ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation

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

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

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

Xilin Liu
Huawei Cloud Computing Technologies CO., LTD.
liuxilin3@huawei.com

Yuchi Ma
Huawei Cloud Computing Technologies CO., LTD.
mayuchi1@huawei.com

Abstract

Retrieval-Augmented Generation (RAG) has proven effective in integrating external knowledge into large language models (LLMs) for solving question-answer (QA) tasks. The state-of-the-art RAG approaches often use the graph data as the external data since they capture the rich semantic information and link relationships between entities. However, existing graph-based RAG approaches cannot accurately identify the relevant information from the graph and also consume large numbers of tokens in the online retrieval process. To address these issues, we introduce a novel graph-based RAG approach, called Attributed Community-based Hierarchical RAG (ArchRAG), by augmenting the question using attributed communities, and also introducing a novel LLM-based hierarchical clustering method. To retrieve the most relevant information from the graph for the question, we build a novel hierarchical index structure for the attributed communities and develop an effective online retrieval method. Experimental results demonstrate that ArchRAG outperforms existing methods in both accuracy and token cost. Moreover, ArchRAG has been successfully applied to domain knowledge QA in Huawei Cloud Computing.

PVLDB Reference Format:
Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation . PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/sam234990/ArchRAG.

1 Introduction

Large Language Models (LLMs) like GPT-4 [1], Qwen2.5 [75], and LLaMA3.1 [11] have received tremendous attention from both industry and academia [17, 26, 36, 39, 48, 66, 69, 83]. Despite their remarkable success in question-answering (QA) tasks, they may still generate wrong answers due to a lack of domain-specific and real-time updated knowledge outside their pre-training corpus [50]. To enhance the trustworthiness and interpretability of LLMs, Retrieval-Augmented Generation (RAG) methods [15, 16, 24, 26, 72, 78, 81] have emerged as a core approach, which often retrieve relevant information from documents, relational data, and graph data to facilitate the QA tasks. RAG has been widely applied in various fields, such as healthcare [39, 66, 83], finance [36, 48], and education [17, 69], and supports many data management tasks, including NL2SQL [13, 33], data cleaning [14, 35, 46, 51], knob tuning [18, 32], DBMS diagnosis [56, 84, 85], and SQL rewrite [37, 59].

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

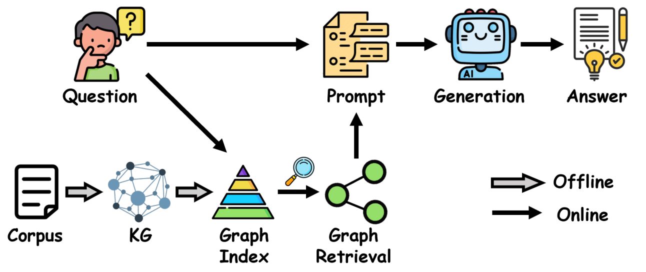


Figure 1: The general workflow of graph-based RAG, which retrieves relevant information (e.g., nodes, subgraphs, or textual information) to facilitate the LLM generation.

The state-of-the-art RAG approaches often use the graph data as the external data since they capture the rich semantic information and link relationships between entities. Given a question Q, the key idea of graph-based RAG is to retrieve relevant information (e.g., nodes, subgraphs, or textual information) from the graph, incorporate them with Q as the prompt, and feed them into the LLM, as illustrated in Figure 1. As shown in the literature, a few recent graph-based RAG methods have emerged for both *abstract question* [12, 20] and *specific question* [21, 34, 54, 70, 71], where the former targets high-level themes (e.g., "What are the potential impacts of LLMs on education?"), while the latter focuses on entity-centric details (e.g., "Who has won the Turing Award in 2024?").

Prior wroks. In the past year, a surge of graph-based RAG methods [21, 34, 42, 58, 70, 71] has emerged, each proposing different retrieval strategies to extract detailed information for response generation. Among them, GraphRAG [12], proposed by Microsoft, is the most prominent and the first to leverage community summarization for abstract QA. It builds a knowledge graph (KG) from the external corpus, detects communities using Leiden [62], and generates a summary for each community using LLMs. For abstract questions that require high-level information, it adopts a Global Search approach, traversing all communities and using LLMs to

Table 1: Comparison of representative RAG methods and our ArchRAG. Note: For the abstract question comparison, unsupported methods are marked with "/", LightRAG refers to the LightRAG-Hybrid, and GraphRAG refers to the GraphRAG-Global Search.

RAG method	Method Type	Retrieval method	Question	Attributed community	Avg. tokens per abstract question	Summarization rating
Zero-shot / CoT [31]	Inference-only	None	Specific	✗	/	/
Vanilla RAG	Retrieval-only	Vector search	Specific	✗	/	/
RAPTOR [54]	Graph-based RAG	Vector search	Specific	✗	/	/
HippoRAG [21]	Graph-based RAG	Entity recognition + PPR	Specific	✗	/	/
LightRAG [20]	Graph-based RAG	Extract keywords + Vector search	Specific / Abstract	✗	7,243	★★★
GraphRAG [12]	Graph-based RAG	Traverse	Specific / Abstract	✗	1,066,655	★★★★
ArchRAG (ours)	Graph-based RAG	Hierarchical search	Specific & Abstract	✓	6,746	★★★★★

retrieve the most relevant summaries. In contrast, for specific questions, it employs a Local Search method to retrieve entities, relevant text chunks, and low-level communities, providing the multi-hop detailed information for accurate answers. Table 1 presents a comparison of representative methods.

Although some methods claim that GraphRAG [12] underperforms and is difficult to apply in practice [42, 79], our re-examination shows that it is primarily constrained by the following limitations (**L**): **L1.** *Low community quality*: GraphRAG uses the Leiden algorithm to detect communities, but this approach relies solely on graph structure and ignores the rich semantics of nodes and edges. As a result, the detected communities often consist of different themes, which leads to the poor quality of community summaries and further decreases its performance. **L2.** *Limited compatibility*: While GraphRAG employs Global and Local Search strategies, each retrieves graph elements at only one granularity, making it inadequate for simultaneously addressing both abstract and specific questions and limiting its applicability in real-world open-ended scenarios. **L3.** *High generation cost*: Although GraphRAG performs well on abstract questions, analyzing all communities with LLMs is both time- and token-consuming. For example, GraphRAG detects 2,984 communities in the Multihop-RAG [61] dataset, and answering just 100 questions incurs a cost of approximately $650 and 106 million tokens[1], which is an impractical overhead.

To tackle the above limitations of GraphRAG, in this paper, we propose a novel graph-based RAG approach, called **A**tt**r**ibuted **C**ommunity-based **H**ierarchical **RAG** (ArchRAG). ArchRAG leverages attributed communities (ACs) and introduces an efficient hierarchical retrieval strategy to adaptively support both abstract and specific questions. To mitigate **L1**, we detect high-quality ACs by exploiting both links and the attributes of nodes, ensuring that each AC comprises nodes that are not only densely connected but also share similar semantic themes [86]. We further propose a novel LLM-based iterative framework for hierarchical AC detection, which can incorporate any existing community detection methods [19, 62, 63, 74, 86]. In each iteration, we detect ACs based on both attribute similarity and connectivity, summarize each AC using an LLM, and construct a higher-level graph by treating each AC as a node, connecting pairs with similar summaries. By iterating the above steps multiple times, we obtain a set of ACs that can be organized into a hierarchical tree structure.

To effectively address **L2**, we organize all ACs and entities into a hierarchical index and retrieve relevant elements from all levels to support both abstract and specific questions. Entities offer fine-grained details, while LLM-generated AC summaries capture relational structures and provide high-level condensed overviews [2, 54], making them suitable for both multi-hop reasoning and abstract insight extraction. To support efficient retrieval across levels, we propose C-HNSW (Community-based HNSW), a novel hierarchical index structure inspired by the HNSW algorithm [43] for approximate nearest neighbor (ANN) search.

