5.53	0.9868	0.6731	0.2833	0.0323	5.11
		HyDE	2.57	4.88	6.33	3.02	0.9554	0.3162	0.0408	0.0210	2.39
		A-RAG	2.85	4.13	6.55	4.07	0.9238	0.7569	0.2667	0.4882	6.58
	MultiHop-RAG	**MAD-RAG**	**8.70**	**6.73**	**8.30**	**8.00**	**0.9916**	**0.6631**	**0.8333**	**0.5704**	-
		KET-RAG	1.28	1.30	2.70	1.32	0.8667	0.1504	0.0128	0.2833	-
		LightRAG	2.05	3.27	5.43	2.45	0.9632	0.2740	0.1352	0.0525	-
		GraphRAG	2.13	4.15	4.73	2.43	0.8810	0.5458	0.0426	0.1541	-
		HyDE	1.68	2.52	4.57	2.03	0.9843	0.1843	0.0112	0.0521	-
		A-RAG	2.15	3.58	6.28	3.42	0.6654	0.3422	0.0571	0.1326	-
	MuSiQue	**MAD-RAG**	6.57	6.55	7.58	6.13	**0.9662**	0.4476	0.4347	**0.7006**	-
		KET-RAG	5.00	3.87	5.72	4.52	0.4171	0.3330	0.5832	0.2167	-
		LightRAG	**8.15**	**7.00**	**8.57**	**7.85**	0.9167	**0.5809**	**0.7167**	0.2970	
		GraphRAG	3.85	3.88	5.70	3.72	0.3498	0.3017	0.2315	0.1519	-
		HyDE	2.13	1.87	2.97	2.08	0.8524	0.2001	0.0284	0.0132	-
		A-RAG	4.85	4.73	7.03	4.87	0.9313	0.4379	0.4167	0.2833	-
Semantic Queries	**Weighted average across datasets**	**MAD-RAG**	**8.20**	**6.96**	**8.37**	**7.65**	0.9742	0.6577	0.6384	**0.8500**	-
		KET-RAG	6.55	5.43	7.57	6.16	0.8280	0.4575	0.6959	0.4014	-
		LightRAG	7.44	6.73	8.28	7.45	**0.9880**	0.5522	**0.7275**	0.3774	-
		GraphRAG	6.74	6.50	7.35	6.56	0.8063	0.5714	0.4248	0.1524	-
		HyDE	4.60	3.68	4.99	4.29	0.8303	0.1409	0.1893	0.1090	-
		A-RAG	7.60	6.60	8.24	7.25	0.8560	**0.6670**	0.5434	0.4295	-
	Structural-API	**MAD-RAG**	8.58	7.45	**9.03**	8.06	**1.0000**	**0.8869**	0.7738	**0.8502**	**8.52**
		KET-RAG	6.44	5.68	8.53	6.42	**1.0000**	0.6891	**0.9513**	0.2696	6.96
		LightRAG	8.94	7.89	8.97	8.45	0.9957	0.8227	0.9189	0.3305	7.34
		GraphRAG	**9.12**	**8.87**	8.93	**8.98**	0.9872	0.7849	0.5071	0.2527	6.45
		HyDE	5.17	3.85	5.28	4.59	0.8680	0.1165	0.1419	0.1348	2.17
		A-RAG	7.63	6.76	8.46	7.34	**1.0000**	0.8503	0.6350	0.5196	8.42
	MultiHop-RAG	**MAD-RAG**	8.45	7.00	8.28	7.88	0.9689	0.6066	**0.5692**	**0.8450**	-
		KET-RAG	6.41	5.45	7.12	5.83	0.5619	0.2662	0.4286	0.7131	-
		LightRAG	6.09	6.07	7.86	6.75	**1.0000**	0.4764	0.4233	0.2904	-
		GraphRAG	**8.72**	**7.58**	**8.83**	**8.15**	**1.0000**	**0.6747**	0.5146	0.0623	-
		HyDE	6.15	5.03	6.42	5.85	0.7752	0.1139	0.2829	0.1417	-
		A-RAG	7.63	6.72	7.84	7.28	0.7384	0.4945	0.5650	0.3411	-
	MuSiQue	**MAD-RAG**	**7.58**	**6.42**	7.79	7.00	0.9536	0.4797	0.5722	**0.8552**	-
		KET-RAG	6.81	5.16	7.07	6.22	0.9221	0.4173	0.7079	0.2216	-
		LightRAG	7.28	6.23	8.01	**7.15**	**0.9683**	0.3574	**0.8403**	0.5114	-
		GraphRAG	2.37	3.04	4.28	2.56	0.4317	0.2548	0.2528	0.1423	-
		HyDE	2.47	2.14	3.28	2.42	0.8478	0.1923	0.1431	0.0506	-
		A-RAG	7.54	6.33	**8.43**	7.12	0.8295	**0.6561**	0.4301	0.4277	-

Table 2: Comparison of MAD-RAG against baselines across three datasets under structural and semantic queries.

both semantic and structural queries. This observation can be attributed to our optimized agent coordination and GraphRAG's comparatively slower community traversal. Two lightweight GraphRAG variants lead the timing performance.

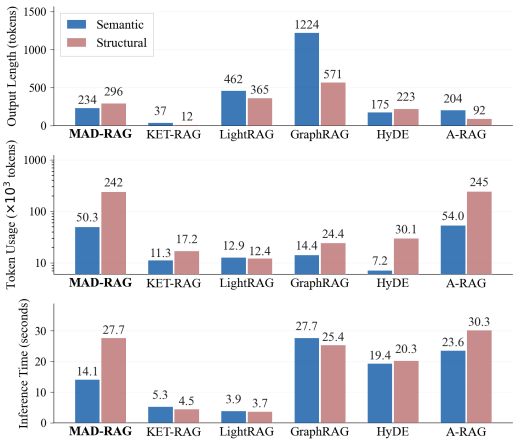

Figure 4: Resource consumption of MAD-RAG and baselines per query, weighted averaged over 3 datasets.

6.3 Ablation on Individual Pipeline (RQ-3)

We split the "full" mode across two pipelines into "structural-only" and "semantic-only" modes to analyze each pipeline's contribution in Table 3. For structural queries, the structural-only mode significantly outperforms semantic-only mode on the API catalog and MultiHop-RAG datasets, indicating that Cypher-based graph traversal addresses the factual precision that embedding-based search cannot solve. However, due to the lack of metadata, the structural pipeline under-performs on MuSiQue dataset. For semantic queries, while semantic-only mode outperforms the structural-only mode across all datasets, the full mode further improves upon it, suggesting that structural retrieval provides valuable factual grounding that complements semantic context.

