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 questing-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 [84], and Llama 3.1 [11] has sparked a revolution in the field of artificial intelligence [19, 28, 41, 49, 60, 78, 80, 91]. Despite their remarkable comprehension and generation capabilities, LLMs may still generate incorrect outputs due to a lack of domain-specific knowledge, real-time updated information, and proprietary knowledge, which are outside LLMs' pre-training corpus, known as "hallucination" [64].

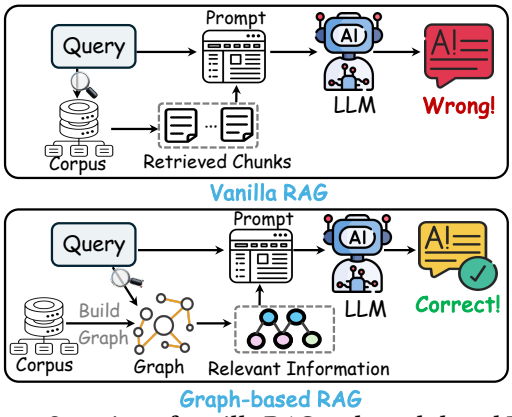


Figure 1: Overview of vanilla RAG and graph-based RAG.

To bridge this gap, the Retrieval Augmented Generation (RAG) technique [15, 18, 27, 29, 83, 87, 89] has been proposed, which supplements LLM with external knowledge to enhance its factual accuracy and trustworthiness. Consequently, RAG techniques have been widely applied in various fields, especially in domains where LLMs need to generate reliable outputs, such as healthcare [49, 78, 91], finance [41, 60], and education [19, 80]. Moreover, RAG has proven highly useful in many data management tasks, including NL2SQL [13, 38], data cleaning [14, 40, 56, 66], knob tuning [20, 37], DBMS diagnosis [70, 92, 93], and SQL rewrite [42, 73]. Due to the important role of the RAG technique in LLM-based applications, numerous RAG methods have been proposed in the past year [27]. 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. Given a user query Q, the key idea of graph-based RAG methods is to retrieve relevant information (e.g., nodes, edges, subgraphs, or textual data) from the graph and then feed them along with Q as a prompt into LLM to generate answers. The overview of naive-based RAG (i.e., vanilla RAG) and graph-based RAG are shown in Figure 1.

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 [12, 21, 22, 30, 39, 64, 68, 81, 82]. 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

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.

Method	Graph Type	Index Component	Retrieval Primitive	Retrieval Granularity	Specific QA	Abstract QA
RAPTOR [68]	Tree	Tree node	Question vector	Tree node	✓	✓
KGP [81]	Passage Graph	Entity	Question	Chunk	✓	✗
HippoRAG [22]	Knowledge Graph	Entity	Entities in question	Chunk	✓	✗
G-retriever [26]	Knowledge Graph	Entity, Relationship	Question vector	Subgraph	✓	✗
ToG [72]	Knowledge Graph	Entity, Relationship	Question	Subgraph	✓	✗
DALK [39]	Knowledge Graph	Entity	Entities in question	Subgraph	✓	✗
LGraphRAG [12]	Textual Knowledge Graph	Entity, Community	Question vector	Entity, Relationship, Chunk, Community	✓	✗
GGraphRAG [12]	Textual Knowledge Graph	Community	Question vector	Community	✗	✓
FastGraphRAG [16]	Textual Knowledge Graph	Entity	Entities in question	Entity, Relationship, Chunk	✓	✓
LLightRAG [21]	Rich Knowledge Graph	Entity, Relationship	Low-level keywords in question	Entity, Relationship, Chunk	✓	✓
GLightRAG [21]	Rich Knowledge Graph	Entity, Relationship	High-level keywords in question	Entity, Relationship, Chunk	✓	✓
HLightRAG [21]	Rich Knowledge Graph	Entity, Relationship	Both high- and low-level keywords	Entity, Relationship, Chunk	✓	✓

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 propose a novel unified framework with four stages, namely ❶ *Graph building*, ❷ *Index construction*, ❸ *Operator configuration*, and ❹ *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 [10, 61]. 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 [61].

Retrieval Augmented Generation. During completing tasks with prompting, LLMs often generate erroneous or meaningless responses, i.e., the hallucination problem [28]. 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 [18]: *(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 [64]. 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 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. Compared to vanilla RAG, the key difference in graph-based RAG methods can be summarized from two perspectives:

- *Retrieval source:* Vanilla RAG retrieves knowledge directly from external chunks, whereas graph-based RAG methods

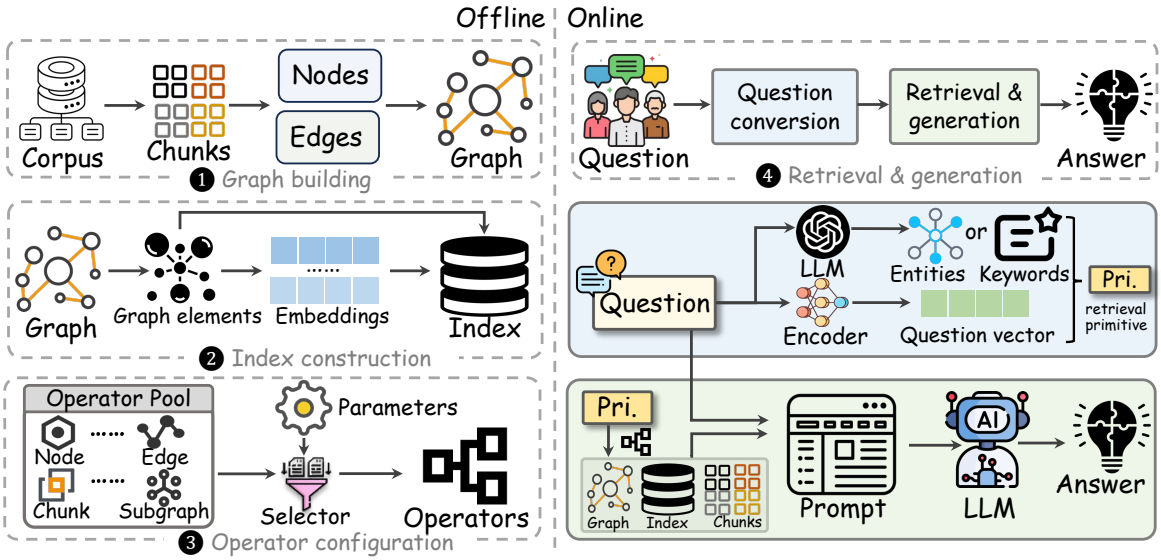


Figure 2: Workflow of graph-based RAG methods under our unified framework.

retrieve information from a graph constructed using these chunks.

- *Retrieval element:* Given a user question Q, vanilla RAG aims to retrieve the most relevant chunks, while graph-based RAG methods focus on finding useful information from the graph, such as nodes, relationships, or subgraphs.