To mitigate the high generation cost caused by traversing all communities (**L3**), we propose a hierarchical search combined with an adaptive filtering strategy, which enables efficient selection of the most relevant ACs and entities while maintaining performance. Specifically, we design an efficient hierarchical search algorithm over the proposed C-HNSW index, which supports top-k nearest neighbor search across multiple levels, thereby facilitating access to multi-level relevant information. Furthermore, the adaptive filtering mechanism identifies the most informative results at each level, making the retrieved information complementary.

We have extensively evaluated ArchRAG by conducting various experiments on real-world datasets, and the results show that ArchRAG outperforms existing approaches in answering both abstract and specific questions. Particularly, ArchRAG achieves a 10% high accuracy compared to state-of-the-art graph-based RAG methods on specific QA tasks and shows significant improvement on abstract QA tasks as well. Moreover, ArchRAG is very token-efficient, saving up to 250 times the token usage compared to GraphRAG [12]. In addition, ArchRAG has been successfully applied to domain knowledge QA in Huawei Cloud Computing, demonstrating strong practical performance.

In summary, our main contributions are as follows:

- We present a novel graph-based RAG approach by using ACs that are organized hierarchically and detected by an LLM-based hierarchical clustering method.
- To index ACs, we propose a novel hierarchical index structure called C-HNSW and also develop an efficient online retrieval method.
- Extensive experiments show that ArchRAG is both highly effective and efficient, and achieves state-of-the-art performance in answering both abstract and specific QA tasks.

2 Preliminaries

Existing graph-based RAG methods often follow a general framework that leverages external structured knowledge graphs to improve contextual understanding of LLMs and generate more informed responses [50]. Typically, it involves two phases:

(1) **Offline indexing:** Building a KG $G(V, E)$ from a given corpus D where each vertex represents an entity and each

[1]The cost of GPT-4o is $10/M tokens for output and $2.50/M tokens for input (for details, please refer to openai.com/api/pricing).

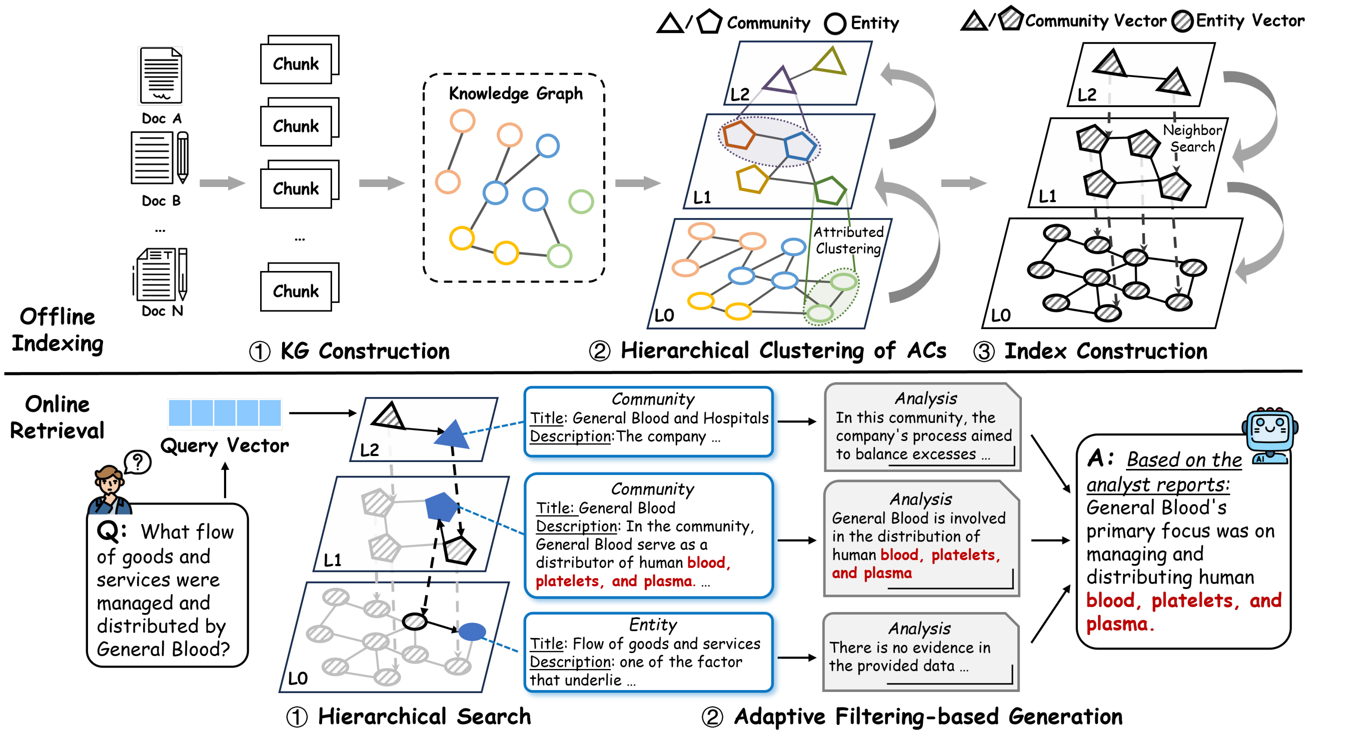

Offline Indexing
Doc A
Doc B
...
Doc N
Chunk
Chunk
Chunk
...
Chunk
Knowledge Graph
△/⬠ Community ○ Entity
L2
L1
L0
Attributed Clustering
△/⬠ Community Vector ⊘ Entity Vector
Neighbor Search
① KG Construction
② Hierarchical Clustering of ACs
③ Index Construction
Online Retrieval
Query Vector
Q: What flow of goods and services were managed and distributed by General Blood?
Community
Title: General Blood and Hospitals
Description: The company ...
Analysis
In this community, the company's process aimed to balance excesses ...
Community
Title: General Blood
Description: In the community, General Blood serve as a distributor of human blood, platelets, and plasma. ...
Analysis
General Blood is involved in the distribution of human blood, platelets, and plasma
Entity
Title: Flow of goods and services
Description: one of the factor that underlie ...
Analysis
There is no evidence in the provided data ...
A: Based on the analyst reports: General Blood's primary focus was on managing and distributing human blood, platelets, and plasma.
① Hierarchical Search
② Adaptive Filtering-based Generation

Figure 2: ArchRAG consists of two phases: offline indexing and online retrieval. For the online retrieval phase, we show an example of using ArchRAG to answer a question in the HotpotQA dataset.

edge denotes the relationship between two entities, and constructing an index based on the KG.

(2) **Online retrieval:** Retrieving the relevant information (e.g., nodes, subgraphs, or textual information) from KG using the index and providing the retrieved information to the LLM to improve the accuracy of the response.

3 Our Approach ArchRAG

As illustrated in Figure 2, our proposed ArchRAG consists of two phases. In the *offline indexing* phase, ArchRAG first constructs a KG from the corpus, then detects ACs by a novel LLM-based hierarchical clustering method, and finally builds the Community-based HNSW (C-HNSW) index. During the *online retrieval* phase, ArchRAG first converts the question into a query vector, then retrieves relevant information from the C-HNSW index, and finally generates answers through an adaptive filtering-based generation process that identifies the most relevant information.

Unlike GraphRAG, which purely uses links to generate communities, we propose to detect communities that can be organized into a hierarchical tree structure by exploiting both links and attributes in the knowledge graph, also called ACs. An AC is a group of entities with a summary, where the entities are not only densely connected but also share similar themes, providing detailed information for answering specific questions, and the summary can offer a condensed view for answering abstract questions since it enables more focused engagement with the entities' attributes. Moreover, in the hierarchical tree structure, the lower-level communities and entities contain detailed knowledge from the KG, while higher-level communities provide global summaries, which naturally enable ArchRAG to address questions with different granularity in abstraction.