6.4 Model Consistency across Different LLM Backends (RQ-4)

Figure 5 reveals the evaluation outcomes across all models. For semantic queries, Claude-Sonnet-4.6

Query Type	Dataset	Approach	LLM-as-a-Judge (content quality)				RAGAs (retrieval effectiveness)			
			Comprehensiveness	Diversity	Empowerment	Overall	Faithfulness	Answer Relevance	Context Recall	Context Precision
Structural	Structural-API	Struct. only	**8.87**	6.32	8.68	**8.02**	**1.0000**	**0.9637**	**0.8542**	**0.8612**
		Sem. only	5.43	5.13	7.62	5.27	**1.0000**	0.7525	0.4332	0.4163
		Full	8.72	**7.30**	**9.05**	7.87	**1.0000**	0.8799	0.8511	0.5612
	MultiHop-RAG	Struct. only	**9.00**	**6.85**	**9.43**	**8.15**	**1.0000**	**0.9315**	0.8242	**0.9347**
		Sem. only	2.28	3.15	5.30	2.27	0.8929	0.3832	0.0160	0.1289
		Full	8.70	6.73	8.30	8.00	0.9916	0.6631	**0.8333**	0.5704
	MuSiQue	Struct. only	1.00	1.04	2.07	1.08	0.7595	0.0000	0.0000	0.0000
		Sem. only	**7.13**	5.43	7.00	**6.72**	**1.0000**	**0.5709**	**0.7165**	**0.7284**
		Full	6.57	**6.55**	**7.58**	6.13	0.9667	0.4476	0.4347	0.7006
Semantic	Structural-API	Struct. only	4.11	3.46	7.04	4.32	0.6000	0.4819	0.0738	0.1385
		Sem. only	8.45	7.26	**9.12**	**8.17**	0.9615	0.8879	0.7504	**1.0000**
		Full	**8.58**	**7.45**	9.03	8.00	**1.0000**	**0.8888**	**0.7738**	0.8502
	MultiHop-RAG	Struct. only	7.18	5.68	6.83	6.54	0.5835	**0.6121**	0.2812	0.4318
		Sem. only	7.57	6.61	8.09	7.11	**1.0000**	0.4194	0.4255	0.5680
		Full	**8.45**	**7.00**	**8.28**	**7.88**	0.9683	0.6066	**0.5692**	**0.8450**
	MuSiQue	Struct. only	2.53	2.24	4.33	3.17	0.5952	0.0614	0.1462	0.1350
		Sem. only	7.39	5.91	7.46	6.82	0.8690	0.4582	**0.7185**	0.7106
		Full	**7.58**	**6.42**	**7.79**	**7.00**	**0.9500**	**0.4797**	0.5722	**0.8552**

Table 3: Ablation study on the structural-only, semantic-only, and full modes with cross-pipeline enhancement.

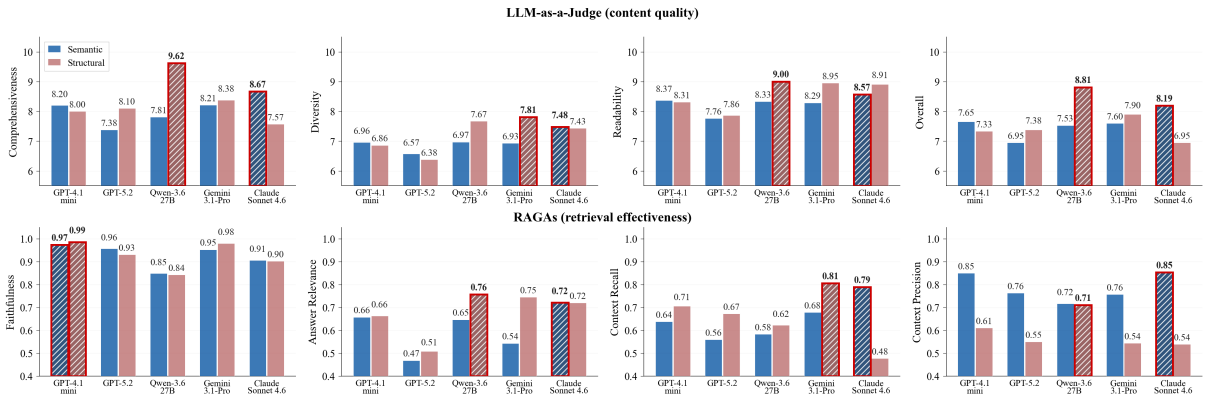

Figure 5: MAD-RAG performance with different LLMs, weighted averaged over 3 datasets.

consistently exhibits dominant performance across most metrics, highlighting its strength in understanding semantic-based prompts. In contrast, for structural queries, Qwen3.6-27B outperforms the three top-tier models in 5 out of 8 metrics. This superior performance might be attributed to the use of another Qwen embedding model for vector representations in structural graph construction, suggesting that a shared embedding space can be beneficial. Notably, GPT-4.1-mini surpasses its larger counterpart GPT-5.2 across all metrics. This finding challenges the common assumption that the latest or larger models inherently lead to better performance. More importantly, it shows that MAD-RAG's effectiveness is robust, performing well across models of varying sizes.

7 Conclusions and Future Work

Facilitated by the dual graph bases and multi-agent pipelines, MAD-RAG achieves substantial improvement over existing baselines in structural information retrieval while preserving the superior capability to understand semantic context comprehensively. It delivers stable performance across LLMs of all sizes, scalable for various resource-constrained environments in industry. Compared with standard GraphRAG and A-RAG, MAD-RAG requires lower inference latency, yet agent orchestration – essential for all agentic RAG approaches, results in processing overhead, making it slower than lightweight GraphRAG variants. Despite this trade-off, MAD-RAG demonstrates that structural metadata-augmented agentic retrieval can effectively balance factual fidelity and contextual richness in complex information retrieval tasks.

As for future work, we encourage exploring alternative agentic RAG architectures, incorporating specialized agents to further enhance the performance while optimizing the resource consumption. Moreover, as LLMs continue to evolve, benchmarking new models within this agentic framework will provide valuable insights. Beyond the API-catalog dataset, the research community also requires more structured, domain-specific benchmarks to advance the field.

Limitations

Despite thorough investigation, we recognize several limitations in this study.

Due to hardware constraints and limited access to proprietary model APIs, our work was confined to the selected models. Expanding the experiments to other prominent LLMs, such as the latest GPT-5.5, Claude-Opus-4.6, and Gemini-3.5 from Google, would further validate our conclusions.

We do not explicitly control dataset heterogeneity, such as metadata structure complexity, document length, and domain terminology, which may influence LLMs and evaluation metrics. Follow-up work are motivated to stratify results by metadata complexity.

