

In-depth Analysis of Graph-based RAG in a Unified Framework [Experiment, Analysis & Benchmark]

Yingli Zhou
CUHK-Shenzhen

Yaodong Su
CUHK-Shenzhen

Youran Sun
CUHK-Shenzhen

Shu Wang
CUHK-Shenzhen

Taotao Wang
CUHK-Shenzhen

Runyuan He
CUHK-Shenzhen

Yongwei Zhang
Huawei Cloud

Sicong Liang
Huawei Cloud

Xilin Liu
Huawei Cloud

Yuchi Ma
Huawei Cloud

Yixiang Fang
CUHK-Shenzhen

ABSTRACT

Graph-based Retrieval-Augmented Generation (RAG) has proven effective in integrating external knowledge into large language models (LLMs), improving their factual accuracy, adaptability, interpretability, and trustworthiness. A number of graph-based RAG methods have been proposed in the literature. However, these methods have not been systematically and comprehensively compared under the same experimental settings. In this paper, **we first summarize a unified framework to incorporate all graph-based RAG methods from a high-level perspective.** We then extensively compare representative graph-based RAG methods over a range of questioning-answering (QA) datasets – from specific questions to abstract questions – and examine the effectiveness of all methods, providing a thorough analysis of graph-based RAG approaches. As a byproduct of our experimental analysis, we are also able to identify new variants of the graph-based RAG methods over specific QA and abstract QA tasks respectively, by combining existing techniques, which outperform the state-of-the-art methods. Finally, based on these findings, we offer promising research opportunities. We believe that a deeper understanding of the behavior of existing methods can provide new valuable insights for future research.

PVLDB Reference Format:

Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, and Yixiang Fang. In-depth Analysis of Graph-based RAG in a Unified Framework [Experiment, Analysis & Benchmark]. PVLDB, 18(1): XXX-XXX, 2025. doi:XX.XX/XXX.XX

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/JayLZhou/GraphRAG>.

1 INTRODUCTION

The development of Large Language Models (LLMs) like GPT-4 [1], Qwen2.5 [89], and Llama 3.1 [14] has sparked a revolution in the field of artificial intelligence [22, 31, 46, 54, 63, 83, 84, 96]. Despite their remarkable comprehension and generation capabilities, LLMs may still generate incorrect outputs due to a lack of domain-specific

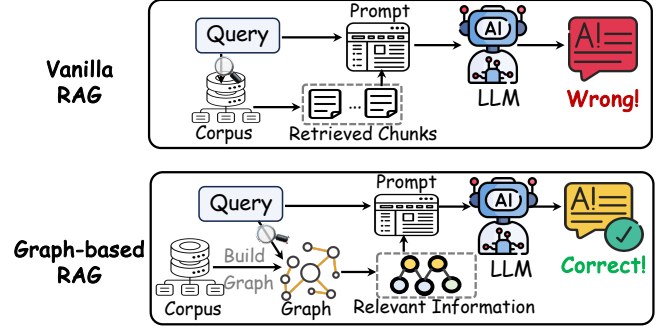


Figure 1: Overview of vanilla RAG and graph-based RAG.

knowledge, real-time updated information, and proprietary knowledge, which are outside LLMs’ pre-training corpus, known as “hallucination” [67].

To bridge this gap, the Retrieval Augmented Generation (RAG) technique [18, 21, 30, 32, 87, 91, 94] has been proposed, which supplements LLM with external knowledge to enhance its factual accuracy and trustworthiness. **Given a user query Q , the key idea of naive-based RAG [42] (i.e., vanilla RAG) is to retrieve relevant chunks from the external corpus, and then feed them along with Q as a prompt into LLM to generate answers.** Consequently, RAG techniques have been widely applied in various fields, especially in domains where LLMs need to generate reliable outputs, such as healthcare [54, 83, 96], finance [46, 63], and education [22, 84]. Moreover, RAG has proven highly useful in many data management tasks, including NL2SQL [16, 43], data cleaning [17, 45, 59, 70], knob tuning [23, 41], DBMS diagnosis [74, 97, 98], and SQL rewrite [47, 78]. **In turn, the database community has recently begun to actively explore how to build efficient and reliable RAG systems [2, 36].** Due to the important role of the RAG technique in LLM-based applications, numerous RAG methods have been proposed in the past year [30]. Among these methods, the state-of-the-art RAG approaches typically use the graph data as the external data (also called graph-based RAG), since they capture the rich semantic information and link relationships between entities. **Unlike vanilla RAG, graph-based RAG methods retrieve relevant information related to the query Q —such as nodes, relationships, or subgraphs—from the graph, and then incorporate this information into the prompt along with Q for the LLM to generate an answer.** The overview of naive-based RAG and graph-based RAG are shown in Figure 1.

Recently, several mainstream database systems have started supporting graph-based RAG, including PostgreSQL [68], Neo4j [62], and Databricks [11]. At the same time, graph-based RAG has become a core component in modern graph-native agentic systems,

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment. Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097. doi:XX.XX/XXX.XX

Table 1: Classification of existing representative graph-based RAG methods. Index Component indicates which graph elements (e.g., nodes, relationships, communities) are stored in the index.

Method	Graph Type	Index Component	Retrieval Primitive	Retrieval Granularity	Specific QA	Abstract QA
RAPTOR [72]	Tree	Tree node	Question vector	Tree node	✓	✓
KGP [85]	Passage Graph	Entity	Question	Chunk	✓	✗
HippoRAG [25]	Knowledge Graph	Entity	Entities in question	Chunk	✓	✗
G-retriever [29]	Knowledge Graph	Entity, Relationship	Question vector	Subgraph	✓	✗
ToG [77]	Knowledge Graph	Entity, Relationship	Question	Subgraph	✓	✗
DALK [44]	Knowledge Graph	Entity	Entities in question	Subgraph	✓	✗
LGraphRAG [15]	Textual Knowledge Graph	Entity, Community	Question vector	Entity, Relationship, Chunk, Community	✓	✗
GGraphRAG [15]	Textual Knowledge Graph	Community	Question vector	Community	✗	✓
FastGraphRAG [19]	Textual Knowledge Graph	Entity	Entities in question	Entity, Relationship, Chunk	✓	✓
LLightRAG [24]	Rich Knowledge Graph	Entity, Relationship	Low-level keywords in question	Entity, Relationship, Chunk	✓	✓
GLightRAG [24]	Rich Knowledge Graph	Entity, Relationship	High-level keywords in question	Entity, Relationship, Chunk	✓	✓
HLightRAG [24]	Rich Knowledge Graph	Entity, Relationship	Both high- and low-level keywords	Entity, Relationship, Chunk	✓	✓

such as LangGraph [40] and Chat2Graph [6]. Following the success of graph-based RAG, researchers from fields such as database, data mining, machine learning, and natural language processing have designed efficient and effective graph-based RAG methods [15, 24, 25, 33, 44, 67, 72, 85, 86]. In Table 1, we summarize the key characteristics of 12 representative graph-based RAG methods based on the graph types they rely on, their index components, retrieval primitives and granularity, and the types of tasks they support. After a careful literature review, we make the following observations. First, no prior work has proposed a unified framework to abstract the graph-based RAG solutions and identify key performance factors. Second, existing works focus on evaluating the overall performance, but not individual components. Third, there is no existing comprehensive comparison between all these methods in terms of accuracy and efficiency.

Our work. To address the above issues, in this paper, we conduct an in-depth study on graph-based RAG methods. We first **summarize a novel unified framework with four stages**, namely ① *Graph building*, ② *Index construction*, ③ *Operator configuration*, and ④ *Retrieval & generation*, which captures the core ideas of all existing methods. Under this framework, we systematically compare 12 existing representative graph-based RAG methods. We conduct comprehensive experiments on the widely used question-answering (QA) datasets, including the specific and abstract questions, which evaluate the effectiveness of these methods in handling diverse query types and provide an in-depth analysis.

In summary, our principal contributions are as follows.

- Summarize a novel unified framework with four stages for graph-based RAG solutions from a high-level perspective (Sections 3 ~ 6).
- Conduct extensive experiments from different angles using various benchmarks, providing a thorough analysis of graph-based RAG methods. Based on our analysis, we identify new variants of graph-based RAG methods, by combining existing techniques, which outperform the state-of-the-art methods (Section 7).
- Summarize lessons learned and propose practical research opportunities that can facilitate future studies (Section 8).

The rest of the paper is organized as follows. In Section 2, we present the preliminaries and introduce a novel unified framework for graph-based RAG solutions in Section 3. In Sections 4 through

6, we compare the graph-based RAG methods under our unified framework. The comprehensive experimental results and analysis are reported in Section 7. We present the learned lessons and a list of research opportunities in Section 8, and Section 9 reviews related work while Section 10 summarizes the paper.

2 PRELIMINARIES

In this section, we review some key concepts of LLM and the general workflow of graph-based RAG methods.

2.1 Large Language Models (LLMs)

We introduce some fundamental concepts of LLMs, including LLM prompting and retrieval augmented generation (RAG).

LLM Prompting. After instruction tuning on large corpus of human interaction scenarios, LLM is capable of following human instructions to complete different tasks [13, 64]. Specifically, given the task input, we construct a prompt that encapsulates a comprehensive task description. The LLM processes this prompt to fulfill the task and generate the corresponding output. Note that pre-training on trillions of bytes of data enables LLM to generalize to diverse tasks by simply adjusting the prompt [64].

Retrieval Augmented Generation. During completing tasks with prompting, LLMs often generate erroneous or meaningless responses, i.e., the hallucination problem [31]. To mitigate the problem, retrieval augmented generation (RAG) is utilized as an advanced LLM prompting technique by using the knowledge within the external corpus, typically including two major steps [21]: (1) *retrieval*: given a user question Q , using the index to retrieve the most relevant (i.e., top- k) chunks to Q , where the large corpus is first split into smaller chunks, and (2) *generation*: guiding LLM to generate answers with the retrieved chunks along with Q as a prompt.

2.2 Graph-based RAG

Unlike vanilla RAG, graph-based RAG methods employ graph structures built from external corpus to enhance contextual understanding in LLMs and generate more informed and accurate responses [67]. Typically, graph-based RAG methods are composed of three major stages: (1) *graph building*: given a large corpus \mathcal{D} with d chunks, for each chunk, an LLM extracts nodes and edges, which are then combined to construct a graph \mathcal{G} ; (2) *retrieval*: given a user question Q , using the index to retrieve the most relevant

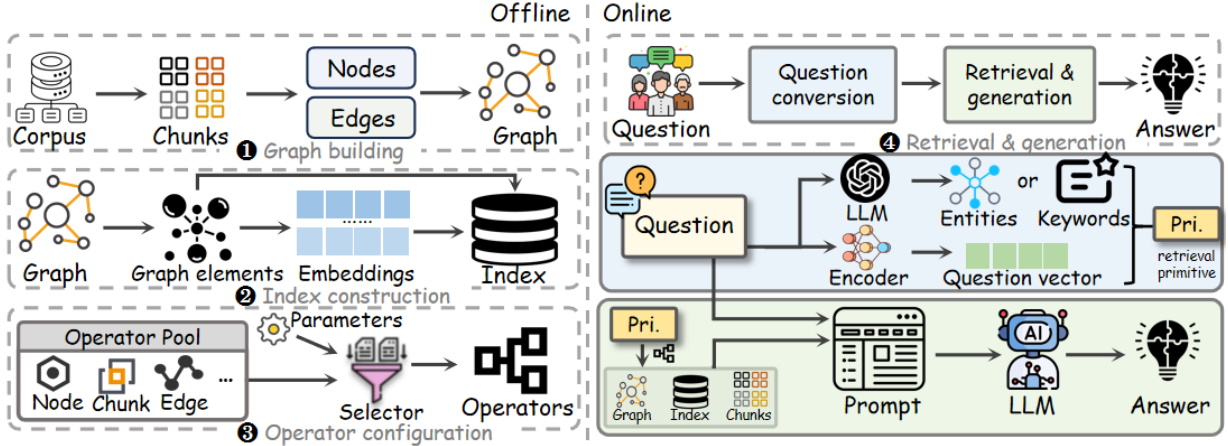


Figure 2: **Workflow of graph-based RAG methods under our unified framework.**

information (e.g., nodes or subgraphs) from \mathcal{G} , and (3) *generation*: guiding LLM to generate answers by incorporating the retrieved information into the prompt along with Q .

3 A UNIFIED FRAMEWORK

In this section, we develop a novel unified framework, consisting of four stages: ❶ *Graph building*, ❷ *Index construction*, ❸ *Operator configuration*, and ❹ *Retrieval & generation*, which can cover all existing graph-based RAG methods, as shown in Algorithm 1.

Algorithm 1: A unified framework for graph-based RAG

input : Corpus \mathcal{D} , user question Q , and parameters \mathcal{P}
output : The answers for user question Q

- 1 $C \leftarrow \text{split } \mathcal{D} \text{ into multiple chunks;}$
// (1) Graph building.
- 2 $\mathcal{G} \leftarrow \text{GraphBuilding}(C);$
// (2) Index construction.
- 3 $\mathcal{I} \leftarrow \text{IndexConstruction}(\mathcal{G}, C);$
// (3) Operator configuration.
- 4 $\mathcal{O} \leftarrow \text{OperatorConfiguration}(\mathcal{P});$
// (4) Retrieve relevant information and generate response.
- 5 $\mathcal{R} \leftarrow \text{Retrieval\&generation}(\mathcal{G}, \mathcal{I}, \mathcal{O}, Q, C);$
- 6 **return** $\mathcal{R};$

Specifically, given the large corpus \mathcal{D} , we first split it into multiple chunks C (line 1). We then sequentially execute operations in the following four stages (lines 2-5): (1) Build the graph \mathcal{G} for input chunks C (Section 4); (2) Construct the index based on the graph \mathcal{G} from the previous stage (Section 5); (3) Configure the retriever operators for subsequent retrieving stages (Section 6), and (4) For the input user question Q , retrieve relevant information from \mathcal{G} using the selected operators and feed them along with the question Q into the LLM to generate the answer. The workflow of graph-based RAG methods under our framework is shown in Figure 2. We note that graph-based RAG methods differ across the four stages. Specifically, as shown in Table 1, different methods construct distinct types of graphs in Stage ❶, build different indices in Stage ❷, and retrieve information at varying levels of granularity. Consequently, they employ different operators for retrieval, leading to variations in both Stage ❸ and Stage ❹. Note that the query is first converted

into a retrieval primitive, which serves as the basis for retrieval in each method (see Section 6.3).

4 GRAPH BUILDING

The *graph building* stage aims to transfer the input corpus into a graph, serving as a fundamental component in graph-based RAG methods. Before building a graph, the corpus is first split into smaller chunks. Then, an LLM or other tools are used to construct nodes and edges based on these chunks, as shown in Figure 2❶. We note that this preprocessing step is essential for all RAG methods. Further details are provided in our technical report [71]. There are five types of graphs, each with a corresponding construction method; we present a brief description of each graph type and its construction method below:

❶ *Passage Graph*. In the passage graph (PG), each chunk represents a node, and edges are built by the entity linking tools [85]. If two chunks contain a number of the same entities larger than a threshold, we link an edge for these two nodes.

❷ *Tree*. The tree is constructed in a progressive manner, where each chunk represents the leaf node in the tree. Then, it uses an LLM to generate higher-level nodes. Specifically, at the i -th layer, the nodes of $(i + 1)$ -th layer are created by clustering nodes from the i -th layer that does not yet have parent nodes. For each cluster with more than two nodes, the LLM generates a virtual parent node with a high-level summary of its child node descriptions.

❸ *Knowledge Graph*. The knowledge graph (KG) is constructed by extracting entities and relationships from each chunk, where each entity represents an object and the relationship denotes the semantic relation between two entities.

❹ *Textual Knowledge Graph*. A textual knowledge graph (TKG) is a specialized KG (following the same construction step as KG), with the key difference being that in a TKG, each entity and relationship is assigned a brief textual description.

❺ *Rich Knowledge Graph*. The rich knowledge graph (RKG) is an extended version of TKG, containing more information, including textual descriptions for entities and relationships, as well as keywords for relationships. We summarize the key characteristics of each graph type in Table 2, considering their contained attributes (e.g., inclusion of entity names and descriptions), the time and number of tokens required for construction, the resulting graph size, and the richness of information contained within each graph. In

Table 2: Comparison of different types of graph.

Graph	Entity Name	Entity Type	Entity Description	Relationship Name	Relationship Keyword	Relationship Description	Edge Weight	Token Consuming	Graph Size	Information Richness	Construction Time
Tree	✗	✗	✗	✗	✗	✗	✗	★	★	★★	★★
PG	✓	✗	✗	✓	✗	✗	✗	N/A	★★★★	★	★★★★
KG	✓	✗	✗	✓	✗	✗	✓	★★	★★	★★★	★★
TKG	✓	✓	✓	✗	✗	✓	✓	★★★	★★★	★★★★	★★★
RKG	✓	✓	✓	✗	✓	✓	✓	★★★	★★★	★★★★★	★★★

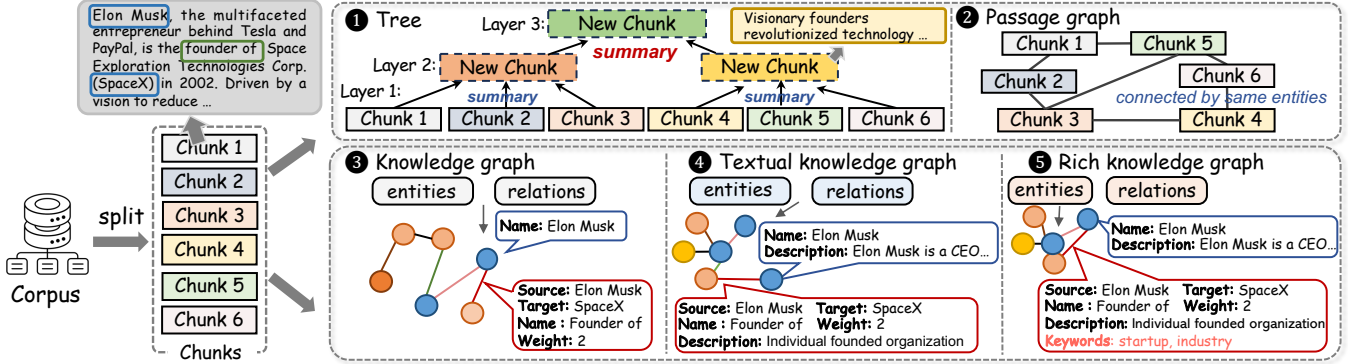


Figure 3: Examples of five types of graphs.

addition, we provide a case study of how the five types of graphs are constructed from the external corpus in Figure 3. The detailed descriptions are shown in our technical report [71].

5 INDEX CONSTRUCTION

To support efficient online querying, existing graph-based RAG methods typically include an index-construction stage, which involves storing graph elements – such as entities and relationships – in a vector database and computing community reports to enable efficient online retrieval, as shown in Figure 2. Generally, there are three types of indices, ① *Node index*, ② *Relationship index*, and ③ *Community index*, where for the first two types, we use the well-known text-encoder models, such as BERT [12] or ColBERT [39] to generate embeddings for nodes or relationships in the graph.

① *Node index* stores the graph nodes in the vector database. For RAPTOR, G-retriever, DALK, FastGraphRAG, LGraphRAG, LLightRAG, and HLighRAG, all nodes in the graph are directly stored in the vector database. For each node in KG, its embedding vector is generated by encoding its entity name, while for nodes in Tree, TKG, and RKG, the embedding vectors are generated by encoding their associated textual descriptions. In KGP, it stores the TF-IDF matrix [27], which represents the term-weight distribution across different nodes (i.e., chunks) in the index.

② *Relationship index* stores the relationships of the graph in a vector database, where for each relationship, its embedding vector is generated by encoding a description that combines its associated context (e.g., description) and the names of its linked entities.

③ *Community index* stores the community reports for each community, where communities are generated by the clustering algorithm and the LLM produces the reports. Specifically, Leiden [80] algorithm is utilized by LGraphRAG and GGraphRAG.

Remark. *Relationship index* tends to have a larger size, whereas the *Community index* is more compact but incurs the highest construction cost in terms of tokens and time, and provides the detailed comparisons of these indices in our technical report [71].

6 RETRIEVAL AND GENERATION

In this section, we explore the key steps in graph-based RAG methods, i.e., selecting operators, and using them to retrieve relevant information to question Q .

6.1 Retrieval operators

In this subsection, we demonstrate that the retrieval stage of various graph-based RAG methods can be abstracted into a modular sequence of operators. Different methods select and compose these operators in distinct ways, enabling flexible and extensible retrieval pipelines. By systematically analyzing existing graph-based RAG implementations, we identify a comprehensive set of 19 retrieval operators, and based on the granularity of retrieval, we classify the operators into five categories. We note that most operators are derived from designs described in the original papers—though often unnamed—so we assign them meaningful and consistent names. For the remaining operators, which are not explicitly defined in the literature, we extract and summarize them based on source code analysis. Importantly, by selecting and arranging these operators in different sequences, all existing (and potentially future) graph-based RAG methods can be implemented.