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 $\mathcal{D}$, and user question Q
output: The answers for user question Q

1 $C \leftarrow$ split $\mathcal{D}$ into multiple chunks;
// (1) Graph building.
2 $\mathcal{G} \leftarrow$ GraphBuilding(C);
// (2) Index construction.
3 $\mathcal{I} \leftarrow$ IndexConstruction($\mathcal{G}$, C);
// (3) Operator configuration.
4 $\mathcal{O} \leftarrow$ OperatorConfiguration( );
// (4) Retrieve relevant information and generate response.
5 $\mathcal{R} \leftarrow$ Retrieval&generation($\mathcal{G}$, $\mathcal{I}$, $\mathcal{O}$, Q);
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. Note that the first three stages are executed offline, enabling support for efficient online queries once completed, we present the workflow of graph-based RAG methods under our framework in Figure 2.

Table 2: Comparison of different types of graphs.

Attributes	Tree	PG	KG	TKG	RKG
Original Chunk	✓	✓	✗	✗	✗
Entity Name	✗	✗	✓	✓	✓
Entity Type	✗	✗	✗	✓	✓
Entity Description	✗	✗	✗	✓	✓
Relationship Name	✗	✗	✓	✗	✗
Relationship Keyword	✗	✗	✗	✗	✓
Relationship Description	✗	✗	✗	✓	✓
Edge Weight	✗	✗	✓	✓	✓

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 first step is splitting the corpus into smaller chunks, followed by using an LLM or other tools to create nodes and edges based on these chunks. 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 [81]. 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 characters of each graph type in Table 2.

5 INDEX CONSTRUCTION

To support efficient online querying, existing graph-based RAG methods typically include an index-construction stage, which involves storing entities or relationships in the vector database, and computing community reports for efficient online retrieval. Generally, there are three types of indices, ❶ **Node Index**, ❷ **Relationship Index**, and ❸ **Community Index**, where for the first two types, we use the well-known text-encoder models, such as BERT [9], BGE-M3 [55], or ColBert [35] to generate embeddings for nodes or relationships in the graph.

❶ **Node Index** stores the graph nodes in the vector database. For `RAPTOR`, `G-retriever`, `DALK`, `FastGraphRAG`, `LGraphRAG`, `LLightRAG`, and `HLightRAG`, 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 [24], which represents the term-weight distribution across different nodes (i.e., chunks) in the index.

❷ **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.

❸ **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 [75] algorithm is utilized by `LGraphRAG` and `GGraphRAG`.

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 exhibit that the retrieval stage in various graph-based RAG methods can be decoupled into a series of operators, with different methods selecting specific operators and combining them in various ways. By selecting and arranging these operators in different sequences, all existing (and potentially future) graph-based RAG methods can be implemented. Through an in-depth analysis of all implementations, we distill the retrieval process into a set of 19 operators, forming an operator pool. Based on the granularity of retrieval, we classify the operators into five categories:

• **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. ❶ `VDB` leverages the vector database to retrieve nodes by computing the vector similarity with the query vector. ❷ `RelNode` extracts nodes from the provided relationships. ❸ `PPR` uses the Personalized PageRank (PPR) algorithm [25] 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`, where the first one is based on the `Link` operator in the relationship type, specifically, we use the relationship scores and the relationship-chunk interactions to select the top-k chunks, where the score of each chunk is obtained by the summing the scores of all relationships extracted from it. The `FromRel` operator retrieves chunks that "contain" the given relationships. The `Occurrence` operator selects the top-k chunks based on the given relationships, assigning each chunk a score by counting the number of times it contains both entities in a relationship.

• **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 [24]. 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. The `Entity` operator aims to obtain the communities containing the specified entities. Here, all identified

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

communities are sorted based on their rating (generated by the LLM), and then the top-k communities are returned. The Leiden algorithm generates hierarchical communities, where higher layers represent more abstract, high-level information. The Layer operator is used to retrieve all communities below the required layers.

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 parameters. Those parameters define two key aspects: (1) which operators to use, and (2) how to combine or apply the selected operators.

In Table 3, we present how the existing graph-based RAG methods utilize our provided operators to assemble their retrieval stages. 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 graph-based 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 P, where P 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 P 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, while the latter, used in GGraphRAG, analyzes the retrieved communities one by one, first, each community is used to answer the question independently in parallel, and then all relevant partial answers are summarized into a final answer.

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	29,898	Specific QA
HotpotQA	8,495,056	3,702	66,581	Specific QA
ALCE	13,490,670	948	89,562	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 re-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 Figure 3, and make our unified system available in: https://github.com/JayLZhou/GraphRAG/tree/master.

▶ **Benchmark Dataset.** We employ 11 real-world datasets for evaluating the performance of each graph-based RAG method, including both specific and abstract questions.

- *Specific.* The question in this group is detail-oriented and typically references specific entities within the graph (e.g.,

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**†

"Who won the 2024 U.S. presidential election?"). We categorize the questions into two groups based on complexity: *Simple* and *Complex*. The former has answers directly available in one or two text chunks, requiring no reasoning across chunks, which includes three datasets: Quality [62], PopQA [53], and HotpotQA [85]. The latter involves reasoning across multiple chunks, understanding implicit relationships, and synthesizing knowledge, including datasets: MultihopQA [74], MusiqueQA [76], and ALCE [17].

- *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 [65], MultihopSum [74], Agriculture [65], CS [65], and Legal [65].

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 existing works [12, 21] and generate questions using one of the most advanced LLM, GPT-4o. Specifically, for each dataset, we generate 125 questions by prompting GPT-4o, following the approach in [21]. The prompt template used for question generation is provided in our technical report [67]. 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 [53, 69]. For the ALCE dataset, answers are typically full sentences rather than

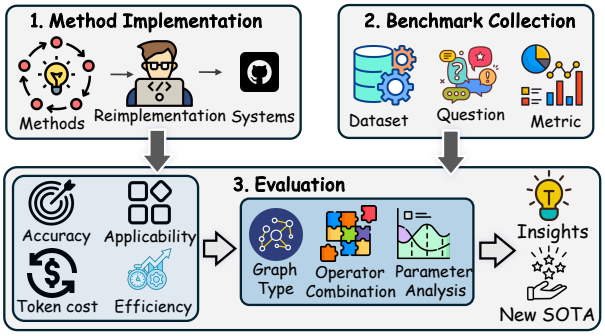


Figure 3: Workflow of our empirical study.

specific options or words. Following existing works [17, 68], we use string recall (STRREC), string exact matching (STREM), and string hit (STRHIT) as evaluation metrics. For abstract QA tasks, we follow prior work [12] and use a head-to-head comparison approach using an LLM evaluator (i.e., GPT-4o). This is mainly because LLMs have demonstrated strong capabilities as evaluators of natural language generation, often achieving state-of-the-art or competitive results when compared to human judgments [77, 90]. Here, we utilize four evaluation dimensions: Comprehensiveness, Diversity, Empowerment, and Overall for abstract QA tasks.

