In-depth Analysis of Graph-based RAG in a Unified Framework

Yingli Zhou
The Chinese University of Hong Kong, Shenzhen, China
yinglizhou@link.cuhk.edu.cn

Yaodong Su
The Chinese University of Hong Kong, Shenzhen, China
224045002@link.cuhk.edu.cn

Youran Sun
The Chinese University of Hong Kong, Shenzhen, China
youransun@link.cuhk.edu.cn

Shu Wang
The Chinese University of Hong Kong, Shenzhen, China
shuwang3@link.cuhk.edu.cn

Taotao Wang
The Chinese University of Hong Kong, Shenzhen, China
taotaowang@link.cuhk.edu.cn

Runyuan He
The Chinese University of Hong Kong, Shenzhen, China
123090163@link.cuhk.edu.cn

Yongwei Zhang
Huawei Cloud Computing Technologies Co., Ltd.
zhangyongwei11@huawei.com

Sicong Liang
Huawei Cloud Computing Technologies Co., Ltd.
liangsicong2@huawei.com

Xilin Liu
Huawei Cloud Computing Technologies Co., Ltd.
liuxilin3@huawei.com

Yuchi Ma
Huawei Cloud Computing Technologies Co., Ltd.
mayuchi1@huawei.com

Yixiang Fang*
The Chinese University of Hong Kong, Shenzhen, China
fangyixiang@cuhk.edu.cn

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. PVLDB, 18(13): 5623 - 5637, 2025.
doi:10.14778/3773731.3773738

*Yixiang Fang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 13 ISSN 2150-8097.
doi:10.14778/3773731.3773738

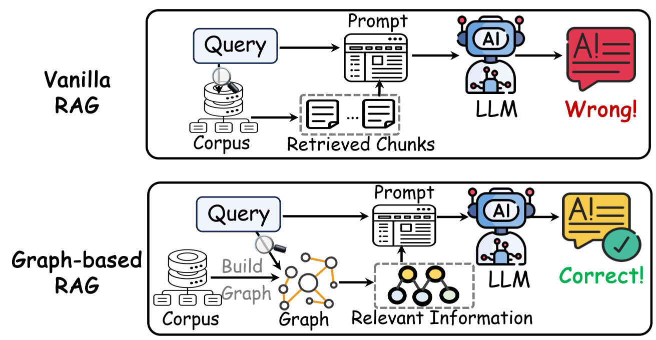


Figure 1: Overview of vanilla RAG and graph-based RAG.

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 [87], and Llama 3.1 [14] has sparked a revolution in the field of artificial intelligence [22, 31, 46, 55, 60, 80, 81, 95]. 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].

To bridge this gap, the Retrieval Augmented Generation (RAG) technique [18, 21, 30, 32, 84, 89, 93] has been proposed, which supplements LLM with external knowledge to enhance its factual accuracy and trustworthiness. Given a user query Q, the key idea of naive-based RAG [41] (i.e., vanilla RAG) is to retrieve relevant chunks from the external corpus, and then feed them along with Q as a prompt into LLM to generate answers. Consequently, RAG techniques have been widely applied in various fields, especially in domains where LLMs need to generate reliable outputs, such as

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

Method	Graph Type	Index Component	Retrieval Primitive	Retrieval Granularity	Specific QA	Abstract QA
`RAPTOR` [69]	Tree	Tree node	Question vector	Tree node	✓	✓
`KGP` [82]	Passage Graph	Entity	Question	Chunk	✓	✗
`HippoRAG` [25]	Knowledge Graph	Entity	Entities in question	Chunk	✓	✗
`G-retriever` [29]	Knowledge Graph	Entity, Relationship	Question vector	Subgraph	✓	✗
`ToG` [74]	Knowledge Graph	Entity, Relationship	Question	Subgraph	✓	✗
`DALK` [43]	Knowledge Graph	Entity	Entities in question	Subgraph	✓	✗
`LGraphRAG` [15]	Textual Knowledge Graph	Entity, Community	Question vector	Entity, Relationship, Chunk, Community	✓	✗
`GGraphRAG` [15]	Textual Knowledge Graph	Community	Question vector	Community	✗	✓
`FastGraphRAG` [19]	Textual Knowledge Graph	Entity	Entities in question	Entity, Relationship, Chunk	✓	✓
`LLightRAG` [24]	Rich Knowledge Graph	Entity, Relationship	Low-level keywords in question	Entity, Relationship, Chunk	✓	✓
`GLightRAG` [24]	Rich Knowledge Graph	Entity, Relationship	High-level keywords in question	Entity, Relationship, Chunk	✓	✓
`HLightRAG` [24]	Rich Knowledge Graph	Entity, Relationship	Both high- and low-level keywords	Entity, Relationship, Chunk	✓	✓

healthcare [55, 80, 95], finance [46, 60], and education [22, 81]. Moreover, RAG has proven highly useful in many data management tasks, including NL2SQL [16, 42], data cleaning [17, 44, 58, 67], knob tuning [23, 40], DBMS diagnosis [71, 96, 97], and SQL rewrite [48, 75]. In turn, the database community has recently begun to actively explore how to build efficient and reliable RAG systems [2, 36]. Due to the important role of the RAG technique in LLM-based applications, numerous RAG methods have been proposed in the past year [30, 45]. Among these methods, the state-of-the-art RAG approaches typically use the graph data as the external data (also called graph-based RAG), since they capture the rich semantic information and link relationships between entities. Unlike vanilla RAG, graph-based RAG methods retrieve relevant information related to the query Q—such as nodes, relationships, or subgraphs—from the graph, and then incorporate this information into the prompt along with Q for the LLM to generate an answer. The overview of naive-based RAG and graph-based RAG is shown in Figure 1.