- *Node type*. This type of operator focuses on retrieving “important” nodes for a given question, and based on the selection policy, there are seven different operators to retrieve nodes. ① VDB leverages the vector database to retrieve nodes by computing the vector similarity with the query vector. ② ReINode extracts nodes from

the provided relationships. ❸ PPR uses the Personalized PageRank (PPR) algorithm [28] to identify the top- k similar nodes to the question, where the restart probability of each node is based on its similarity to the entities in the given question. ❹ Agent utilizes the capabilities of LLMs to select nodes from a list of candidate nodes. ❺ Onehop selects the one-hop neighbor entities of the given entities. ❻ Link selects the top-1 most similar entity for each entity in the given set from the vector database. ❼ TF-IDF retrieves the top- k relevant entities by ranking them based on term frequency and inverse document frequency from the TF-IDF matrix.

- *Relationship type.* These operators are designed to retrieve relationships from the graph that are most relevant to the user question. There are four operators: ❶ VDB, ❷ Onehop, ❸ Aggregator, and ❹ Agent. Specifically, the VDB operator also uses the vector database to retrieve relevant relationships. The Onehop operator selects relationships linked by one-hop neighbors of the given selected entities. The Aggregator operator builds upon the PPR operator in the node operator. Given the PPR scores of entities, the most relevant relationships are determined by leveraging entity-relationship interactions. Specifically, the score of each relationship is obtained by summing the scores of the two entities it connects. Thus, the top- k relevant relationships can be selected. The key difference for the Agent operator is that, instead of using a candidate entity list, it uses a candidate relationship list, allowing the LLM to select the most relevant relationships based on the question.

- *Chunk type.* The operators in this type aim to retrieve the most relevant chunks to the given question. There are three operators: ❶ Aggregator, ❷ FromRel, and ❸ Occurrence. Specifically, Aggregator uses the relationship score vector from the Link operator and a relationship-chunk interaction matrix to aggregate chunk scores via matrix multiplication, selecting the top- k chunks with the highest scores. For FromRel: Given a set of relationships, all chunks containing at least one of them are retrieved. The Occurrence selects the top- k chunks based on the given relationships. Specifically, for each relationship, we identify its two associated entities. If both entities appear in the same chunk, the chunk’s score is incremented by 1. After processing all relationships, the top- k chunks with the highest scores are selected.

- *Subgraph type.* There are three operators to retrieve the relevant subgraphs from the graph \mathcal{G} : The ❶ KhopPath operator aims to identify k -hop paths in \mathcal{G} by iteratively finding such paths where the start and end points belong to the given entity set. After identifying a path, the entities within it are removed from the entity set, and this process repeats until the entity set is empty. Note that if two paths can be merged, they are combined into one path. For example, if we have two paths $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow C \rightarrow D$, we can merge them into a single path $A \rightarrow B \rightarrow C \rightarrow D$. The ❷ Steiner operator first identifies the relevant entities and relationships, then uses these entities as seed nodes to construct a Steiner tree [27]. The ❸ AgentPath operator aims to identify the most relevant k -hop paths to a given question, by using LLM to filter out the irrelevant paths.

- *Community type.* Only the LGraphRAG and GGraphRAG using the community operators, which includes two detailed operators, ❶ Entity, and ❷ Layer. This operator first identifies communities that contain the specified entities, with each community maintaining an associated entity list. It then ranks the selected communities based on relevance scores assigned by the LLM, returning the top- k

highest-scoring ones. Each community is associated with a layer attribute, and the Layer operator retrieves all communities at or below the specified layer.

6.2 Operator configuration

Under our unified framework, any existing graph-based RAG method can be implemented by leveraging the operator pool along with specific method parameter \mathcal{P} , as shown in Figure 2❸. Instead, \mathcal{P} acts as a control module that configures the retrieval pipeline for a given graph-based RAG method by determining: (1) which atomic operators should be used in the method; and (2) the execution order of these operators within the retrieval process.

In Table 3, we present how the existing graph-based RAG methods utilize our provided operators to assemble their retrieval stages. For example, LLIGHTRAG first applies the VDB operator to retrieve relevant nodes, then uses the Onehop operator to retrieve relevant relationships, and finally employs the Occurrence operator to obtain the relevant chunks. In the above example, we can set $\mathcal{P} = \langle \text{VDB, Onehop, Occurrence} \rangle$. Essentially, the parameter \mathcal{P} represents the retrieval configuration to distinguish the retrieval stage for each specific graph-based RAG method. Due to this independent and modular decomposition of all graph-based RAG methods, we not only gain a deeper understanding of how these approaches work but also gain the flexibility to combine these operators to create new methods. Besides, new operators can be easily created, for example, we can create a new operator VDB within the community type, which allows us to retrieve the most relevant communities by using vector search to compare the semantic similarity between the question and the communities. In our later experimental results (see Exp.5 in Section 7.3), thanks to our modular design, we can design a new state-of-the-art RAG method by first creating two new operators and combining them with the existing operators.

6.3 Retrieval & generation

In the *Retrieval & generation* stage, the graph-based RAG methods first go through a *Question conversion* stage (see the second subfigure on the right side of Figure 2), which aims to transfer the user input question Q into the retrieval primitive \mathcal{D} , where \mathcal{D} denotes the atomic retrieval unit, such as entities or keywords in Q , and the embedding vector of Q .

In the *Question conversion* stage, DALK, HippoRAG, and ToG extract entities from the question; KGP directly uses the original question as the retrieval primitive. The three versions of LighTRAG extract keywords from the question as the retrieval primitive, and the remaining methods use the embedding vector of Q .

Based on the retrieval primitive \mathcal{D} and the selected operators, the most relevant information to Q is retrieved and combined with Q to form the final prompt for LLM response generation. Generally, there are two types of answer generation paradigms: ❶ Directly and ❷ Map-Reduce. The former directly utilizes the LLM to generate the answer. The Map-Reduce strategy in GGraphRAG prompts the LLM to generate partial answers and confidence scores from each retrieved community. These (answer, score) pairs are ranked, and the top ones are appended to form the final prompt for answer generation. An example is shown in [71].

Table 3: Operators utilized in graph-based RAG methods; “N/A” means that this type of operator is not used.

Method	Node	Relationship	Chunk	Subgraph	Community
RAPTOR	VDB	N/A	N/A	N/A	N/A
KGP	TF-IDF	N/A	N/A	N/A	N/A
HippoRAG	Link + PPR	Aggregator	Aggregator	N/A	N/A
G-retriever	VDB	VDB	N/A	Steiner	N/A
ToG	Link + Onehop + Agent	Onehop + Agent	N/A	N/A	N/A
DALK	Link + Onehop + Agent	N/A	N/A	KhopPath + AgentPath	N/A
FastGraphRAG	Link + VDB + PPR	Aggregator	Aggregator	N/A	N/A
LGraphRAG	VDB	Onehop	Occurrence	N/A	Entity
RGraphRAG	N/A	N/A	N/A	N/A	Layer
LLightRAG	VDB	Onehop	Occurrence	N/A	N/A
GLightRAG	FromRel	VDB	FromRel	N/A	N/A
HLightRAG	VDB + FromRel	Onehop + VDB	Occurrence + FromRel	N/A	N/A

Table 4: Datasets used in our experiments; The underlined number of chunks denotes that the dataset is pre-split into chunks by the expert annotator.

Dataset	# of Tokens	# of Questions	# of Chunks	QA Type
MultihopQA	1,434,889	2,556	609	Specific QA
Quality	1,522,566	4,609	265	Specific QA
PopQA	2,630,554	1,172	33,595	Specific QA
MusiqueQA	3,280,174	3,000	<u>29,898</u>	Specific QA
HotpotQA	8,495,056	3,702	66,581	Specific QA
ALCE	13,490,670	948	<u>89,562</u>	Specific QA
Mix	611,602	125	61	Abstract QA
MultihopSum	1,434,889	125	609	Abstract QA
Agriculture	1,949,584	125	12	Abstract QA
CS	2,047,923	125	10	Abstract QA
Legal	4,774,255	125	94	Abstract QA

7 EXPERIMENTS

We now present the experimental results. Section 7.1 discusses the setup. We discuss the results for specific QA and abstract QA tasks in Sections 7.2 and 7.3, respectively.

7.1 Setup

► **Workflow of our evaluation.** We present the first open-source testbed for graph-based RAG methods, which (1) collects and implements 12 representative methods within a unified framework (as depicted in Section 3). (2) supports a fine-grained comparison over the building blocks of the retrieval stage with up to 100+ variants, and (3) provides a comprehensive evaluation over 11 datasets with various metrics in different scenarios. We summarize the workflow of our empirical study in [71].

► **Benchmark Dataset.** We employ 11 widely used real-world datasets [15, 24, 25, 32] to evaluate the performance of each graph-based RAG method. These datasets span various corpus domains and cover diverse task types.

- *Specific.* This group focuses on detail-oriented questions referencing specific entities (e.g., “Who won the 2024 U.S. presidential election?”). We divide them into two types based on complexity: *Simple* (answerable from one or two chunks without reasoning): Quality [65], PopQA [57], HotpotQA [90]) and *Complex* (requiring multi-hop reasoning and synthesis): MultihopQA [79], MusiqueQA [81], ALCE [20]).
- *Abstract.* Unlike the previous groups, the questions in this category are not centered on specific factual queries. Instead, they involve abstract, conceptual inquiries that encompass broader topics, summaries, or overarching themes.

An example of an abstract question is: “How does artificial intelligence influence modern education?”. The abstract question requires a high-level understanding of the dataset contents, including five datasets: Mix [69], MultihopSum [79], Agriculture [69], CS [69], and Legal [69].

Their statistics, including the numbers of tokens, and questions, and the question-answering (QA) types are reported in Table 4. For specific (both complex and simple) QA datasets, we use the questions provided by each dataset. While for abstract QA datasets, We follow the question generation method introduced in [15], using GPT-4o to generate 125 questions per dataset with controlled difficulty, which is also aligned with existing works [15, 19, 24, 67]. The details and prompt template used for question generation are provided in our technical report [71]. Note that MultihopQA and MultihopSum originate from the same source, but differ in the types of questions they include—the former focuses on complex QA tasks, while the latter on abstract QA tasks.

► **Evaluation Metric.** For the specific QA tasks, we use Accuracy and Recall to evaluate performance on the first five datasets based on whether gold answers are included in the generations instead of strictly requiring exact matching, following [57, 73]. For the ALCE dataset, answers are typically full sentences rather than specific options or words. Following existing works [20, 72], we use string recall (STRREC), string exact matching (STREM), and string hit (STRHIT) as evaluation metrics. For abstract QA tasks, we adopt four evaluation metrics following prior works [15, 24]: *Comprehensiveness*, *Diversity*, *Empowerment*, and *Overall*, which assess answer quality from different perspectives. We employ a head-to-head comparison strategy using GPT-4o as the evaluator. Specifically, for each pair of answers, the LLM is prompted to judge which one is better with respect to a given metric, rather than assigning explicit scores. This comparative approach is motivated by the strong performance of LLMs as evaluators of natural language generation, often matching or exceeding human judgments [82, 95]. We provide detailed descriptions of the evaluation protocol and example case studies for all four metrics in our technical report [71].

► **Implementation.** We implement all the algorithms in Python with our proposed unified framework and try our best to ensure a native and effective implementation. All experiments are run on 350 Ascend 910B-3 NPUs [34]. Besides, Zeroshot [5], and vanilla RAG (denoted by VanillaRAG [42]) are also included in our study, which typically represent the model’s inherent capability and the performance improvement brought by basic RAG, respectively. If a

Table 5: Comparison of methods on different datasets, where **Purple** denotes the best result, and **Orange** denotes the best result excluding the best one; For the three largest datasets, we replace the clustering method in RAPTOR from Gaussian Mixture to K-means, as the former fails to finish within two days; The results of this version (i.e., K-means) are marked with [†].

Method	MultihopQA		Quality	PopQA		MusiqueQA		HotpotQA		ALCE		
	Accuracy	Recall	Accuracy	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall	STRREC	STREM	STRHIT
ZeroShot	49.022	34.256	37.058	28.592	8.263	1.833	5.072	35.467	42.407	15.454	3.692	30.696
VanillaRAG	50.626	36.918	39.141	60.829	27.058	17.233	27.874	50.783	57.745	34.283	11.181	63.608
G-retriever	42.019	43.116	31.807	17.084	6.075	2.733	11.662	—	—	9.754	2.215	19.726
ToG	41.941	38.435	34.888	47.677	23.727	9.367	20.536	—	—	13.975	3.059	29.114
KGP	48.161	36.272	33.955	57.255	24.635	17.333	27.572	—	—	27.692	8.755	51.899
DALK	53.952	47.232	34.251	45.604	19.159	11.367	22.484	33.252	47.232	21.408	4.114	44.937
LLightRAG	44.053	35.528	34.780	38.885	16.764	9.667	19.810	34.144	41.811	21.937	5.591	43.776
GLightRAG	48.474	38.365	33.413	20.944	8.146	7.267	17.204	25.581	33.297	17.859	3.587	37.131
HLightRAG	50.313	41.613	34.368	41.244	18.071	11.000	21.143	35.647	43.334	25.578	6.540	50.422
FastGraphRAG	52.895	44.278	37.275	53.324	22.433	13.633	24.470	43.193	51.007	30.190	8.544	56.962
HippoRAG	53.760	47.671	48.297	59.900	24.946	17.000	28.117	50.324	58.860	23.357	6.962	43.671
LGraphRAG	55.360	50.429	37.036	45.461	18.657	12.467	23.996	33.063	42.691	28.448	8.544	54.747
RAPTOR	56.064	44.832	56.997	62.545	27.304	24.133[†]	35.595[†]	55.321[†]	62.424[†]	35.255[†]	11.076[†]	65.401[†]

method cannot finish in two days, we mark its result as N/A in the figures and “—” in the tables.

► **Hyperparameter Settings.** In our experiments, we use Llama-3-8B [14] as the default LLM, not only because it is the most widely adopted model in recent RAG studies [93], but also due to its strong capabilities in language understanding and reasoning [14], as well as its practical efficiency for deployment. For LLM, we set the maximum token length to 8,096, and use greedy decoding to generate one sample for the deterministic output. For each method requiring top- k selection (e.g., chunks or entities), we set $k = 4$ to accommodate the token length limitation. We use one of the most advanced text-encoding models, BGE-M3 [58], as the embedding model across all methods to generate embeddings for vector search. If an expert annotator pre-splits the dataset into chunks, we use those as they preserve human insight. Otherwise, following existing works [15, 24], we divide the corpus into 1,200-token chunks. For other hyperparameters of each method, we adopt the original code settings when available; otherwise, we reproduce them based on the configurations described in the corresponding papers.

7.2 Evaluation for specific QA

In this section, we evaluate the performance of different methods on specific QA tasks.

► **Exp.1. Overall performance.** We report the metric values of all algorithms on specific QA tasks in Table 5. We can make the following observations and analyses: (1) Generally, the RAG technique significantly enhances LLM performance across all datasets, and the graph-based RAG methods (e.g., HippoRAG and RAPTOR) typically exhibit higher accuracy than VanillaRAG. However, if the retrieved elements are not relevant to the given question, RAG may degrade the LLM’s accuracy. For example, on the Quality dataset, compared to Zeroshot, RAPTOR improves accuracy by 53.80%, while G-retriever decreases it by 14.17%. This is mainly because, for simple QA tasks, providing only entities and relationships from a subgraph is insufficient to answer such questions effectively.

(2) For specific QA tasks, retaining the original text chunks is crucial for accurate question answering, as the questions and answers in these datasets are derived from the text corpus. This may explain

why G-retriever, ToG, and DALK, which rely solely on graph structure information, perform poorly on most datasets. However, on MultihopQA, which requires multi-hop reasoning, DALK effectively retrieves relevant reasoning paths, achieving accuracy and recall improvements of 6.57% and 27.94% over VanillaRAG, respectively.

(3) If the dataset is pre-split into chunks by the expert annotator, VanillaRAG often performs better compared to datasets where chunks are split based on the token size, and we further investigate this phenomenon later in our technical report [71].

(4) RAPTOR often achieves the best performance among most datasets, especially for simple questions. For complex questions, RAPTOR also performs exceptionally well. This is mainly because, for such questions, high-level summarized information is crucial for understanding the underlying relationships across multiple chunks. Hence, as we shall see, LGraphRAG is expected to achieve similar results, as it also incorporates high-level information (i.e., a summarized report of the most relevant community for a given question). However, we only observe this effect on the MultihopQA dataset. For the other two complex QA datasets, LGraphRAG even underperforms compared to VanillaRAG. Meanwhile, RAPTOR still achieves the best performance on these two datasets. We hypothesize that this discrepancy arises from differences in how high-level information is retrieved (See Table 3).

(5) For the three largest datasets, the K-means [27]-based RAPTOR (denoted as RAPTOR-K) also demonstrates remarkable performance. This suggests that the clustering method used in RAPTOR merely impacts overall performance. This may be because different clustering methods share the same key idea: grouping similar items into the same cluster. Therefore, they may generate similar chunk clusters. We note that RAPTOR-K achieves comparable or even better performance than RAPTOR, and the detailed results are shown in technical report [71]. If RAPTOR does not finish constructing the graph within two days, we use RAPTOR-K instead.

Remark. We note that not all graph-based RAG methods consistently outperform the baseline VanillaRAG on every question. By carefully analyzing the failure case of the top-performing methods, we examine why HippoRAG, RAPTOR, and LGraphRAG sometimes fall short in specific QA tasks. In short, the main failure reasons are: incorrect or incomplete entity extraction and irrelevant chunk

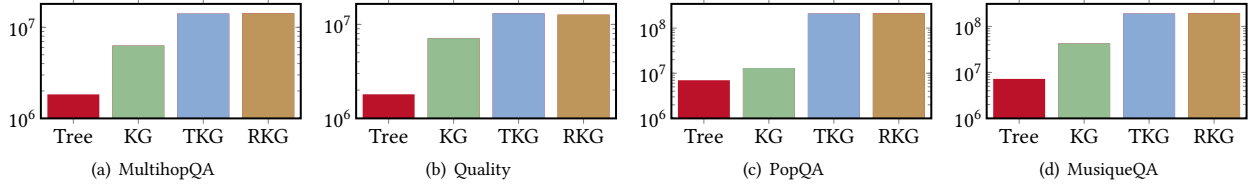


Figure 4: Token cost of graph building on specific QA datasets.

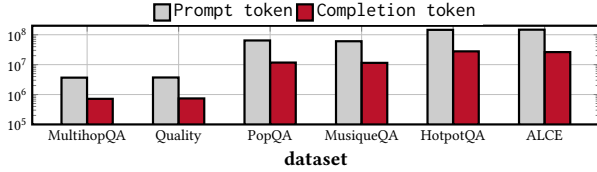


Figure 5: Token cost of index construction on specific QA. retrieval via Personalized PageRank for HippoRAG; low-quality cluster summaries for RAPTOR; and irrelevant community reports (via the Entity operator) and unrelated chunks (via the Occurrence operator) for LGraphRAG. Please see the failure cases in [71].

► **Exp.2. Token costs of graph and index building.** In this experiment, we first report the token costs of building four types of graphs across all datasets. Notably, building PG incurs no token cost, as it does not rely on the LLM for graph construction. We only report the results on four datasets here, and leave the remaining results in our technical report [71]. As shown in Figure 4(a) to (d), we observe the following: (1) Building trees consistently requires the least token cost, while TKG and RKG incur the highest token costs, with RKG slightly exceeding TKG. In some cases, RKG requires up to 40× more tokens than trees. (2) KG falls between these extremes, requiring more tokens than trees but fewer than TKG and RKG. This trend aligns with the results in Table 2, where graphs with more attributes require higher token costs for construction. (3) Recall that the token cost for an LLM call consists of two parts: the prompt token, which accounts for the tokens used in providing the input, and the completion part, which includes the tokens generated by the model as a response. We can see that regardless of the graph type, the prompt part always incurs higher token costs than the completion part; the detailed results are shown in [71].

We then examine the token costs of index building across all datasets. Since only LGraphRAG and GGraphRAG require an LLM for index construction, we report only the token costs for generating community reports in Figure 5. We can see that the token cost for index construction is nearly the same as that for building TKG. This is mainly because it requires generating a report for each community, and the number of communities is typically large, especially in large datasets. For example, the HotpotQA dataset contains 57,384 communities, significantly increasing the overall token consumption. That is to say, on large datasets, the two versions of GraphRAG often take more tokens than other methods in the offline stage.

► **Exp.3. Evaluation of the generation costs.** In this experiment, we evaluate the time and token costs for each method in specific QA tasks. Specifically, we report the average time and token costs for each query across all datasets in Table 6. We only report the results on three datasets here, and leave the remaining results in [71]. It is not surprising that ZeroShot and VanillaRAG are the most cost-efficient methods in terms of both time and token consumption. In terms of token cost, RAPTOR and HippoRAG are generally more efficient than other graph-based RAG methods, as

Table 6: Time and token costs of all methods on specific QA.