▶ **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 [31]. Besides, Zeroshot, and VanillaRAG 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 method cannot finish in two days, we mark its result as **N/A** in the figures and "—" in the tables.

▶ **Hyperparameter Settings.** In our experiment, we use Llama-3-8B [11] as the default LLM, which is widely used in existing RAG methods [88]. For LLM, we set the maximum token length to 8,000, and use greedy decoding to generate one sample for the

Table 6: 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**

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 [55], 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 [12, 21], we divide the corpus into 1,200-token chunks. For other hyper-parameters of each method, we follow the original settings in their available code.

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 [67].

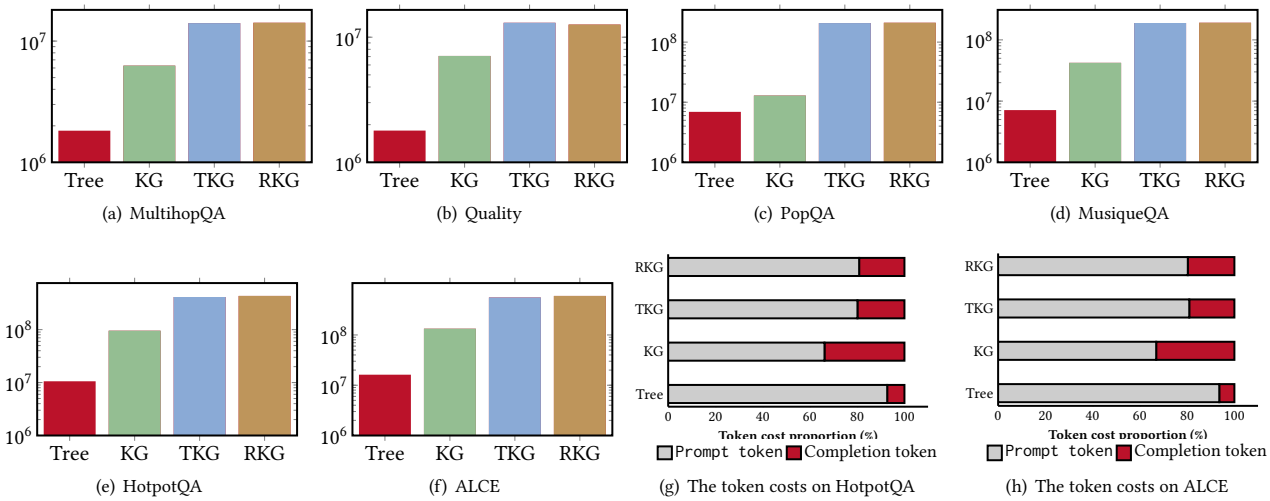
(4) RAPTOR often achieves the best performance among most datasets, especially for simple questions. For complex questions, RAPTOR also performs exceptionally well. This is mainly because, for such questions, high-level summarized information is crucial for understanding the underlying relationships 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 operators used for each method in Table 3). Specifically, RAPTOR leverages similarity-based vector search, comparing the embedding similarity between the given question and the high-level text summary. In contrast, LGraphRAG selects communities based on whether they contain the given entities and the rating score generated by the LLM. However, this approach may retrieve communities that do not align with the question's semantic meaning, potentially degrading LLM performance. To further investigate this issue, we conduct an extra ablation study later (See Exp.4 in this Section).

(5) For the three largest datasets, the K-means [24]-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. To verify this, we compare RAPTOR-K with RAPTOR on the first three datasets, and present results in Table 6. We observe that RAPTOR-K achieves comparable or even better performance than RAPTOR. In the remaining part of our experiments, if RAPTOR does not finish constructing the graph within two days, we use RAPTOR-K instead.

▶ **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. As shown in Figure 4(a) to (f), we observe the following: (1) Building trees consistently require 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. Here, we report the token costs for prompt and completion on HotpotQA and ALCE datasets in Figure 4(g) to (h). The other datasets exhibit similar trends, we include their results in our technical report [67]. We conclude that, regardless of the graph type, the prompt part always incurs higher token costs than the completion part.

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 7 (These results may vary upon rerunning due to the inherent uncertainty of the LLM.). It is not surprising that ZeroShot and VanillaRAG



(a) MultihopQA (b) Quality (c) PopQA (d) MusiqueQA

(e) HotpotQA (f) ALCE (g) The token costs on HotpotQA (h) The token costs on ALCE

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

Table 7: 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

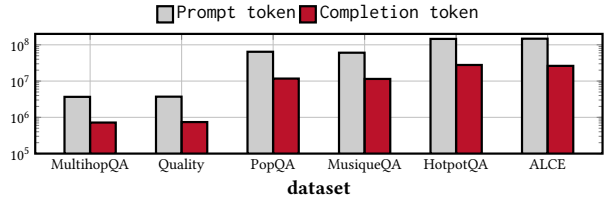


Figure 5: Token cost of index construction in specific QA.

are the most cost-efficient methods in terms of both time and token consumption. Among all graph-based RAG methods, RAPTOR and HippoRAG are typically the most cost-efficient, as 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 [67], which explains why in some datasets, VanillaRAG is even faster than ZeroShot.

► **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 8. 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

Table 8: 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%

Table 9: 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	✓

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 9. 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 10. We make the following analysis: (1) Community reports serve as effective high-level information for complex QA tasks. For instance, VGraphRAG-CC achieves comparable or even better performance than RAPTOR, highlighting the value of community reports. (2) The retrieval strategy in the original LGraphRAG, which selects communities and chunks based on the frequency of relevant entities in a given question, may not be optimal in some cases. This is verified by VGraphRAG-CC consistently outperforming GraphRAG-CC, which suggests that a vector-based retrieval approach is a better way than the heuristic rule-based way. (3) For multi-hop reasoning tasks (e.g., MultihopQA), entity and relationship information can serve as an auxiliary signal, it helps LLM link relevant information (i.e., entity and relationship descriptions), and guide the reasoning process. This is supported by LGraphRAG outperforming both GraphRAG-CC and VGraphRAG-CC on the MultihopQA dataset, indicating the importance of structured graph information for multi-hop reasoning.

▶ **Exp.5. 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, this step is the same as VGraphRAG-CC. 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 10, we can see that VGraphRAG performs best on all complex QA datasets. For example, on the ALCE dataset, it improves STRREC, STREM, and STRHIT by 8.47%, 13.18%, and 4.93%, respectively, compared to VGraphRAG-CC. Meanwhile, compared to RAPTOR, our new algorithm VGraphRAG improves Accuracy by 6.42% on the MultihopQA dataset and 11.6% on the MusiqueQA dataset, respectively.