Moreover, to answer questions, we need to efficiently identify highly relevant ACs at different levels of the hierarchical structure. Considering that our hierarchical ACs and the HNSW index [43] are similar in structure, we propose to map entities and ACs into high-dimensional nodes and then build a unified C-HNSW index for them, which allows us to efficiently identify the highly relevant ACs.

3.1 Offline Indexing

3.1.1 KG construction. Our first step is to build a KG by prompting the LLM to extract entities and relations from each chunk of the text corpus D. Specifically, all text contexts are segmented into chunks based on specified chunk length, enabling the LLM to extract entities and relations from each chunk using in-context learning [4], thus forming subgraphs. These subgraphs are then merged, with entities and relations that appear repeatedly across multiple subgraphs being consolidated by the LLM to generate a complete description. Finally, we get a KG, denoted by $G(V, E)$, where V and E are sets of vertices and edges, respectively, and each vertex and edge is associated with some textual attributes.

3.1.2 LLM-based hierarchical clustering. We propose an iterative LLM-based hierarchical clustering framework that supports arbitrary graph augmentation (e.g., KNN connections and CODICIL [53]) and graph clustering algorithms (e.g., weighted Leiden [62], weighted spectral clustering, and SCAN [74]). Specifically, we propose to augment the KG by linking entities if their attribute similarities are larger than a threshold, and then associate each pair of linked entities with a weight denoting their attribute similarity value. Next, we generate the ACs using any given graph clustering algorithm. In this way, both node attributes and structural links are jointly considered during community detection.

Algorithm 1 shows the above iterative clustering process. Given a graph augmentation method and clustering algorithm, we perform the following steps in each iteration: (1) augmenting the graph (line 3); (2) computing the edge weights (lines 4-5); (3) clustering the augmented graph (line 6); (4) generating a summary for each community using LLM (line 7); and (5) building a new attributed graph where each node denotes an AC and two nodes are linked if their community members are connected (line 9). We repeat the iterations until the stopping condition (such as insufficient nodes or reaching the specified level limit) is met (line 10). Since each iteration corresponds to one layer, all the ACs HC can be organized into a multi-layer hierarchical tree structure, denoted by Δ, where each community in one layer includes multiple communities in the next layer. We also provide a running example for Algorithm 1 in the appendix of our technical report [68].

Algorithm 1: LLM-based hierarchical clustering

```
input  :KG $G(V,E)$, augment function Aug, graph clustering
        algorithm GCluster, and termination condition $T$
1  $T \leftarrow$ False, $HC \leftarrow \emptyset$;
2  repeat
3      $G'(V,E') \leftarrow$ Aug($G(V,E)$);
4      for each $e' = (u,v) \in E'$ do
5          update the weight of $e'$ as $1 - \cos(z_u, z_v)$;
6      $C \leftarrow$ GCluster($G'(V,E')$);
7      for each $c \in C$ do generate the summary of $c$ by LLM ;
8      $HC \leftarrow HC \cup C$;
9      $G(V,E) \leftarrow$ build a new graph using $C$ and $E'$;
10     update $T$ according to $G(V,E)$;
11 until $T$=True;
12 return $HC$;
```

3.1.3 *C-HNSW index.*

Given a query, to efficiently identify the most relevant information from each layer of the hierarchical tree Δ, a naive method is to build a vector database for the ACs in each layer, which is costly in both time and space. To tackle this issue, we propose to build a single hierarchical index for all the communities. Recall that the ACs in Δ form a tree structure, and the number of nodes decreases as the layer level increases. Since this tree structure is similar to the HNSW (Hierarchical Navigable Small World) index which is the most well-known index for efficient ANN search [43] , we propose to map entities and ACs of Δ into high-dimensional nodes, and then build a unified Community-based HNSW (C-HNSW) index for them.

• **The structure of C-HNSW.** Conceptually, the C-HNSW index is a list of simple graphs with links between them, denoted by $\mathcal{H} = (\mathcal{G}, L_{inter})$ with $\mathcal{G} = \{G_0 = (V_0, E_0), G_1 = (V_1, E_1), \cdots, G_L = (V_L, E_L)\}$, where G_i is a simple graph and each node of the simple graph corresponds to an AC or entity. The number L of layers (simple graphs) of $\mathcal{H}$ is the same as the that of Δ. Specifically, for each AC or entity in the i-the layer of Δ, we map it to a high-dimensional node in the i-th layer of $\mathcal{H}$ by using a language model (e.g., nomic-embed-text [49]).

After obtaining the high-dimensional nodes, we establish two types of links between them, i.e., *intra-layer* and *inter-layer* links:

- **Intra-layer links:** These links exist between nodes in the same layers. Specifically, for each node in each layer, we link it to at least M nearest neighbors within the same layer, where M is a predefined value, and the nearest neighbors are determined according to a given distance metric d. Thus, all the intra-layer links are edges in all the simple graphs:

$$L_{intra} = \bigcup_{i=0}^{L} E_i \tag{1}$$

- **Inter-layer links:** These links cross two adjacent layers. Specifically, we link each node in each layer to its nearest neighbor in the next layer. As a result, all the inter-layer links can be represented as follows:

$$L_{inter} = \bigcup_{i=1}^{L} \{(v, \psi(v)) | v \in V_i, \psi(v) \in V_{i-1}\}, \tag{2}$$

  where $\psi(\cdot) : V_i \rightarrow V_{i-1}$ is the injective function that identifies the nearest neighbor of each node in the lower layer.

For example, in Figure 2, the C-HNSW index has three layers (simple graphs), incorporating all the ACs. Within each layer, each node is connected to its two nearest neighbors via intra-layer links, denoted by undirected edges. The inter-layer links are represented by arrows, e.g., the green community at layer L_1 is connected to the green entity (its nearest neighbor at layer L_0).

Intuitively, since the two types of links above are established based on nearest neighbors, C-HNSW allows us to quickly search the relevant information for a query by traversing along with these links. Note that C-HNSW is different from HNSW since it has intra-layer links, and each node exists in only one layer.

• **The construction of C-HNSW.** A naive approach to constructing the C-HNSW index is to build nodes first and then establish the two types of links by finding the nearest neighbors of each node. However, the process of finding the nearest neighbor is costly. To accelerate the construction, we propose a top-down approach by borrowing the idea of HNSW construction. Specifically, by leveraging the procedure SearchLayer of Hierarchical search, which will be introduced in Section 3.2.1, nodes are progressively inserted into the index starting from the top layer, connecting intra-layer links within the same layer and updating the inter-layer links. For lack of space, we give the details of the construction algorithm in the appendix of our technical report [68].

3.2 Online retrieval

In the online retrieval phase, after obtaining the query vector for a given question, ArchRAG generates the final answer by first conducting a hierarchical search on the C-HNSW index and then analyzing and filtering the retrieved information.

3.2.1 *Hierarchical search.*

We propose an efficient and fast retrieval algorithm, hierarchical search, to retrieve high-level dimensional nodes from each layer of the C-HNSW index. Intuitively, retrieving nodes from a given layer in C-HNSW requires starting from the top layer and searching downward through two types of links (i.e., *intra-layer* and *inter-layer* links) to locate the nearest neighbors at the given layer. In contrast, our hierarchical search algorithm accelerates this process by reusing intermediate results, the nearest neighbors found in higher layers, as the starting node for lower layers. This approach avoids redundant computations that would otherwise arise from repeatedly searching from the top layer, thereby enabling efficient multi-layer retrieval, as illustrated in Algorithm 2.