Method	MultihopQA		PopQA		ALCE	
	time	token	time	token	time	token
ZeroShot	3.23 s	270.3	1.17 s	82.2	2.41 s	177.2
VanillaRAG	2.35 s	3,623.4	1.41 s	644.1	1.04 s	849.1
G-retriever	6.87 s	1,250.0	37.51 s	3,684.5	101.16 s	5,096.1
ToG	69.74 s	16,859.6	42.02 s	11,224.2	34.94 s	11,383.2
KGP	38.86 s	13,872.2	37.49 s	6,738.9	105.09 s	9,326.6
DALK	28.03 s	4,691.5	16.33 s	2,496.5	17.04 s	4,071.9
LLightRAG	19.28 s	5,774.1	10.71 s	2,447.5	10.34 s	4,427.9
GLightRAG	18.37 s	5,951.5	12.10 s	3,255.6	13.02 s	4,028.1
HLightRAG	19.31 s	7,163.2	17.71 s	5,075.8	16.55 s	6,232.3
FastGraphRAG	7.17 s	5,874.8	13.25 s	6,157.0	25.82 s	6,010.9
HippoRAG	3.46 s	3,261.1	2.32 s	721.3	2.94 s	858.2
LGraphRAG	2.98 s	6,154.9	1.72 s	4,325.2	2.11 s	5,441.1
RAPTOR	3.18 s	3,210.0	1.36 s	1,188.3	1.54 s	793.6

they share a similar retrieval stage with VanillaRAG. The main difference lies in the chunk retrieval operators they use. Besides, KGP and ToG are the most expensive methods, as they rely on the agents (i.e., different roles of the LLM) for information retrieval during prompt construction. The former utilizes the LLM to reason the next required information based on the original question and retrieved chunks, while the latter employs LLM to select relevant entities and relationships for answering the question. On the other hand, the costs of LLightRAG, GLightRAG, and HLightRAG gradually increase, aligning with the fact that more information is incorporated into the prompt construction. All three methods are more expensive than LGraphRAG in specific QA tasks, as they use LLM to extract keywords in advance. Moreover, the time cost of all methods is proportional to the completion token cost. We present the results in our technical report [71], which explains why in some datasets, VanillaRAG is even faster than ZeroShot.

► **Exp.4. Detailed analysis for RAPTOR and LGraphRAG.** Due to space limitations, we highlight only the key insights derived from our analysis of RAPTOR and LGraphRAG. First, RAPTOR reveals that a significantly higher proportion of retrieved high-level information (i.e., content from non-leaf nodes) appears in complex QA tasks compared to simple ones, suggesting that high-level information is crucial for multi-hop reasoning. Second, we find that community reports serve as more effective high-level information than the chunk summaries used in RAPTOR, and the similarity-driven community retrieval strategy proves more robust than the Entity operator employed in LGraphRAG. Finally, for multi-hop reasoning tasks like MultihopQA, entity and relationship information provides valuable auxiliary signals that help the LLM connect relevant facts and guide the reasoning process. The detailed analysis and respective experiment results are shown in [71].

► **Exp.5. Effect of chunk size.** We evaluate the impact of chunk size on all RAG methods for specific QA by splitting the corpus into chunks of 600, 1200, and 2400 tokens. To this end, we construct three new datasets—PopAll, HotpotAll, and ALCEAll—by re-chunking the full corpora of PopQA, HotpotQA, and ALCE based on token

Table 7: Comparison of methods on different datasets under different chunk sizes.

Method	Chunk Size	MultihopQA		PopAll		HotpotAll		ALCEAll		
		Accuracy	Recall	Accuracy	Recall	Accuracy	Recall	STRREC	STREM	STRHIT
VanillaRAG	600	54.421	42.740	57.255	24.171	49.190	56.935	30.174	8.333	57.911
	1,200	50.626	36.918	57.041	25.877	44.254	52.511	29.334	8.228	56.329
	2,400	50.665	37.172	47.677	19.122	27.553	34.293	26.350	7.490	51.371
DALK	600	55.986	48.202	41.243	16.131	32.766	41.737	21.734	4.536	44.304
	1,200	53.952	47.232	42.602	17.024	30.416	39.544	21.327	4.430	43.987
	2,400	53.208	46.829	45.318	18.651	28.633	37.826	20.350	4.430	41.456
HippoRAG	600	47.144	41.210	62.401	26.892	50.783	58.454	27.025	8.122	51.477
	1,200	53.760	47.671	60.472	25.041	55.267	62.862	21.633	5.696	41.561
	2,400	52.152	46.601	50.751	19.986	45.624	53.597	26.477	6.118	52.848
LGraphRAG	600	55.282	46.267	53.181	26.292	41.194	49.801	33.692	10.971	62.447
	1,200	55.360	50.429	39.814	17.998	30.686	38.824	27.785	8.017	52.954
	2,400	54.930	44.588	43.317	20.185	37.061	45.366	28.398	7.806	54.008
RAPTOR	600	56.729	46.358	61.830	28.176	56.132	63.584	35.111	12.236	63.186
	1,200	56.064	44.832	47.963	21.399	31.983	39.864	34.044	10.971	62.342
	2,400	56.299	44.610	48.177	21.289	31.983	39.122	33.432	10.654	61.181

Table 8: Comparison of our newly designed methods on specific datasets with complex questions.

Dataset	Metric	LGraphRAG	RAPTOR	VGraphRAG
MultihopQA	Accuracy	55.360	56.064	59.664
	Recall	50.429	44.832	50.893
MusiqueQA	Accuracy	12.467	24.133	26.933
	Recall	23.996	35.595	40.026
ALCE	STRREC	28.448	35.255	41.023
	STREM	8.544	11.076	15.401
	STRHIT	54.747	65.401	71.835

length. This re-chunking is necessary because these datasets are pre-split by expert annotators, which may not accurately reflect the effects of chunk sizes. We only report the results of five methods in Table 7, and present the remaining results in [71]. We can see that: (1) For simple QA datasets (e.g., PopAll and HotpotAll), smaller chunk sizes generally yield better performance. This is because such questions often require information that is directly available in a single chunk or two. Smaller chunks provide more focused and precise context, improving answer accuracy. (2) We note that performance on simple QA tasks is highly sensitive to chunk size, while for complex QA tasks, performance remains relatively stable across different chunk sizes. This is because complex questions typically require reasoning across multiple chunks, making them less dependent on individual chunk granularity.

► **Exp.6. New SOTA algorithm.** Based on the above analysis, we aim to develop a new state-of-the-art method for complex QA datasets, denoted as VGraphRAG. Specifically, our algorithm first retrieves the top- k entities and their corresponding relationships, this step is the same as LGraphRAG. Next, we adopt the vector search-based retrieval strategy to select the most relevant communities and chunks. Then, by combining the four elements above, we construct the final prompt of our method to effectively guide the LLM in generating accurate answers. The results are also shown in Table 8, we can see that VGraphRAG performs best on all complex QA datasets. For example, compared to RAPTOR, our new algorithm VGraphRAG improves Accuracy by 6.42% on the MultihopQA dataset and 11.6% on the MusiqueQA dataset, respectively.

Table 9: Effect of LLM backbones for specific QA task.

Method	LLM backbone	MultihopQA		ALCEAll		
		Accuracy	Recall	STRREC	STREM	STRHIT
ZeroShot	Llama-3-8B	49.022	34.256	15.454	3.692	30.696
	Qwen-2.5-32B	45.070	33.332	30.512	10.127	56.118
	Llama-3-70B	55.908	52.987	31.234	7.170	61.920
	GPT-4o-mini	59.546	48.322	34.965	10.232	66.245
VanillaRAG	Llama-3-8B	50.626	36.918	29.334	8.228	56.329
	Qwen-2.5-32B	56.299	47.660	39.490	14.873	69.937
	Llama-3-70B	56.768	49.127	34.961	9.810	68.038
	GPT-4o-mini	59.311	47.941	35.735	10.127	68.249
HLightRAG	Llama-3-8B	50.313	41.613	22.475	6.329	43.776
	Qwen-2.5-32B	53.678	51.403	34.168	10.971	63.819
	Llama-3-70B	57.081	54.510	29.548	8.228	57.911
	GPT-4o-mini	55.829	46.424	41.334	15.506	71.730
HippoRAG	Llama-3-8B	53.760	47.671	21.633	5.696	41.561
	Qwen-2.5-32B	48.083	40.488	37.419	13.397	66.245
	Llama-3-70B	57.277	57.736	32.904	9.916	32.534
	GPT-4o-mini	67.723	55.482	39.274	12.447	72.046
RAPTOR	Llama-3-8B	56.064	44.832	34.044	10.971	62.342
	Qwen-2.5-32B	60.485	56.359	39.267	13.924	70.359
	Llama-3-70B	63.028	61.042	37.286	12.236	68.671
	GPT-4o-mini	60.603	51.521	29.770	8.017	58.861
VGraphRAG	Llama-3-8B	59.664	50.893	35.213	11.603	64.030
	Qwen-2.5-32B	57.277	55.151	39.234	14.557	69.831
	Llama-3-70B	67.567	68.445	37.576	12.447	69.198
	GPT-4o-mini	68.193	56.564	43.963	18.038	74.473

► **Exp.7. Effect of LLM backbone.** We evaluate the impact of different LLM backbones—Llama-3-8B [14], Qwen-2.5-32B [88], Llama-3-70B [14], and GPT-4o-mini—on the MultihopQA and ALCEAll datasets. The main results are shown in Table 9, while the remaining parts are presented in [71]. We make the following observations: (1) Stronger models generally yield better performance, especially in the ZeroShot setting, which most directly reflects the inherent capabilities of the underlying LLM. (2) The three variants of LLightRAG, LLightRAG, GLightRAG, and HLighRAG as well as LGraphRAG, achieve significant performance improvements when using more powerful LLMs. This can be attributed to their reliance on Rich Knowledge Graphs and Textual Knowledge Graphs, where stronger LLMs contribute to the construction of higher-quality graphs. (3) HippoRAG shows notably superior performance when using GPT-4o-mini compared to other LLM backbones. We attribute this to GPT-4o-mini’s ability to extract more accurate entities from the question and to construct higher-quality knowledge graphs,

thereby improving the retrieval of relevant chunks and the final answer accuracy. (4) Regardless of the LLM backbone, our proposed method VGraphRAG consistently achieves the best performance, demonstrating the advantages of our proposed unified framework.

7.3 Evaluation for abstract QA

In this section, we evaluate the performance of different methods on abstract QA tasks.

► **Exp.1. Overall Performance.** We evaluate the performance of methods that support abstract QA (see Table 1) by presenting head-to-head win rate percentages, comparing the performance of each row method against each column method. Here, we denote VR, RA, GS, LR, and FG as VanillaRAG, RAPTOR, GGraphRAG with high-layer communities (i.e., two-layer for this original implementation), HLightRAG and FastGraphRAG, respectively. The results are shown in Figure 6 to Figure 10, and we can see that: (1) Graph-based RAG methods often outperform VanillaRAG, primarily because they effectively capture inter-connections among chunks. (2) Across all four metrics, GGraphRAG stands out across all metrics. It achieves the highest Comprehensiveness by leveraging community-level retrieval to reduce fragmented evidence and capture broader context. For *Diversity*, both RAPTOR and GGraphRAG perform well by aggregating content across clusters or communities, covering a wide range of subtopics. On *Empowerment*, GGraphRAG and LightRAG lead by integrating structured elements such as entities and relations, helping the LLM generate more grounded and actionable answers. Overall, GGraphRAG consistently ranks first, with RAPTOR typically second, demonstrating the value of high-level summaries and the effectiveness of Map-Reduce.

► **Exp.2. Evaluation of the generation costs.** In this experiment, we present the time and token costs for each method in abstract QA tasks. As shown in Table 10, GGraphRAG is the most expensive method, as expected, while other graph-based methods exhibit comparable costs, although they are more expensive than VanillaRAG. For example, on the MutihopSum dataset, GGraphRAG requires $57 \times$ more time and $210 \times$ more tokens per query compared to VanillaRAG. Specifically, each query in GGraphRAG takes around 9 minutes and consumes 300K tokens, making it impractical for real-world scenarios. This is because, to answer an abstract question, GGraphRAG needs to analyze all retrieved communities, which is highly time- and token-consuming, especially when the number of communities is large (e.g., in the thousands).

► **Exp.3. New SOTA algorithm.** While the GGraphRAG shows remarkable performance in abstract QA, its time and token costs are not acceptable in practice, since given a question Q , GGraphRAG needs to use LLM to analyze all communities via Map-Reduce. (See Section 6.3) To alleviate this issue, we propose a cost-efficient variant of GGraphRAG, named CheapRAG. Instead of applying the LLM to analyze all communities, CheapRAG first computes the vector similarity between each community and the query to filter out irrelevant ones. It then applies the LLM only to the most relevant communities, significantly reducing token costs compared to GGraphRAG. Moreover, we observe that many top-performing methods, such as RAPTOR, HLightRAG, and GGraphRAG, all leverage the original chunks. This suggests that original chunks remain useful for certain questions. Hence, CheapRAG also incorporates original chunks into its retrieval process. After retrieving the top- k most relevant

communities and chunks, CheapRAG adopts a Map-Reduce strategy: the LLM generates partial answers for each selected community and chunk independently, and then summarizes them into a final response. As shown in Figure 11 and Table 10, CheapRAG not only achieves better performance than GGraphRAG but also significantly reduces token costs (in most cases). For example, on the MutihopSum dataset, CheapRAG reduces token costs by $100\times$ compared to GGraphRAG, while achieving better answer quality. We leave improving the answer diversity of CheapRAG to future work.

► **Exp.4. Effect of chunk size and LLM backbone.** We also study the impact of chunk size and LLM backbone on abstract QA tasks, following the same experimental setup as in Section 7.2. Due to space limitations, we report the results based on the “Overall” metric in Figure 12, with additional details provided in [71]. Our key observations are as follows: (1) The performance of GGraphRAG remains stable across different chunk sizes, likely due to its use of the Map-Reduce strategy for final answer synthesis, which mitigates the influence of chunk granularity. (2) In contrast, methods like FastGraphRAG and VanillaRAG show greater variance across chunk sizes, as their performance relies heavily on the granularity of individual chunks—smaller chunks tend to provide more precise information, directly impacting retrieval and generation quality. (3) Regardless of chunk size, RAPTOR and GGraphRAG consistently achieve the best performance, reaffirming our earlier conclusion that high-level structural information is essential for abstract QA tasks. (4) All methods still lag behind GGraphRAG, further highlighting that community-level information is particularly beneficial for abstract QA tasks. In addition, we evaluate our newly proposed method CheapRAG against the baselines under varying chunk sizes and LLM backbones. As shown in [71], CheapRAG consistently achieves the best performance across all settings.

8 LESSONS AND OPPORTUNITIES

We summarize the lessons (L) for practitioners and propose practical research opportunities (O) based on our observations.

Lessons:

► **L1.** In Figure 13, we depict a roadmap of the recommended RAG methods, highlighting which methods are best suited for different scenarios. **It is derived from all conducted experiments, which is an overall conclusion for both graph-based RAG methods.**

► **L2.** Chunk quality is critical to the overall performance of RAG methods, and human experts typically produce more effective chunking than approaches based solely on token length. **(See results in Table 7 and Figure 12.)**

► **L3.** For complex questions in specific QA, high-level information is typically needed, as they capture the complex relationship among chunks, and the vector search-based retrieval strategy is better than the rule-based (e.g., Entity operator) one. **This lesson is supported by the results in Tables 5 and 8.**

► **L4.** Community reports provide a more effective high-level structure than summarized chunk clusters for abstract QA tasks, as they better capture diversified topics and overarching themes within local modules of the corpus. **(See results in Figures 6 ~ 12).**

► **L5.** Original chunks are useful for all QA tasks, as they provide essential textual descriptions for augmenting or completing information needed to answer questions. When attempting to design new graph-based RAG methods, incorporating the relevant original

	VR	RA	GS	LR	FG
VR	50	58	30	36	93
RA	42	50	39	26	82
GS	70	61	50	15	89
LR	64	74	85	50	98
FG	7	18	11	2	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	66	58	35	90
RA	34	50	54	20	76
GS	42	46	50	26	86
LR	65	80	74	50	96
FG	10	24	14	4	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	60	29	28	92
RA	40	50	45	22	82
GS	71	54	50	12	88
LR	72	78	88	50	98
FG	8	18	12	2	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	60	19	32	93
RA	40	50	44	24	82
GS	81	56	50	14	88
LR	68	76	86	50	98
FG	7	18	12	2	50

(d) Overall

Figure 6: The abstract QA results on Mix dataset.

	VR	RA	GS	LR	FG
VR	50	50	2	46	95
RA	50	50	47	48	94
GS	78	53	50	79	96
LR	54	52	21	50	92
FG	5	6	4	8	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	64	58	64	93
RA	36	50	42	49	85
GS	42	55	50	52	92
LR	36	51	48	50	88
FG	7	15	8	12	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	52	36	39	95
RA	48	50	45	45	93
GS	64	54	50	41	97
LR	61	55	59	50	95
FG	5	7	3	5	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	52	44	46	95
RA	48	50	45	47	94
GS	56	55	50	52	97
LR	54	53	48	50	93
FG	5	6	3	7	50

(d) Overall

Figure 7: The abstract QA results on MultihopSum dataset.

	VR	RA	GS	LR	FG
VR	50	32	39	54	85
RA	68	50	19	73	94
GS	61	81	50	62	89
LR	46	27	38	50	78
FG	15	6	11	22	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	32	45	59	77
RA	68	50	16	76	90
GS	55	84	50	63	82
LR	41	24	37	50	71
FG	23	10	18	29	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	24	41	52	85
RA	76	50	22	76	96
GS	59	78	50	58	91
LR	48	24	42	50	81
FG	15	4	9	19	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	30	38	53	85
RA	70	50	16	76	95
GS	62	84	50	62	90
LR	47	24	38	50	79
FG	15	5	10	21	50

(d) Overall

Figure 8: The abstract QA results on Agriculture dataset.

	VR	RA	GS	LR	FG
VR	50	22	25	36	80
RA	78	50	55	69	99
GS	75	45	50	64	97
LR	64	31	36	50	95
FG	20	1	3	5	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	18	25	37	75
RA	82	50	51	79	99
GS	75	49	50	63	91
LR	63	21	37	50	93
FG	25	1	9	7	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	15	24	29	80
RA	85	50	59	72	100
GS	76	41	50	60	96
LR	71	28	40	50	98
FG	20	0	4	2	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	15	22	30	79
RA	85	50	54	73	67
GS	78	46	50	62	97
LR	70	27	38	50	97
FG	21	33	3	3	50

(d) Overall

Figure 9: The abstract QA results on CS dataset.

	VR	RA	GS	LR	FG
VR	50	26	31	41	93
RA	74	50	27	67	95
GS	69	73	50	62	97
LR	59	33	38	50	97
FG	7	5	3	3	50

(a) Comprehensiveness

	VR	RA	GS	LR	FG
VR	50	36	32	45	90
RA	64	50	68	68	93
GS	68	33	50	66	94
LR	55	32	34	50	93
FG	10	7	6	7	50

(b) Diversity

	VR	RA	GS	LR	FG
VR	50	24	29	34	95
RA	76	50	31	67	96
GS	71	69	50	60	96
LR	66	33	40	50	97
FG	5	4	4	3	50

(c) Empowerment

	VR	RA	GS	LR	FG
VR	50	26	30	37	94
RA	74	50	31	67	96
GS	70	69	50	62	96
LR	63	33	38	50	97
FG	6	4	4	3	50

(d) Overall

Figure 10: The abstract QA results on Legal dataset.

	Com.	Div.	Emp.	Overall
VR	68	56	55	62
RA	77	56	64	70
GS	73	65	66	69
LR	68	22	34	44
FG	97	81	92	93

(a) Mix

	Com.	Div.	Emp.	Overall
VR	74	47	64	68
RA	68	48	64	67
GS	72	43	65	68
LR	66	45	56	64
FG	100	86	98	99

(b) MultihopSum

	Com.	Div.	Emp.	Overall
VR	84	54	66	75
RA	68	42	43	54
GS	81	38	62	71
LR	84	54	62	74
FG	99	79	94	94

(c) Agriculture

	Com.	Div.	Emp.	Overall
VR	86	47	71	78
RA	66	19	39	49
GS	66	21	48	51
LR	80	34	54	66
FG	98	79	95	96

(d) CS

	Com.	Div.	Emp.	Overall
VR	82	45	62	68
RA	63	27	40	46
GS	64	26	38	46
LR	76	42	44	59
FG	97	90	94	97

(e) Legal

Figure 11: Comparison of our newly designed method on abstract QA datasets.

	VR	RA	GS	LR	FG
VR	50	27	30	78	67
RA	73	50	54	90	92
GS	70	46	50	86	71
LR	22	10	14	50	29
FG	33	8	19	71	50

(a) chunk size=600

	VR	RA	GS	LR	FG
VR	50	23	37	62	68
RA	77	50	66	83	90
GS	63	34	50	70	72
LR	38	17	30	50	54
FG	32	10	28	46	50

(b) chunksize=2400

	VR	RA	GS	LR	FG
VR	50	19	10	18	68
RA	81	50	24	61	88
GS	90	76	50	80	93
LR	82	39	20	50	91
FG	32	12	7	9	50

(c) Qwen-2.5-32B

	VR	RA	GS	LR	FG
VR	50	12	6	18	78
RA	88	50	30	76	97
GS	94	70	50	83	99
LR	82	24	17	50	95
FG	22	3	1	5	50

(d) Llama-3-70B

	VR	RA	GS	LR	FG
VR	50	72	4	13	60
RA	28	50	4	10	37
GS	96	96	50	90	95
LR	87	90	10	50	87
FG	40	63	5	13	50

(e) GPT-4o-mini

Figure 12: Performance with different chunk sizes and LLM backbones on the MultihopSum dataset.