Recently, several mainstream database systems have started supporting graph-based RAG, including PostgreSQL [65], Neo4j [59], and Databricks [11]. At the same time, graph-based RAG has become a core component in modern graph-native agentic systems, such as LangGraph [39] and Chat2Graph [5]. 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 [9, 15, 24, 25, 33, 43, 47, 64, 69, 82, 83, 85, 90, 99]. In Table 1, we summarize the key characteristics of 12 representative graph-based RAG methods based on the graph types they rely on, their index components, retrieval primitives and granularity, and the types of tasks they support. After a careful literature review, we make the following observations. First, no prior work has proposed a unified framework to abstract the graph-based RAG solutions and identify key performance factors. Second, existing works focus on evaluating the overall performance, but not individual components. Third, there is no existing comprehensive comparison between all these methods in terms of accuracy and efficiency.

Our work. To address the above issues, in this paper, we conduct an in-depth study on graph-based RAG methods. We first summarize a novel unified framework with four stages, namely 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 $\sim$ 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. Section 9 reviews related work while Section 10 summarizes the paper.

2 PRELIMINARIES

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

2.1 Large Language Models (LLMs)

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

LLM Prompting. After instruction tuning on large corpus of human interaction scenarios, LLM is capable of following human instructions to complete different tasks [13, 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].

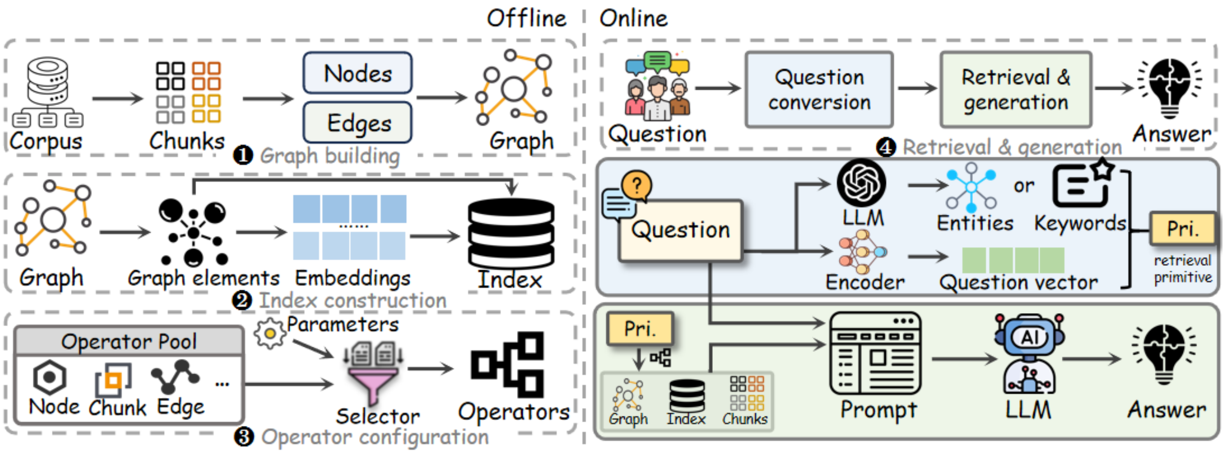


Figure 2: Workflow of graph-based RAG methods under our unified framework.

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

2.2 Graph-based RAG

Unlike vanilla RAG, graph-based RAG methods employ graph structures built from external corpus to enhance contextual understanding in LLMs and generate more informed and accurate responses [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.

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.

Specifically, given the large corpus $\mathcal{D}$, we first split it into multiple chunks C (line 1). We then sequentially execute operations in the following four stages (lines 2-5): (1) Build the graph $\mathcal{G}$ for input chunks C (Section 4); (2) Construct the index based on the graph $\mathcal{G}$ from the previous stage (Section 5); (3) Configure the retriever operators for subsequent retrieving stages (Section 6), and (4) For the input user question Q, retrieve relevant information from $\mathcal{G}$ using the selected operators and feed them along with the question Q into the LLM to generate the answer. The workflow of graph-based RAG methods under our framework is shown in Figure 2. We note that graph-based RAG methods differ across the four stages. Specifically, as shown in Table 1, different methods construct distinct types of graphs in Stage ❶, build different indices in Stage ❷, and retrieve information at varying levels of granularity. Consequently, they employ different operators for retrieval, leading to variations in both Stage ❸ and Stage ❹. Note that the query is first converted into a retrieval primitive, which serves as the basis for retrieval in each method (see Section 6.3).

Algorithm 1: A unified framework for graph-based RAG

input : Corpus $\mathcal{D}$, user question Q, and parameters $\mathcal{P}$
output: The answers for user question Q
1 $C \leftarrow$ 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`($\mathcal{P}$);
// (4) Retrieve relevant information and generate response.
5 $\mathcal{R} \leftarrow$ `Retrieval&generation`($\mathcal{G}$, $\mathcal{I}$, $\mathcal{O}$, Q, C);
6 **return** $\mathcal{R}$;

4 GRAPH BUILDING

The *graph building* stage aims to transfer the input corpus into a graph, serving as a fundamental component in graph-based RAG methods. Before building a graph, the corpus is first split into smaller chunks. Then, an LLM or other tools are used to construct nodes and edges based on these chunks, as shown in Figure 2❶. We note that this preprocessing step is essential for all RAG methods. Further details are provided in our technical report [68]. 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 [82]. If two chunks contain a number of the same entities larger than a threshold, we link an edge for these two nodes.