Considering the domain expertise of all participants, we only deploy human evaluation on the API catalog dataset in software engineering domain to ensure the evaluation quality. The remaining datasets include domain knowledge beyond our team's expertise. Therefore, no human assessment has been conducted.

The current framework does not support multimodal content, e.g., images, media, which may limit scalability across different types of data. Extracting structural information from multimodal data would be a fresh topic to explore.

Potential Risks

Despite RAG's hallucination-reducing design, there remains a slight chance that LLMs may still generate inaccurate responses when retrieved context is incomplete, ambiguous, or contradictory.

External knowledge bases may contain sensitive or proprietary information. Integrating such sources into an agentic RAG framework like MAD-RAG raises concerns around unauthorized data exposure and compliance with data protection regulations, e.g. GDPR. Retrieved information may reflect biases present in the knowledge base, which could propagate into generated responses.

Ethical Considerations

All datasets used in this work are publicly available and utilized in accordance with their respective licenses. The datasets contain no personal data, ensuring full compliance with data privacy regulations.

Acknowledgments

Generative AI tools were used solely to improve the writing quality of this paper, not to generate new content, and are not considered authors of this work.

References

Alibaba. 2026. Qwen3.6-Plus: Towards real world agents.

Anthropic. 2026. Introducing Claude Sonnet 4.6.

Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. 2024. A survey on RAG with llms. *Procedia computer science*, 246:3781–3790.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. RQ-RAG: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Google DeepMind. 2026. Gemini 3.1 pro.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Shaohan Wang, Pengyu Wang, Xiaorui Wang, and Zhendong Mao. 2026. A-RAG: Scaling agentic retrieval-augmented generation via hierarchical retrieval interfaces. *arXiv preprint arXiv:2602.03442*.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, St. Julians, Malta. Association for Computational Linguistics.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada. Association for Computational Linguistics.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. 2026. A survey on LLM-as-a-Judge. *The Innovation*, page 101253.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*,

pages 10746–10761, Suzhou, China. Association for Computational Linguistics.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2025. GRAG: Graph retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4145–4157, Albuquerque, New Mexico. Association for Computational Linguistics.

Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. KET-RAG: A cost-efficient multi-granular indexing framework for Graph-RAG. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 1003–1012, New York, NY, USA. Association for Computing Machinery.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438, Suzhou, China. Association for Computational Linguistics.

Haoran Luo, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and 1 others. 2026. HyperGraphRAG: Retrieval-augmented generation via hypergraph-structured knowledge representation. *Advances in Neural Information Processing Systems*, 38:152206–152234.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

Shengyu Mao, Yong Jiang, Boli Chen, Xiao Li, Peng Wang, Xinyu Wang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2024. RaFe: Ranking feedback improves query rewriting for RAG. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 884–901, Miami, Florida, USA. Association for Computational Linguistics.

OpenAI. 2025. Introducing GPT-5.2.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2025. Graph retrieval-augmented generation: A survey. *ACM Trans. Inf. Syst.*, 44(2).

Theofanis P Raptis, Andrea Passarella, and Marco Conti. 2019. Data management in industry 4.0: State of the art and open challenges. *Ieee Access*, 7:97052–97093.

Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. 2024. HybridRAG: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, page 608–616, New York, NY, USA. Association for Computing Machinery.

Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*.

Jiashuo Sun, Xianrui Zhong, Sizhe Zhou, and Jiawei Han. 2026. DynamicRAG: Leveraging outputs of large language model as feedback for dynamic reranking in retrieval-augmented generation. *Advances in Neural Information Processing Systems*, 38:168661–168692.

Yixuan Tang and Yi Yang. 2024. MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*.

Yongjian Tang, Rakebul Hasan, and Thomas Runkler. 2024. FsPONER: Few-shot prompt optimization for named entity recognition in domain-specific scenarios. In *ECAI 2024*, pages 3757–3764. IOS Press.

Yongjian Tang, Doruk Tuncel, Christian Koerner, and Thomas Runkler. 2025. The few-shot dilemma: Over-prompting large language models. In *2025 3rd International Conference on Foundation and Large Language Models (FLLM)*, pages 134–141.

Link to portal. Xcelerator developer portal.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

A API Catalog Dataset Construction

We curate the API Catalog dataset from a publicly accessible developer portal (to portal), where each API entry contains a structured JSON metadata field recording the API name, description, and multi-valued attributes including tags, product line, vertical domain, and resource type, alongside unstructured API documentation.

Since the API Catalog dataset and other commonly used datasets (Tang and Yang, 2024; Trivedi et al., 2022) are all JSON-based, a unified preprocessing schema is implemented to extract plain text from each file. The plain text is used to build the semantic knowledge graph G_u following the method in Section 4.2.2, while the corresponding structured fields are parsed into structural records $s_k = [(p_k, \delta(p_k)), A_k]$, where p_k is the primary entity name, $\delta(p_k)$ its description, and A_k the attribute set, to construct G_s as described in Section 4.2.1. Table 4 reports detailed statistics of the API Catalog dataset.

Metric	Value
Product families	35
Number of APIs	138
Text documents	332
Total tokens	93,334
Structural knowledge graph nodes	239
Structural knowledge graph edges	1,704
Semantic knowledge graph nodes	4675
Semantic knowledge graph edges	6916

Table 4: API Catalog dataset statistics.

B Human Experts Evaluation on API Dataset

Human evaluation directly reflects the practical engineering value by assessing whether answers are useful in real-world scenarios.

Query Type	Approach	Readability	Usefulness	Detail Coverage	Average
Structural	**MAD-RAG**	8.72	**8.35**	**8.83**	**8.63**
	KET-RAG	**8.90**	2.18	2.48	4.52
	LightRAG	4.32	6.97	7.60	6.30
	GraphRAG	3.95	4.57	6.82	5.11
	HyDE	5.08	1.03	1.05	2.39
	A-RAG	8.23	5.67	5.83	6.58
Semantic	**MAD-RAG**	8.24	**8.35**	**8.98**	**8.52**
	KET-RAG	**9.12**	6.96	4.81	6.96
	LightRAG	5.10	8.13	8.79	7.34
	GraphRAG	4.64	6.15	8.56	6.45
	HyDE	4.28	1.24	1.00	2.17
	A-RAG	8.32	8.21	8.73	8.42

Table 5: Human evaluation on the API Catalog dataset.