Table 10: The average time and token costs on abstract QA datasets.

Dataset	VanillaRAG		RAPTOR		GGraphRAG		HLightRAG		FastGraphRAG		CheapRAG	
	time	token	time	token	time	token	time	token	time	token	time	token
Mix	18.7 s	4,114	35.5 s	4,921	72.2 s	10,922	22.6 s	5,687	20.9 s	4,779	27.3 s	11,720
MultihopSum	9.1 s	1,680	32.7 s	4,921	521.0 s	353,889	33.7 s	5,329	34.4 s	5,839	54.1 s	3,784
Agriculture	17.4 s	5,091	20.7 s	3,753	712.3 s	448,762	25.3 s	4,364	28.8 s	5,640	47.1 s	10,544
CS	17.8 s	4,884	32.7 s	4,921	442.0 s	322,327	51.4 s	4,908	28.2 s	5,692	48.8 s	17,699
Legal	26.2 s	2,943	59.8 s	3,573	231.2 s	129,969	31.1 s	4,441	34.0 s	5,411	34.8 s	14,586

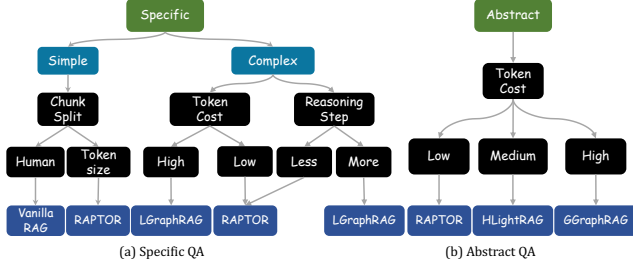


Figure 13: The taxonomy tree of RAG methods.

chunks is not a bad idea. Based on the results in Tables 5 and 8, and Figures 6 ~ 12, we conclude this lesson.

Opportunities:

► **O1.** All existing graph-based RAG methods (both specific QA and abstract QA) assume the setting of the external corpus is static. What if the external knowledge source evolves over time? For example, Wikipedia articles are constantly evolving, with frequent updates to reflect new information. Can we design graph-based RAG methods that efficiently and effectively adapt to such dynamic changes in external knowledge sources?

► **O2.** The quality of a graph plays a key role in determining the effectiveness of graph-based RAG methods. However, evaluating graph quality before actually handling a question remains a critical challenge that needs to be addressed. Existing graph construction methods consume a substantial number of tokens and often produce graphs with redundant entities or miss potential relationships, so designing a cost-efficient yet effective construction method is a meaningful research direction.

► **O3.** In many domains, the corpus is private (e.g., finance, legal, and medical), and retrieving the relevant information from such corpus can reveal information about the knowledge source. Designing a graph-based RAG method that incorporates local differential privacy is an interesting research problem.

► **O4.** How to use LLMs and graph-based RAG methods to facilitate query optimization, such as generating efficient query plans and execution strategies, for database systems. More details are in our technical report [71].

9 RELATED WORKS

In this section, we mainly review the related works of existing RAG methods. We also present the applications of RAG in various areas, particularly in data management area.

• **RAG methods.** RAG has been proven to be very effective in migrating the “hallucination” of LLMs [5, 7–9, 35, 75]. Recently, most RAG approaches [15, 24, 25, 44, 67, 72, 85, 86] have adopted graph as the external knowledge to organize the information and relationships within documents, achieving improved overall retrieval

performance, which is extensively reviewed in this paper. Nevertheless, there is a lack of a comprehensive work comparison between all graph-based RAG methods in terms of accuracy and efficiency. We note that there exists an empirical study [26] that compares Microsoft’s methods (LGraphRAG and GGraphRAG) with the standard VanillaRAG, and a few survey papers on graph-based RAG systems [37, 92]. However, our work differs significantly from these in both scope and depth. First, our work focus on systematically comparing the different graph-based RAG methods, and conduct a stage-wise comparison across a unified framework. This allows us to identify core design principles and enables the construction of a new state-of-the-art method through component recombination. Second, unlike survey papers that offer high-level overviews, our work provides deep empirical analysis and practical insights grounded in extensive experiments.

• **RAG applications.** Due to the wealth of developer experience captured in a vast array of database forum discussions, recent studies [7, 16, 23, 41, 48, 74, 78, 98, 99] have begun leveraging RAGs to enhance database performance. For instance, GPTuner [41] proposes to enhance database knob tuning using RAG by leveraging domain knowledge to identify important knobs and coarsely initialize their values for subsequent refinement. Besides, D-Bot [98] proposes an LLM-based database diagnosis system, which can retrieve relevant knowledge chunks and tools, and use them to identify typical root causes accurately. In addition, RAG-based SQL rewriting systems [48, 76, 78] have recently attracted significant attention. These methods retrieve domain-specific knowledge from database forums and official documentation to enhance SQL rewriting and optimization. The RAG-based data analysis systems have also been studied [3, 10, 49–53, 66]. For applications in other areas, we refer readers to recent RAG surveys [31, 93].

10 CONCLUSIONS

In this paper, we provide an in-depth experimental evaluation and comparison of existing graph-based Retrieval-Augmented Generation (RAG) methods. We first provide a novel unified framework, which can cover all the existing graph-based RAG methods, using an abstraction of a few key operations. We then thoroughly analyze and compare different graph-based RAG methods under our framework. We further systematically evaluate these methods from different angles using various datasets for both specific and abstract question-answering (QA) tasks, and also develop variations by combining existing techniques, which often outperform state-of-the-art methods. From extensive experimental results and analysis, we have identified several important findings and analyzed the critical components that affect the performance. In addition, we have summarized the lessons learned and proposed practical research opportunities that can facilitate future studies.

REFERENCES

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Floren-
cia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal
Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*
(2023).
- [2] Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit
Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. 2025.
Cache-Craft: Managing Chunk-Caches for Efficient Retrieval-Augmented Gener-
ation. *arXiv preprint arXiv:2502.15734* (2025).
- [3] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry
Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A Shah, et al. 2024.
The Design of an LLM-powered Unstructured Analytics System. *arXiv preprint*
arXiv:2409.00847 (2024).
- [4] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi.
2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection.
arXiv preprint arXiv:2310.11511 (2023).
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan,
Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda
Askell, et al. 2020. Language models are few-shot learners. *Advances in neural*
information processing systems 33 (2020), 1877–1901.
- [6] Chat2Graph. 2025. About Chat2Graph: Graph Native Agentic System. <https://chat2graph.vercel.app/chat2graph/en-us/introduction>.
- [7] Sibe Chen, Ju Fan, Bin Wu, Nan Tang, Chao Deng, Pengyi Wang, Ye Li, Jian Tan,
Feifei Li, Jingren Zhou, et al. 2024. Automatic Database Configuration Debugging
using Retrieval-Augmented Language Models. *arXiv preprint arXiv:2412.07548*
(2024).
- [8] Sibe Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei
Zhang, and Surajit Chaudhuri. 2024. Auto-Formula: Recommend Formulas in
Spreadsheets using Contrastive Learning for Table Representations. *Proceedings*
of the ACM on Management of Data 2, 3 (2024), 1–27.
- [9] Sibe Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xi-
aoyong Du. 2023. Haipipe: Combining human-generated and machine-generated
pipelines for data preparation. *Proceedings of the ACM on Management of Data* 1,
1 (2023), 1–26.
- [10] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang,
Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. SEED: Domain-Specific
Data Curation With Large Language Models. *arXiv e-prints* (2023), arXiv–2310.
- [11] Databricks. 2024. Building, Improving, and Deploying Knowledge Graph
RAG Systems on Databricks. [https://neo4j.com/docs/neo4j-graphrag-python/
current/](https://neo4j.com/docs/neo4j-graphrag-python/current/).
- [12] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for
language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [13] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia,
Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning.
arXiv preprint arXiv:2301.00234 (2022).
- [14] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad
Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan,
et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).
- [15] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva
Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph
rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*
(2024).
- [16] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li,
Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language
models and large language models for zero-shot nl2sql. *Proceedings of the VLDB*
Endowment 17, 11 (2024), 2750–2763.
- [17] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and
Xiaoyong Du. 2024. Cost-effective in-context learning for entity resolution: A
design space exploration. In *2024 IEEE 40th International Conference on Data*
Engineering (ICDE). IEEE, 3696–3709.
- [18] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin,
Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards
retrieval-augmented large language models. In *Proceedings of the 30th ACM*
SIGKDD Conference on Knowledge Discovery and Data Mining. 6491–6501.
- [19] FastGraphRAG. 2024. FastGraphRAG. [https://github.com/circlemind-ai/fast-
graphrag](https://github.com/circlemind-ai/fast-graphrag).
- [20] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large
language models to generate text with citations. *arXiv preprint arXiv:2305.14627*
(2023).
- [21] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai,
Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large
language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
- [22] Aashish Ghimire, James Prather, and John Edwards. 2024. Generative AI in
Education: A Study of Educators’ Awareness, Sentiments, and Influencing Factors.
arXiv preprint arXiv:2403.15586 (2024).
- [23] Victor Giannakouris and Immanuel Trummer. 2024. $\{\lambda\}$ -Tune: Har-
nessing Large Language Models for Automated Database System Tuning. *arXiv*
preprint arXiv:2411.03500 (2024).
- [24] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. LightRAG:
Simple and Fast Retrieval-Augmented Generation. *arXiv e-prints* (2024), arXiv–
2410.
- [25] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su.
2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large
Language Models. *arXiv preprint arXiv:2405.14831* (2024).
- [26] Haoyu Han, Harry Shomer, Yu Wang, Yongjia Lei, Kai Guo, Zhigang Hua, Bo
Long, Hui Liu, and Jiliang Tang. 2025. Rag vs. graphrag: A systematic evaluation
and key insights. *arXiv preprint arXiv:2502.11371* (2025).
- [27] Jiawei Han, Jian Pei, and Hanghang Tong. 2022. *Data mining: concepts and*
techniques. Morgan kaufmann.
- [28] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th*
international conference on World Wide Web. 517–526.
- [29] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann
LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented
generation for textual graph understanding and question answering. *arXiv*
preprint arXiv:2402.07630 (2024).
- [30] Yucheng Hu and Yuxing Lu. 2024. Rag and rau: A survey on retrieval-augmented
language model in natural language processing. *arXiv preprint arXiv:2404.19543*
(2024).
- [31] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Hao-
tian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al.
2023. A survey on hallucination in large language models: Principles, taxonomy,
challenges, and open questions. *arXiv preprint arXiv:2311.05232* (2023).
- [32] Yizheng Huang and Jimmy Huang. 2024. A Survey on Retrieval-Augmented Text
Generation for Large Language Models. *arXiv preprint arXiv:2404.10981* (2024).
- [33] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. KET-RAG: A Cost-
Efficient Multi-Granular Indexing Framework for Graph-RAG. *arXiv preprint*
arXiv:2502.09304 (2025).
- [34] huawei. 2019. Ascend GPU. [https://e.huawei.com/ph/products/computing/
ascend](https://e.huawei.com/ph/products/computing/ascend).
- [35] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park.
2024. Adaptive-rag: Learning to adapt retrieval-augmented large language mod-
els through question complexity. *arXiv preprint arXiv:2403.14403* (2024).
- [36] Wenqi Jiang, Marco Zeller, Roger Waleffe, Torsten Hoefler, and Gustavo Alonso.
2024. Chameleon: A Heterogeneous and Disaggregated Accelerator System for
Retrieval-Augmented Language Models. *Proceedings of the VLDB Endowment* 18,
1 (2024), 42–52.
- [37] Vikas Kamra, Lakshya Gupta, Dhruv Arora, and Ashwin Kumar Yadav. 2024.
Enhancing Document Retrieval Using AI and Graph-Based RAG Techniques. In
2024 5th International Conference on Communication, Computing & Industry 6.0
(C2I6). IEEE, 1–7.
- [38] Omar Khattab, Arnab Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav
Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi,
Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls
into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).
- [39] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage
search via contextualized late interaction over bert. In *Proceedings of the 43rd*
International ACM SIGIR conference on research and development in Information
Retrieval. 39–48.
- [40] LangGraph. 2024. LangGraph. <https://www.langchain.com/langgraph>.
- [41] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng,
Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2024. Gptuner: A manual-
reading database tuning system via gpt-guided bayesian optimization. *Proced-*
ings of the VLDB Endowment 17, 8 (2024), 1939–1952.
- [42] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir
Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rock-
täschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp
tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
- [43] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The
Dawn of Natural Language to SQL: Are We Fully Ready? *arXiv preprint*
arXiv:2406.01265 (2024).
- [44] Dawei Li, Shu Yang, Zhen Tan, Jae Young Baik, Sukwon Yun, Joseph Lee, Aaron
Chacko, Bojian Hou, Duy Duong-Tran, Ying Ding, et al. 2024. DALK: Dynamic
Co-Augmentation of LLMs and KG to answer Alzheimer’s Disease Questions
with Scientific Literature. *arXiv preprint arXiv:2405.04819* (2024).
- [45] Lan Li, Liri Fang, and Vette I Torvik. 2024. AutoDCWorkflow: LLM-based
Data Cleaning Workflow Auto-Generation and Benchmark. *arXiv preprint*
arXiv:2412.06724 (2024).
- [46] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language
models in finance: A survey. In *Proceedings of the fourth ACM international*
conference on AI in finance. 374–382.
- [47] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024.
LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for
Boosting Query Efficiency. *arXiv preprint arXiv:2404.12872* (2024).
- [48] Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2025.
LLM-R2: A Large Language Model Enhanced Rule-based Rewrite System for
Boosting Query Efficiency. *Proceedings of the VLDB Endowment* 1, 18 (2025),
53–65.
- [49] Chen Liang, Donghua Yang, Zheng Liang, Zhiyu Liang, Tianle Zhang, Boyu Xiao,
Yuqing Yang, Wenqi Wang, and Hongzhi Wang. 2025. Revisiting Data Analysis
with Pre-trained Foundation Models. *arXiv preprint arXiv:2501.01631* (2025).

- [50] Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. 2025. TWIX: Automatically Reconstructing Structured Data from Templated Documents. *arXiv preprint arXiv:2501.06659* (2025).
- [51] Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeigham, Aditya G Parameswaran, and Eugene Wu. 2024. Towards Accurate and Efficient Document Analytics with Large Language Models. *arXiv preprint arXiv:2405.04674* (2024).
- [52] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. *arXiv preprint arXiv:2405.14696* (2024).
- [53] Chunwei Liu, Gerardo Vitagliano, Brandon Rose, Matt Prinz, David Andrew Samson, and Michael Cafarella. 2025. PalimpChat: Declarative and Interactive AI analytics. *arXiv preprint arXiv:2502.03368* (2025).
- [54] Lei Liu, Xiaoyan Yang, Junchi Lei, Xiaoyang Liu, Yue Shen, Zhiqiang Zhang, Peng Wei, Jinjie Gu, Zhixuan Chu, Zhan Qin, et al. 2024. A Survey on Medical Large Language Models: Technology, Application, Trustworthiness, and Future Directions. *arXiv preprint arXiv:2406.03712* (2024).
- [55] llamaindex. 2023. llamaindex. <https://www.llamaindex.ai/>.
- [56] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
- [57] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511* (2022).
- [58] Multi-Linguality Multi-Functionality Multi-Granularity. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. (2024).
- [59] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using LLMs and Data Lakes. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4421–4424.
- [60] nebula. 2024. nebula. <https://www.databricks.com/blog/building-improving-and-deploying-knowledge-graph-rag-systems-databricks>.
- [61] neo4j. 2006. neo4j. <https://neo4j.com/>.
- [62] neo4j rag. 2024. Neo4j GraphRAG. <https://neo4j.com/docs/neo4j-graphrag-python/current/>.
- [63] Yuqi Nie, Yaxuan Kong, Xiaowen Dong, John M Mulvey, H Vincent Poor, Qingsong Wen, and Stefan Zohren. 2024. A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges. *arXiv preprint arXiv:2406.11903* (2024).
- [64] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.
- [65] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. 2021. QuALITY: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608* (2021).
- [66] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. Lotus: Enabling semantic queries with llms over tables of unstructured and structured data. *arXiv preprint arXiv:2407.11418* (2024).
- [67] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921* (2024).
- [68] PostgreSQL. 2024. Azure Database for PostgreSQL Blog. <https://techcommunity.microsoft.com/blog/adforpostgresql/introducing-the-graphrag-solution-for-azure-database-for-postgresql/4299871>.
- [69] Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. Memorag: Moving towards next-gen rag via memory-inspired knowledge discovery. *arXiv preprint arXiv:2409.05591* (2024).
- [70] Yichen Qian, Yongyi He, Rong Zhu, Jintao Huang, Zhijian Ma, Haibin Wang, Yaohua Wang, Xiuyu Sun, Defu Lian, Bolin Ding, et al. 2024. UniDM: A Unified Framework for Data Manipulation with Large Language Models. *Proceedings of Machine Learning and Systems* 6 (2024), 465–482.
- [71] The Technique Report. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework (technical report). https://github.com/JayLZhou/GraphRAG/blob/master/VLDB2025_GraphRAG.pdf.
- [72] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059* (2024).
- [73] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).
- [74] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gandghariaiah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents. (2024).
- [75] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kalurachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics* 11 (2023), 1–17.
- [76] Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2025. QUITTE: A Query Rewrite System Beyond Rules with LLM Agents. *arXiv preprint arXiv:2506.07675* (2025).
- [77] Jiashuo Sun, Chengjin Xu, Luminyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. In *The Twelfth International Conference on Learning Representations*.
- [78] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).
- [79] Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391* (2024).
- [80] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific reports* 9, 1 (2019), 1–12.
- [81] Harsh Trivedi, Niranjana Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics* 10 (2022), 539–554.
- [82] Jian Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. 2023. Is chatgpt a good nlg evaluator? a preliminary study. *arXiv preprint arXiv:2303.04048* (2023).
- [83] Jinqiang Wang, Huansheng Ning, Yi Peng, Qikai Wei, Daniel Tesfai, Wenwei Mao, Tao Zhu, and Runhe Huang. 2024. A Survey on Large Language Models from General Purpose to Medical Applications: Datasets, Methodologies, and Evaluations. *arXiv preprint arXiv:2406.10303* (2024).
- [84] Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105* (2024).
- [85] Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, 19206–19214.
- [86] Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, and Vicente Grau. 2024. Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation. *arXiv preprint arXiv:2408.04187* (2024).
- [87] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Mao, Xue Liu, Tei-Wei Kuo, Nan Guan, et al. 2024. Retrieval-augmented generation for natural language processing: A survey. *arXiv preprint arXiv:2407.13193* (2024).
- [88] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
- [89] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 Technical Report. *arXiv preprint arXiv:2412.15115* (2024).
- [90] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600* (2018).
- [91] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. 2024. Evaluation of Retrieval-Augmented Generation: A Survey. *arXiv preprint arXiv:2405.07437* (2024).
- [92] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *arXiv preprint arXiv:2501.13958* (2025).
- [93] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhen-gran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, et al. 2024. Raglab: A modular and research-oriented unified framework for retrieval-augmented generation. *arXiv preprint arXiv:2408.11381* (2024).
- [94] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).
- [95] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2023), 46595–46623.
- [96] Yanxin Zheng, Wensheng Gan, Zefeng Chen, Zhenlian Qi, Qian Liang, and Philip S Yu. 2024. Large language models for medicine: a survey. *International Journal of Machine Learning and Cybernetics* (2024), 1–26.
- [97] Yihang Zheng, Bo Li, Zhenghao Lin, Yi Luo, Xuanhe Zhou, Chen Lin, Jinsong Su, Guoliang Li, and Shifu Li. 2024. Revolutionizing Database Q&A with Large Language Models: Comprehensive Benchmark and Evaluation. *arXiv preprint arXiv:2409.04475* (2024).
- [98] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-bot: Database diagnosis

system using large language models. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2514–2527.

[99] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: Large language model meets database. *Data Science and Engineering* 9, 1 (2024), 102–111.

Table 11: Comparison RAPTOR and RAPTOR-K.

Method	MultihopQA		Quality	PopQA	
	Accuracy	Recall	Accuracy	Accuracy	Recall
RAPTOR	56.064	44.832	56.997	62.545	27.304
RAPTOR-K	56.768	44.208	54.567	64.761	28.469

A ADDITIONAL EXPERIMENTS

A.1 Results on specific QA tasks

In this Section, we present the additional results on the specific QA tasks.

► **Exp.1. Comparison RAPTOR-K and RAPTOR.** We compare RAPTOR-K with RAPTOR on the first three datasets, and present results in Table 11. We observe that RAPTOR-K achieves comparable or even better performance than RAPTOR.

► **Exp.2. Token costs of graph and index building.** We report the token costs of building four types of graphs across HotpotQA and ALCE datasets in Figure 14. Recall that the token cost for an LLM call consists of two parts: the prompt token, which accounts for the tokens used in providing the input, and the completion part, which includes the tokens generated by the model as a response. Here, we report the token costs for prompt and completion on HotpotQA and ALCE in Figure 14(c) to (d), and show the results on other datasets in Figure 15. We conclude that, regardless of the graph type, the prompt part always incurs higher token costs than the completion part.