Algorithm 2: Hierarchical search

```
input  : $\mathcal{H} = (\mathcal{G}, L_{inter})$, $q$, $k$.
1  $s \leftarrow$ a random node in the highest layer $L$;
2  $R \leftarrow \emptyset$;
3  for $i \leftarrow L, \cdots, 0$ do
4      $R_i \leftarrow$ SearchLayer ($G_l = (V_l, E_l), q, s, 1, l$);
5      $R \leftarrow R \cup R_i$;
6      $c \leftarrow$ get the nearest node from $R_i$;
7      $s \leftarrow$ find the node in layer $i-1$ via the inter-layer links of $c$;
8  return $R$;
9  Procedure SearchLayer($G_i = (V_i, E_i), q, s, k, i$):
10     $V \leftarrow \{s\}, K \leftarrow \{s\}, Q \leftarrow$ initialize a queue containing $s$;
11     while $|K| > 0$ do
12         $c \leftarrow$ nearest node in $Q$;
13         $f \leftarrow$ furthest node in $K$;
14         if $d(c, q) > d(f, q)$ then break ;
15         for each neighbor $x \in N(c)$ in $G_i$ do
16             if $x \in V$ then continue;
17             $V \leftarrow V \cup \{x\}$;
18             $f \leftarrow$ furthest node in $K$;
19             if $d(x, q) < d(f, q)$ or $|K| < k$ then
20                 $Q \leftarrow Q \cup \{x\}, K \leftarrow K \cup \{x\}$;
21                 if $|K| > k$ then remove $f$ from $K$;
22     return $K$;
```

Given the C-HNSW $\mathcal{H}$, query point q, and the number k of nearest neighbors to retrieve at each layer, the hierarchical search algorithm can be implemented by the following iterative process:

(1) Start from a random node at the highest layer L, which serves as the starting node for layer L (line 1).
(2) For each layer i from the top layer L down to layer 0, the algorithm begins at the starting node and performs a greedy traversal (i.e., the `SearchLayer` procedure) to find the set R_i of the k nearest neighbors of q. The set R_i is then merged into the final result set R (lines 4–5).
(3) The closest neighbor c of q is then obtained from R_i, and the algorithm proceeds to the next layer by traversing the inter-layer link of c, using it as the starting node for the subsequent search (lines 6–7).

Specifically, the greedy traversal strategy compares the distance between the query point and the visited nodes during the search process. It achieves this by maintaining a candidate expansion queue Q and a dynamic nearest neighbor set K, which contains k elements:

- Expansion Queue Q: For each neighbor x of a visited node, if $d(x, q) < d(f, q)$, where f is the furthest node from R to q, then x is added to the expansion queue.
- Dynamic Nearest Neighbor Set K: Nodes added to C are used to update K, ensuring that it maintains no more than k elements, where k is the specified number of query results.

In the greedy traversal, if a node x expanded from Q satisfies $d(n, q) > d(n, f)$, where f is the furthest node from K to q, then the traversal stops.

After completing the hierarchical search and obtaining the ACs and entities from each layer, we further extract their associated textual information. In particular, at the bottom layer, we also extract the relationships between the retrieved entities, resulting in the textual subgraph representation denoted as R_0. We denote all the retrieved textual information from each layer as R_i, where $i \in 0, 1, \ldots, L$, which will be used in the subsequent adaptive filtering-based generation process.

3.2.2 Adaptive filtering-based generation. While some optimized LLMs support longer text inputs, they may still encounter issues such as the "lost in the middle" dilemma [40]. Thus, direct utilization of retrieved information comprising multiple text segments for LLM-based answer generation risks compromising output accuracy.

To mitigate this limitation, we propose an adaptive filtering-based method that harnesses the LLM's inherent reasoning capabilities. We first prompt the LLM to extract and generate an analysis report from the retrieved information, identifying the parts that are most relevant to answering the query and assigning relevance scores to these reports. Then, all analysis reports are integrated and sorted, ensuring that the most relevant content is used to summarize the final response to the query, with any content exceeding the text limit being truncated. This process can be represented as:

$$A_i = LLM(P_{filter} || R_i) \tag{3}$$

$$Output = LLM(P_{merge} || Sort(\{A_0, A_1, \cdots, A_n\})) \tag{4}$$

where P_{filter} and P_{merge} represent the prompts for extracting relevant information and summarizing, respectively, A_i, $i \in 0 \cdots n$ denotes the filtered analysis report. The sort function orders the content based on the relevance scores from the analysis report.

3.2.3 Complexity analysis. We now analyze the complexity of our ArchRAG during the online retrieval phase. In light of the high token consumption associated with LLMs, we examine both the time and token cost complexity to ensure scalability and efficiency. We also provide the complexity analysis of the index phase and the proof of each lemma in the appendix of our technical report [68].

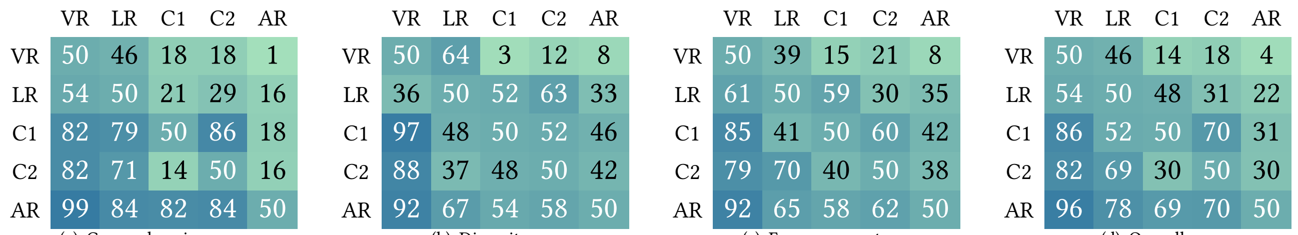
LEMMA 1. *Given a C-HNSW with L layers constructed from a large text corpus or a large set of text documents, the time complexity of a sequentially executed single online retrieval query in ArchRAG is $O(e + LkI + Lk\log(n))$, where I is the generation time of the LLM for a single inference, e is the time cost of computing the query embedding, k is the number of nodes retrieved at each layer, and n is the number of nodes at the lowest layer in C-HNSW.*

LEMMA 2. *Given a C-HNSW with L layers constructed from a large text corpus or a large set of text documents, the number of tokens used for a single online retrieval query in ArchRAG is $O(kL(c + P))$, where k is the number of neighbors retrieved at each layer, c is the average length of the retrieved content, and P is the length of the prompt.*

4 Experiments

Our experiments aim to answer following research questions (RQs):

- *RQ1*: How does ArchRAG perform compared with existing baselines on specific and abstract QA tasks?
- *RQ2*: How efficient is the ArchRAG approach?
- *RQ3*: What is the quality of our LLM-based ACs, and how does it impact the performance of RAG?
- *RQ4*: How robust is ArchRAG?



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

Figure 3: The abstract QA task results are presented as head-to-head win rate percentages, comparing the performance of the row method over the column method. VR, LR, C1, C2, and AR denote Vanilla RAG, LightRAG-Hybrid, GraphRAG-Global with high-level communities, GraphRAG-Global with intermediate-level communities, and ArchRAG, respectively.

4.1 Setup

Table 2: Datasets used in our experiments. BLEU, MET, and ROU denote BLEU-1, METEOT, and ROUGE-L F1.

Dataset	Multihop-RAG	HotpotQA	NarrativeQA
Passages	609	9,221	1,572
Tokens	1,426,396	1,284,956	121,152,448
Nodes	23,353	37,436	650,571
Edges	30,716	30,758	679,426
Questions	2,556	1,000	43,304
Metrics	Accuracy, Recall	Accuracy, Recall	BLEU, MET, ROU

Datasets. We consider both specific and abstract QA tasks. For the specific QA task, we use the following datasets: Multihop-RAG [61], HotpotQA [76], and NarrativeQA [30], all of which are extensively utilized within the QA and Graph-based RAG research communities [3, 21, 25, 54, 73, 77]. Multihop-RAG is designed to evaluate retrieval and reasoning across news articles in various categories. HotpotQA is a QA dataset featuring natural, multi-hop questions. NarrativeQA is a dataset that comprises QA pairs based on the full texts of books and movie transcripts. Notably, NarrativeQA is substantially larger than the datasets used in GraphRAG [12] by approximately two orders of magnitude. Besides, we follow the GraphRAG [12] method for the abstract QA task. We reuse the Multihop-RAG corpus and prompt LLM to generate questions that convey a high-level understanding of dataset contents. The statistics of these datasets are reported in Table 2.