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) Among graph-based RAG methods, GGraphRAG and RAPTOR generally outperform HLightRAG and FastGraphRAG as they integrate high-level summarized text into the prompt, which is essential for abstract QA tasks. In contrast, the latter two rely solely on low-level graph structures (e.g., entities and relationships) and original text chunks, limiting their effectiveness in handling abstract questions. (3) On almost all datasets, GGraphRAG consistently achieves the best performance, aligning with findings from existing work [12]. This suggests that community reports are highly effective in capturing high-level structured knowledge and relational dependencies among chunks, meanwhile, the Map-Reduce strategy further aids in filtering irrelevant retrieved content. (4) Sometimes, RAPTOR outperforms GGraphRAG, likely because the textual information in the original chunks is crucial for answering certain questions.

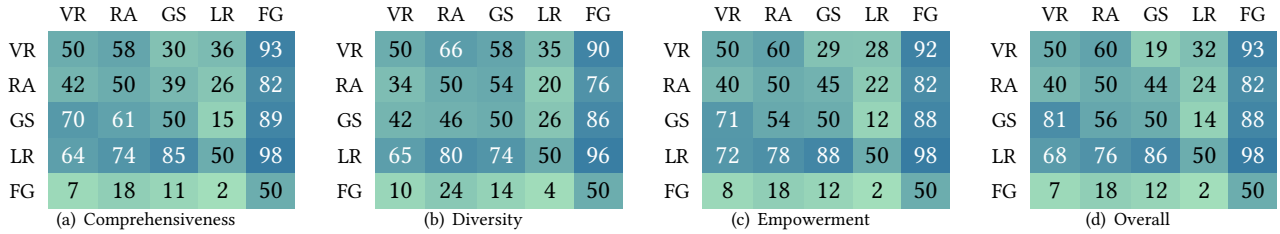
▶ **Exp.2. Token costs of graph and index building.** The token costs of the graph and index building across all abstract QA datasets are shown in Figures 11 and 12 respectively. The conclusions are highly similar to the Exp.2 in Section 7.2.

▶ **Exp.3. 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 11, 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 × more time and 210 × 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.4. New SOTA algorithm.** While the GGraphRAG shows remarkable performance in abstract QA, its time and token costs are not acceptable in practice. Based on the above analysis, we aim to design a cost-efficient version of GGraphRAG, named CheapRAG. Our design is motivated by the fact that: *only a few communities (typically less than 10) are useful for generating responses.* However, in GGraphRAG, all communities within a specified number of layers must be retrieved and analyzed, which is highly token-consuming. Besides, as discussed earlier, original chunks are also valuable for certain questions. Therefore, our new algorithm, CheapRAG, incorporates these useful chunks. Specifically, given a new question Q,

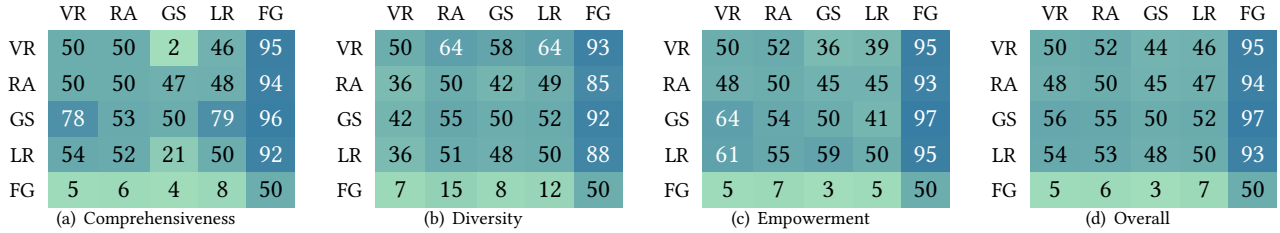
Table 10: Comparison of our newly designed methods on specific datasets with complex questions.

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**



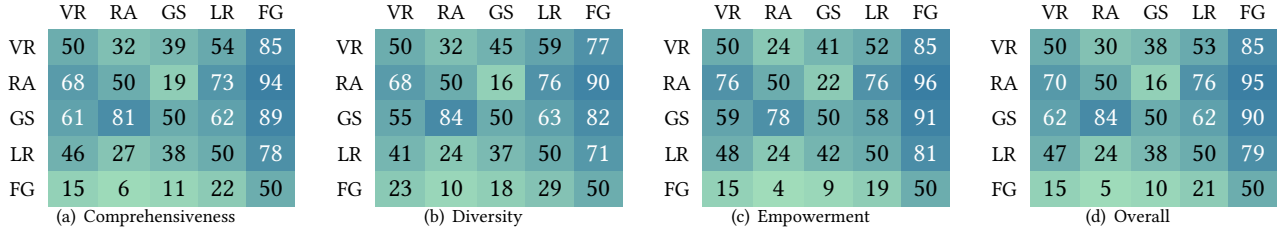
(a) Comprehensiveness (b) Diversity (c) Empowerment (d) Overall

Figure 6: The abstract QA results on Mix dataset.



(a) Comprehensiveness (b) Diversity (c) Empowerment (d) Overall

Figure 7: The abstract QA results on MultihopSum dataset.



(a) Comprehensiveness (b) Diversity (c) Empowerment (d) Overall

Figure 8: The abstract QA results on Agriculture dataset.

our algorithm adopts a vector search-based retrieval strategy to select the most relevant communities and chunks. Next, we apply a `Map-Reduce` strategy to generate the final answer. As shown in Figure 13 and Table 11, `CheapRAG` not only achieves better performance than `GGraphRAG` but also significantly reduces token costs (in most cases). For example, on the MultihopSum dataset, `CheapRAG` reduces token costs by 100× compared to `GGraphRAG`, while achieving better answer quality. While the diversity of answers generated by `CheapRAG` is not yet optimal, we leave this as a future work.

More Analysis. In addition, we present a detailed analysis of the graph-based RAG methods in our technical report [67], including *Effect of chunk size*, *Effect of base model*, and *The size of graph*. Due to the limited space, we only summarize the key conclusions here: (1) Chunk quality is crucial for all methods. (2) Stronger LLM backbone models can further enhance performance for all methods. (3) The constructed graphs are typically very sparse, with their size proportional to the number of chunks.

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 14, we depict a roadmap of the recommended RAG methods, highlighting which methods are best suited for different scenarios.

▶ **L2.** Chunk quality is very important for the overall performance of all RAG methods, and human experts are better at splitting chunks than relying solely on token size.

▶ **L3.** For complex questions in specific QA, high-level information is typically needed, as they capture the complex relationship among

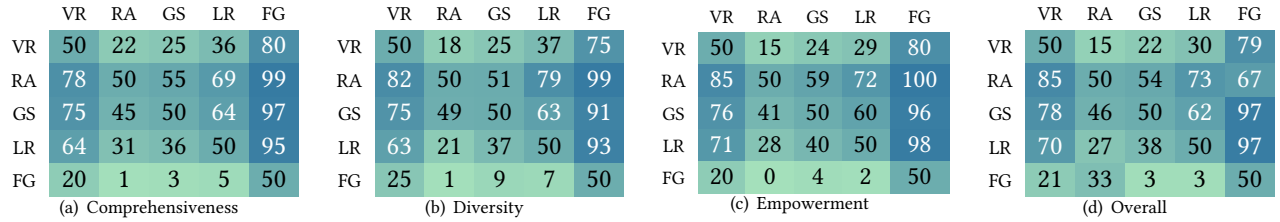


Figure 9: The abstract QA results on CS dataset.