❷ *Tree.* The tree is constructed in a progressive manner, where each chunk represents the leaf node in the tree. Then, it uses an LLM to generate higher-level nodes. Specifically, at the i-th layer, the nodes of $(i + 1)$-th layer are created by clustering nodes from

the i-th layer that does not yet have parent nodes. For each cluster with more than two nodes, the LLM generates a virtual parent node with a high-level summary of its child node descriptions.
❸ *Knowledge Graph.* The knowledge graph (KG) is constructed by extracting entities and relationships from each chunk, where each entity represents an object and the relationship denotes the semantic relation between two entities.
❹ *Textual Knowledge Graph.* A textual knowledge graph (TKG) is a specialized KG (following the same construction step as KG), with the key difference being that in a TKG, each entity and relationship is assigned a brief textual description.
❺ *Rich Knowledge Graph.* The rich knowledge graph (RKG) is an extended version of TKG, containing more information, including textual descriptions for entities and relationships, as well as keywords for relationships. We summarize the key characteristics of each graph type in Table 2, considering their contained attributes (e.g., inclusion of entity names and descriptions), the time and number of tokens required for construction, the resulting graph size, and the richness of information contained within each graph. In addition, we provide a case study of how the five types of graphs are constructed from the external corpus in Figure 3. The detailed descriptions are shown in our technical report [68].

5 INDEX CONSTRUCTION

To support efficient online querying, existing graph-based RAG methods typically include an index-construction stage, which involves storing graph elements — such as entities and relationships — in a vector database and computing community reports to enable efficient online retrieval, as shown in Figure 2❷. 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 [12] or ColBert [38] 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 [27], 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 [77] algorithm is utilized by `LGraphRAG` and `GGraphRAG`.

Remark. *Relationship index* tends to have a larger size, whereas the *Community index* is more compact incurs the highest construction cost in terms of tokens and time, and provides the detailed comparisons of these indices in our technical report [68].

6 RETRIEVAL AND GENERATION

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

6.1 Retrieval operators

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

• *Node type.* This type of operator focuses on retrieving "important" nodes for a given question, and based on the selection policy, there are seven different operators to retrieve nodes. 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 [28] to identify the top-k similar nodes to the question, where the restart probability of each node is based on its similarity to the entities in the given question. ❹ `Agent` utilizes the capabilities of LLMs to select nodes from a list of candidate nodes. ❺ `Onehop` selects the one-hop neighbor entities of the given entities. ❻ `Link` selects the top-1 most similar entity for each entity in the given set from the vector database. ❼ `TF-IDF` retrieves the top-k relevant entities by ranking them based on term frequency and inverse document frequency from the TF-IDF matrix.

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

• *Chunk type.* The operators in this type aim to retrieve the most relevant chunks to the given question. There are three operators: ❶ `Aggregator`, ❷ `FromRel`, and ❸ `Occurrence`. Specifically, `Aggregator` uses the relationship score vector from the `Link` operator and a relationship–chunk interaction matrix to aggregate chunk scores via matrix multiplication, selecting the top-k chunks with the highest scores. For `FromRel`: Given a set of relationships, all chunks

Table 2: Comparison of different types of graph.

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

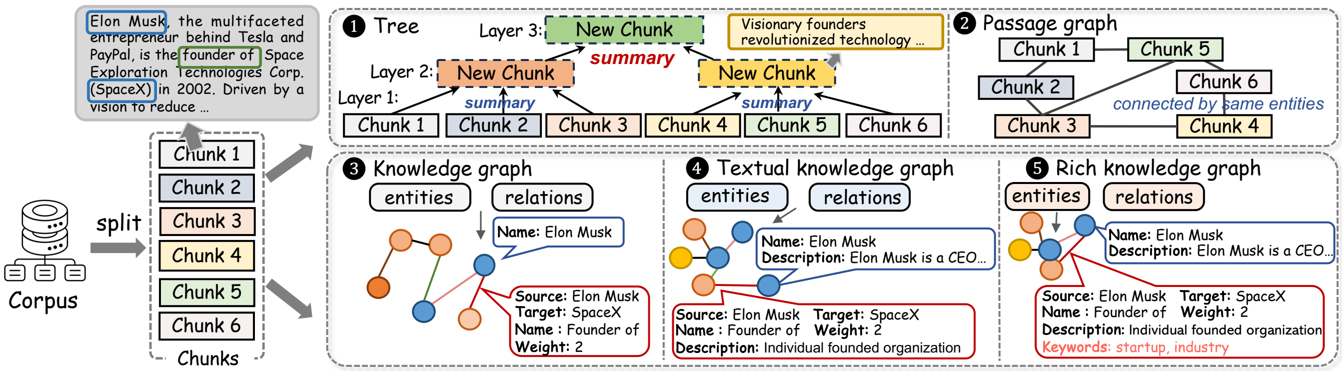


Figure 3: Examples of five types of graphs.

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

containing at least one of them are retrieved. The `Occurrence` selects the top-k chunks based on the given relationships. Specifically, for each relationship, we identify its two associated entities. If both entities appear in the same chunk, the chunk's score is incremented by 1. After processing all relationships, the top-k chunks with the highest scores are selected.