► **Exp.3. Evaluation of the generation costs.** In this experiment, we evaluate the time and token costs for each method in specific QA tasks. Specifically, we report the average time and token costs for each query across all datasets in Table 12, the conclusions are consistent with those reported in our manuscript. As shown in Figure 16, we present the average token costs for prompt tokens and completion tokens across all questions in all specific QA datasets. We can observe that the running time of each method is highly proportional to the completion token costs, which aligns with the computational paradigm of the Transformer architecture.

► **Exp.4. Detailed analysis for RAPTOR and LGraphRAG.** Our first analysis about RAPTOR aims to explain why RAPTOR outperforms VanillaRAG. Recall that in RAPTOR, for each question Q , it retrieves the top- k items across the entire tree, meaning the retrieved items may originate from different layers. That is, we report the proportion of retrieved items across different tree layers in Table 13. As we shall see, for the MultihopQA and MusiqueQA datasets, the proportion of retrieved high-level information (i.e., items not from leaf nodes) is significantly higher than in other datasets. For datasets requiring multi-hop reasoning to answer questions, high-level information plays an essential role. This may explain why RAPTOR outperforms VanillaRAG on these two datasets.

We then conduct a detailed analysis of LGraphRAG on complex questions in specific QA datasets by modifying its retrieval methods or element types. By doing this, we create three variants of LGraphRAG, and we present the detailed descriptions for each variant in Table 14. Here, VGraphRAG-CC introduces a new retrieval strategy. Unlike LGraphRAG, it uses vector search to retrieve the top- k elements (i.e., chunks or communities) from the vector database. Eventually, we evaluate their performance on the three complex QA datasets and present the results in Table 15.

► **Exp.5. Effect of the chunk size.** Recall that our study includes some datasets that are pre-split by the export annotator. To

investigate this impact, we re-split the corpus into multiple chunks based on token size for these datasets instead of using their original chunks. Here, we create three new datasets from HotpotQA, PopQA, and ALCE, named HotpotAll, PopAll, and ALCEAll, respectively.

For each dataset, we use Original to denote its original version and New chunk to denote the version after re-splitting. We report the results of graph-based RAG methods on both the original and new version datasets in Figure 17, we can see that: (1) The performance of all methods declines, mainly because rule-based chunk splitting (i.e., by token size) fails to provide concise information as effectively as expert-annotated chunks. (2) Graph-based methods, especially those relying on TKG and RKG, are more sensitive to chunk quality. This is because the graphs they construct encapsulate richer information, and coarse-grained chunk splitting introduces potential noise within each chunk. Such noise can lead to inaccurate extraction of entities or relationships and their corresponding descriptions, significantly degrading the performance of these methods. (3) As for token costs, all methods that retrieve chunks incur a significant increase due to the larger chunk size in New chunk compared to Original, while other methods remain stable. These findings highlight that chunk segmentation quality is crucial for the overall performance of all RAG methods.

We evaluate all 12 RAG methods under chunk sizes of 600, 1200, and 2400 tokens, as shown in Table 16, and further select the top-performing eight methods to analyze how their performance varies with chunk size, as shown in Figure 18. We observe that the results across different settings remain largely consistent with the conclusions drawn using the default chunk size of 1200 in our submitted manuscript. Specifically, we make the following observations and analysis: (1) For simple QA datasets (e.g., PopAll and HotpotAll), smaller chunk sizes generally yield better performance. This is because such questions often require information that is directly available in a single chunk or two. Smaller chunks provide more focused and precise context, improving answer accuracy. (2) Across all chunk sizes, HippoRAG and RAPTOR perform best on simple QA tasks, while RAPTOR and LGraphRAG outperform others on complex QA datasets (e.g., MultihopQA, ALCEAll). These observations align with the results reported in our manuscript. (3) The effect of chunk size varies by task type: performance on simple QA tasks is highly sensitive to chunk size, while for complex QA tasks, performance remains relatively stable across different chunk sizes. This is because complex questions typically require reasoning across multiple chunks, making them less dependent on individual chunk granularity.

► **Exp.6. Effect of the LLM backbones.** For each system, we evaluate multiple LLM backbones, including Llama-3-8B, Qwen-2.5-32B, Llama-3-70B, and GPT-4o-mini. These cover three open-source models ranging from small to large scales, as well as one proprietary yet strong GPT-family model. In this experiment, we fix the chunk size at 1200 tokens. As shown in Table 17, we evaluate each RAG method and the ZeroShot baseline on the MultihopQA and ALCEAll datasets using different LLM backbones. Based on the results, we can see that: (1) Stronger models generally yield better performance, especially in the ZeroShot setting, which most directly reflects the inherent capabilities of the underlying LLM. (2) We observe that the three variants of LightRAG, LLightRAG, GLightRAG, and HLightRAG as well as LGraphRAG, achieve significant performance improvements when using more powerful LLMs.

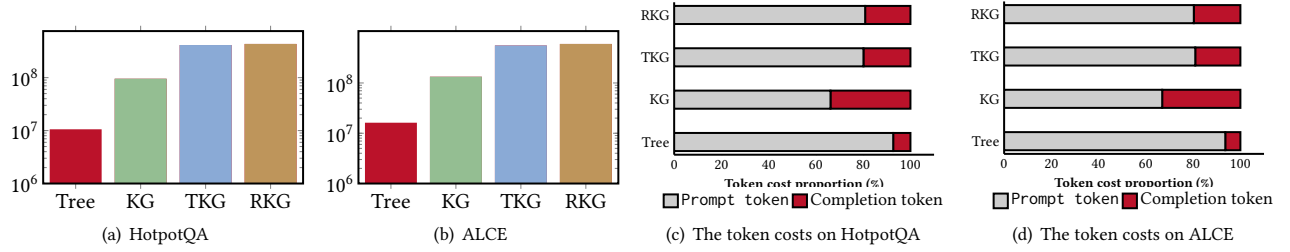


Figure 14: Token cost of graph building on specific QA datasets.

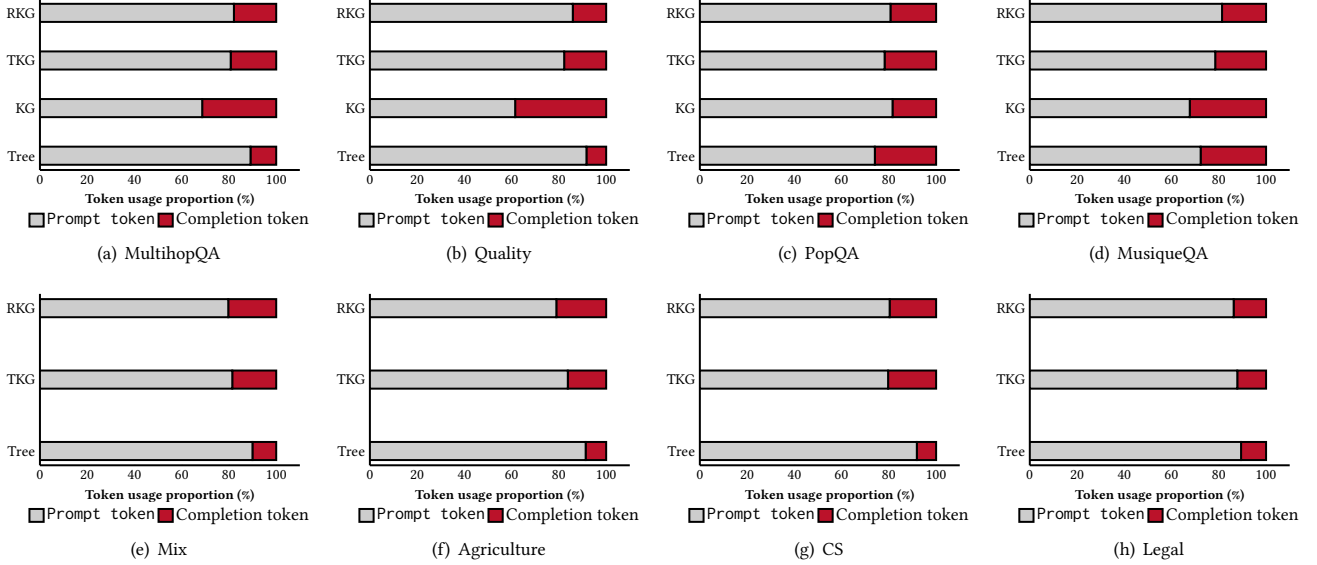


Figure 15: Proportion of the token costs for prompt and completion in graph building stage across all datasets.

Table 12: The average time and token costs of all methods on specific QA datasets.

Method	MultihopQA		Quality		PopQA		MusiqueQA		HotpotQA		ALCE	
	time	token	time	token	time	token	time	token	time	token	time	token
ZeroShot	3.23 s	270.3	1.47 s	169.1	1.17 s	82.2	1.73 s	137.8	1.51 s	125.0	2.41 s	177.2
VanillaRAG	2.35 s	3,623.4	2.12 s	4,502.0	1.41 s	644.1	1.31 s	745.4	1.10 s	652.0	1.04 s	849.1
G-retriever	6.87 s	1,250.0	5.18 s	985.5	37.51 s	3,684.5	31.21 s	3,260.5	—	—	101.16 s	5,096.1
ToG	69.74 s	16,859.6	37.03 s	10,496.4	42.02 s	11,224.2	53.55 s	12,480.8	—	—	34.94 s	11,383.2
KGP	38.86 s	13,872.2	35.76 s	14,092.7	37.49 s	6,738.9	39.82 s	7,555.8	—	—	105.09 s	9,326.6
DALK	28.03 s	4,691.5	13.23 s	2,863.6	16.33 s	2,496.5	17.48 s	3,510.9	21.33 s	3,989.7	17.04 s	4,071.9
LLightRAG	19.28 s	5,774.1	15.76 s	5,054.5	10.71 s	2,447.5	13.95 s	3,267.6	13.94 s	3,074.2	10.34 s	4,427.9
GLightRAG	18.37 s	5,951.5	15.97 s	5,747.3	12.10 s	3,255.6	15.20 s	3,260.8	13.95 s	3,028.7	13.02 s	4,028.1
HLightRAG	19.31 s	7,163.2	21.49 s	6,492.5	17.71 s	5,075.8	20.93 s	5,695.3	19.58 s	4,921.7	16.55 s	6,232.3
FastGraphRAG	7.17 s	5,874.8	3.48 s	6,138.9	13.25 s	6,157.0	15.19 s	6,043.5	28.71 s	6,029.8	25.82 s	6,010.9
HippoRAG	3.46 s	3,261.1	3.03 s	3,877.6	2.32 s	721.3	2.69 s	828.4	3.12 s	726.4	2.94 s	858.2
LGraphRAG	2.98 s	6,154.9	3.77 s	6,113.7	1.72 s	4,325.2	2.66 s	4,675.7	2.05 s	4,806.2	2.11 s	5,441.1
RAPTOR	3.18 s	3,210.0	2.46 s	4,140.7	1.36 s	1,188.3	1.85 s	1,742.9	1.48 s	757.6	1.54 s	793.6

Table 14: Descriptions of the different variants of LGraphRAG.

Name	Retrieval elements	New retrieval strategy
LGraphRAG	Entity, Relationship, Community, Chunk	✗
GraphRAG-ER	Entity, Relationship	✗
GraphRAG-CC	Community, Chunk	✗
VGraphRAG-CC	Community, Chunk	✓
VGraphRAG	Entity, Relationship, Community, Chunk	✓

Table 13: Proportion of retrieved nodes across tree layers.

Layer	MultihopQA	Quality	PopQA	MusiqueQA	HotpotQA	ALCE
0	59.3%	76.8%	76.1%	69.3%	89.7%	90.6%
1	27.5%	18.7%	16.5%	28.1%	9.5%	8.8%
> 1	13.2%	4.5%	7.4%	2.6%	0.8%	0.6%

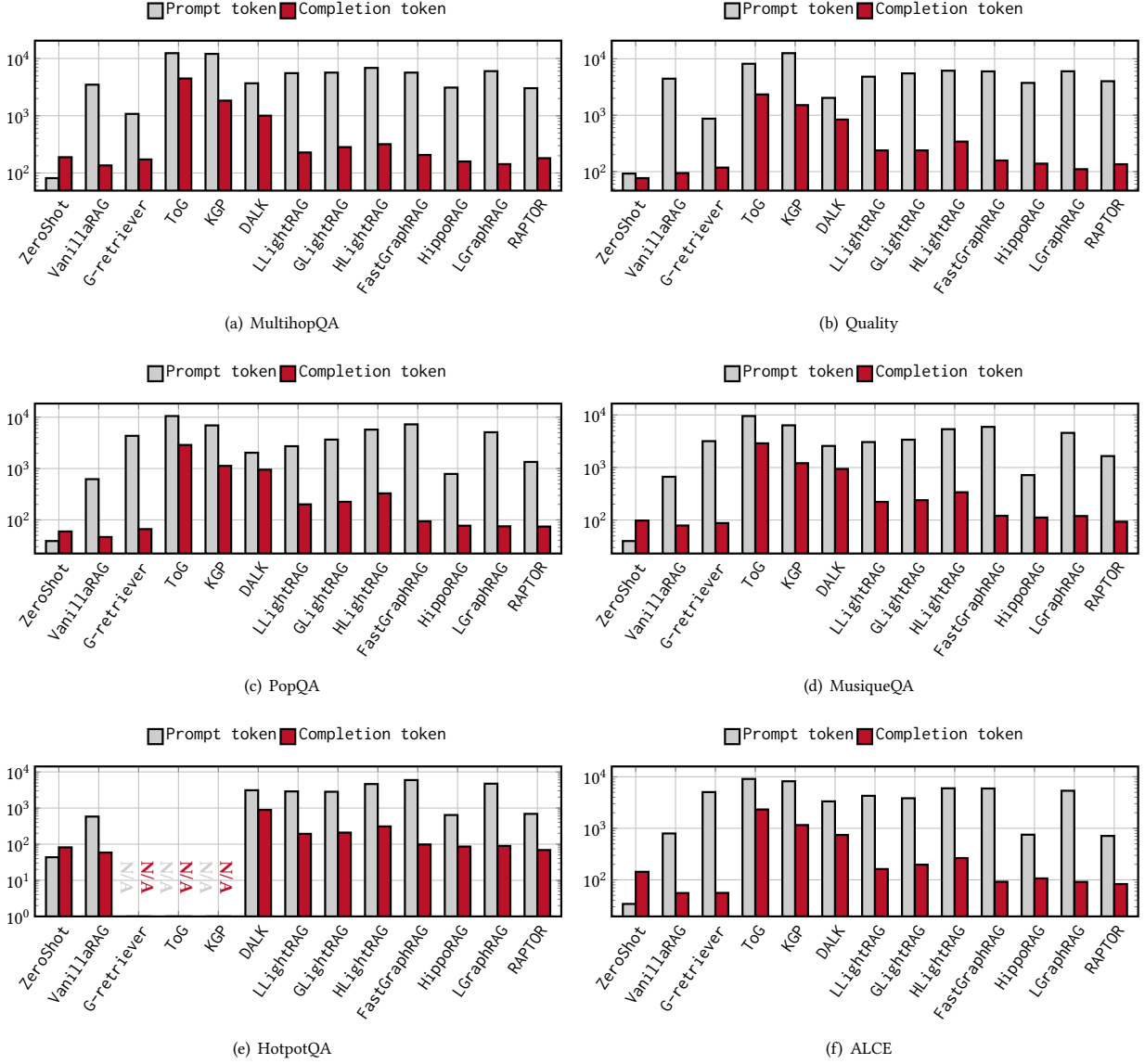


Figure 16: Token costs for prompt and completion tokens in the generation stage across all datasets.

Table 15: Comparison of our newly designed methods on specific datasets with complex questions.

Dataset	Metric	ZeroShot	VanillaRAG	LGraphRAG	RAPTOR	GraphRAG-ER	GraphRAG-CC	VGraphRAG-CC	VGraphRAG
MultihopQA	Accuracy	49.022	50.626	55.360	56.064	52.739	52.113	55.203	59.664
	Recall	34.526	36.918	50.429	44.832	45.113	43.770	46.750	50.893
MusiqueQA	Accuracy	1.833	17.233	12.467	24.133	11.200	13.767	22.400	26.933
	Recall	5.072	27.874	23.996	35.595	22.374	25.707	35.444	40.026
ALCE	STRREC	15.454	34.283	28.448	35.255	26.774	35.366	37.820	41.023
	STREM	3.692	11.181	8.544	11.076	7.5949	11.920	13.608	15.401
	STRHIT	30.696	63.608	54.747	65.401	52.743	64.662	68.460	71.835

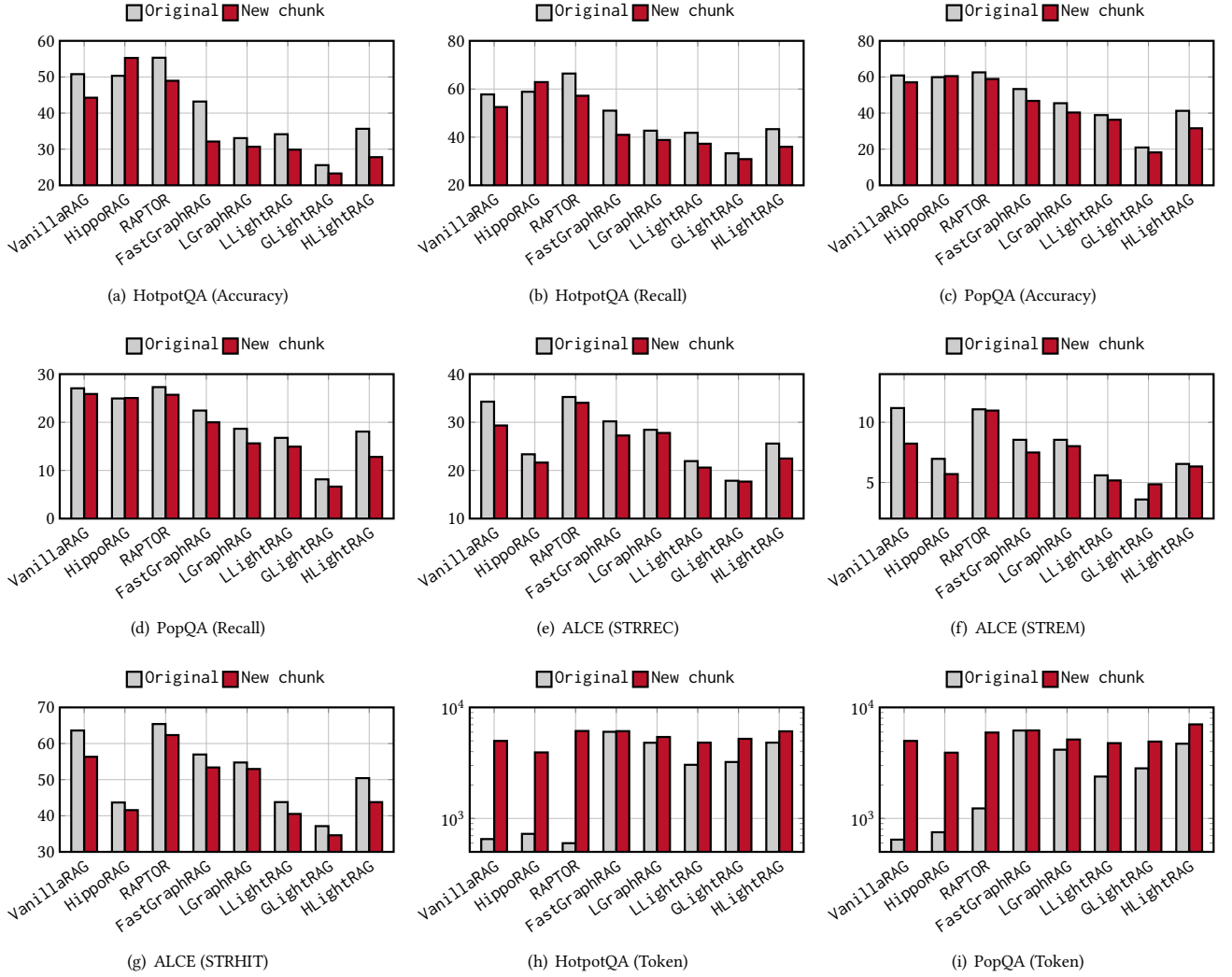


Figure 17: Effect of chunk quality on the performance of specific QA tasks.

Table 16: Comparison of methods on different datasets under different chunk sizes, where Purple denotes the best result, and Orange denotes the best result excluding the best one.