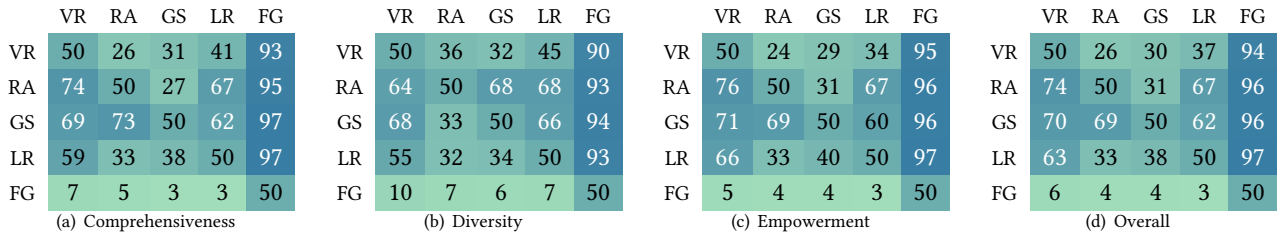


Figure 10: The abstract QA results on Legal dataset.

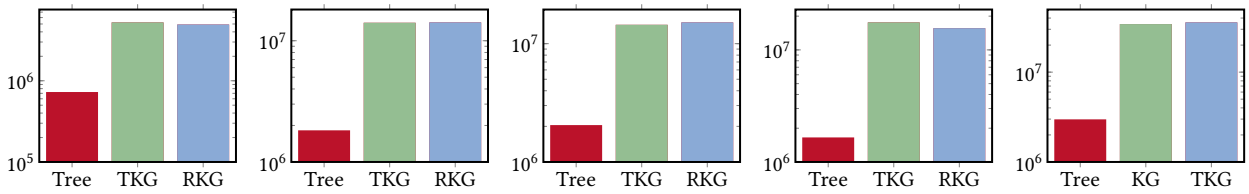


Figure 11: Token cost of the graph building on abstract QA datasets.

Table 11: 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

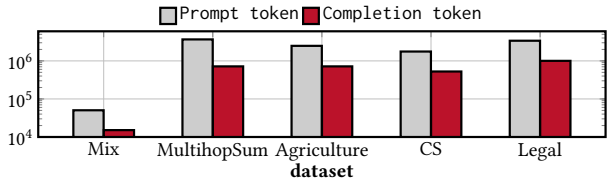


Figure 12: Token cost of index construction in abstract QA.

chunks, and the vector search-based retrieval strategy is better than the rule-based (e.g., `Entity` operator) one.

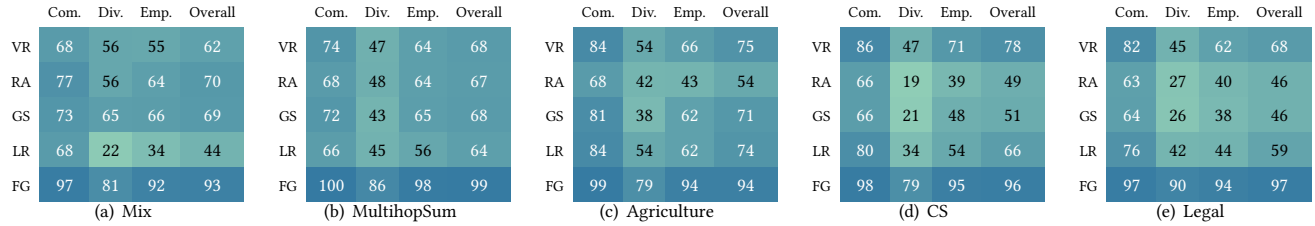
▶ **L4.** Community reports provide a more effective high-level structure than summarized chunk clusters for abstract QA tasks, as they better capture diversified topics and overarching themes within local modules of the corpus.

▶ **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 chunks is not a bad idea.

Opportunities:

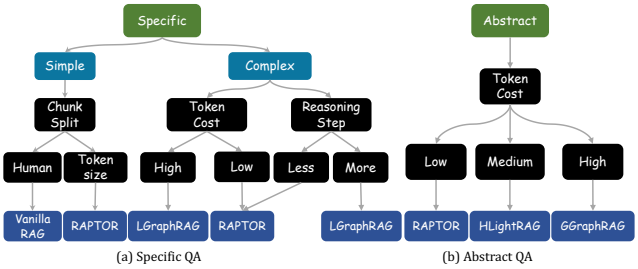
▶ **O1.** All existing graph-based RAG methods (both specific QA and abstract QA) assume the setting of the external corpus is static. What if the external knowledge source evolves over time? For example, Wikipedia articles are constantly evolving, with frequent updates to reflect new information. Can 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.



(a) Mix (b) MultihopSum (c) Agriculture (d) CS (e) Legal

Figure 13: Comparison of our newly designed method on abstract QA datasets.



(a) Specific QA (b) Abstract QA

Figure 14: The taxonomy tree of RAG methods.

▶ **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.** 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.

▶ **O5.** The well-known graph database systems, such as Neo4j [59] and Nebula [58], support transferring the corpus into a knowledge graph via LLM. However, enabling these popular systems to support the various graph-based RAG methods presents an exciting opportunity. More details are in our technical report [67].

9 RELATED WORKS

In this section, we review the related works, including Retrieval-Augmentation-Generation (RAG) frameworks, and LLMs for graph data management. Besides, works on employing LLM for enhancing database performance are also discussed.

• **RAG frameworks.** RAG has been proven to excel in many tasks, including open-ended question answering [32, 71], programming context [5–7], SQL rewrite [43, 73], automatic DBMS configuration debugging [70, 93], and data cleaning [56, 57, 66]. The naive RAG technique relies on retrieving query-relevant information from external knowledge bases to mitigate the "hallucination" of LLMs. Recently, most RAG approaches [12, 21, 22, 39, 64, 68, 81, 82] 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. In terms of open-source software, a variety of graph databases are supported by both the LangChain [36] and LlamaIndex [50] libraries, while a more general class of graph-based RAG applications is also emerging, including systems that can create and reason over knowledge graphs in both Neo4j [59] and NebulaGraph [58]. For more details, please refer to the recent surveys of graph-based RAG methods [23, 64].

• **LLMs for graph data management.** Recent advances in LLMs have demonstrated superiority in natural language understanding and reasoning, which offers opportunities to leverage LLMs to graph data management. These include using LLMs for knowledge graph (KG) creation [81] and completion [86], as well as for the extraction of causal graphs [4] from source texts. They also include advanced methods by employing KG to enhance the LLM reasoning. For instance, KD-CoT [79] retrieves facts from an external knowledge graph to guide the CoT performed by LLMs. RoG [51] proposes a planning-retrieval-reasoning framework that retrieves reasoning paths from KGs to guide LLMs conducting trust-worthy reasoning. To capture graph structure, GNN-RAG [54] adopts a lightweight graph neural network to retrieve elements from KGs effectively. StructGPT [33] and ToG [72] treat LLMs as agents that interact with KGs to find reasoning paths.