In Figure 5, MAD-RAG achieves the highest average score for both structural and semantic queries across all three dimensions. Among the baselines, each approach exhibits weaknesses in different aspects: KET-RAG scores high on readability but low on usefulness and detail coverage, reflecting its tendency to produce overly brief answers that lack contextual details. GraphRAG and HyDE suffer from poor readability – HyDE frequently claims insufficient context while GraphRAG generates redundant, disorganized responses that engineers find difficult to follow. And A-RAG performs reasonably on readability but falls short on detail coverage for structural queries.

C MAD-RAG Cost Analysis

In addition to the token consumption analysis in Section 6.2, we compare the cost of each approach using GPT-4.1-mini in Figure 6.

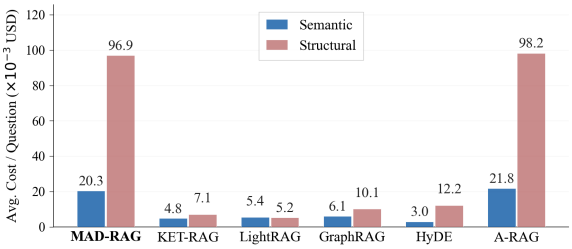

Figure 6: Average per-query cost of MAD-RAG and all baselines, weighted averaged over 3 datasets.

The cost distribution scales proportionally with the token consumption pattern in Figure 4. As expected, non-agentic baselines results in lower costs, while the agent-based frameworks – MAD-RAG and A-RAG – incur higher costs due to the multi-round coordination throughout the dual pipelines. Comparing the two agent-based approaches, MAD-RAG incurs slightly lower costs than A-RAG, spending 0.097$ per structural query. While maintaining this manageable inference cost, it demonstrates superior performance across answer correctness and retrieval precision, achieving a significantly better performance-to-cost ratio than existing agentic RAG systems (as demonstrated in Table 2 and 5). Notably, MAD-RAG is the only approach capable of handling structural queries fairly well (as evidenced in our case study in Appendix H), making this cost investment worthwhile.

Note: For detailed prompts and case studies, please refer to the appendix sections below. (page 12 - 22)

D Prompt for Knowledge Graph Construction

This section present the prompts used to extract entity and relation for semantic knowledge graph construction, which is an extension of Section 4.2.2.

Entity Extraction — System Prompt

You are an information extraction specialist. Your task is to analyze the provided text and identify structured knowledge in the form of entities and their interconnections.

Your Objective:
Parse the input text to discover notable concepts, actors, objects, and places, then map how they relate to each other. All outputs must be grounded exclusively in the given text — never infer beyond what is stated.

Entity Extraction Protocol:
• Scan the text for distinct, meaningful real-world or conceptual items.
• For each discovered item, determine:
1. name: A canonical label in title case. Use identical naming if the same entity appears multiple times.
2. type: Classify into one of {entity_types}. If no category is appropriate, assign "other".
3. description: A single-sentence summary of what this entity is or does, derived only from the text.
entity|entity_name|entity_type|entity_description

Relationship Extraction Protocol:
• Identify explicit connections between the entities you extracted above.
• If a statement links more than two entities simultaneously, break it into separate pairwise connections.
• For each connection, determine:
1. source/target_entity: The two linked entities (title case, matching your entity names exactly).
2. relationship_keywords: A few comma-separated terms capturing the nature of the link. Never use | inside this field.
3. relationship_description: One sentence explaining why these two entities are connected.
• Format each relationship as exactly 5 pipe-delimited fields on one line:
relation|source_entity|target_entity|relationship_keywords|relationship_description

Output Rules:
• Entities come first, relationships follow, ordered from most central to least.
• All relationships are bidirectional unless the text explicitly states directionality. Do not output the same pair twice.
• Write in third person throughout. Reference entities by name, never use "this paper", "the author", "we", or any pronoun standing in for an entity.
• Naming consistency is critical: if you call something "Machine Learning" in one entity, do not call it "ML" in a relationship.

Example:
Input:
Dr. Maria Chen presented her findings on neural plasticity at the 2024 Neuroscience Summit held in Geneva. Her research, conducted at the Brainwave Institute, demonstrated that targeted stimulation could accelerate recovery in stroke patients.

Entity Types:
["Person","Organization","Location","Event","Concept","Artifact"]

Output:
entity|Maria Chen|person|Maria Chen is a researcher who presented findings on neural plasticity at the 2024 Neuroscience Summit.

entity|Brainwave Institute|organization|Brainwave Institute is the research institution where Maria Chen conducted her study on targeted stimulation and stroke recovery.

entity|2024 Neuroscience Summit|event|The 2024 Neuroscience Summit is a scientific conference held in Geneva where neural plasticity research was presented.

entity|Geneva|location|Geneva is the host city of the 2024 Neuroscience Summit.

```
entity|Neural Plasticity|concept|Neural plasticity refers to the brain's ability to reorganize itself, studied by Maria Chen in the context of stroke recovery.

relation|Maria Chen|2024 Neuroscience Summit|presentation, research dissemination|Maria Chen presented her research findings at the 2024 Neuroscience Summit.

relation|Maria Chen|Brainwave Institute|affiliation, research|Maria Chen conducted her neural plasticity research at the Brainwave Institute.

relation|2024 Neuroscience Summit|Geneva|event location|The 2024 Neuroscience Summit was held in Geneva.

relation|Maria Chen|Neural Plasticity|research focus, discovery|Maria Chen's research demonstrated new findings about neural plasticity and stroke recovery.
```

Entity Extraction — User Prompt

Task:
Given the text below, perform entity and relationship extraction following the system instructions exactly.

Rules:
1. Produce only the structured extraction output, nothing else.
2. All extracted entities and relationships must be in {language}.
3. Preserve proper nouns in their original form.

Entity types to consider:
[{entity_types}]

Input:
{Input}

Output:

Note:
For detailed agent prompts,
please refer to the appendix sections on the next page.

E Prompt of Agents

In this section, we demonstrate all prompts of agents in both pipelines. The prompts utilize techniques including role assignment, task description, few shot examples, rules and constraints, and output format. Notably, dynamic loading of entity and relation from structural graph to prompt inform agent the ground truth data and equips agents with flexibility to various datasets and scenarios.