Method	Chunk Size	MultihopQA		PopAll		HotpotAll		ALCEAll		
		Accuracy	Recall	Accuracy	Recall	Accuracy	Recall	STRREC	STREM	STRHIT
VanillaRAG	600	54.421	42.740	57.255	24.171	49.190	56.935	30.174	8.333	57.911
	1200	50.626	36.918	57.041	25.877	44.254	52.511	29.334	8.228	56.329
	2400	50.665	37.172	47.677	19.122	27.553	34.293	26.350	7.490	51.371
G-retriever	600	44.953	47.322	19.014	6.895	18.882	27.041	10.506	2.532	21.414
	1200	42.019	43.116	20.157	6.567	19.098	27.262	13.061	2.848	26.160
	2400	41.667	36.611	20.229	6.763	27.958	36.940	14.822	2.954	30.591
ToG	600	42.214	37.823	49.035	22.642	—	—	16.240	3.376	33.439
	1200	41.941	38.435	49.750	22.767	—	—	15.909	3.059	33.755
	2400	41.667	36.611	44.961	20.374	—	—	14.822	2.954	30.591
KGP	600	46.831	34.714	41.101	18.064	—	—	26.799	7.806	51.793
	1200	48.161	36.272	41.887	18.498	—	—	27.551	7.806	52.637
	2400	48.083	36.486	40.815	18.060	—	—	26.370	6.540	51.477
DALK	600	55.986	48.202	41.243	16.131	32.766	41.737	21.734	4.536	44.304
	1200	53.952	47.232	42.602	17.024	30.416	39.544	21.327	4.430	43.987
	2400	53.208	46.829	45.318	18.651	28.633	37.826	20.350	4.430	41.456
LLightRAG	600	46.675	39.190	38.671	16.040	32.334	39.753	20.084	4.641	40.717
	1200	44.053	35.528	36.312	14.947	29.876	37.258	20.594	5.169	40.506
	2400	43.858	36.878	35.811	14.230	29.336	36.621	19.576	5.485	38.397
GLightRAG	600	47.222	36.885	20.801	6.860	24.419	31.852	18.319	5.063	36.814
	1200	48.474	38.365	18.227	6.643	23.285	30.888	17.686	4.852	34.599
	2400	48.592	41.213	14.653	5.890	19.665	27.566	13.493	3.376	26.688
HLightRAG	600	50.196	40.626	34.668	15.326	30.632	38.194	23.833	5.907	47.363
	1200	50.313	41.613	31.594	12.812	27.796	36.010	22.475	6.329	43.776
	2400	49.648	41.775	30.093	11.683	26.958	34.387	20.816	5.485	40.823
FastGraphRAG	600	50.000	44.736	49.392	22.326	35.467	43.534	30.116	9.705	55.907
	1200	52.895	44.278	46.748	19.996	32.118	40.966	27.258	7.490	53.376
	2400	47.261	46.251	32.809	12.879	26.445	34.800	22.020	6.435	42.300
HippoRAG	600	47.144	41.210	62.401	26.892	50.783	58.454	27.025	8.122	51.477
	1200	53.760	47.671	60.472	25.041	55.267	62.862	21.633	5.696	41.561
	2400	52.152	46.601	50.751	19.986	45.624	53.597	26.477	6.118	52.848
LGraphRAG	600	55.282	46.267	53.181	26.292	41.194	49.801	33.692	10.971	62.447
	1200	55.360	50.429	39.814	17.998	30.686	38.824	27.785	8.017	52.954
	2400	54.930	44.588	43.317	20.185	37.061	45.366	28.398	7.806	54.008
RAPTOR	600	56.729	46.358	61.830	28.176	56.132	63.584	35.111	12.236	63.186
	1200	56.064	44.832	47.963	21.399	31.983	39.864	34.044	10.971	62.342
	2400	56.299	44.610	48.177	21.289	31.983	39.122	33.432	10.654	61.181

This can be attributed to their reliance on Rich Knowledge Graphs and Textual Knowledge Graphs, where stronger LLMs contribute to the construction of higher-quality graphs. (3) HippoRAG shows notably superior performance when using GPT-4o-mini compared to other LLM backbones. We attribute this to GPT-4o-mini’s ability to extract more accurate entities from the question and to construct higher-quality knowledge graphs, thereby improving the retrieval

of relevant chunks and the final answer accuracy. (4) Regardless of the LLM backbone, our proposed method VGraphRAG consistently achieves the best performance, demonstrating the advantages of our proposed unified framework.

► **Exp.7. The size of graph.** For each dataset, we report the size of five types of graphs in Table 18. We observe that PG is typically denser than other types of graphs, as they connect nodes based on

Table 17: The specific QA performance comparison of graph-based RAG methods with different LLM backbones.

Method	LLM backbone	MultihopQA		ALCEAll		
		Accuracy	Recall	STRREC	STREM	STRHIT
ZeroShot	Llama-3-8B	49.022	34.256	15.454	3.692	30.696
	Qwen-2.5-32B	45.070	33.332	30.512	10.127	56.118
	Llama-3-70B	55.908	52.987	31.234	7.170	61.920
	GPT-4o-mini	59.546	48.322	34.965	10.232	66.245
VanillaRAG	Llama-3-8B	50.626	36.918	29.334	8.228	56.329
	Qwen-2.5-32B	56.299	47.660	39.490	14.873	69.937
	Llama-3-70B	56.768	49.127	34.961	9.810	68.038
	GPT-4o-mini	59.311	47.941	35.735	10.127	68.249
G-retriever	Llama-3-8B	42.019	43.116	13.061	2.848	26.160
	Qwen-2.5-32B	43.075	33.864	34.678	13.608	60.338
	Llama-3-70B	50.430	50.144	24.575	5.907	49.051
	GPT-4o-mini	56.534	46.427	31.681	7.068	62.764
ToG	Llama-3-8B	41.941	38.435	15.909	3.059	33.755
	Qwen-2.5-32B	34.390	35.566	19.167	4.430	39.662
	Llama-3-70B	42.762	38.495	18.449	3.270	38.819
	GPT-4o-mini	41.862	30.247	28.405	7.068	56.962
KGP	Llama-3-8B	48.161	36.272	27.551	7.806	52.637
	Qwen-2.5-32B	61.463	57.340	40.608	14.873	71.097
	Llama-3-70B	55.008	47.878	35.420	10.549	68.379
	GPT-4o-mini	63.146	55.789	38.015	12.764	69.620
DALK	Llama-3-8B	53.952	47.232	21.327	4.430	43.987
	Qwen-2.5-32B	32.003	20.158	16.653	4.430	33.333
	Llama-3-70B	60.524	55.086	28.980	5.696	60.338
	GPT-4o-mini	66.980	57.687	31.813	8.333	62.236
LLightRAG	Llama-3-8B	44.053	35.528	20.594	5.169	40.506
	Qwen-2.5-32B	48.552	45.387	31.549	9.916	59.177
	Llama-3-70B	57.081	54.510	28.636	8.228	56.013
	GPT-4o-mini	52.113	38.923	38.446	15.084	33.456
GLightRAG	Llama-3-8B	48.474	38.365	17.686	4.852	34.599
	Qwen-2.5-32B	52.582	48.236	27.961	9.599	53.165
	Llama-3-70B	55.986	51.713	23.553	5.807	45.992
	GPT-4o-mini	55.125	47.899	39.095	15.506	67.933
HLightRAG	Llama-3-8B	50.313	41.613	22.475	6.329	43.776
	Qwen-2.5-32B	53.678	51.403	34.168	10.971	63.819
	Llama-3-70B	57.081	54.510	29.548	8.228	57.911
	GPT-4o-mini	55.829	46.424	41.334	15.506	71.730
FastGraphRAG	Llama-3-8B	52.895	44.278	27.258	7.490	53.376
	Qwen-2.5-32B	46.088	50.370	31.387	10.021	59.388
	Llama-3-70B	54.069	55.787	35.658	12.236	65.612
	GPT-4o-mini	66.080	57.007	23.521	8.228	42.827
HippoRAG	Llama-3-8B	53.760	47.671	21.633	5.696	41.561
	Qwen-2.5-32B	48.083	40.488	37.419	13.397	66.245
	Llama-3-70B	57.277	57.736	32.904	9.916	32.534
	GPT-4o-mini	67.723	55.482	39.274	12.447	72.046
LGraphRAG	Llama-3-8B	55.360	50.429	27.785	8.017	52.954
	Qwen-2.5-32B	49.531	52.113	35.406	12.553	63.924
	Llama-3-70B	58.060	55.390	34.256	10.232	66.561
	GPT-4o-mini	65.415	50.216	36.890	11.287	69.304
RAPTOR	Llama-3-8B	56.064	44.832	34.044	10.971	62.342
	Qwen-2.5-32B	60.485	56.359	39.267	13.924	70.359
	Llama-3-70B	63.028	61.042	37.286	12.236	68.671
	GPT-4o-mini	60.603	51.521	29.770	8.017	58.861
VGraphRAG	Llama-3-8B	59.664	50.893	35.213	11.603	64.030
	Qwen-2.5-32B	57.277	55.151	39.234	14.557	69.831
	Llama-3-70B	67.567	68.445	37.576	12.447	69.198
	GPT-4o-mini	68.193	56.564	43.963	18.038	74.473

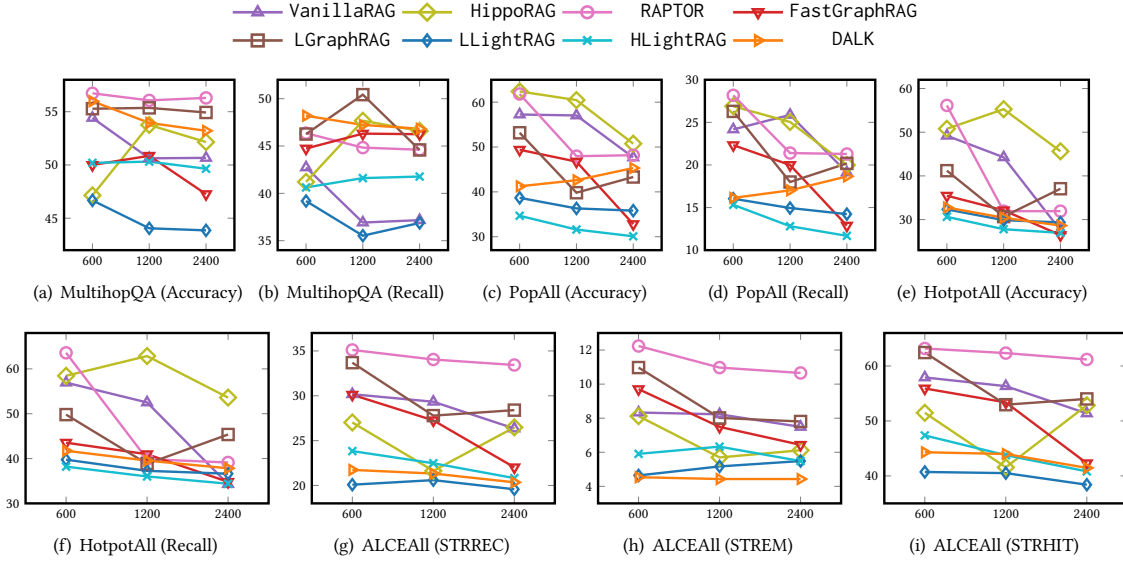


Figure 18: Effect of chunk size.

shared entity relationships, where each node represents a chunk in PG. In fact, the probability of two chunks sharing at least a few entities is quite high, leading to a high graph density (i.e., the ratio of edges to nodes), sometimes approaching a clique (fully connected graph). In contrast, KG, TKG, and RKG are much sparser since they rely entirely on LLMs to extract nodes and edges. This sparsity is primarily due to the relatively short and incomplete outputs typically generated by LLMs, which miss considerable potential node-edge pairs. Interestingly, the size or density of the constructed graph has not shown a strong correlation with the final performance of graph-based RAG methods. This observation motivates us to explore a method for evaluating the quality of the constructed graph before using it for LLM-based question answering.



Figure 19: The failure cases of HippoRAG.

► **Exp.8. Failure cases of graph-based RAG methods.** We would like to clarify that the results presented in our paper represent the average performance across all questions within each

dataset. Thus, not all graph-based RAG methods consistently outperform the baseline VanillaRAG on every question. We conduct a detailed failure case analysis focusing on the top-performing methods. Specifically, we examine why RAPTOR, HippoRAG, RAPTOR and LGraphRAG sometimes fall short in specific QA tasks. Please refer to the detailed analysis provided below.

► **The key failure reasons of HippoRAG:**

- (1) *Incorrect or incomplete entity extraction from the question.* Consider the example in Figure 19(a), where HippoRAG extracts the entity "Lyric Street Records" from the question and then applies Personalized PageRank based on this entity to retrieve relevant chunks. However, to answer this question correctly, two key entities are required: "Lyric Street Records" and "If You Ever Get Lonely". Since HippoRAG fails to extract the latter, it retrieves chunks that are insufficient to provide a correct answer.
- (2) *Retrieval of irrelevant chunks by Personalized PageRank.* In another example shown in Figure 19(b), HippoRAG correctly extracts the relevant entities ["Grown-ups", "Allo 'Allo!", "actor"] from the question. However, its chunk retriever strategy—Personalized PageRank—tends to favor chunks where these entities appear frequently, regardless of whether the content is semantically relevant to the question. As a result, the retrieved chunks may not align with the actual intent of the question, leading to an incorrect final answer.

► **The key failure reason of Raptor:**

- *Low-quality cluster summaries.* As illustrated in Figure 20, Raptor retrieves cluster summaries generated from groups of similar chunks. However, chunks within the same cluster may mention various loosely related facts that are topically similar but not logically unified. For example, the summary "Top Scorers of 2023" in Figure 20 contains some loosely related facts about scorers in 2023, which are too general to provide a precise answer. When summarizing such content, the LLM tends to produce generic or fragmented summaries

Table 18: The size of each graph type across all datasets.

Dataset	Tree		PG		KG		TKG		RKG	
	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges	# of vertices	# of edges
MultihopQA	2,053	2,052	1,658	564,446	35,953	37,173	12,737	10,063	18,227	12,441
Quality	1,862	1,861	1,518	717,468	28,882	30,580	10,884	8,992	13,836	9,044
PopQA	38,325	38,324	32,157	3,085,232	260,202	336,676	179,680	205,199	188,946	215,623
MusiqueQA	33,216	33,215	29,898	3,704,201	228,914	295,629	153,392	183,703	149,125	188,149
HotpotQA	73,891	73,890	66,559	13,886,807	511,705	725,521	291,873	401,693	324,284	436,362
ALCE	99,303	99,302	89,376	22,109,289	610,925	918,499	306,821	475,018	353,989	526,486
Mix	719	718	1,778	1,225,815	28,793	34,693	7,464	2,819	7,701	3,336
MultihopSum	2,053	2,052	N/A	N/A	N/A	N/A	12,737	10,063	18,227	12,441
Agriculture	2,156	2,155	N/A	N/A	N/A	N/A	15,772	7,333	17,793	12,600
CS	2,244	2,243	N/A	N/A	N/A	N/A	10,175	6,560	12,340	8,692
Legal	5,380	5,379	N/A	N/A	N/A	N/A	15,034	10,920	16,565	17,633

MultihopQA
Which rugby team, featured in articles from 'The Independent - Sports' and 'The Roar | Sports Writers Blog', **faced home defeats to Ireland, South Africa, and Argentina**, aimed to utilize a numerical advantage by kicking for the corner, and has players striving to conclude their careers on a high note, while also having lost to Argentina both in Christchurch and previously in Sydney?

Retrieved summaries

1. Top Scorers of 2023.
2. Australian Davis Cup Team Advances to Final.
3. West Indies vs England T20 International Cricket Match.
4. CONCACAF Nations League Quarterfinal Second Leg.
5. France vs Italy Rugby World Cup Match.
 - France defeated Italy 40-0 in a Rugby World Cup match, securing their spot in the quarterfinals.
 - **France's top spot** in Pool A is secured, and **they will face either Ireland, South Africa, or Scotland ...**
6. England Cricket Team Struggles.
7. Australian Women's Cricket Team.

Raptor's Output

Overall, based on the articles, it is clear that **France** is the team that ...

Answer: [All Blacks]

Figure 20: The failure case of Raptor.

MultihopQA
What entity, discussed in articles from both **The Verge** and **Fortune**, was involved in **implementing a system** to prevent liquidation due to software issues, took on losses to maintain another company's balance sheet, and claimed to have acted legally in its business practices as a customer, payment processor, and market maker?

Retrieved Entities: [the verge, Google, mastercard, ...]

Retrieved communities

Google Community: Influence, Innovation, and Ethical Concerns

- Google's Influence in the Technology Industry
- Google's AI Capabilities
- Google's Partnerships and Collaborations
- Google's Impact on the Technology Industry
- Google's Potential for Bias and Misuse

LGraphRAG's Output

Output: [Google]

Answer: [Alameda Research]

Figure 21: The failure case of LGraphRAG (I).

MultihopQA
Which company, featured in articles from both **Wired** and **Cnbc | World Business News Leader**, introduced an **invite-only deal system** during a summer event for its members and is also considered to provide a life-changing opportunity for its sellers?

Retrieved Relationships: [(OpenAI, pricing schema), (eater, wine), (green monday, newegg)]

Retrieved chunks

The chunks contain the (OpenAI, pricing schema)

Will that fact lead startups building modern AI tools to pursue more traditional SaaS pricing? (The **OpenAI pricing schema** based on tokens and usage led us to this question.) The trajectory of usage-based pricing has organically aligned with the needs of large language models, given that there is significant variation in prompt/output sizes and resource utilization per user. OpenAI itself racks upward of \$700,000 per day on compute, so to achieve profitability, these operation costs ...

LGraphRAG's Output

Output: [OpenAI]

Answer: [Amazon]

Figure 22: The failure case of LGraphRAG (II).

that fail to capture the key information required to answer the question.

►The key failure reasons of LGraphRAG:

- **Irrelevant community reports retrieved by Entity operator.** Consider the example in Figure 21, where LGraphRAG first extracts entities such as ["The Verge", "Google", "Mastercard", "..."] from the question. It then applies the Entity operator to retrieve communities whose reports contain these entities. Among them, communities with frequent mentions of "Google" are prioritized. However, these retrieved communities turn out to be irrelevant to the actual question, as the method relies solely on surface-level entity frequency while ignoring semantic relevance.
- **Irrelevant chunks retrieved by Occurrence operator.** Consider the example in Figure 22, where LGraphRAG extracts relationships such as [(“OpenAI”, “pricing schema”), (“Eater”, “wine”), (“greenmonday”, “newegg”)] from the question. It then applies the Occurrence operator to retrieve chunks that contain the relationship (“OpenAI”, “pricing schema”) with high frequency. Based on these chunks, LGraphRAG generates the incorrect answer “OpenAI”. The key reason is that the retrieved chunks, despite their frequent mentions of certain relationships, are not semantically relevant to the question. The method relies on co-occurrence frequency rather than actual contextual relevance.

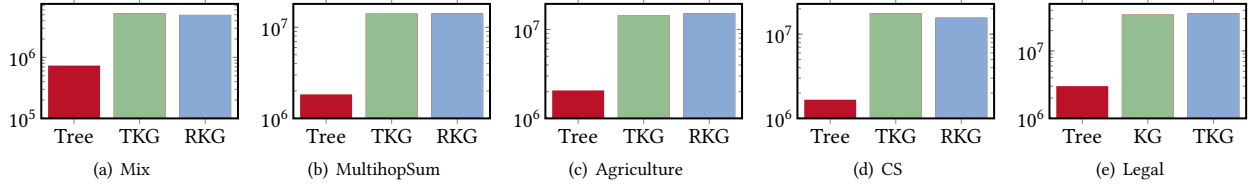


Figure 23: Token cost of the graph building on abstract QA datasets.

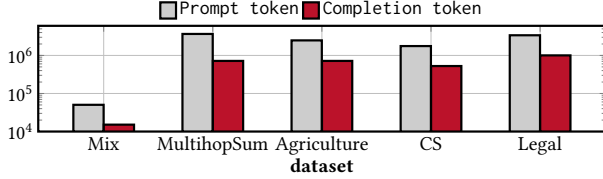


Figure 24: Token cost of index construction in abstract QA.

A.2 Results on abstract QA tasks

In this subsection, we present the addition results on abstract QA tasks.

► **Exp.1. Token costs of graph and index building.** The token costs of the graph and index building across all abstract QA datasets are shown in Figures 23 and 24 respectively. The conclusions are highly similar to the Exp.2 in Section 7.2.

► **Exp.2. Effect of chunk size.** We report the performance of various RAG methods on abstract QA tasks under different chunk sizes in Figures 25 to 27. Our key observations are as follows: (1) The performance of GGraphRAG remains stable across different chunk sizes, likely due to its use of the Map-Reduce strategy for final answer synthesis, which mitigates the influence of chunk granularity. (2) In contrast, methods like FastGraphRAG and VanillaRAG show greater variance across chunk sizes, as their performance relies heavily on the granularity of individual chunks—smaller chunks tend to provide more precise information, directly impacting retrieval and generation quality. (3) Regardless of chunk size, RAPTOR and GGraphRAG consistently achieve the best performance, reaffirming our earlier conclusion that high-level structural information is essential for abstract QA tasks.

Additionally, we evaluate our newly proposed method CheapRAG across different chunk sizes. As shown in Figure 28, CheapRAG generally outperforms the five baselines. Notably, under the 600-token setting, CheapRAG surpasses GGraphRAG in more cases. We attribute this to the higher precision of smaller chunks, which enhances the effectiveness of semantic similarity-based retrieval in CheapRAG, compared to the entity frequency-based retrieval strategy used in GGraphRAG.