Baselines. Our experiments consider three model configurations:

- **Inference-only:** Using an LLM for direct question answering without any retrieval data, i.e., Zero-Shot and CoT [31].
- **Retrieval-only:** Retrieval models extract relevant chunks from all documents and use them as prompts for large models. We select strong and widely used retrieval models: BM25 [52] and vanilla RAG.
- **Graph-based RAG:** These methods leverage graph data during retrieval. We select RAPTOR [54], HippoRAG [21], GraphRAG [12], and LightRAG [20]. Particularly, GraphRAG has two versions, i.e., GraphRAG-Global and GraphRAG-Local, which use global and local search methods respectively. Similarly, LightRAG integrates local search, global search, and hybrid search, denoted by LightRAG-Low, LightRAG-High, and LightRAG-Hybrid, respectively.

In GraphRAG-Global, all communities below the selected level are first retrieved, and then the LLM is used to filter out irrelevant communities. This process can be viewed as utilizing the LLM as a *retriever* to find relevant communities within the corpus. According to the selected level of communities [12], GraphRAG-Global can be further categorized into C1 and C2, representing high-level and intermediate-level communities, respectively, with C2 as the default.

Metrics. For the specific QA tasks, we use Accuracy and Recall to evaluate performance on the first two datasets based on whether gold answers are included in the generations instead of strictly requiring exact matching, following [3, 44, 55]. We also use the official metrics of BLEU, METEOR, and ROUGE-l F1 in the NarrativeQA dataset. For the abstract QA task, we follow prior work [12] and adopt a head-to-head comparison approach using an LLM evaluator (GPT-4o). 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 [65, 82]. Overall, we utilize four evaluation dimensions: Comprehensiveness, Diversity, Empowerment, and Overall.

Implementation details. We mainly use Llama 3.1-8B [11] as the default LLM model and use nomic-embed-text [49] as the embedding model to encode all texts. We use KNN for graph augmentation and the weighted Leiden algorithm for community detection. For each retrieval item k, we search the same number of items at each layer, with $k = 5$ as the default. All RAG methods are required to complete index construction and query execution within 3 days, respectively. Our codes are available at: https://github.com/sam234990/ArchRAG. Additional details are provided in the appendix of our technical report [68].

4.2 Overall results

To answer the question *RQ1*, we compare our method with baseline methods in solving both abstract and specific QA tasks.

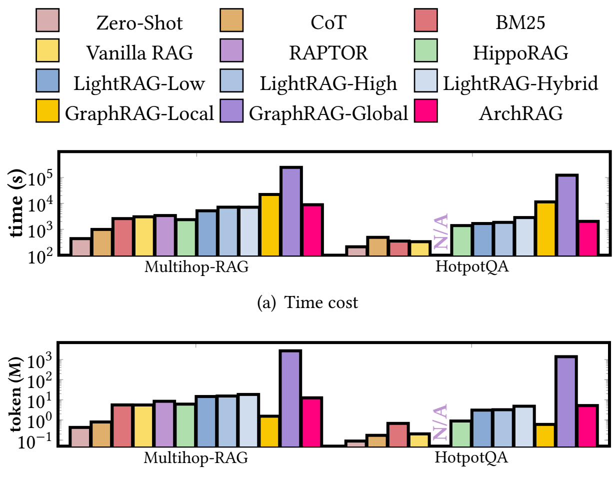
• **Results of abstract QA tasks.** We compare ArchRAG against baselines across four dimensions on the Multihop-RAG dataset. In this comparison, we only compare the LightRAG-Hybrid method, as it represents the best version of LightRAG methods [20]. As shown in Figure 3, GraphRAG-Global outperforms other baseline methods, while our method achieves comparable performance on the diversity and empowerment dimensions and significantly surpasses it on the comprehensive dimension. Overall, by leveraging attributed communities, ArchRAG demonstrates superior performance in addressing abstract QA tasks.

• **Results of specific QA tasks.** Table 3 reports the performance of each method on three datasets. Note that GraphRAG-Global fails

Table 3: Performance comparison of different methods across various datasets for solving specific QA tasks. The best and second-best results are marked in bold and underlined, respectively. OOT: Method did not finish querying within 3 days.

Method Type	Method	Multihop-RAG		HotpotQA		NarrativeQA		
		(Accuracy)	(Recall)	(Accuracy)	(Recall)	(BLEU-1)	(METEOR)	(ROUGE-L F1)
Inference-only	Zero-shot	47.7	23.6	28.0	31.8	8.0	7.9	8.6
	CoT	54.5	28.7	32.5	39.7	5.0	8.1	6.4
Retrieval-only	BM25	37.6	19.4	49.7	53.6	2.0	4.9	2.8
	Vanilla RAG	58.6	31.4	50.6	56.1	2.0	4.9	2.8
Graph-based RAG	RAPTOR	59.1	34.1	N/A	N/A	5.5	12.5	9.1
	HippoRAG	38.9	19.1	51.3	56.8	2.2	5.0	2.8
	LightRAG-Low	44.1	25.1	34.1	41.8	4.5	8.7	6.6
	LightRAG-High	48.5	28.7	25.6	33.3	4.4	8.1	6.1
	LightRAG-Hybrid	50.3	30.3	35.6	43.3	5.0	9.4	7.0
	GraphRAG-Local	40.1	23.8	29.7	35.5	3.9	3.3	3.5
	GraphRAG-Global	45.9	28.4	33.5	42.6	OOT	OOT	OOT
Our proposed	ArchRAG	**68.8**	**37.2**	**65.4**	**69.2**	**11.5**	**15.6**	**17.6**

to complete querying on the NarrativeQA dataset within the 3-day time limit. RAPTOR is unable to build the index on datasets like HotpotQA, which contains a large number of text chunks. Its Gaussian Mixture Model (GMM) clustering algorithm requires prohibitive computational time and suffers from non-termination issues during clustering. Clearly, ArchRAG demonstrates a substantial performance advantage over other baseline methods on these datasets. The experimental results suggest that not all communities are suitable for specific QA tasks, as the GraphRAG-Global performs poorly. Furthermore, GraphRAG does not consider node attributes during clustering, which causes the community's summary to become dispersed, making it difficult for the LLM to extract relevant information. Thus, we gain an interesting insight: *LLM may not be a good retriever, but is a good analyzer.* We further analyze the reasons behind the underperformance of each graph-based RAG method and support our claims with empirical evidence in the appendix of our technical report [68].

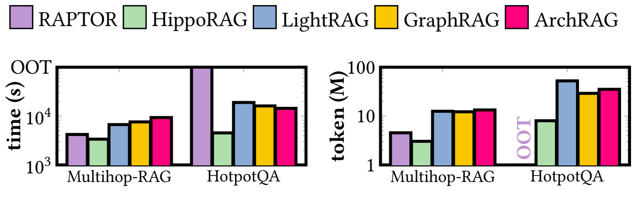


(a) Time cost

(b) Token cost

Figure 4: Comparison of query efficiency.

• **Efficiency of ArchRAG.** To answer the question *RQ2*, we compare the time cost and token usage of ArchRAG with those of other baseline methods. As shown in Figure 4, ArchRAG demonstrates significant time and cost efficiency for online queries. For example, token usage on the HotpotQA dataset is cut by **250×** with ArchRAG compared to GraphRAG-Global, from 1,394M tokens down to 5.1M tokens. Besides, Figure 5 shows the index construction time and token usage for different methods. The cost of building an index for ArchRAG is similar to that of GraphRAG, but due to the need for community summarization, both take a higher time cost and token usage than HippoRAG.