E.1 Query Router

Query Classifier — System Prompt

```
You are a query classifier for a dual-pipeline retrieval system.

Pipelines:
• STRUCTURAL: Executes Cypher queries on a Neo4j knowledge graph.
  Best for: counts, statistics, aggregations, entity listings, relationship lookups, simple factual
attribute queries.
• SEMANTIC: Retrieves and synthesizes text from indexed documents via vector search.
  Best for: detailed explanations, content understanding, capabilities/features, multi-hop text
reasoning, analysis.

---------------------------------------------------------------------------------------------
Knowledge Graph Schema (dynamically loaded from Neo4j at initialization)
The Neo4j database contains {total} entities across {N} types:
  • {label_1} ({count_1} entities): {name_1}, {name_2}, ..., {name_10}...
  • {label_2} ({count_2} entities): {name_1}, {name_2}, ..., {name_10}...
  • ...
Relationships: RELATED_TO (entity↔entity), SIMILAR_TO (similarity edges).
Each entity has 'name' and 'description' properties.
---------------------------------------------------------------------------------------------

Classification Rules (apply in priority order):
'structural' — the question:
  1. Asks for COUNTS or TOTALS ('how many X', 'number of', 'count')
  2. Asks for AGGREGATION ('which X has the most/least/fewest')
  3. Asks to LIST or FILTER entities by criteria
  4. Asks about RELATIONSHIPS between entities in the graph
  5. Asks for a SHORT FACTUAL ANSWER (a name, date, number) that can be found by traversing graph
nodes — AND the question is concise without lengthy contextual descriptions

'semantic' — the question:
  1. Asks WHAT something DOES, PROVIDES, SUPPORTS, or HOW it WORKS
  2. Asks for DETAILED DESCRIPTIONS or EXPLANATIONS of capabilities/features
  3. Requires READING and UNDERSTANDING document/article CONTENT
  4. Involves INTERPRETATION, ANALYSIS, or OPINION
  5. Asks about PROCESSES, MECHANISMS, or WORKFLOWS
  6. Describes a COMPLEX SCENARIO or NARRATIVE in the question itself (e.g. 'Which individual is
implicated in both X and Y as reported by Z') → even if the final answer is a short fact, the
REASONING requires reading and synthesizing document content → SEMANTIC

Key Rules:
• Do NOT classify as 'structural' just because an entity name appears in the query. The TYPE OF
ANSWER determines the classification, not whether entities are mentioned.
• 'What data/capabilities/features does X provide?' → SEMANTIC (needs document content even though
X is an entity)
• 'How many X exist?' / 'Which X has the most Y?' → STRUCTURAL (answerable by a graph query)
• When uncertain, prefer 'semantic'.

Output Format (strict):
  Line 1: ONE word — 'structural' or 'semantic'
  Line 2: comma-separated entity mentions (if structural); empty line (if semantic)
```

Query Classifier — User Prompt

```
Query:
{raw_question}
```

E.2 Structural Information Retrieval Pipeline

This section presents the prompts used in the structural pipeline for the Entity Manager, Cypher Writer, and Answer Formulator. The Cypher Executor requires no prompt as it directly executes the Cypher queries produced by the Cypher Writer. All prompted agents receive both a system prompt and a user prompt, with dynamic content such as entity and relation lists and the original query loaded at runtime to ensure timeliness and effectiveness. Each prompt follows a consistent design pattern: role assignment and task description are provided upfront, followed by clearly enumerated constraints and rules. For the Cypher Writer, few-shot examples are included to guide entity and relation extraction, enabling the LLM to generate correct and executable Cypher queries.

Entity Manager — System Prompt

```
You are an Entity Manager for a Neo4j knowledge graph.

Task: Map raw entity mentions from the user query to the EXACT node names below.

Schema: each node has a label (its type) and a 'name' property.
Known entities by label:

DYNAMIC: Full entity catalog
Known node names (use ONLY these exact values):
  Label '{label_1}': {name_1}, {name_2}, ..., {name_k}
  Label '{label_2}': {name_1}, {name_2}, ..., {name_m}
  ...

Output Format: a JSON object with exactly these keys: {label_1}, {label_2}, ...
Each key holds a list of matched node name strings (use EXACT values from the catalog).
Use [] if no match. Output ONLY valid JSON — no markdown fences, no explanation.
```

Entity Manager — User Prompt

```
Raw entity:
{entity_mentions}
```

Cypher Writer — System Prompt

```
You are a Neo4j Cypher expert for a label-based knowledge graph.

Task:Given the user requirements about entities and relationships, write correct and executable
Cypher Queries for the semantic graph to extract entities and relationships.

Schema:
Nodes — each node has exactly ONE label (its type). Properties:
  name      : string (unique within label)
  description : string (may be empty for attribute nodes)

DYNAMIC: Labels in this graph: {label_1}, {label_2}, ..., {label_n}

Relationships:
  RELATED_TO : connects primary nodes to attribute nodes (bidirectional)
  SIMILAR_TO : connects primary nodes to each other (bidirectional), property: similarity (float)

DYNAMIC: Full entity catalog
Known node names (use ONLY these exact values):
  Label '{label_1}': {name_1}, {name_2}, ..., {name_k}
  Label '{label_2}': {name_1}, {name_2}, ..., {name_m}
  ...
```

Query Patterns (replace <name> with exact node name, <Label> with node label):
Find a node by name:
MATCH (n:‘<Label>’ {name: ‘<name>’}) RETURN n.name AS name, n.description AS description

Find all nodes related to a given node:
MATCH (a:‘<Label>’ {name: ‘<name>’})-[:RELATED_TO]-(b) RETURN labels(b)[0] AS type, b.name AS name, b.description AS description LIMIT 25

Find nodes of a specific type related to a given node:
MATCH (a:‘<Label1>’ {name: ‘<name>’})-[:RELATED_TO]-(b:‘<Label2>’) RETURN b.name AS name, b.description AS description LIMIT 25

Find similar nodes (via SIMILAR_TO edge):
MATCH (a:‘<Label>’ {name: ‘<name>’})-[r:SIMILAR_TO]-(b) RETURN b.name AS name, b.description AS description, r.similarity AS similarity ORDER BY r.similarity DESC LIMIT 10

Find all nodes of a type:
MATCH (n:‘<Label>’) RETURN n.name AS name LIMIT 30

Find which primary nodes share a common attribute node:
MATCH (a:‘<PrimaryLabel>’)-[:RELATED_TO]->(c:‘<AttrLabel>’ {name: ‘<name>’})<-[:RELATED_TO]-(b:‘<PrimaryLabel>’) WHERE a <> b RETURN a.name AS node1, b.name AS node2, c.name AS shared LIMIT 20

Rules:
- ONLY use node names from the ‘Known node names’ list above — never invent names
- Use backtick-quoted labels: ‘Label’ (required if label contains special chars)
- Output ONE Cypher query per line — no explanation, no comments, no semicolons
- You can write multiple queries, but DO NOT write duplicate queries
- If retrying, fix the reported error and re-output all corrected queries

Cypher Writer — User Prompt

User Requirement:
{user_requirement}
Matched Entities (JSON):
{matched_entities}
[Optional: Previous errors to fix: {executor_error}]

Answer Formulator — System Prompt

You are a knowledge graph query assistant.