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

• *Community type.* Only the `LGraphRAG` and `GGraphRAG` using the community operators, which includes two detailed operators, ❶ `Entity`, and ❷ `Layer`. This operator first identifies communities that contain the specified entities, with each community maintaining an associated entity list. It then ranks the selected communities based on relevance scores assigned by the LLM, returning the top-k highest-scoring ones. Each community is associated with a layer attribute, and the `Layer` operator retrieves all communities at or below the specified layer.

6.2 Operator configuration

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

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

6.3 Retrieval & generation

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

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

Based on the retrieval primitive $\mathcal{D}$ and the selected operators, the most relevant information to Q is retrieved and combined with Q to form the final prompt for LLM response generation. Generally, there are two types of answer generation paradigms: ❶ `Directly` and ❷ `Map-Reduce`. The former directly utilizes the LLM to generate the answer. The `Map-Reduce` strategy in `GGraphRAG` prompts the LLM to generate partial answers and confidence scores from each retrieved community. These (answer, score) pairs are ranked, and the top ones are appended to form the final prompt for answer generation. An example is shown in [68].

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 [68].

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	<u>33,595</u>	Specific QA
MusiqueQA	3,280,174	3,000	<u>29,898</u>	Specific QA
HotpotQA	8,495,056	3,702	<u>66,581</u>	Specific QA
ALCE	13,490,670	948	<u>89,562</u>	Specific QA
Mix	611,602	125	61	Abstract QA
MultihopSum	1,434,889	125	609	Abstract QA
Agriculture	1,949,584	125	12	Abstract QA
CS	2,047,923	125	10	Abstract QA
Legal	4,774,255	125	94	Abstract QA

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

- *Specific.* This group focuses on detail-oriented questions referencing specific entities (e.g., "Who won the 2024 U.S. presidential election?"). We divide them into two types based on complexity: *Simple* (answerable from one or two chunks without reasoning): Quality [62], PopQA [56], HotpotQA [88]) and *Complex* (requiring multi-hop reasoning and synthesis): MultihopQA [76], MusiqueQA [78], ALCE [20]).
- *Abstract.* Unlike the previous groups, the questions in this category are not centered on specific factual queries. Instead, they involve abstract, conceptual inquiries that encompass broader topics, summaries, or overarching themes. An example of an abstract question is: "How does artificial intelligence influence modern education?". The abstract question requires a high-level understanding of the dataset contents, including five datasets: Mix [66], MultihopSum [76], Agriculture [66], CS [66], and Legal [66].

Their statistics, including the numbers of tokens, and questions, and the question-answering (QA) types are reported in Table 4. For specific (both complex and simple) QA datasets, we use the questions provided by each dataset. While for abstract QA datasets, We follow the question generation method introduced in [15], using GPT-4o to generate 125 questions per dataset with controlled difficulty, which is also aligned with existing works [15, 19, 24, 64]. The details and prompt template used for question generation are provided in our technical report [68]. 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 [56, 70]. For the ALCE dataset, answers are typically full sentences rather than specific options or words. Following existing works [20, 69], we use string recall (STRREC), string exact matching (STREM), and

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**†

string hit (STRHIT) as evaluation metrics. For abstract QA tasks, we adopt four evaluation metrics following prior works [15, 24]: Comprehensiveness, Diversity, Empowerment, and Overall, which assess answer quality from different perspectives. We employ a head-to-head comparison strategy using GPT-4o as the evaluator. Specifically, for each pair of answers, the LLM is prompted to judge which one is better with respect to a given metric, rather than assigning explicit scores. This comparative approach is motivated by the strong performance of LLMs as evaluators of natural language generation, often matching or exceeding human judgments [79, 94]. We provide detailed descriptions of the evaluation protocol and example case studies for all four metrics in our technical report [68].

▶ **Implementation.** We implement all the algorithms in Python with our proposed unified framework and try our best to ensure a native and effective implementation. All experiments are run on 350 Ascend 910B-3 NPUs [34]. Besides, Zeroshot [4], and vanilla RAG (denoted by VanillaRAG) [41] 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 experiments, we use Llama-3-8B [14] as the default LLM, not only because it is the most widely adopted model in recent RAG studies [92], but also due to its strong capabilities in language understanding and reasoning [14], as well as its practical efficiency for deployment. For LLM, we set the maximum token length to 8,096, and use greedy decoding to generate one sample for the deterministic output. For each method requiring top-k selection (e.g., chunks or entities), we set $k = 4$ to accommodate the token length limitation. We use one of the most advanced text-encoding models, BGE-M3 [57], as the embedding model across all methods to generate embeddings for vector search. If an expert annotator pre-splits the dataset into chunks, we use those as they preserve human insight. Otherwise, following existing works [15, 24], we divide the corpus into 1,200-token chunks. For other hyperparameters of each method, we adopt the original code settings when available; otherwise, we reproduce them based on the configurations described in the corresponding papers.

7.2 Evaluation for specific QA

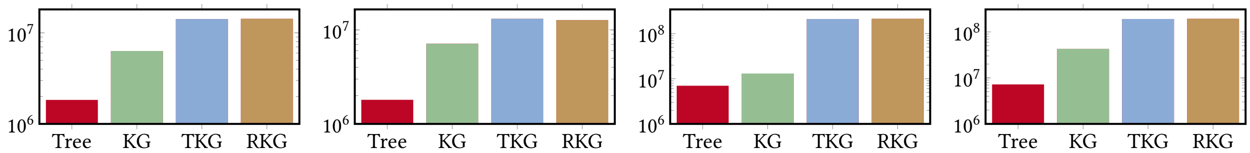
In this section, we evaluate the performance of different methods on specific QA tasks.