• **LLMs for database.** Due to the wealth of developer experience captured in a vast array of database forum discussions, recent studies [5, 13, 20, 37, 43, 73, 93, 94] have begun leveraging LLMs to enhance database performance. For instance, GPTuner [37] proposes to enhance database knob tuning using LLM by leveraging domain knowledge to identify important knobs and coarsely initialize their values for subsequent refinement. Besides, D-Bot [93] proposes an LLM-based database diagnosis system, which can retrieve relevant knowledge chunks and tools, and use them to identify typical root causes accurately. The LLM-based data analysis systems and tools are also been studied [2, 8, 44–48, 63].

To the best of our knowledge, our work is the first study that provides a unified framework for all existing graph-based RAG methods and compares existing solutions comprehensively via in-depth experimental results.

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, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] 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).

[3] 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).

[4] Taiyu Ban, Lyvzhou Chen, Xiangyu Wang, and Huanhuan Chen. 2023. From query tools to causal architects: Harnessing large language models for advanced causal discovery from data. *arXiv preprint arXiv:2306.16902* (2023).

[5] Sibei 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).

[6] Sibei 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.

[7] Sibei Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong 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.

[8] 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.

[9] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[10] 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).

[11] 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).

[12] 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).

[13] 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.

[14] 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.

[15] 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.

[16] FastGraphRAG. 2024. FastGraphRAG. https://github.com/circlemind-ai/fast-graphrag.

[17] 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).

[18] 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).

[19] 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).

[20] Victor Giannankouris and Immanuel Trummer. 2024. {\lambda}-Tune: Harnessing Large Language Models for Automated Database System Tuning. *arXiv preprint arXiv:2411.03500* (2024).

[21] 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.

[22] 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).

[23] Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309* (2024).

[24] Jiawei Han, Jian Pei, and Hanghang Tong. 2022. *Data mining: concepts and techniques*. Morgan kaufmann.

[25] Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*. 517–526.

[26] 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).

[27] 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).

[28] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian 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).

[29] Yizheng Huang and Jimmy Huang. 2024. A Survey on Retrieval-Augmented Text Generation for Large Language Models. *arXiv preprint arXiv:2404.10981* (2024).

[30] 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).

[31] huawei. 2019. Ascend GPU. https://e.huawei.com/ph/products/computing/ascend.

[32] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403* (2024).

[33] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 9237–9251.

[34] Omar Khattab, Arnav 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).

[35] 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.

[36] Langchian. 2023. Langchian. https://python.langchain.com/docs/additional_resources/arxiv_references/.

[37] 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. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1939–1952.

[38] 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).

[39] 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).

[40] Lan Li, Liri Fang, and Vetle I Torvik. 2024. AutoDCWorkflow: LLM-based Data Cleaning Workflow Auto-Generation and Benchmark. *arXiv preprint arXiv:2412.06724* (2024).

[41] 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.

[42] 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).

[43] 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.

[44] 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).

[45] Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. 2025. TWIX: Automatically Reconstructing Structured Data from Templatized Documents. *arXiv preprint arXiv:2501.06659* (2025).

[46] 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).

[47] 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).

[48] 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).

[49] 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).
[50] llamaindex. 2023. llamaindex. https://www.llamaindex.ai/.
[51] Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2023. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061* (2023).
[52] 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.
[53] 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).
[54] Costas Mavromatis and George Karypis. 2024. GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning. *arXiv preprint arXiv:2405.20139* (2024).
[55] Multi-Linguality Multi-Functionality Multi-Granularity. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. (2024).
[56] 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.
[57] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment* 16, 4 (2022), 738–746.
[58] nebula. 2010. nebula. https://www.nebula-graph.io/.
[59] neo4j. 2006. neo4j. https://neo4j.com/.
[60] 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).
[61] 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.
[62] 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).
[63] 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).
[64] 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).
[65] 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).
[66] 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.
[67] 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.
[68] 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).
[69] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, 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).
[70] Vikramank Singh, Kapil Eknath Vaidya, Vinayshekhar Bannihatti Kumar, Sopan Khosla, Murali Narayanaswamy, Rashmi Gangadharaiah, and Tim Kraska. 2024. Panda: Performance debugging for databases using LLM agents. (2024).
[71] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, 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.
[72] Jiashuo Sun, Chengjin Xu, Lumingyuan 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*.
[73] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).
[74] Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391* (2024).
[75] 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.
[76] Harsh Trivedi, Niranjan 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.
[77] Jiaan 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).
[78] 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).
[79] Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. 2023. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259* (2023).
[80] 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).
[81] 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.
[82] 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).
[83] Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, 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).
[84] 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).
[85] 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).
[86] Liang Yao, Jiazhen Peng, Chengsheng Mao, and Yuan Luo. 2023. Exploring large language models for knowledge graph completion. *arXiv preprint arXiv:2308.13916* (2023).
[87] 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).
[88] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran 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).
[89] 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).
[90] 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.
[91] 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.
[92] 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).
[93] 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.
[94] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: Large language model meets database. *Data Science and Engineering* 9, 1 (2024), 102–111.

A ADDITIONAL EXPERIMENTS

A.1 More analysis

In this Section, we present a more detailed analysis of graph-based RAG methods from the following angles.

▶ **Exp.1. 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 15, 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.

▶ **Exp.2. Effect of the base model.** In this experiment, we evaluate the effect of the LLM backbone by replacing Llama-3-8B with Llama-3-70B [11] on the MultihopQA and ALCEAll datasets. We make the following observations: (1) All methods achieve performance improvements when equipped with a more powerful backbone LLM (i.e., Llama-3-70B) compared to Llama-3-8B. For example, on the ALCEALL dataset, replacing the LLM from 8B to 70B, `ZeroShot` improves STRREC, STREM, and STRHIT by 102.1%, 94.2%, and 101.7%, respectively. (2) Our proposed method, `VGraphRAG`, consistently achieves the best performance regardless of the LLM backbone used. For example, on the MultihopQA dataset, `VGraphRAG` with Llama-3-70B achieves 7.20% and 12.13% improvements over `RAPTOR` with Llama-3-70B in terms of Accuracy and Recall, respectively. (3) Under both Llama-3-70B and Llama-3-8B, while all methods show improved performance, they exhibit similar trends to those observed with Llama-3-8B. For instance, `RAPTOR` remains the best-performing method among all existing graph-based RAG approaches, regardless of the LLM used.

▶ **Exp.3. The size of graph.** For each dataset, we report the size of five types of graphs in Table 13. We observe that PG is typically denser than other types of graphs, as they connect nodes based on 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.

A.2 Additional results on token costs.