Task: Given the user’s query and structured results from a Neo4j knowledge graph, formulate a clear, helpful answer.

Guidelines:
- Directly address the user’s question based on the query results provided.
- If the results contain multiple records, summarize them concisely rather than dumping raw data.
- If the results are empty or indicate no match, state that clearly and suggest possible reasons.
- Keep the answer concise and factual; do not hallucinate information beyond what the results show.

Answer Formulator — User Prompt

User Requirement:
{user_requirement}
Database Query Results:
{query_results}

E.3 Semantic Information Retrieval Pipeline

The agents in the semantic information retrieval pipeline are prompted with clear structure including role assignments, task descriptions, constraints, few shot examples, dynamically loaded original query and information flow.

Semantic Query Generator — System Prompt

You are a semantic query generator for knowledge enrichment.

You are Given:
• The user's ORIGINAL QUERY.
• STRUCTURED RESULTS from a knowledge graph that already answered the query with factual entity and relation data.

Task:
Based on the structured results, generate semantic search queries that will retrieve additional, deeper information from an unstructured document store — information that the structured graph alone could not provide.

Rules:
• Generate one query per line.
• Each query should reference specific names or concepts from the structured results.
• Do NOT repeat the structured facts — generate queries to find NEW information.
• Do NOT answer anything — only output the queries.
• Output ONLY the queries, one per line — no numbering, no headers, no explanation.

Semantic Query Generator — User Prompt

Original Query:
{original_query}

Structured Results:
{structural_answer}

Keyword Extraction — System Prompt

You serve as a query decomposition module within a document retrieval pipeline. Your job is to break down an incoming question into two tiers of search terms that will drive both broad relationship matching and precise entity lookup.

Tier 1 — Relation Terms:
Capture the overarching relationships, interactions, and connections implied by the question. These terms help locate documents describing how concepts relate to each other, the type of association being queried, and the broader context linking entities together.

Tier 2 — Entity Terms:
Capture concrete names, technical vocabulary, product identifiers, dates, acronyms, and any other precise tokens that refer to specific real-world entities mentioned in the query.

Extraction Guidelines:
• Every term you produce must originate directly from the query text. Do not invent related concepts the user did not mention.
• Both tiers must be non-empty for any coherent question.
• Favor meaningful multi-word phrases over isolated words. "latest financial report" is one keyword, not three.
• If the query is trivial, gibberish, or a simple greeting, return empty lists for both tiers.
• All keywords must be expressed in {language}. Proper nouns that lack a standard translation stay in their original form.

Response Format:
Return exactly one JSON object with keys "high_level_keywords" and "low_level_keywords", each mapping to a list of strings. No commentary, no code fences, no additional text — the output will be fed directly to a JSON parser.

Example:

Query: "What are the main challenges of deploying large language models in production?"
{
"high_level_keywords": ["deployment challenges", "production obstacles", "operational difficulties"],
"low_level_keywords": ["large language models", "latency", "scaling", "model serving", "GPU costs"]
}

Given Query:
{query}

Output:

Cross-Pipeline Answer Synthesizer — System Prompt

You are a knowledgeable assistant that combines structured data with document context.

You are Given:
• ORIGINAL QUESTION: the user's question
• STRUCTURAL ANSWER: a precise answer from a knowledge graph query
• RETRIEVED CONTEXT: additional information from document search

Task: produce a well-structured answer following this format:

1. **Direct Answer** (1–2 sentences): Answer the original question directly and concisely, using the STRUCTURAL ANSWER as the factual basis. If the structural answer is a number or short fact, state it clearly first.

2. **Details** (optional, only if retrieved context adds value): Expand with relevant background, explanations, or examples from the RETRIEVED CONTEXT. Keep it concise and well-organized.

Rules:
• The direct answer MUST come first and MUST be based on the structural answer.
• Do NOT contradict the structural answer.
• Do NOT hallucinate — only use information from the provided inputs.
• If the retrieved context adds nothing useful beyond the structural answer, just give the direct answer without expansion.
• Use Markdown formatting for readability.
• Be concise. Avoid filler phrases and unnecessary repetition.
• You MUST answer in the SAME language as the ORIGINAL QUESTION.

Cross-Pipeline Answer Synthesizer — User Prompt

Original Question:
{original_query}

Structural Answer (from knowledge graph):
{structural_answer}

Retrieved Context (from document search):
{rag_context}

Note:
For detailed prompt of LLM-as-a-Judge,
please refer to the appendix sections on the next page.

F Prompt of LLM-as-a-Judge

The evaluation prompt instructs the LLM judge to score generated answers across four dimensions:comprehensiveness, diversity, empowerment, and overall quality within a 1–10 scale. LLM is forced to provide explicit reason for a fair score. Additionally, a false refusal penalty is applied to answers that decline to respond despite a valid reference answer existing. The judge is required to compare the generated answer against the reference and return a structured output containing both numerical scores and corresponding reasons.

LLM-as-a-Judge Prompt

You are an expert evaluator for question-answering systems. You will be given a question, an AI-generated answer, and optionally a reference answer. Rate the answer on the following dimensions using a scale of 1-10.

Critical Rules:
1. Accuracy is the MOST important factor across ALL dimensions.
2. For factual questions, a concise correct answer (e.g. just a name, number, or short phrase) is perfectly valid — do NOT penalize brevity.
3. If the AI answer says "I don't know", "no information found", "the context doesn't contain", or otherwise refuses to answer, BUT a reference answer exists and is specific, then the answer is WRONG. Apply these maximum scores:
- comprehensiveness: 1-2
- diversity: 1-2
- Readability: 2-3
- overall: 1-2

4. ALWAYS compare the AI answer against reference answer. An answer that contradicts or fails to match the reference answer is incorrect regardless of how well-written it is.

Respond ONLY with a JSON object:

```
{
"comprehensiveness": <1-10>,   "comprehensiveness_reason": "...",
"diversity": <1-10>,   "diversity_reason": "...",
"Readability": <1-10>,   "Readability_reason": "...",
"overall": <1-10>,   "overall_reason": "..."
}
```

Comprehensiveness: Does the answer correctly and completely address the question?
9-10: Fully correct, addresses all sub-parts. No key information missing.
7-8: Correct on the main point but misses minor details.
5-6: Partially correct — addresses some aspects but misses important parts.
3-4: Mostly incorrect or only tangentially related.
1-2: Completely wrong, irrelevant, empty, OR refuses to answer when a correct answer exists.