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

(2) For specific QA tasks, retaining the original text chunks is crucial for accurate question answering, as the questions and answers in these datasets are derived from the text corpus. This may explain why G-retriever, ToG, and DALK, which rely solely on graph structure information, perform poorly on most datasets. However, on MultihopQA, which requires multi-hop reasoning, DALK effectively retrieves relevant reasoning paths, achieving accuracy and recall improvements of 6.57% and 27.94% over VanillaRAG, respectively.

(3) If the dataset is pre-split into chunks by the expert annotator, VanillaRAG often performs better compared to datasets where chunks are split based on the token size, and we further investigate this phenomenon later in our technical report [68].

(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



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

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

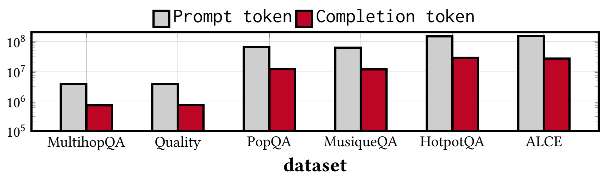


Figure 5: Token cost of index construction on specific QA.

question). However, we only observe this effect on the MultihopQA dataset. For the other two complex QA datasets, LGraphRAG even underperforms compared to VanillaRAG. Meanwhile, RAPTOR still achieves the best performance on these two datasets. We hypothesize that this discrepancy arises from differences in how high-level information is retrieved (See Table 3).

(5) For the three largest datasets, the K-means [27]-based RAPTOR (denoted as RAPTOR-K) also demonstrates remarkable performance. This suggests that the clustering method used in RAPTOR merely impacts overall performance. This may be because different clustering methods share the same key idea: grouping similar items into the same cluster. Therefore, they may generate similar chunk clusters. We note that RAPTOR-K achieves comparable or even better performance than RAPTOR, and the detailed results are shown in technical report [68]. If RAPTOR does not finish constructing the graph within two days, we use RAPTOR-K instead.

Remark. We note that not all graph-based RAG methods consistently outperform the baseline VanillaRAG on every question. By carefully analyzing the failure case of the top-performing methods, we examine why HippoRAG, RAPTOR, and LGraphRAG sometimes fall short in specific QA tasks. Please see the failure cases in [68].

▶ **Exp.2. Token costs of graph and index building.** In this experiment, we first report the token costs of building four types of graphs across all datasets. Notably, building PG incurs no token cost, as it does not rely on the LLM for graph construction. We only report the results on four datasets here, and leave the remaining results in our technical report [68]. As shown in Figure 4(a) to (d), we observe the following: (1) Building trees consistently requires the least token cost, while TKG and RKG incur the highest token costs, with RKG slightly exceeding TKG. In some cases, RKG requires up to 40× more tokens than trees. (2) KG falls between these extremes, requiring more tokens than trees but fewer than TKG and RKG. This trend aligns with the results in Table 2, where graphs with more attributes require higher token costs for construction. (3) Recall that the token cost for an LLM call consists of two parts: the prompt token, which accounts for the tokens used in providing the input, and the completion part, which includes the tokens generated by the model as a response. We can see that regardless of the graph type, the prompt part always incurs higher token costs than the completion part; the detailed results are shown in [68].

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

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

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

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

▶ **Exp.3. Evaluation of the generation costs.** In this experiment, we evaluate the time and token costs for each method in specific QA tasks. Specifically, we report the average time and token costs for each query across all datasets in Table 6. We only report the results on three datasets here, and leave the remaining results in [68]. It is not surprising that ZeroShot and VanillaRAG are the most cost-efficient methods in terms of both time and token consumption. In terms of token cost, RAPTOR and HippoRAG are generally more efficient than other graph-based RAG methods, as 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 [68], which explains why in some datasets, VanillaRAG is even faster than ZeroShot.

▶ **Exp.4. Detailed analysis for RAPTOR and LGraphRAG.** Due to space limitations, we highlight only the key insights derived from our analysis of RAPTOR and LGraphRAG. First, RAPTOR reveals that a significantly higher proportion of retrieved high-level information (i.e., content from non-leaf nodes) appears in complex QA tasks compared to simple ones, suggesting that high-level information is crucial for multi-hop reasoning. Second, we find that community reports serve as more effective high-level information than the chunk summaries used in RAPTOR, and the similarity-driven community retrieval strategy proves more robust than the Entity operator employed in LGraphRAG. Finally, for multi-hop reasoning tasks like MultihopQA, entity and relationship information provide valuable auxiliary signals that help the LLM connect relevant facts and guide the reasoning process. The detailed analysis and respective experiment results are shown in [68].

▶**Exp.5. Effect of chunk size.** We evaluate the impact of chunk size on all RAG methods for specific QA by splitting the corpus into chunks of 600, 1200, and 2400 tokens. To this end, we construct three new datasets—PopAll, HotpotAll, and ALCEAll—by re-chunking the full corpora of PopQA, HotpotQA, and ALCE based on token length. This re-chunking is necessary because these datasets are pre-split by expert annotators, which may not accurately reflect the effects of chunk sizes. We only report the results of five methods in Table 7, and present the remaining results in [68]. We can see that: (1) For simple QA datasets (e.g., PopAll and HotpotAll), smaller chunk sizes generally yield better performance. This is because such questions often require information that is directly available in a single chunk or two. Smaller chunks provide more focused and precise context, improving answer accuracy. (2) We note that performance on simple QA tasks is highly sensitive to chunk size, while for complex QA tasks, performance remains relatively stable across different chunk sizes. This is because complex questions typically require reasoning across multiple chunks, making them less dependent on individual chunk granularity.