As shown in Figure 16, we present the proportions of token costs for the additional eight datasets, which exhibit trends similar to those observed in HotpotQA and ALCE.

A.3 Additional results of generation token costs.

As shown in Figure 17, 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.

A.4 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 [3, 53, 69]. 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 18. 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 [12, 21], 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 19.

A.5 Implementation details

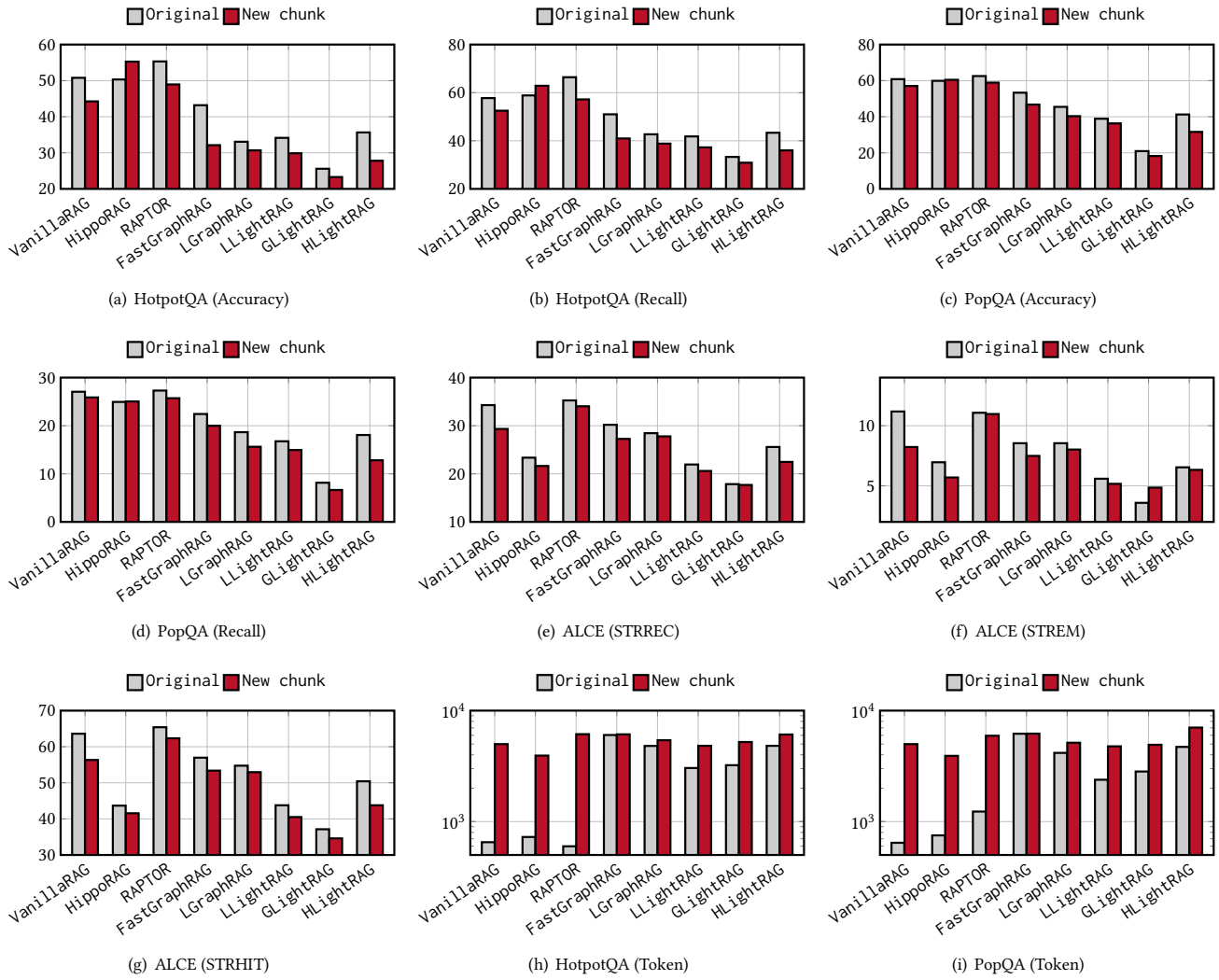
In this subsection, we present more details about our system implementation. Specifically, we use HNSW [52] from Llama-index [50] (a well-known open-source project) as the default vector database for efficient vector search. In addition, for each method, we optimize efficiency by batching or parallelizing operations such as encoding nodes or chunks, and computing personalized page rank, among others, during the retrieval stage.

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`.



(a) HotpotQA (Accuracy) (b) HotpotQA (Recall) (c) PopQA (Accuracy)

(d) PopQA (Recall) (e) ALCE (STRREC) (f) ALCE (STREM)

(g) ALCE (STRHIT) (h) HotpotQA (Token) (i) PopQA (Token)

Figure 15: Effect of chunk quality on the performance of specific QA tasks.

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 section, 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

▶ **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

Table 12: The specific QA performance comparison of graph-based RAG methods with different LLM backbone models.

Method	LLM backbone	MultihopQA		ALCEAll		
		Accuracy	Recall	STRREC	STREM	STRHIT
Zeroshot	Llama-3-8B	49.022	34.256	15.454	3.692	30.696
	Llama-3-70B	55.908	52.987	31.234	7.170	61.920
VanillaRAG	Llama-3-8B	50.626	36.918	29.334	8.228	56.329
	Llama-3-70B	56.768	49.127	34.961	9.810	68.038
HippoRAG	Llama-3-8B	53.760	47.671	21.633	5.696	41.561
	Llama-3-70B	57.277	57.736	32.904	9.916	32.534
RAPTOR	Llama-3-8B	56.064	44.832	34.044	10.971	62.342
	Llama-3-70B	**63.028**	**61.042**	**37.286**	**12.236**	**68.671**
FastGraphRAG	Llama-3-8B	52.895	44.278	27.258	7.490	53.376
	Llama-3-70B	54.069	55.787	35.658	12.236	65.612
LGraphRAG	Llama-3-8B	55.360	50.429	27.785	8.017	52.954
	Llama-3-70B	58.060	55.390	34.256	10.232	66.561
VGraphRAG	Llama-3-8B	59.664	50.893	35.213	11.603	64.030
	Llama-3-70B	**67.567**	**68.445**	**37.576**	**12.447**	**69.198**

Table 13: 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

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.