Diversity: Does the answer draw from multiple information sources or perspectives?
9-10: Synthesizes multiple sources, provides supporting evidence, examples, or different angles.
7-8: Provides the core information with some supporting context or evidence.
5-6: Single-perspective response with minimal supporting detail.
3-4: Very narrow, generic, or lacks any grounding in the knowledge base.
1-2: A guess with no evidence, OR refuses to answer when information should be available.

Empowerment: Is the answer well-organized, understandable, and supportive for decision making ?
9-10: Perfectly clear, logically structured, immediately understandable.
7-8: Clear and readable with minor formatting or structural issues.
5-6: Understandable but somewhat disorganized, verbose, or contains unnecessary information.
3-4: Confusing, poorly structured, OR excessively verbose. A well-written but WRONG answer.
1-2: Incoherent, garbled, or unreadable.

Overall: Holistic impression considering accuracy, usefulness, conciseness, and clarity.
9-10: Correct, concise, well-presented. The user gets exactly what they need.
7-8: Correct and useful. May have minor verbosity or structural issues.
5-6: Correct but significantly flawed, excessively verbose, repetitive, poorly organized.
3-4: Mostly unhelpful, wrong, misleading, too vague, OR so verbose that useful content is lost.
1-2: Completely useless, wrong, or harmful, or refusing to answer when a correct answer exists.

G Query Example

Tables 6–8 present representative queries from each evaluation dataset, illustrating the distinction between semantic queries which require reasoning over unstructured text and structural queries which require aggregation or counting over structured metadata.

Table 6: Query examples from the API dataset.

Type	Query	Reference Answer
Semantic	What data can the Building X Energy API provide?	The Building X Energy API enables reading energy consumption, emission, and cost data for a specific location or meter. Users can specify desired granularity levels and time periods. Emissions are calculated based on defined emission conversion factors, and costs are derived from meter readings.
Semantic	What types of real-time data can the ANYmal robotic platform stream during inspections?	The ANYmal robotic platform can stream acoustic, thermal, gas, and visual feedback in real time. It supports data formats including JPG, WAV, E57, PLY, and DAE. It also provides WebRTC-based streaming video for telepresence and detailed feedback on robot pose and location.
Structural	How many APIs use OpenAPI as their resource type?	90
Structural	Which product line has the most APIs?	Insights Hub and Industrial IoT

Table 7: Query examples from the MultiHop-RAG dataset.

Type	Query	Reference Answer
Semantic	Why is it paradoxical that those most worried about AI are also racing to build it?	Fear of others building it irresponsibly becomes their justification to build it themselves Fear accelerates the arms race rather than stopping it.
Semantic	How does Amazon's "anti-discounting strategy" hurt consumers despite claiming to promote low prices?	Sellers raise prices on all platforms to avoid Amazon's punishment algorithm, creating an artificial price floor everywhere. Consumers pay more across the entire web, not just on Amazon.
Structural	Which news source has the most articles in the dataset?	TechCrunch
Structural	Which person appears in articles from the most different news categories?	Taylor Swift (entertainment + sports via Travis Kelce)

Table 8: Query examples from the Musique dataset.

Type	Query	Reference Answer
Semantic	What is the goal of the group that European Movement Germany is a member of? (Content summary)	European integration
Semantic	Why did the 2010 US tomato shortage primarily affect fast food chains rather than home cooks? (Content understanding)	Because Florida's tomato crop supplies commercial-scale distribution; home gardeners grow their own or buy from local/California sources.
Structural	Where is Ulrich Walter's employer headquartered? (Entity in the content)	Cologne
Structural	Who was the sibling of Nannina de' Medici? (Entity in the content)	Giuliano de' Medici

H Case Study

In this section, we present a case study comparing MAD-RAG with the baselines on the API catalog. As shown in Table 9 and 10, we select one query for both query types and demostrate the generated responses. MAD-RAG is the only approach that provides the factual correctness to the structural query with detailed elaboration, while others provide incorrect numbers or answers with less readability. For the semantic query, MAD-RAG and LightRAG covers all core concepts accurately without redundancy, while maintaining moderate response length for better readability. In contrast, GraphRAG suffer from over-retrieval, producing verbose responses. HyDE and A-RAG retrieve content that is topically related but contextually irrelevant. KET-RAG generates correct and brief answers, however lack of sufficient detail.

Table 9: Sructural query: How many APIs use OpenAPI as their resource type?