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

▶ **Exp.7. Effect of LLM backbone.** We evaluate the impact of different LLM backbones—Llama-3-8B [14], Qwen-2.5-32B [86], Llama-3-70B [14], and GPT-4o-mini—on the MultihopQA and ALCEAll datasets. The main results are shown in Table 9, while the remaining parts are presented in [68]. We make the following observations: (1) Stronger models generally yield better performance, especially in the Zeroshot setting, which most directly reflects the inherent capabilities of the underlying LLM. (2) The three variants of LightRAG, LLightRAG, GLightRAG, and HLightRAG as well as LGraphRAG, achieve significant performance improvements when using more powerful LLMs. This can be attributed to their reliance on Rich Knowledge Graphs and Textual Knowledge Graphs, where stronger LLMs contribute to the construction of higher-quality graphs. (3) HippoRAG shows notably superior performance when using GPT-4o-mini compared to other LLM backbones. We attribute this to GPT-4o-mini's ability to extract more accurate entities from the question and to construct higher-quality knowledge graphs, thereby improving the retrieval of relevant chunks and the final answer accuracy. (4) Regardless of the LLM backbone, our proposed method VGraphRAG consistently achieves the best performance, demonstrating the advantages of our proposed unified framework.

7.3 Evaluation for abstract QA

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

▶ **Exp.2. Evaluation of the generation costs.** In this experiment, we present the time and token costs for each method in abstract QA tasks. As shown in Table 10, GGraphRAG is the most expensive method, as expected, while other graph-based methods exhibit comparable costs, although they are more expensive than VanillaRAG. For example, on the MutihopSum dataset, GGraphRAG requires 57 × 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.3. New SOTA algorithm.** While the GGraphRAG shows remarkable performance in abstract QA, its time and token costs are not acceptable in practice, since given a question Q, GGraphRAG needs to use LLM to analyze all communities via Map-Reduce. (See Section 6.3) To alleviate this issue, we propose a cost-efficient variant of GGraphRAG, named CheapRAG. Instead of applying the LLM to analyze all communities, CheapRAG first computes the vector similarity between each community and the query to filter out irrelevant ones. It then applies the LLM only to the most relevant communities, significantly reducing token costs compared to GGraphRAG. Moreover, we observe that many top-performing methods, such as RAPTOR, HLightRAG, and GGraphRAG, all leverage the original chunks. This suggests that original chunks remain useful for certain questions. Hence, CheapRAG also incorporates original chunks into its retrieval process. After retrieving the top-k most relevant communities and chunks, CheapRAG adopts a Map-Reduce strategy: the LLM generates partial answers for each selected community and chunk independently, and then summarizes them into a final

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

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

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

(a) Comprehensiveness

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

(b) Diversity

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

(c) Empowerment

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

(d) Overall

Figure 6: The abstract QA results on Mix dataset.

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

(a) Comprehensiveness

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

(b) Diversity

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

(c) Empowerment

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

(d) Overall

Figure 7: The abstract QA results on MultihopSum dataset.

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

(a) Comprehensiveness

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

(b) Diversity

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

(c) Empowerment

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

(d) Overall

Figure 8: The abstract QA results on Agriculture dataset.

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

(a) Comprehensiveness

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

(b) Diversity

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

(c) Empowerment

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

(d) Overall

Figure 9: The abstract QA results on CS dataset.

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

(a) Comprehensiveness

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

(b) Diversity

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

(c) Empowerment

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

(d) Overall

Figure 10: The abstract QA results on Legal dataset.

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

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

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

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

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

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

(a) chunk size=600	VR	RA	GS	LR	FG
VR	50	27	30	78	67
RA	73	50	54	90	92
GS	70	46	50	86	71
LR	22	10	14	50	29
FG	33	8	19	71	50

(b) chunksize=2400	VR	RA	GS	LR	FG
VR	50	23	37	62	68
RA	77	50	66	83	90
GS	63	34	50	70	72
LR	38	17	30	50	54
FG	32	10	28	46	50

(c) Qwen-2.5-32B	VR	RA	GS	LR	FG
VR	50	19	10	18	68
RA	81	50	24	61	88
GS	90	76	50	80	93
LR	82	39	20	50	91
FG	32	12	7	9	50

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

(e) GPT-4o-mini	VR	RA	GS	LR	FG
VR	50	72	4	13	60
RA	28	50	4	10	37
GS	96	96	50	90	95
LR	87	90	10	50	87
FG	40	63	5	13	50

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

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

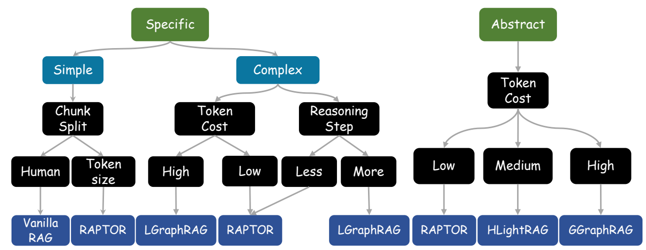
Dataset	Metric	LGraphRAG	RAPTOR	VGraphRAG
MultihopQA	Accuracy	55.360	56.064	**59.664**
	Recall	50.429	44.832	**50.893**
MusiqueQA	Accuracy	12.467	24.133	**26.933**
	Recall	23.996	35.595	**40.026**
ALCE	STRREC	28.448	35.255	**41.023**
	STREM	8.544	11.076	**15.401**
	STRHIT	54.747	65.401	**71.835**

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

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

response. As shown in Figure 11 and Table 10, CheapRAG not only achieves better performance than GGraphRAG but also significantly reduces token costs (in most cases). For example, on the Multihop-Sum dataset, CheapRAG reduces token costs by 100× compared to GGraphRAG, while achieving better answer quality. We leave improving the answer diversity of CheapRAG to future work.