(a) Time cost

(b) Token cost

Figure 5: Comparison of indexing efficiency.

• **Efficiency of hierarchical search.** We compare our hierarchical search based on C-HNSW with a baseline approach (Base-HNSW), which independently builds a vector index for attributed communities at each layer and performs retrieval separately for each. As shown in Figure 6, on the large-scale synthetic dataset, C-HNSW achieves up to a 5.4× speedup (On level 1, C-HNSW takes 1.861 seconds, while Base-HNSW takes 10.125 seconds.) and is on average 3.5× faster than Base-HNSW. Additional details are provided in the appendix of our technical report [68].

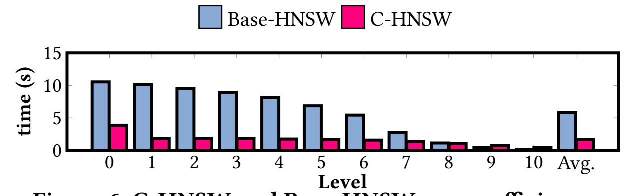


Figure 6: C-HNSW and Base-HNSW query efficiency.

4.3 Detailed analysis

To answer the question *RQ3*, we first evaluate the community qualities under our proposed LLM-based attributed clustering framework and then conduct an ablation study.

• **Community quality of different clustering methods.** We evaluate the community quality of our proposed LLM-based hierarchical clustering framework using four clustering algorithms (weighted Leiden, Spectral Clustering [63], SCAN [74], and node2vec [19] with KMeans), combined with two graph augmentation techniques (the KNN algorithm and CODICIL [53]). The resulting communities are assessed using the Calinski-Harabasz Index (CHI) [5] and Cosine Similarity (Sim) [7], where higher values indicate better quality. Further details on the clustering implementation and evaluation metrics can be found in the appendix of our technical report [68]. Figures 7 and 8 show that combining KNN or CODICIL with the weighted Leiden algorithm significantly enhances community detection quality.

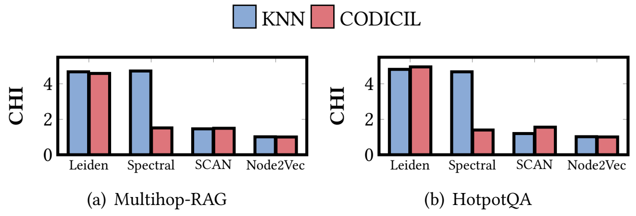


Figure 7: Community quality evaluated by CH Index.

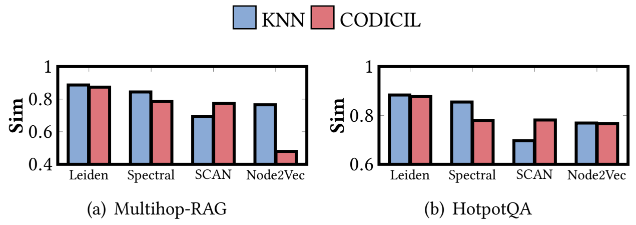


Figure 8: Community quality evaluated by Cosine Similarity.

• **Ablation study.** To evaluate the effect of the attributed community in ArchRAG, we design some variants of ArchRAG and conduct the ablation experiments.

- Spec.: Replaces the Weighted Leiden community detection algorithm with spectral clustering.
- Spec. (No Aug): Removes the graph augmentation and uses spectral clustering.
- Leiden: Replaces our clustering framework with the Leiden algorithm.
- Single-Layer: Replaces our hierarchical index and search with a single-layer community.
- Entity-Only: Generate the response using entities only.
- Direct Prompt: Direct prompts the LLM to generate the response without the adaptive filtering-based generation.

The first two variants replace the algorithm in the LLM-based hierarchical clustering framework, while the last three variants modify key components of ArchRAG: attributes, hierarchy, and communities. As shown in Table 4, the performance of ArchRAG on specific QA tasks decreases when each feature is removed, with the removal of the community component resulting in the most significant drop. Additionally, the direct variant demonstrates that the adaptive filtering-based generation process can effectively extract relevant information from retrieved elements.

Table 4: Comparing the performance of different variants of ArchRAG on the specific QA tasks.

Method variants	Multihop-RAG		HotpotQA	
	(Accuracy)	(Recall)	(Accuracy)	(Recall)
ArchRAG	68.8	37.2	65.4	69.2
- Spec.	67.1	36.7	60.5	63.7
- Spec. (No Aug)	62.8	34.2	64.8	63.2
- Leiden	63.2	34.1	61.7	64.8
- Single-Layer	63.8	36.4	60.1	63.6
- Entity-Only	61.2	34.7	59.9	63.1
- Direct Prompt	59.9	29.6	40.7	45.4

In addition, we evaluate the performance of ArchRAG under different LLM backbones and various top-k retrieval content to address question *RQ4*. ArchRAG consistently achieves state-of-the-art performance across all LLM backbones and remains robust under varying top-k values. Detailed results and additional comparison experiments are provided in the appendix of our technical report [68].

5 Related Work

In this section, we review the related works, including Retrieval-Augmentation-Generation (RAG) approaches, and LLMs for graph mining and learning.

• **RAG approaches.** RAG has been proven to excel in many tasks, including open-ended question answering [28, 57], programming context [9, 10], SQL rewrite [38, 59], and data cleaning [46, 47, 51]. The naive RAG technique relies on retrieving query-relevant contexts from external knowledge bases to mitigate the "hallucination" of LLMs. Recently, many RAG approaches [12, 20, 21, 23, 34, 42, 42, 54, 70] have adopted graph structures to organize the information and relationships within documents, achieving improved overall retrieval performance. For more details, please refer to the recent survey of graph-based RAG methods [50].

• **LLM for graph mining.** Recent advances in LLMs have offered opportunities to leverage LLMs in graph mining. These include using LLMs for KG construction [87], and employing KG to enhance the LLM reasoning [27, 29, 41, 45, 58, 67, 70]. For instance, KD-CoT [67] retrieves facts from an external knowledge graph to guide the CoT performed by LLMs. RoG [41] proposes a planning-retrieval-reasoning framework that retrieves reasoning paths from KGs to guide LLMs conducting faithful reasoning. KET-RAG [27] introduces an efficient indexing approach and enhances the final generation quality by incorporating keyword-based retrieval. StructGPT [29] and ToG [58] treat LLMs as agents that interact with KGs to find reasoning paths leading to the correct answers. Besides, LLM is used to generate code to solve complex graph mining problems, e.g., the shortest path computation [6, 8, 60, 80].

6 Conclusion

In this paper, we propose ArchRAG, a novel graph-based RAG approach, by augmenting the question using attributed communities from the knowledge graph built on the external corpus, and building a novel index for efficient retrieval of relevant information. Our experiments show that ArchRAG is highly effective and efficient. In the future, we will explore fast parallel graph-based RAG methods to process large-scale external corpus.

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] Stefanos Angelidis and Mirella Lapata. 2018. Summarizing opinions: Aspect extraction meets sentiment prediction and they are both weakly supervised. *arXiv preprint arXiv:1808.08858* (2018).

[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] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[5] Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods* 3, 1 (1974), 1–27.

[6] Yukun Cao, Shuo Han, Zengyi Gao, Zezhong Ding, Xike Xie, and S Kevin Zhou. 2024. Graphinsight: Unlocking insights in large language models for graph structure understanding. *arXiv preprint arXiv:2409.03258* (2024).

[7] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 380–388.

[8] N. Chen, Y. Li, J. Tang, and J. Li. 2024. Graphwiz: An instruction-following language model for graph computational problems. In *KDD*.

[9] 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.

[10] 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.

[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] 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).

[17] 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).

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

[19] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[20] 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.

[21] 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).