Method	**Response**
Reference	90
MAD-RAG	There are 90 APIs that use OpenAPI as their resource type. OpenAPI serves as a widely adopted standard for defining and documenting RESTful APIs, providing an interface specification and promoting a standardized approach to interface development. Its broad utility and acceptance are evident in its widespread use across various product lines for resource modeling. Examples of APIs leveraging OpenAPI include those within Industrial Asset Hub, Industrial Edge, and the SiGREEN REST API, as well as the Resource Access Management API.
A-RAG	The number of APIs that use OpenAPI as their resource type can be counted from the chunks provided. Here is the count from each chunk: - Chunk 49: 8 APIs - Chunk 71: 7 APIs - Chunk 98: 10 APIs (partial list visible) - Chunk 118: 8 APIs - Chunk 249: 7 APIs (partial list visible) Summing these up, there are at least 40 APIs that use OpenAPI as their resource type.
KET-RAG	At least 38 APIs use OpenAPI as their resource type.
GraphRAG	The provided data lists a significant number of APIs that use OpenAPI as their resource type. OpenAPI is a widely adopted open standard specification used for designing and describing RESTful web services, ensuring standardized and interoperable API communication. Within the datasets, many APIs across various product lines and vertical domains provide OpenAPI resource types. For instance, APIs such as the AI Asset Manager OpenAPI V2 and V3, Industrial Information Hub Essentials API, OT Companion APIs (Asset Management and Protection Settings), various Building X APIs (Activities API, Energy API, Security Workflow API, and many others), Siemens eMobility APIs (e.g., Charger Status API, Remote Command API), and many specialized domain APIs like Electrification X API, SiGREEN suite APIs, and RailXplore Fusion API all use OpenAPI as their specification resource. Moreover, the AI Asset Manager itself provides APIs explicitly formatted in OpenAPI V2 and V3, highlighting its emphasis on standardized API descriptions to encourage interoperability and ease of integration across diverse industrial and cloud platforms. Given the extensive list, it can be concluded that dozens of APIs utilize OpenAPI resource types. Specifically, the data shows at least around 40 named APIs across different sectors and product lines are explicitly tagged as providing OpenAPI resource types. This widespread adoption underscores OpenAPI's role as a foundational standard in related ecosystems for API management, facilitating consistent interface design, automation, and integration. In summary, more than 40 distinct APIs in the provided data use OpenAPI as their resource type, demonstrating its broad and critical usage in industrial, building technology, sustainability, digital industries, and e-mobility domains.
HyDE	Based on the provided context, only one API explicitly mentions using OpenAPI as its resource type. The API serving the Digital Industries vertical domain provides OpenAPI resource types. No other APIs in the context specify OpenAPI as the resource type. Answer: One API uses OpenAPI as its resource type.
LightRAG	A significant number of APIs utilize OpenAPI as their resource type, indicating its widespread use as a standard for defining and documenting RESTful APIs. The following APIs are identified as providing OpenAPI as a resource type: Accounts API, Activities API, Agent Management Async API, Agent Management Service, AI Asset Manager OpenAPI V2, AI Asset Manager OpenAPI V3, Asset Management Service, Authorization API, Battery Passport REST API, Building Geometry API, Building Operations API, Building Structure API, Case Management Management Service, Charger Control API, Charger Status API, Client API: Registry API, Commanding API, COMOS Web REST API, Data Exchange Service, Device Discovery API, Drivetrain Analyzer Cloud API, Electrification X API, Energy API, Energy and Environment APIs, Event Management, Fire API, Flow Creator Configurator APIs, Gateway Onboarding API, gWAP API, Identity Management Service, IE Device API, IE Portal API, IEM API, IEM V2 API, Industrial Information Hub APIs, Industrial Information Hub Essentials API, Integrated Data Lake Service, Inventory API, IoT File Service, IoT Time Series Aggregates Service, IoT Time Series Bulk Service, IoT Time Series Service, Job Manager Service, Lifecycle Twin API, Location Services APIs, Message Broker Service, Mcenter - Openness APIs of Applications, MindConnect API, Model Management Service, Nurse Call System API, Oauth Authorization Server, Occupancy APIs, Opcenter Intelligence Service, Optimize My Plant API, OT Companion Asset Management API, OT Companion Protection Settings API, Point Value Ingest API, Polarion REST API, Product Information Hub API, Railigent X Train API, RailXplore Fusion API, Remote Access API, Remote Command API, Reservations API, Resource Access Management, RFID Management API, Rules Management Service, Security Identities And Privileges API, Security Intrusion API, Security Workflow API, Server API: Device Discover API, Server API: Driver Event API, Server API: Driver Info API, Server API: Generic Data API, SiGREEN My Task API, SiGREEN PCF API, SiGREEN Procurement API, SiGREEN Product API, SieSmart API, SITRANS SCM IQ API, Tenant Management Service, Token Management Service, Transaction Management API, Transport Carbon Calculator API, Usage Transparency Service, User Configuration API, VDV-261 API, Vision Payload Specification, Visual Flow Creator Service, Workflow Executor API

Table 10: Semantic query: How does the Building X Building Operations API organize its resources?

Method	**Response**
Reference	The Building Operations API is organized around Device, Point, and PointValue resources. Devices host multiple points (sensors or command receivers). Point values represent measurements or set values with time-interval filtering. The API also supports Events for non-normal states like faults and alarms, Workorders for tracking maintenance and repair tasks with priority and due dates, and Teams for organizing people responsible for building operations.
MAD-RAG	The Building X Building Operations API organizes its resources around three main types: Device: This refers to electronic equipment with computing capabilities, like pumps or sensors. A device can host multiple points. Point: These are interfaces to data within a system that provide meaning and classify the data, such as a temperature set point. PointValue: This represents the measured data coming from these points. Users can specify the time range for retrieving this data. This organization allows the API to manage tasks like reading data from points and sending commands to equipment within building systems.
A-RAG	The Building X Building Operations API organizes its resources by providing a means to define and manage access policies for fine-grained access control of various business resources. It also allows the organization of groups of resources which can be part of these policies. The API has certain limitations such as limits on the number of existing policies per owner/tenant, the number of subjects/rules per policy, and the number of actions/resources per rule. Additionally, there are specific notes on URL formatting and region code specifications for subjects or resources to ensure proper API usage.
KET-RAG	The Building Operations API is structured around Device, Point, and PointValue resources.
GraphRAG	The Building X Building Operations API organizes its resources through a comprehensive, hierarchical, and functionally categorized structure designed to support efficient management, control, and integration within smart building environments. This organization aligns physical building contexts, operational functions, user roles, and security components to facilitate scalable and secure building operations management. Hierarchical Physical and Logical Structure: At the core, the API arranges resources to mirror the real-world physical and organizational structure of buildings and their components. The primary top-level entities and hierarchical relationships include: Customers and Partitions (companies owning assets, logical subdivisions to group multiple buildings), Sites and Buildings (geographic clusters hosting multiple Buildings), Locations and Spaces (finer physical or functional subdivisions within Buildings), Floors and Tagged Elements, Equipment and Devices (sensors, pumps, valves, actuators linked to spatial entities, hosting Points). Functional and Operational Resource Categories: UserGroups, Roles, and Machine Users for access management; Authorization and Security; Alarms, Events, and Workflows; Device Management (Device Groups, Device Orders, Head End Systems); Data Points and Monitoring (Points as semantic interfaces, PointValues storing time-series data). Integration and Ecosystem Context: Building X Platform and Cloud Integration; Open API Portfolio and Mendix Connector; Siemens Xcelerator Ecosystem; Documentation and Collaboration. In summary, the API organizes resources through a multi-layered hierarchical structure reflecting physical and functional layout while incorporating operational, security, and user management categories.
HyDE	The Building X Building Operations API organizes its resources by managing various building-related entities such as organization structure, areas, fixtures, plugloads, conference rooms, and emergency lighting controls. It provides access to lighting controls, sensors, and energy data to optimize building efficiency. This resource organization supports functionalities related to energy and environment management within the building technology domain. Specifically, the API allows users to manage: Organization structure, Areas within the building, Fixtures (lighting and other devices), Plugloads (electrical loads connected to plugs), Conference rooms, Emergency lighting controls. This structured approach enables comprehensive control and monitoring of building operations to enhance energy efficiency and operational effectiveness. The API is part of the Enlighted product line and is tagged under Energy and Environment, Tracking, and Energy, serving the Building Technology vertical domain.
LightRAG	The Building X Building Operations API organizes its resources around Device, Point, and PointValue concepts. Specifically: A Device refers to electronic equipment with computing abilities, such as pumps, sensors, or automation devices. A Point represents the interface to data within a system, classifying it and providing semantic context, like a temperature set point. A device can host multiple points, which can either sense information (e.g., temperature, air quality) or receive and process commands. A PointValue is an object representing a measurement or a set value of a Point. The measured data from data points is referred to as point values, and users can specify the time interval for data retrieval. This structure enables the API to read data from data points and issue commands to actuator points, supporting functionalities such as changing thermostat set-point value.