▶ **Exp.4. Effect of chunk size and LLM backbone.** We also study the impact of chunk size and LLM backbone on abstract QA tasks, following the same experimental setup as in Section 7.2. Due to space limitations, we report the results based on the "Overall" metric in Figure 12, with additional details provided in [68]. Our key observations are as follows: (1) The performance of GGraphRAG remains stable across different chunk sizes, likely due to its use of the Map-Reduce strategy for final answer synthesis, which mitigates the influence of chunk granularity. (2) In contrast, methods like FastGraphRAG and VanillaRAG show greater variance across chunk sizes, as their performance relies heavily on the granularity of individual chunks—smaller chunks tend to provide more precise information, directly impacting retrieval and generation quality. (3) Regardless of chunk size, RAPTOR and GGraphRAG consistently achieve the best performance, reaffirming our earlier conclusion that high-level structural information is essential for abstract QA tasks. (4) All methods still lag behind GGraphRAG, further highlighting that community-level information is particularly beneficial for abstract QA tasks. In addition, we evaluate our newly proposed method CheapRAG against the baselines under varying chunk sizes and LLM backbones. As shown in [68], CheapRAG consistently achieves the best performance across all settings.



(a) Specific QA (b) Abstract QA

Figure 13: The taxonomy tree of RAG methods.

8 LESSONS AND OPPORTUNITIES

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

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

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

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

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

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

chunks, and the vector search-based retrieval strategy is better than the rule-based (e.g., `Entity` operator) one. This lesson is supported by the results in Tables 5 and 8.

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

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

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

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

▶ **O4.** How to use LLMs and graph-based RAG methods to facilitate query optimization, such as generating efficient query plans and execution strategies, for database systems. More details are in our technical report [68].

9 RELATED WORKS

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

• **RAG methods.** RAG has been proven to be very effective in migrating the "hallucination" of LLMs [4, 6–8, 35, 72]. Recently, most RAG approaches [15, 24, 25, 43, 64, 69, 82, 83] have adopted graph as the external knowledge to organize the information and relationships within documents, achieving improved overall retrieval performance, which is extensively reviewed in this paper. Nevertheless, there is a lack of a comprehensive work comparison between all graph-based RAG methods in terms of accuracy and efficiency. We note that there exists an empirical study [26] that compares Microsoft's methods (`LGraphRAG` and `GGraphRAG`) with the standard `VanillaRAG`, and a few survey papers on graph-based RAG systems [37, 91]. However, our work differs significantly from these in both scope and depth. First, our work focus on systematically comparing the different graph-based RAG methods, and conduct a stage-wise comparison across a unified framework. This allows us to identify core design principles and enables the construction of a new state-of-the-art method through component recombination. Second, unlike survey papers that offer high-level overviews, our work provides deep empirical analysis and practical insights grounded in extensive experiments.

• **RAG applications.** Due to the wealth of developer experience captured in a vast array of database forum discussions, recent studies [6, 16, 23, 40, 49, 71, 75, 97, 98] have begun leveraging RAGs to enhance database performance. For instance, GPTuner [40] proposes to enhance database knob tuning using RAG by leveraging domain knowledge to identify important knobs and coarsely initialize their values for subsequent refinement. Besides, D-Bot [97] proposes an LLM-based database diagnosis system, which can retrieve relevant knowledge chunks and tools, and use them to identify typical root causes accurately. In addition, RAG-based SQL rewriting systems [49, 73, 75] have recently attracted significant attention. The RAG-based data analysis systems have also been studied [3, 10, 50–54, 63]. For applications in other areas, we refer readers to recent RAG surveys [31, 92].

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.

11 ACKNOWLEDGMENTS

This work was supported in part by the 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund under Grant 2025A0505000045, Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010 001), Shenzhen Research Institute of Big Data under Grant SIF20240002, and Huawei Collaboration Fund under Grant TC20240920019.

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] Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. 2025. Cache-Craft: Managing Chunk-Caches for Efficient Retrieval-Augmented Generation. *arXiv preprint arXiv:2502.15734* (2025).

[3] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A Shah, et al. 2024. The Design of an LLM-powered Unstructured Analytics System. *arXiv preprint arXiv:2409.00847* (2024).

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Chat2Graph. 2025. About Chat2Graph: Graph Native Agentic System. https://chat2graph.vercel.app/chat2graph/en-us/introduction.

[6] 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).

[7] 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.

[8] 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.

[9] Shengyuan Chen, Chuang Zhou, Zheng Yuan, Qinggang Zhang, Zeyang Cui, Hao Chen, Yilin Xiao, Jiannong Cao, and Xiao Huang. 2025. You Don't Need Pre-built Graphs for RAG: Retrieval Augmented Generation with Adaptive Reasoning Structures. *arXiv preprint arXiv:2508.06105* (2025).