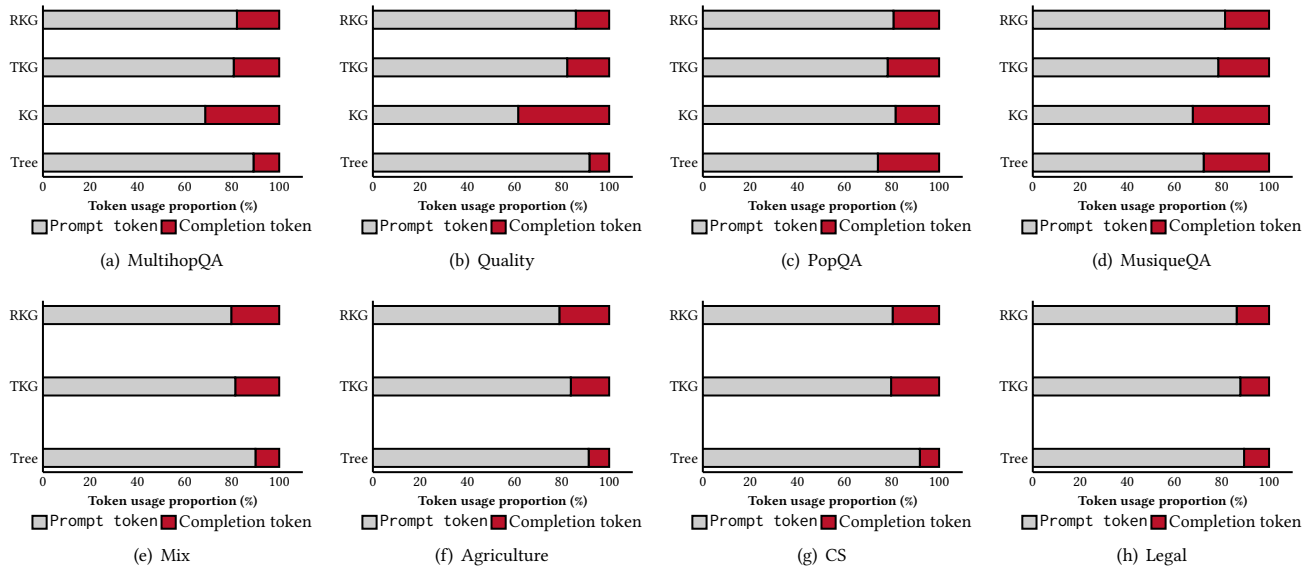
▶ **O9.** Existing methods do not fully leverage the graph structure; they typically rely on simple graph patterns (e.g., nodes, edges, or k-hop paths). Although GraphRAG adopts a hierarchical community structure (detecting by the Leiden algorithm), this approach does not consider node attributes, potentially compromising the quality of the communities. That is, determining which graph structures are superior remains an open question.

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 retrieval-enhanced models.

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.



(a) MultihopQA (b) Quality (c) PopQA (d) MusiqueQA

(e) Mix (f) Agriculture (g) CS (h) Legal

Figure 16: Proportion of the token costs for prompt and completion in graph building stage across all datasets.

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.

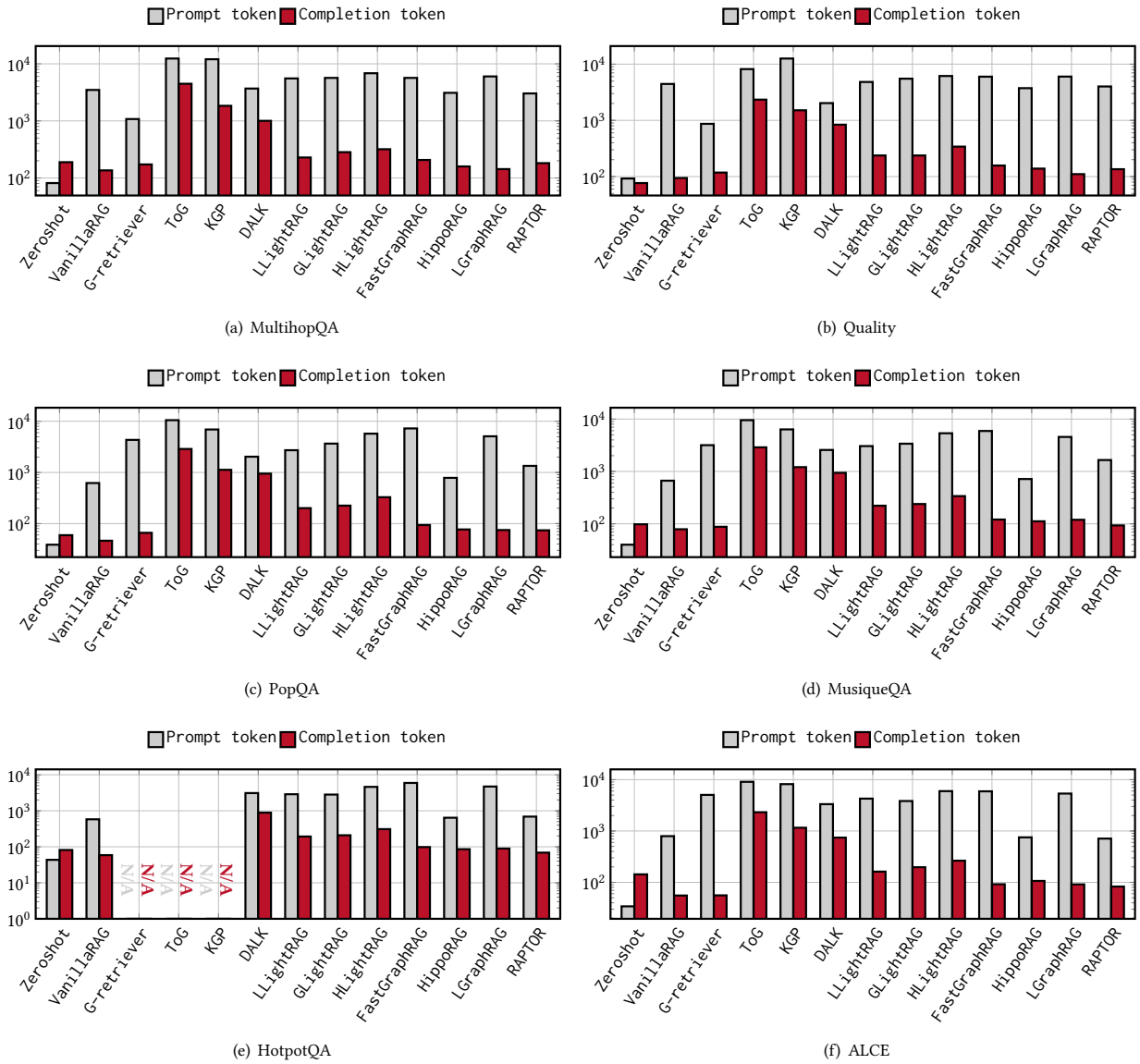
Finally, our framework contributes to the broader research community by providing a standardized methodology to assess graph-based RAG approaches. The introduction of a unified evaluation testbed ensures reproducibility, promotes fair a benchmark, and facilitates future innovations in RAG-based LLM applications.

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 [34], 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.

C PROMPT



(a) MultihopQA

(b) Quality

(c) PopQA

(d) MusiqueQA

(e) HotpotQA

(f) ALCE

Figure 17: Token costs for prompt and completion tokens in the generation stage across all datasets.

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 18: The prompt for generating abstract questions.

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 19: The prompt for the evaluation of abstract QA.