► **Exp.3. Effect of LLM backbone.** We evaluate all methods that support abstract QA on the MultihopSum dataset, using different LLM backbones. Results for Llama-3-8B are shown in Figure 26, while Figures 29 to 31 present results for the other models. We observe that, across different backbones, the performance of each method on abstract QA tasks remains relatively stable—especially when compared to the fluctuations seen in specific QA tasks. We note that all methods still lag behind GGraphRAG, further highlighting that community-level information is particularly beneficial for abstract QA tasks. Moreover, we compare our newly proposed method, CheapRAG, with five strong baselines under different LLM backbones. As shown in Figure 32, CheapRAG exhibits remarkable

performance improvements as the model capacity increases. Notably, under the GPT-4o-mini backbone, CheapRAG achieves near-universal wins across all evaluated cases, clearly demonstrating its strong generalization ability and effectiveness.

► **Exp.4. More analysis.** We have further analyzed the performance results across different methods, and summarize the key insights as follows:

- **Comprehensiveness:** GGraphRAG consistently achieves the highest scores, highlighting the strength of community-level retrieval in capturing global context. By grouping semantically related content, communities help reduce fragmented evidence and support more holistic answers.
- **Diversity:** Both RAPTOR and GGraphRAG perform strongly by aggregating information across multiple clusters or communities. This enables the generation of responses that span diverse subtopics while maintaining relevance.
- **Empowerment:** GGraphRAG and LightRAG jointly lead on this metric. Their retrieval strategies incorporate structured elements—entities, relations, and keywords—that provide concrete grounding for the model to generate actionable, role-relevant responses. This better supports practical decision-making in activity-centered QA.
- **Overall:** GGraphRAG consistently ranks first, with RAPTOR typically second. This highlights the advantage of leveraging high-level summarized information—such as community reports and cluster-level chunks—with the former generally proving more effective. Additionally, the results support the effectiveness of the Map-Reduce mechanism in filtering out irrelevant information during retrieval.

A.3 Evaluation metrics

This section outlines the metrics used for evaluation.

• **Metrics for specific QA Tasks.** We use accuracy as the evaluation metric, based on whether the gold answers appear in the model’s generated outputs, rather than requiring an exact match, following the approach in [4, 57, 73]. This choice is motivated by the uncontrollable nature of LLM outputs, which often makes it difficult to achieve exact matches with standard answers. Similarly, we prefer recall over precision as it better reflects the accuracy of the generated responses.

• **Metrics for abstract QA Tasks.** Building on existing work, we use an LLM to generate abstract questions, as shown in Figure 34. We adopt 125 questions following the prior works, such as GraphRAG [15], LightRAG [24], and FastGraphRAG [19]. In their setup, the number 125 comes from generating $N=5$ user roles, each with N associated tasks, and for each (user, task) pair, N abstract questions—yielding $5 \times 5 \times 5 = 125$ questions per dataset. We follow this standard for consistency and comparability across studies. The reasons of selecting GPT-4o are twofold: (1) at the time

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	27	32	78	56	VR	50	42	17	82	69	VR	50	34	24	82	74	VR	50	27	30	78	67
RA	73	50	61	88	90	RA	58	50	18	87	82	RA	66	50	38	92	91	RA	73	50	54	90	92
GS	68	39	50	81	66	GS	83	82	50	97	92	GS	76	62	50	93	89	GS	70	46	50	86	71
LR	27	12	19	50	29	LR	18	13	3	50	31	LR	18	8	7	50	27	LR	22	10	14	50	29
FG	44	10	34	71	50	FG	31	18	8	69	50	FG	26	9	11	73	50	FG	33	8	19	71	50
(a) Comprehensiveness					(b) Diversity					(c) Empowerment					(d) Overall								

Figure 25: The abstract QA results on the MultihopSum dataset (chunk size = 600).

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	50	2	46	95	VR	50	64	58	64	93	VR	50	52	36	39	95	VR	50	52	44	46	95
RA	50	50	47	48	94	RA	36	50	42	49	85	RA	48	50	45	45	93	RA	48	50	45	47	94
GS	78	53	50	79	96	GS	42	55	50	52	92	GS	64	54	50	41	97	GS	56	55	50	52	97
LR	54	52	21	50	92	LR	36	51	48	50	88	LR	61	55	59	50	95	LR	54	53	48	50	93
FG	5	6	4	8	50	FG	7	15	8	12	50	FG	5	7	3	5	50	FG	5	6	3	7	50
(a) Comprehensiveness					(b) Diversity					(c) Empowerment					(d) Overall								

Figure 26: The abstract QA results on the MultihopSum dataset (chunk size = 1200).

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	22	44	60	67	VR	50	32	17	67	69	VR	50	26	28	66	75	VR	50	23	37	62	68
RA	78	50	73	83	88	RA	68	50	28	82	82	RA	74	50	45	83	91	RA	77	50	66	83	90
GS	56	27	50	61	63	GS	83	72	50	90	90	GS	72	55	50	86	89	GS	63	34	50	70	72
LR	40	17	39	50	53	LR	33	18	10	50	53	LR	34	17	14	50	57	LR	38	17	30	50	54
FG	33	12	37	47	50	FG	31	18	10	47	50	FG	25	9	11	43	50	FG	32	10	28	46	50
(a) Comprehensiveness					(b) Diversity					(c) Empowerment					(d) Overall								

Figure 27: The abstract QA results on the MultihopSum dataset (chunk size = 2400).

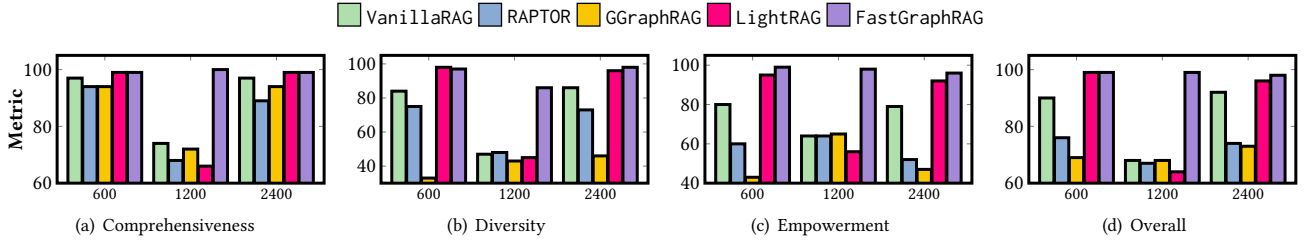


Figure 28: Comparison of CheapRAG and other methods under different chunk sizes on the MultihopSum dataset.

when conducted our experiments, GPT-4o was one of the most advanced LLM, demonstrating strong zero-shot capabilities and superior performance in long-context understanding compared to other models, and (2) it provides the highest fluency, coherence, and factual consistency, ensuring that the generated questions are both challenging and realistic. This effectively supports our exploration of real-world data sensemaking scenarios.

Defining ground truth for abstract questions, especially those involving complex high-level semantics, presents significant challenges. To address this, we adopt an LLM-based multi-dimensional comparison method, inspired by [15, 24], which evaluates comprehensiveness, diversity, empowerment, and overall quality. We use a robust LLM, specifically GPT-4o, to rank each baseline in comparison to our method. The evaluation prompt used is shown in Figure 35. These four metrics are defined [15, 19, 24] as follows:

- **Comprehensiveness.** How much detail does the answer provide to cover all aspects and details of the question?
- **Diversity.** How varied and rich is the answer in providing different perspectives and insights on the question?
- **Empowerment.** How well does the answer help the reader understand and make informed judgments about the topic?

- **Overall.** Select an overall winner based on these categories.

To better illustrate these dimensions, Figure 36 presents examples of both good and bad answers with respect to comprehensiveness, diversity, and empowerment.

Head-to-head comparison. To evaluate abstract QA tasks by head-to-head comparison using an LLM evaluator, selecting four target metrics capturing qualities that are desirable for abstract questions. The answer to an abstract question is not a collection of details from specific texts, but rather a high-level understanding of the dataset’s contents relevant to the query. A good response to an abstract question should perform well across the following four metrics, including “Comprehensiveness”, “Diversity”, “Empowerment”, and “Overall”.

Head-to-head win results illustrate the relative performance among different methods. Each value indicates the percentage of test cases where the row method outperforms the column method - higher values indicate better performance. For example, Figure 37 demonstrates an example head-to-head result under the “comprehensive” dimension on the Mix dataset. The value 30 in the first row and third column indicates that VanillaRAG outperforms GGraphRAG in 30% of cases, suggesting that VanillaRAG performs

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	17	9	14	62	VR	50	22	9	22	73	VR	50	26	12	26	74	VR	50	19	10	18	68
RA	83	50	24	55	86	RA	78	50	26	60	86	RA	74	50	24	63	90	RA	81	50	24	61	88
GS	91	76	50	77	93	GS	91	74	50	83	95	GS	88	76	50	84	94	GS	90	76	50	80	93
LR	86	45	23	50	91	LR	78	40	17	50	90	LR	74	37	16	50	91	LR	82	39	20	50	91
FG	38	14	7	9	50	FG	27	14	5	10	50	FG	26	10	6	9	50	FG	32	12	7	9	50
(a) Comprehensiveness						(b) Diversity						(c) Empowerment						(d) Overall					

Figure 29: The abstract QA results on the MultihopSum dataset (LLM backbone = Qwen-2.5-32B).

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	10	18	16	70	VR	50	15	1	19	67	VR	50	13	4	29	81	VR	50	12	6	18	78
RA	90	50	49	68	96	RA	85	50	5	68	91	RA	87	50	20	74	97	RA	88	50	30	76	97
GS	82	51	50	63	90	GS	99	95	50	99	99	GS	96	80	50	91	99	GS	94	70	50	83	99
LR	84	32	37	50	94	LR	81	32	1	50	88	LR	71	26	9	50	93	LR	82	24	17	50	95
FG	30	4	10	6	50	FG	33	9	1	12	50	FG	19	3	1	7	50	FG	22	3	1	5	50
(a) Comprehensiveness						(b) Diversity						(c) Empowerment						(d) Overall					

Figure 30: The abstract QA results on the MultihopSum dataset (LLM backbone = Llama-3-70B).

	VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG		VR	RA	GS	LR	FG
VR	50	66	4	12	44	VR	50	61	6	15	54	VR	50	79	8	31	74	VR	50	72	4	13	60
RA	34	50	5	9	31	RA	39	50	6	14	39	RA	21	50	4	14	40	RA	28	50	4	10	37
GS	96	95	50	81	94	GS	94	94	50	85	95	GS	92	96	50	92	95	GS	96	96	50	90	95
LR	88	91	19	50	82	LR	85	86	15	50	87	LR	69	86	8	50	86	LR	87	90	10	50	87
FG	56	69	6	18	50	FG	46	61	5	13	50	FG	26	60	5	14	50	FG	40	63	5	13	50
(a) Comprehensiveness						(b) Diversity						(c) Empowerment						(d) Overall					

Figure 31: The abstract QA results on the MultihopSum dataset (LLM backbone = GPT-4o-mini).

	Com.	Div.	Emp.	Overall		Com.	Div.	Emp.	Overall		Com.	Div.	Emp.	Overall		Com.	Div.	Emp.	Overall
VR	74	47	64	68	VR	99	99	99	99	VR	98	97	91	96	VR	99	99	99	99
RA	68	48	64	67	RA	87	96	91	93	RA	93	77	61	73	RA	99	99	99	99
GS	72	43	65	68	GS	90	81	60	82	GS	95	40	76	88	GS	64	90	72	80
LR	66	45	56	64	LR	99	99	97	99	LR	98	94	76	88	LR	93	99	96	98
FG	100	86	98	99	FG	99	99	99	99	FG	99	99	98	99	FG	99	99	99	99
(a) LLM backbone = Llama-3-8B					(b) LLM backbone = Qwen-2.5-32B					(c) LLM backbone = Llama-3-70B					(d) LLM backbone = GPT-4o-mini				

Figure 32: Comparison of our newly designed method on MultihopSum dataset.

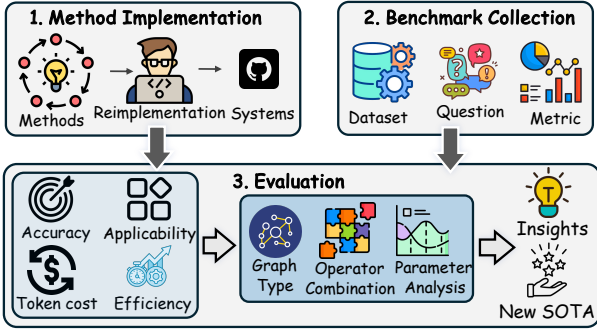


Figure 33: Workflow of our empirical study.

worse than GGraphRAG on this metric. Based on the overall comparison in the figure, the relative ranking of the five methods is: 1. HLightRAG (LR), 2. GGraphRAG (GS), 3. VanillaRAG (VR), 4. RAPTOR (RA), 5. FastGraphRAG (FG).

A.4 Implementation details

In this subsection, we present more details about our system implementation. Specifically, we use HNSW [56] from Llama-index [55] (a well-known open-source project) as the default vector database for efficient vector search. In addition, for each method, we optimize efficiency by batching or parallelizing operations such as encoding nodes or chunks, and computing personalized page rank, among others, during the retrieval stage.

Why Llama-3-8B? Beyond the widespread use of Llama-3-8B in existing works, our choice is further motivated by the following considerations:

- (1) *Strong Capability*: Llama-3-8B exhibits solid language understanding and reasoning abilities [14], which are crucial for RAG workflows. Its effectiveness in integrating retrieved content with internal knowledge enables accurate and contextually relevant response generation.
- (2) *Practical Efficiency*: The 8B variant offers a favorable trade-off between performance and resource efficiency. It supports FP16 inference within 20 GB of GPU memory, allowing deployment on widely available hardware (e.g., a single 24 GB GPU).
- (3) *Open source LLM*: As an open-source LLM, it can be deployed locally, saving the costs of paying for API calls required by proprietary models such as GPT-4o.

Workflow of our evaluation. We present the first open-source testbed for graph-based RAG methods, which (1) collects and reimplements 12 representative methods within a unified framework (as depicted in Section 3), (2) supports a fine-grained comparison over the building blocks of the retrieval stage with up to 100+ variants, and (3) provides a comprehensive evaluation over 11 datasets with various metrics in different scenarios, we summarize the workflow of our empirical study in Figure 33.

A.5 Criteria of selection dataset

Our work aims to systematically analyze various graph-based RAG methods and provide in-depth insights. To this end, we select datasets based on the following criteria:

- (1) *Widely used in the existing works*: All selected datasets are extensively used in the RAG and LLM research communities. For example, the HotpotQA dataset has been cited more than 2,800 times.
- (2) *Diverse corpus domains*: Our selected datasets cover a broad range of corpus domains. For instance, MultihopRAG consists of long English news articles; HotpotQA includes short passages from Wikipedia; and Quality comprises a diverse mixture of sources such as fiction from Project Gutenberg and articles from Slate magazine, with each document containing at least 2,000 tokens. For abstract questions, we include datasets spanning various domains such as agriculture, computer science, and legal texts.
- (3) *Diverse in task types*: Question answering (QA) tasks are typically categorized into abstract and specific questions [24], in which the specific questions can be further divided by complexity into simple and complex types [4, 72]. Specifically, simple questions typically require only one or two text chunks for an answer, without the need for multi-hop reasoning. In contrast, complex questions necessitate reasoning across multiple chunks, understanding implicit relations, and synthesizing information.

B MORE DISCUSSIONS.

B.1 New operators

Here, we introduce the operators that are not used in existing graph-based RAG methods but are employed in our newly designed state-of-the-art methods.

Chunk type. We include a new operator VDB of chunk type, which is used in our VGraphRAG method. This operator is the same as the chunk retrieval strategy of VanillaRAG.

Community type. We also include a new operator VDB of community type, retrieving the top- k communities by vector searching, where the embedding of each community is generated by encoding its community report.

B.2 More Lessons and Opportunities

In this subsection, we show the more lessons and opportunities learned from our study.

Lessons

- **L6.** For large datasets, both versions of the GraphRAG methods incur unacceptable token costs, as they contain a large number of communities, leading to high costs for generating community reports.
- **L7.** Regardless of whether the questions are specific or abstract, they all rely on an external corpus (i.e., documents). For such questions, merely using graph-structure information (nodes, edges, or subgraphs) is insufficient to achieve good performance.
- **L8.** Methods designed for knowledge reasoning tasks, such as DALK, ToG, and G-retriever, do not perform well on document-based QA tasks. This is because these methods are better suited for extracting reasoning rules or paths from well-constructed KGs. However, when KGs are built from raw text corpora, they may not

accurately capture the correct reasoning rules, leading to suboptimal performance in document-based QA tasks.

- **L9.** The effectiveness of RAG methods is highly impacted by the relevance of the retrieved elements to the given question. That is, if the retrieved information is irrelevant or noisy, it may degrade the LLM’s performance. When designing new graph-based RAG methods, it is crucial to evaluate whether the retrieval strategy effectively retrieves relevant information for the given question.

Opportunities

- **O5.** In real applications, external knowledge sources are not limited to text corpora; they may also include PDFs, HTML pages, tables, and other structured or semi-structured data. A promising future research direction is to explore graph-based RAG methods for heterogeneous knowledge sources.
- **O6.** An interesting future research direction is to explore more graph-based RAG applications. For example, applying graph-based RAG to scientific literature retrieval can help researchers efficiently extract relevant studies and discover hidden relationships between concepts. Another potential application is legal document analysis, where graph structures can capture case precedents and legal interpretations to assist in legal reasoning.
- **O7.** The users may request multiple questions simultaneously, but existing graph-based RAG methods process them sequentially. Hence, a promising future direction is to explore efficient scheduling strategies that optimize multi-query handling. This could involve batching similar questions or parallelizing retrieval.
- **O8.** Different types of questions require different levels of information, yet all existing graph-based RAG methods rely on fixed, predefined rules. How to design an adaptive mechanism that can address these varying needs remains an open question.
- **O9.** Existing methods do not fully leverage the graph structure; they typically rely on simple graph patterns (e.g., nodes, edges, or k -hop paths). Although GraphRAG adopts a hierarchical community structure (detecting by the Leiden algorithm), this approach does not consider node attributes, potentially compromising the quality of the communities. That is, determining which graph structures are superior remains an open question.
- **O10.** The well-known graph database systems, such as Neo4j [61] and Nebula [60], support transferring the corpus into a knowledge graph via LLM. However, enabling these popular systems to support the diverse operators required by various graph-based RAG methods presents an exciting opportunity.

B.3 Benefit of our framework

Our framework offers exceptional flexibility by enabling the combination of different methods at various stages. This modular design allows different algorithms to be seamlessly integrated, ensuring that each stage—such as *graph building*, and *retrieval&generation*—can be independently optimized and recombined. For example, methods like HippoRAG, which typically rely on KG, can easily be adapted to use RKG instead, based on specific domain needs.

In addition, our operator design allows for simple modifications—often just a few lines of code—to create entirely new graph-based RAG methods. By adjusting the retrieval stage or swapping components, researchers can quickly test and implement new strategies, significantly accelerating the development cycle of graph-based RAG methods.

The modular nature of our framework is further reinforced by the use of retrieval elements (such as node, relationship, or subgraph)

coupled with retrieval operators. This combination enables us to easily design new operators tailored to specific tasks. For example, by modifying the strategy for retrieving given elements, we can create customized operators that suit different application scenarios.

By systematically evaluating the effectiveness of various retrieval components under our unified framework, we can identify the most efficient combinations of graph construction, indexing, and retrieval strategies. This approach enables us to optimize retrieval performance across a range of use cases, allowing for both the enhancement of existing methods and the creation of novel, state-of-the-art techniques.

Finally, our framework contributes to the broader research community by providing a standardized methodology to assess graph-based RAG approaches. The introduction of a unified evaluation testbed ensures reproducibility, promotes fair benchmarking, and facilitates future innovations in RAG-based LLM applications.

B.4 Limitations

In our empirical study, we put considerable effort into evaluating the performance of existing graph-based RAG methods from various angles. However, our study still has some limitations, primarily due to resource constraints. (1) *Token Length Limitation*: The primary experiments are conducted using Llama-3-8B with a token window size of 8k. This limitation on token length restricted the model’s ability to process longer input sequences, which could potentially impact the overall performance of the methods, particularly in tasks that require extensive context. Larger models with larger token windows could better capture long-range dependencies and deliver more robust results. This constraint is a significant factor that may affect the generalizability of our findings. (2) *Limited Knowledge Datasets*: Our study did not include domain-specific knowledge datasets, which are crucial for certain applications. Incorporating such datasets could provide more nuanced insights and allow for a better evaluation of how these methods perform in specialized settings. (3) *Resource Constraints*: Due to resource limitations, the largest model we utilized is Llama-3-70B, and the entire paper consumes nearly 10 billion tokens. Running larger models, such as GPT-4o (175B parameters or beyond), would incur significantly higher costs, potentially reaching several hundred thousand dollars depending on usage. While we admit that introducing more powerful models could further enhance performance, the 70B model is already a strong choice, balancing performance and resource feasibility. That is to say, exploring the potential of even larger models in future work could offer valuable insights and further refine the findings. (4) *Prompt Sensitivity*: The performance of each method is highly affected by its prompt design. Due to resource limitations, we did not conduct prompt ablation studies and instead used the available prompts from the respective papers. Actually, a fairer comparison would mitigate this impact by using prompt tuning tools, such as DSPy [38], to customize the prompts and optimize the performance of each method.