[10] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. SEED: Domain-Specific Data Curation With Large Language Models. *arXiv e-prints* (2023), arXiv–2310.

[11] Databricks. 2024. Building, Improving, and Deploying Knowledge Graph RAG Systems on Databricks. https://neo4j.com/docs/neo4j-graphrag-python/current/.

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

[13] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[14] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[15] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[16] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot nl2sql. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2750–2763.

[17] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-effective in-context learning for entity resolution: A design space exploration. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3696–3709.

[18] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6491–6501.

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

[20] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. *arXiv preprint arXiv:2305.14627* (2023).

[21] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[22] Aashish Ghimire, James Prather, and John Edwards. 2024. Generative AI in Education: A Study of Educators' Awareness, Sentiments, and Influencing Factors. *arXiv preprint arXiv:2403.15586* (2024).

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

[24] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. LightRAG: Simple and Fast Retrieval-Augmented Generation. *arXiv e-prints* (2024), arXiv–2410.

[25] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831* (2024).

[26] Haoyu Han, Harry Shomer, Yu Wang, Yongjia Lei, Kai Guo, Zhigang Hua, Bo Long, Hui Liu, and Jiliang Tang. 2025. Rag vs. graphrag: A systematic evaluation and key insights. *arXiv preprint arXiv:2502.11371* (2025).

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

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

[29] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *arXiv preprint arXiv:2402.07630* (2024).

[30] Yucheng Hu and Yuxing Lu. 2024. Rag and rau: A survey on retrieval-augmented language model in natural language processing. *arXiv preprint arXiv:2404.19543* (2024).

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

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

[33] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. KET-RAG: A Cost-Efficient Multi-Granular Indexing Framework for Graph-RAG. *arXiv preprint arXiv:2502.09304* (2025).

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

[35] 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).

[36] Wenqi Jiang, Marco Zeller, Roger Waleffe, Torsten Hoefler, and Gustavo Alonso. 2024. Chameleon: A Heterogeneous and Disaggregated Accelerator System for Retrieval-Augmented Language Models. *Proceedings of the VLDB Endowment* 18, 1 (2024), 42–52.

[37] Vikas Kamra, Lakshya Gupta, Dhruv Arora, and Ashwin Kumar Yadav. 2024. Enhancing Document Retrieval Using AI and Graph-Based RAG Techniques. In *2024 5th International Conference on Communication, Computing & Industry 6.0 (C2I6)*. IEEE, 1–7.

[38] Omar Khattab 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.

[39] LangGraph. 2024. LangGraph. https://www.langchain.com/langgraph.

[40] 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.

[41] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[42] 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).

[43] 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).

[44] 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).

[45] Yuhan Li, Zhixun Li, Peisong Wang, Jia Li, Xiangguo Sun, Hong Cheng, and Jeffrey Xu Yu. 2023. A survey of graph meets large language model: Progress and future directions. *arXiv preprint arXiv:2311.12399* (2023).

[46] Yinheng Li, Shaofei Wang, Han Ding, and Hang Chen. 2023. Large language models in finance: A survey. In *Proceedings of the fourth ACM international conference on AI in finance*. 374–382.

[47] Zhixun Li, Liang Wang, Xin Sun, Yifan Luo, Yanqiao Zhu, Dingshuo Chen, Yingtao Luo, Xiangxin Zhou, Qiang Liu, Shu Wu, et al. 2023. GSLB: the graph structure learning benchmark. *Advances in Neural Information Processing Systems* 36 (2023), 30306–30318.

[48] 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).

[49] 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.

[50] 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).
[51] 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).
[52] 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).
[53] 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).
[54] 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).
[55] 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).
[56] 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).
[57] Multi-Linguality Multi-Functionality Multi-Granularity. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. (2024).
[58] 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.
[59] neo4j rag. 2024. Neo4j GraphRAG. https://neo4j.com/docs/neo4j-graphrag-python/current/.
[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] PostgreSQL. 2024. Azure Database for PostgreSQL Blog. https://techcommunity.microsoft.com/blog/adforpostgresql/introducing-the-graphrag-solution-for-azure-database-for-postgresql/4299871.
[66] 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).
[67] 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.
[68] 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.
[69] 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).
[70] 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).
[71] 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).
[72] 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.
[73] Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2025. QUITE: A Query Rewrite System Beyond Rules with LLM Agents. *arXiv preprint arXiv:2506.07675* (2025).
[74] 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*.
[75] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).
[76] Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391* (2024).
[77] 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.
[78] 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.
[79] 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).
[80] 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).
[81] 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).
[82] 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.
[83] 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).
[84] 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).
[85] Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2025. When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation. *arXiv preprint arXiv:2506.05690* (2025).
[86] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
[87] 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).
[88] 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).
[89] 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).
[90] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. 2025. EraRAG: Efficient and Incremental Retrieval Augmented Generation for Growing Corpora. *arXiv preprint arXiv:2506.20963* (2025).
[91] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *arXiv preprint arXiv:2501.13958* (2025).
[92] 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).
[93] 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).
[94] 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.
[95] 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.
[96] 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).
[97] 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.
[98] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. Db-gpt: Large language model meets database. *Data Science and Engineering* 9, 1 (2024), 102–111.
[99] Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2025. LinearRAG: Linear Graph Retrieval Augmented Generation on Large-scale Corpora. *arXiv preprint arXiv:2510.10114* (2025).