[22] Rujun Han, Yuhao Zhang, Peng Qi, Yumo Xu, Jenyuan Wang, Lan Liu, William Yang Wang, Bonan Min, and Vittorio Castelli. 2024. RAG-QA Arena: Evaluating Domain Robustness for Long-form Retrieval Augmented Question Answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 4354–4374.

[23] 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).

[24] 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).

[25] Ziniu Hu, Yichong Xu, Wenhao Yu, Shuohang Wang, Ziyi Yang, Chenguang Zhu, Kai-Wei Chang, and Yizhou Sun. 2022. Empowering language models with knowledge graph reasoning for question answering. *arXiv preprint arXiv:2211.08380* (2022).

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

[27] 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).

[28] 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).

[29] 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. *arXiv preprint arXiv:2305.09645* (2023).

[30] Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics* 6 (2018), 317–328.

[31] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.

[32] 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.

[33] 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).

[34] 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).

[35] 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).

[36] 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.

[37] 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).

[38] 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.

[39] 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).

[40] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[41] 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).

[42] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2024. Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation. *arXiv preprint arXiv:2407.10805* (2024).

[43] 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.

[44] 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).

[45] Costas Mavromatis and George Karypis. 2024. GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning. *arXiv preprint arXiv:2405.20139* (2024).

[46] 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.

[47] 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.

[48] 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).

[49] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. 2024. Nomic Embed: Training a Reproducible Long Context Text Embedder. arXiv:2402.01613 [cs.CL]

[50] 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).

[51] 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.

[52] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer, 232–241.

[53] Yiye Ruan, David Fuhry, and Srinivasan Parthasarathy. 2013. Efficient community detection in large networks using content and links. In *Proceedings of the 22nd international conference on World Wide Web*. 1089–1098.

[54] 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).

[55] 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).

[56] 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).

[57] 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.

[58] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *arXiv preprint arXiv:2307.07697* (2023).

[59] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2024. R-Bot: An LLM-based Query Rewrite System. *arXiv preprint arXiv:2412.01661* (2024).

[60] Jianheng Tang, Qifan Zhang, Yuhan Li, and Jia Li. 2024. Grapharena: Benchmarking large language models on graph computational problems. *arXiv preprint arXiv:2407.00379* (2024).

[61] Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391* (2024).

[62] 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.

[63] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17 (2007), 395–416.

[64] Jinyu Wang, Jingjing Fu, Rui Wang, Lei Song, and Jiang Bian. 2025. PIKE-RAG: sPecIalized KnowledgE and Rationale Augmented Generation. *arXiv preprint arXiv:2501.11551* (2025).

[65] 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).

[66] 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).

[67] 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).

[68] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. [n. d.]. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation [Scalable Data Science]. https://github.com/sam234990/ArchRAG.

[69] 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).

[70] 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.

[71] 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).

[72] 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).

[73] Shicheng Xu, Liang Pang, Mo Yu, Fandong Meng, Huawei Shen, Xueqi Cheng, and Jie Zhou. 2024. Unsupervised Information Refinement Training of Large Language Models for Retrieval-Augmented Generation. *arXiv preprint arXiv:2402.18150* (2024).

[74] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. 2007. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 824–833.

[75] 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).

[76] 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).

[77] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[78] 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).

[79] Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. 2024. SiReRAG: Indexing Similar and Related Information for Multihop Reasoning. *arXiv preprint arXiv:2412.06206* (2024).

[80] Qifan Zhang, Xiaobin Hong, Jianheng Tang, Nuo Chen, Yuhan Li, Wenzhong Li, Jing Tang, and Jia Li. 2024. Gcoder: Improving large language model for generalized graph problem solving. *arXiv preprint arXiv:2410.19084* (2024).

[81] 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).

[82] 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.

[83] 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.

[84] 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).

[85] 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.

[86] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. 2009. Graph clustering based on structural/attribute similarities. *Proceedings of the VLDB Endowment* 2, 1 (2009), 718–729.

[87] Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2024. Llms for knowledge graph construction and reasoning: Recent capabilities and future opportunities. *World Wide Web* 27, 5 (2024), 58.

A Method details

A.1 LLM-based hierarchical clustering

For example, as shown in the second step of offline indexing in Figure 2, LLM-based hierarchical clustering first enhances the original KG at layer L_0 by adding similar edges. Next, the weight of each edge is calculated based on the strength of the relationship between the node embeddings, and a weighted clustering algorithm is applied to obtain four communities. Based on the existence of links between nodes within a community, the topology of communities at layer L_1 is constructed, resulting in a graph of communities. This process is repeated, ultimately generating a clustering result consisting of three layers of hierarchical communities.

A.2 More details of C-HNSW

• **Introduction of HNSW.** We provide a brief introduction to HNSW, an efficient Approximate Nearest Neighbor Search (ANNS) technique for vector databases.

Definition 3 (Hierarchical Navigable Small World (HNSW) [43]). HNSW is a graph-based ANNS algorithm that consists of a multi-layered index structure, where each node uniquely corresponds to a vector in the database. Given a set S containing n vectors, the constructed HNSW can be represented as a pair $\mathcal{H} = (\mathbf{G}, C)$. $\mathbf{G} = \{G_0, G_1, \ldots, G_L\}$ is a set of simple graphs (also called layers) $G_i = (V_i, E_i)$, where $i \in \{0, 1, \ldots, L\}$ and $V_L \subset V_{L-1} \subset \cdots \subset V_1 \subset V_0 = S$. C records the inter-layer mappings of edges between the same node across adjacent layers $C = \bigcup_{i=0}^{L-1}\{(v, \phi(v)) | v \in V_i, \phi(v) \in V_{i+1}\}$, where $\phi(v) : V_i \longrightarrow V_{i+1}$ is the mapping function for the same node across two adjacent layers.

The nodes in the multi-layer graph of HNSW are organized in a nested structure, where each node at each layer is connected to its nearest neighbors. During a query, the search begins at the top layer and quickly identifies the node closest to q through a greedy search. Then, through inter-layer mapping, the search proceeds to the next lower layer. This process continues until all approximate nearest neighbors are identified in G_0.

• **The construction of C-HNSW.** The construction of C-HNSW is illustrated in Algorithm 3. The construction algorithm of C-HNSW follows a top-down approach. Using the query process of C-HNSW, we obtain the M nearest neighbors of each node in its layer, and the inter-layer links are continuously updated during this process. When inserting a node x at layer i , if the nearest neighbor at the layer $i + 1$ is c_j, the inter-layer link of c_j is updated in the following two cases:

- Node c_j does not have a inter-layer link to the layer i.
- The distance from node c_j to node x is smaller than the distance from c_j to its previous nearest neighbor x', i.e., $d(c_j, x) < d(c_j, x')$.

After all nodes at layer i ($i < L$) have been inserted, we check each node at layer $i+1$ to ensure it has a inter-layer link, confirming the traverse from the higher layer to the lower layer.

A.3 Complexity analysis of ArchRAG

We now analyze the complexity of our ArchRAG approach. Since the token cost is more important in the era of LLM, we replace the space complexity with the token cost.