These limitations highlight areas for future exploration, and overcoming these constraints would enable a more thorough and reliable evaluation of graph-based RAG methods, strengthening the findings and advancing the research.

B.5 Necessity of chunk splitting in RAG systems

Splitting the input corpus into smaller chunks is a necessary step for all RAG methods, including both non-graph and graph-based variants. The reasons can be summarized in three aspects below:

- (1) *Input length limit of LLMs*. Every LLM has its own input length limitation. For example, ChatGPT-3.5 supports up to 4,096 tokens, while Llama-3-8B allows up to 8,192 tokens. A token is a basic unit of text (roughly a word or subword). However, real-world corpora often contain tens of thousands to millions of tokens—e.g., the ALCE dataset exceeds 13 million tokens—far beyond the processing capability of any existing LLM. Therefore, it is essential to split the corpus into smaller chunks.
- (2) *Relevance filtering and noise reduction*. Even if the full corpus could fit into the LLM, chunking is still necessary because most queries only relate to a small part of the content. That is, given a query Q , inputting the entire corpus into the LLM introduces unnecessary noise. To alleviate this, RAG systems retrieve only the top- k relevant chunks, which helps filter out irrelevant content and improves both accuracy and efficiency.
- (3) *Graph construction in graph-based RAG*. Chunking is equally essential and plays a critical role in all graph-based RAG methods. Below, we outline how different types of graphs rely on chunking:
 - *Knowledge graph (KG), Textual Knowledge graph (TKG), and Rich Knowledge graph (RKG)*: These graphs use LLMs to extract entities and relationships from the input corpus. Due to the token limitation of the LLM, the corpus must first be split into chunks. The LLM is then applied independently to each chunk, and the resulting outputs are aggregated to construct the complete graph. This follows a divide-and-conquer paradigm.
 - *Passage graph (PG)*: In the passage graph, each chunk is treated as a node, and an edge is added between two nodes if their corresponding chunks share at least a certain number of common entities. The graph is constructed by comparing entity overlap across all pairs of chunks.

In summary, chunking is not only necessary due to token limits but also integral to the design of RAG systems.

B.6 Comparison of graphs and indices

Indeed, it is difficult to define a single “best” graph, as each type of graph has its own advantages and disadvantages. To help clarify this, we summarize the key characteristics of each graph type in Table 19. For example, although tree can be constructed with minimal cost, it carries limited information. Methods such as RAPOTR, which are based on tree, demonstrate competitive performance under lightweight settings. However, they may fall short in complex QA and abstract QA tasks due to limited semantic coverage. In contrast, methods such as VGraphRAG and GGraphRAG, which are built upon TKG, benefit from richer entity and relationship representations. This enhanced expressiveness enables more accurate reasoning in information-intensive tasks. However, these advantages come at the cost of increased token consumption and longer construction time. Additionally, while PG avoids token usage during construction, it incurs significant time overhead due to exhaustive pairwise chunk comparisons.

Prompt for generating abstract questions

Prompt:

Given the following description of a dataset:

{description}

Please identify 5 potential users who would engage with this dataset. For each user, list 5 tasks they would perform with this dataset. Then, for each (user, task) combination, generate 5 questions that require a high-level understanding of the entire dataset.

Output the results in the following structure:

- User 1: [user description]
 - Task 1: [task description]
 - Question 1:
 - Question 2:
 - Question 3:
 - Question 4:
 - Question 5:
 - Task 2: [task description]
 - ...
 - Task 5: [task description]
- User 2: [user description]
- ...
- User 5: [user description]
- ...

Note that there are 5 users and 5 tasks for each user, resulting in 25 tasks in total. Each task should have 5 questions, resulting in 125 questions in total. The Output should present the whole tasks and questions for each user.

Output:

Figure 34: The prompt for generating abstract questions.

Table 19: Comparison of different types of graph.

Graph	Entity Name	Entity Type	Entity Description	Relationship Name	Relationship Keyword	Relationship Description	Edge Weight	Token Consuming	Graph Size	Information Richness	Construction Time
Tree	✗	✗	✗	✗	✗	✗	✗	★	★	★★	★
PG	✗	✗	✗	✗	✗	✗	✗	N/A	★★★★	★	★★★★
KG	✓	✗	✗	✓	✗	✗	✓	★★	★★	★★★	★★
TKG	✓	✓	✓	✗	✗	✓	✓	★★★	★★★	★★★★	★★★
RKG	✓	✓	✓	✗	✓	✓	✓	★★★	★★★	★★★★★	★★★

Table 20: Comparison of different types of index.

Index	Index Size	Token Consuming	Construction Time
Node Index	★★	N/A	★
Relationship Index	★★★	N/A	★★
Community Index	★	★★★★	★★★★

Similar to the comparison of graphs, it is also challenging to identify a universally “best” index. Instead, we summarize the characteristics of different indices in Table 20. We can see that the *relationship index* tends to have a larger size, whereas the *community index* is more compact but incurs the highest construction cost in terms of tokens and time. Indeed, in our experiments, we report the token costs associated with constructing the *community index* for each dataset. Notably, for some datasets, such as HotpotQA and ALCE, this cost is comparable to that of building the graph itself, exceeding 10^8 tokens. Despite their differences, all indices share a

common goal of facilitating fast and effective retrieval within graph-based RAG systems. The selection of a specific index should align with the retrieval strategy and the demands of the downstream QA task. For instance, to answer abstract questions that rely on some high-level summaries, it is better to use the community-based indexes (e.g., the index of GraphRAG) to retrieve relevant information since they often provide the summary reports for the communities.

B.7 Relevance to the data management community

We agree that graph-based RAG is very relevant to the information retrieval and NLP communities, but we also believe that it is highly relevant to the graph data management community, since it has received growing interest from both industry and academia. In the industrial area, Ant-Group recently proposed Chat2Graph [6], a graph-native agent system that incorporates graph-based RAG as a core component. Similarly, modern graph database systems such as

Prompt for LLM-based multi-dimensional comparison

Prompt:

You will evaluate two answers to the same question based on three criteria: **Comprehensiveness**, **Diversity**, **Empowerment**, and **Directness**.

- **Comprehensiveness**: How much detail does the answer provide to cover all aspects and details of the question?
- **Diversity**: How varied and rich is the answer in providing different perspectives and insights on the question?
- **Empowerment**: How well does the answer help the reader understand and make informed judgments about the topic?
- **Directness**: How specifically and clearly does the answer address the question?

For each criterion, choose the better answer (either Answer 1 or Answer 2) and explain why. Then, select an overall winner based on these four categories.

Here is the **question**:

Question: {query}

Here are the two answers:

Answer 1: {answer1}

Answer 2: {answer2}

Evaluate both answers using the four criteria listed above and provide detailed explanations for each criterion. Output your evaluation in the following JSON format:

```
{
  "Comprehensiveness": {
    "Winner": "[Answer 1 or Answer 2]",
    "Explanation": "[Provide one sentence explanation here]"
  },
  "Diversity": {
    "Winner": "[Answer 1 or Answer 2]",
    "Explanation": "[Provide one sentence explanation here]"
  },
  "Empowerment": {
    "Winner": "[Answer 1 or Answer 2]",
    "Explanation": "[Provide one sentence explanation here]"
  },
  "Overall Winner": {
    "Winner": "[Answer 1 or Answer 2]",
    "Explanation": "[Briefly summarize why this answer is the overall winner]"
  }
}
```

Output:

Figure 35: The prompt for the evaluation of abstract QA.

Neo4j [62], NebulaGraph [60], and PostgreSQL [68] are beginning to support graph-based RAG methods, further reinforcing the growing intersection between AI and graph data management. In the academia area, interest in RAG is also rising within the data management field, as demonstrated by recent work such as Chameleon [36] (VLDB 2025) and Cache-Craft [2] (SIGMOD 2025).

The core retrieval stage in graph-based RAG systems can be naturally framed as a query optimization process over graph databases, where the goal is to identify the most relevant subgraphs, paths, nodes, or relationships from a large graph G given a natural language query Q . This introduces new challenges for graph query processing and indexing, as many modern graph databases lack native support for the complex retrieval operators required by

graph-based RAG workflows. For instance, while retrieving k -hop neighbors is commonly supported, operators such as efficiently identifying communities containing specific entities, computing Steiner trees, or finding k -hop semantically relevant paths to the query Q remain largely unsupported. Moreover, retrieval operators that rely on LLM-based reasoning or semantic similarity are fundamentally beyond the capabilities of existing graph data systems. Our work highlights these limitations and identifies opportunities for extending graph databases to such operators over large graph data—an emerging direction highly relevant to the graph data management community.

Lastly, our work has garnered significant attention from both academic and industrial areas. Our open-source project has attracted

An example abstract question

Question: How do the syntactic and semantic differences between functional and object-oriented paradigms affect program design?

Responses of different methods

Comprehensiveness:

► **Why it matters:** *A comprehensive answer should mention key differences and explain their design implications.*

► **Comprehensive Answer:** Functional programming emphasizes *pure functions, immutability, and recursion*, reducing side effects and improving testability. Object-oriented programming (OOP) uses *classes, encapsulation, and mutable state*, supporting modularity and reuse. These paradigms shape program design in *syntax* (e.g., *method chaining vs. composition*) and *semantics* (e.g., *stateful vs. stateless logic*). For example, FP promotes *declarative flows* and *better concurrency*, while OOP mirrors *real-world models*. *Understanding both helps in choosing suitable architecture patterns.*

► **Incomprehensive Answer:** Functional programming focuses on using *functions*, and object-oriented programming uses *objects and classes*. These differences change how you write code, but *both are just different ways to program*. Most people use OOP because it's more common. Functional programming is used sometimes, but not always necessary. *It depends on what you're doing*, but overall, they just represent two styles.

► **Explanation:** *The second response omits key distinctions like mutability and side effects, and lacks detail on how such differences influence program structure or architecture.*

Diversity:

► **Why it matters:** *A diverse answer should present multiple angles, use cases, and comparisons.*

► **Diverse Answer:** Functional and object-oriented paradigms differ across many dimensions. Syntactically, FP uses *higher-order functions* and *expressions*, while OOP relies on *method calls* and *class hierarchies*. Semantically, FP enforces *immutability* and *stateless computation*, whereas OOP supports *encapsulated state* and side effects. These lead to varied design strategies: FP suits *parallel computation, mathematical modeling*, and *data transformation*; OOP works well for *user interfaces, simulations*, and *domain modeling*. *Hybrid designs* are also common in modern software.

► **Low Diversity Answer:** Functional programming is good for *parallelism* because it doesn't use shared state. Object-oriented programming helps organize code using *classes*. In general, most developers stick to OOP because it's easier and more practical. *Functional programming is more theoretical*. While they work differently, the main thing to know is that *functional is more mathematical*, and object-oriented is more practical for apps.

► **Explanation:** *The second answer focuses on only one or two differences and lacks variety in perspectives, missing concrete use cases and deeper design implications.*

Empowerment

► **Why it matters:** *An empowering answer helps readers apply the concepts to make better design decisions.*

► **Empowerment Answer:** Understanding the core trade-offs between functional and object-oriented paradigms equips developers to design more robust systems. For instance, choosing *functional principles like immutability* can minimize bugs in *concurrent applications*. OOP's modeling of *real-world entities* simplifies maintenance in large systems. Awareness of these options enables *informed architectural decisions*, such as using a *functional core within an object-oriented shell*. This empowers developers to *select the right paradigm or blend* based on project needs.

► **Low Empowerment Answer:** Object-oriented programming is the *standard approach*, so you should usually stick to that. *Functional programming is more academic* and harder to understand. Unless you're working on something very technical, like machine learning or math-heavy software, *it's not really useful*. Most beginners don't need to worry about it. *Just learn OOP and you'll be fine* for most jobs or projects.

► **Explanation:** *The second answer discourages understanding by dismissing functional programming as impractical, instead of helping the reader see when and why each paradigm is appropriate.*

Figure 36: Representative good and bad answers across three evaluation dimensions. Cyan italics highlight informative content in good answers; gray italics indicate vague or unhelpful points in bad answers.

VanillaRAG outperforms GGraphRAG on 30% of cases

		VR	RA	GS	LR	FG
VanillaRAG	VR	50	58	30	36	93
RAPTOR	VR					
GGraphRAG	RA	42	50	39	26	82
HLightRAG	GS	70	61	50	15	89
FastGraphRAG	LR	64	74	85	50	98
	FG	7	18	11	2	50

Figure 37: An example head-to-head comparison result.

significant attention (e.g., 1.3k+ GitHub stars ¹) and has been successfully adopted by *Huawei Cloud for domain-specific QA*, thanks to its modular architecture and ability to handle complex retrieval workflows efficiently. In summary, as noted in the meta-review, we believe that with the increasing importance of graphs and the growing adoption of AI techniques such as RAG, it is both timely and relevant to the graph data management community.

B.8 More examples

• **An example of building graph.** As shown in Figure 38, we consider a corpus from the IT domain, which is segmented into six chunks. We then illustrate how to construct different types of graphs based on this corpus:

- **Tree:** We first apply the Gaussian Mixture clustering algorithm to group the six chunks into two clusters: (chunk 1, chunk 2, chunk 3) and (chunk 4, chunk 5, chunk 6). Each cluster forms a parent node in the second layer, where an LLM is used to summarize the content of all associated chunks. These two parent nodes are then clustered into a single root node (third layer), whose content is also summarized via LLM. The resulting structure is a three-layer tree with 6 leaf nodes (original chunks), 2 intermediate cluster nodes, and 1 root node.
- **Passage Graph:** We construct a passage graph where each chunk is treated as a node, and an edge is added between two nodes if their corresponding chunks share at least τ common entities. For example, an edge is added between chunk 4 and chunk 6 since they both contain the same entities, e.g., “ChatGPT” and “OpenAI”.
- **Knowledge Graph:** We use an LLM to extract entity–relation triples from each chunk. Each entity includes a name, while each relationship is represented by a name and an associated weight indicating how frequently it appears within the given chunk. For example, from chunk 1, we extract the entity “Elon Musk” and its relationship “Founder of”.
- **Textural & Rich Knowledge Graph:** The two types of graphs are constructed in a similar way: for each chunk, we use an LLM to extract entities and their relationships. Each entity is represented with three attributes—name, type, and description—while each relationship includes a name, description, and weight. The key difference lies in that the Rich Knowledge Graph further annotates each relationship with a set of keywords, providing more semantic cues for retrieval.

¹<https://github.com/JayLZhou/GraphRAG>

• **Details of operators.** Specifically, we abstract three operators under the **Chunk type**: ① Aggregator, ② FromRel, and ③ Occurrence.

- **Aggregator:** This operator relies on two matrices: a score vector $\mathcal{R} \in \mathbb{R}^{1 \times m}$ and an interaction matrix $\mathcal{M} \in \mathbb{R}^{m \times c}$, where m is the number of relationships and c is the number of chunks. Specifically, the i -th entry in \mathcal{R} represents the score of the i -th relationship, while $\mathcal{M}_{i,j} = 1$ if the i -th relationship is extracted from the j -th chunk, and 0 otherwise. Afterwards, the aggregated score of each chunk is computed via matrix multiplication:

$$\Psi = \mathcal{R} \times \mathcal{M},$$

where each entry in $\Psi \in \mathbb{R}^{1 \times c}$ represents the aggregated relationship score of a chunk. Based on Ψ , the top- k chunks with the highest scores are selected for retrieval.

- **FromRel:** For each relationship in the graph, we maintain a mapping to the set of chunks from which it is extracted. This allows us to efficiently retrieve relevant chunks given a set of relationships by computing the union of all chunks associated with these relationships.
- **Occurrence:** For every relationship, we identify its two associated entities. For each entity, we maintain a mapping to the set of chunks from which it is extracted, since an entity can appear in multiple chunks. If a chunk contains both entities of a given relationship, its score (initially set to 0) is incremented by 1. After processing all relationships, we obtain a score for each chunk, and select the top- k chunks based on these scores.

Besides, there are two operators under the **Community type**:

① Entity, and ② Layer:

- **Entity:** This operator retrieves communities that contain the specified entities. Each community maintains a list of associated entities. Retrieved communities are then ranked based on their relevance scores (generated by the LLM), and the top- k communities are returned.
- **Layer:** In GGraphRAG, communities are detected using the Leiden algorithm, resulting in a hierarchical structure where higher layers represent more abstract and coarse-grained information. The Layer operator retrieves all communities at or below a specified layer, allowing access to more fine-grained community information.

• **An example of Map-Reduce strategy.** In the Map-Reduce phase of GGraphRAG, each retrieved community is individually used to answer the question. Specifically, for each community, the LLM is prompted to generate a partial answer along with a confidence score (ranging from 0 to 100), reflecting how well the community summary addresses the query. After processing all communities, we obtain a set of partial answers, each represented as a pair: (answer, score). These partial answers are then ranked in descending order by score and sequentially appended to the prompt for final answer generation. We also provide an example to illustrate this.

EXAMPLE 1. Figure 39 illustrates how the partial answers are generated and used in GGraphRAG. Consider an abstract question: “What are the socio-economic impacts of artificial intelligence on the global labor market?” In this example, three communities are retrieved, and GGraphRAG generates partial answers from each one. For instance, using the first community, it produces a partial answer discussing the impact of AI on the job landscape, which receives a relevance score of

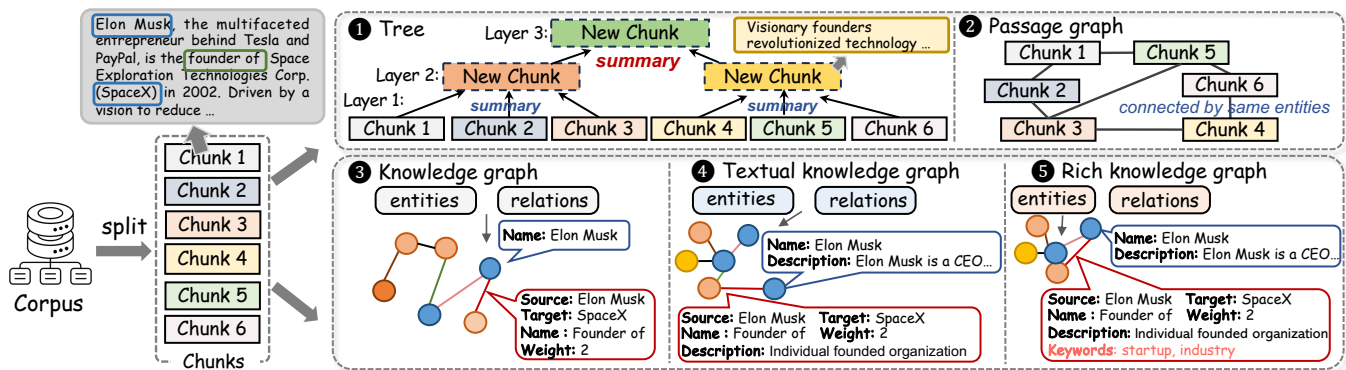


Figure 38: Examples of five types of graphs.

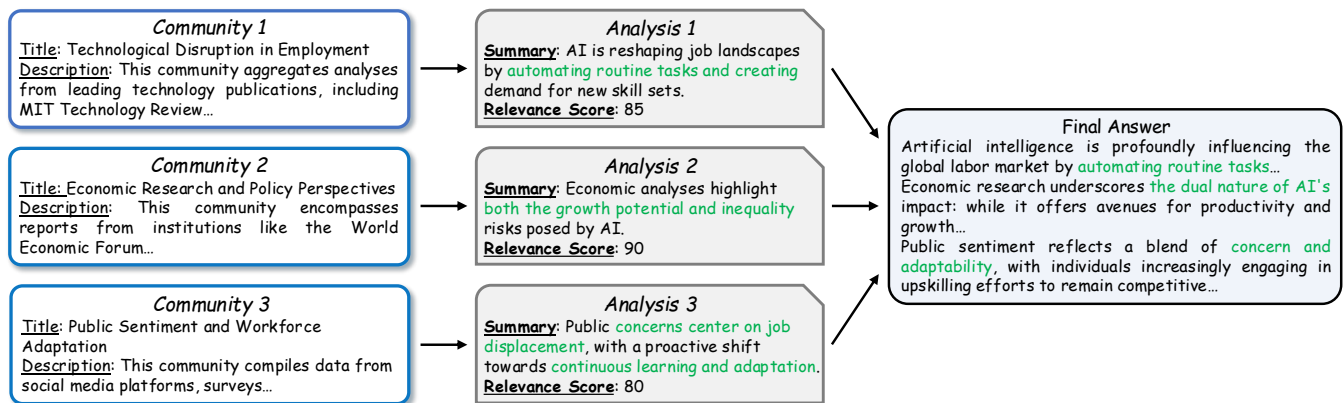


Figure 39: An illustrative example of GGraphRAG's Map-Reduce generation process. The analysis presents partial answers derived from each community, with the portions incorporated into the final answer highlighted in green.

85. These partial answers are then aggregated, allowing GGraphRAG to generate a final, comprehensive response.