Algorithm 3: C-HNSW construction

```
input : The Hierarchical community $\mathcal{HC}$, KG $\mathcal{G}(V, E)$, maximum
        number of connections for each node $M$.
1  $\mathcal{H} \leftarrow \emptyset$;
2  $\mathbf{V} \leftarrow \{\mathcal{HC} \cup V\}$ // Get all nodes of each layer.
3  for each layer $l \leftarrow L \cdots 0$ do
4      for each node $v \in V_l$ do
5          if $l \neq L$ then
6              $R \leftarrow$ SearchLayer($G_l = (V_l, E_l), q, s, 1, l$);
7              $c \leftarrow$ get the nearest node from R;
8              $s \leftarrow$ node in layer $l - 1$ via $c$'s inter-layer link;
9              if $s$ is null or $d(c, s) > d(c, v)$ then
10                 update $v$ as $c$'s inter-layer link
11         if $l = L$ or $s$ is null then $s \leftarrow$ random node in layer $l$;
12         $R \leftarrow$ SearchLayer($G_l = (V_l, E_l), q, s, M, l$);
13         add edges between $v$ and $R$, update $E_l$;
14     $\mathcal{H} \leftarrow \mathcal{H} \cup G_l = (V_l, E_l)$
15 return $\mathcal{H}$;
```

The offline indexing process of ArchRAG includes the KG construction, hierarchical clustering, and C-HNSW construction. The time complexity and token usage are as follows:

Lemma 4. *Given a large text corpus or a large set of text documents with a total of D tokens, the time complexity of the offline indexing process of ArchRAG is* $O(I\frac{D}{w} + \frac{1-a^L}{1-a}(n * t + I\frac{D}{w} + \pi(m) + n\log n))$, *where* $\mathbf{I}$ *is the generation time of the LLM for a single inference, w is the specified token size of one chunk, n and m are the number of entities and relations in the extracted KG, L is the height of the resulting hierarchical community structure, and a is the average ratio of the number of nodes between two consecutive layers, with* $0 < a < 1$. *For a given embedding model, the computation time for the embedding of an entity description is denoted by t, while a specific clustering method is typically a function of m, represented as* $\pi(m)$.

Proof. For the corpus D, we use the LLM to infer and extract the KG from each chunk of size w, resulting in a cost of $O(I\frac{D}{w})$ for constructing the KG. Since the size of all community summaries in a single layer would not exceed the length of the corpus, their LLM inference time is also less than $O(I\frac{D}{w})$. For each layer of clustering, the embedding of each point must be computed, and clustering is performed with time complexity of $\pi(m)$. In the C-HNSW construction, each point requires $O(nlogn)$ time to perform the k-nearest neighbor search and establish connections, which is similar to the proof in [43]. For an L-layer multi-layer graph structure, where the number of nodes decreases by a factor of a between two consecutive layers, the increase in clustering and C-HNSW construction time is given by: $\frac{1-a^L}{1-a}$. □

Lemma 5. *Given a large text corpus or a large set of text documents with a total of D tokens, the number of tokens used in the offline indexing process of ArchRAG is* $O(D(1 + \frac{1-a^L}{1-a}))$, *where L is the height of the resulting hierarchical community structure and a is the average ratio of the number of nodes between two consecutive layers, with* $0 < a < 1$.

Proof. Based on the above proof, the token cost for constructing the KG is $O(D)$, while the token cost for the summaries does not exceed $O(D\frac{1-a^L}{1-a})$. □

Generally, L is $O(\log n)$, where n is the number of extracted entities. However, due to the constraints of community clustering, L is typically constant, usually no greater than 5.

Next, we analyze the time complexity and token usage in the online query process of the ArchRAG.

Lemma 6. *Given a C-HNSW with L layers constructed from a large text corpus or a large set of text documents, the time complexity of a sequentially executed single online retrieval query in ArchRAG is $O(e+LkI+Lk\log(n))$, where I is the generation time of the LLM for a single inference, e is the time cost of computing the query embedding, k is the number of nodes retrieved at each layer, and n is the number of nodes at the lowest layer in C-HNSW.*

Proof. ArchRAG first computes the embedding of the query, which takes $O(e)$ time. For each layer, querying one nearest neighbor takes no more than $O(\log(n))$ time, similar to the proof in [43]. In the Adaptive filtering-based generation, the content of each query is analyzed and inferred, requiring $O(LkI)$ time. Therefore, the total time for the online retrieval query is $O(e + LkI + Lk\log(n))$. □

Lemma 7. *Given a C-HNSW with L layers constructed from a large text corpus or a large set of text documents, the number of tokens used for a single online retrieval query in ArchRAG is $O(kL(c+P)))$, where k is the number of neighbors retrieved at each layer, c is the average length of the retrieved content, and P is the length of the prompt.*

Proof. The token consumption for analyzing all retrieved information is $O(kL(c + P)))$, while the token consumption for generating the final response is of constant order. Therefore, the total token consumption for the online retrieval is $O(kL(c + P)))$. □

In practice, multiple retrieved contents are combined, allowing the LLM to analyze them together, provided the total token count does not exceed the token size limit. As a result, the time and token usage for online retrieval are lower than those required for analysis.

B Experimental details

B.1 Metrics

This section provides additional details on the metrics.

• **Metrics for specific QA tasks.** We choose accuracy as the evaluation metric based on whether the gold answers are included in the model's generations rather than strictly requiring an exact match, following [3, 44, 55]. This is because LLM outputs are typically uncontrollable, making it difficult for them to match the exact wording of standard answers. Similarly, we choose recall as the metric instead of precision, as it better reflects the accuracy of the generated responses. Additionally, when calculating recall, we adopt the same approach as previous methods [3, 21]: if the golden answer or the generated output contains "yes" or "no", the recall for that question is set to 0. Therefore, the recall metric is not perfectly correlated with accuracy.

• **Metrics for abstract QA tasks.** Following existing works, we use an LLM to generate abstract questions, with the prompts shown in Figure 13, defining ground truth for abstract questions, particularly those involving complex high-level semantics, poses significant challenges. We build on existing works [12, 20] to address this and adopt an LLM-based multi-dimensional comparison method (including comprehensiveness, diversity, empowerment, and overall). We employ a robust LLM, specifically GPT-4o-mini, to rank each baseline against our method. Figure 14 shows the evaluation prompt we use.

• **Metrics of community quality.** We select the following metrics to evaluate the quality of the community:

(1) *Calinski-Harabasz Index (CHI)* [5]: A higher value of CHI indicates better clustering results because it means that the data points are more spread out between clusters than they are within clusters. It is an internal evaluation metric where the assessment of the clustering quality is based solely on the dataset and the clustering results and not on external ground-truth labels. The CHI is calculated by between-cluster separation and within-cluster dispersion:

$$CHI = \frac{N-C}{C-1}\frac{\sum_{i=1}^{C} n_i||c_i - c||^2}{\sum_{i=1}^{C}\sum_{x\in C_i}||x-c_i||^2}. \tag{5}$$

N is the number of nodes. C is the number of clusters.n_i is the number of nodes in cluster i. C_i is the $i-th$ cluster. c_i is the centroid of cluster C_i. c is the overall centroid of the datasets. x is the feature of the target node.

(2) *Cosine Similarity (Sim)* [7]: Cosine similarity is a measure of similarity between two non-zero vectors defined in an inner product space. In this paper, for each cluster, we calculate the similarity between the centroid of this cluster and each node in this cluster:

$$Sim = \frac{1}{N}\sum_{i=1}^{C}\sum_{x\in C_i} Cosine(x, c_i). \tag{6}$$

N is the number of nodes. C is the number of clusters. C_i is the i-th cluster. c_i is the centroid of cluster C_i. x is the feature of the target node.

$$Cosine(x,y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}. \tag{7}$$

B.2 Implementation details

We implement our ArchRAG in Python, while C-HNSW is implemented in C++ and provides a Python interface for integration. We implement C-HNSW using the FAISS framework and employ the inner product metric to measure the proximity between two vectors. All the experiments were conducted on a Linux operating system running on a machine with an Intel Xeon 2.0 GHz CPU, 1024GB of memory, and 8 NVIDIA GeForce RTX A5000 GPUs, each with 24 GB of memory. All methods utilize 10 concurrent LLM calls, and to maintain consistency, other parallel computations in the method, such as embedding calculations, also use 10 concurrent threads. Figure 15 demonstrates the prompt used in Adaptive filtering-based generation. Please refer to our repository (https://github.com/sam234990/ArchRAG) to view the detailed prompts.

• **Details of clustering methods.**

The graph augmentation methods we choose are the KNN algorithm, which computes the similarity between each node, and CODICIL [53], which selects and adds the top similar edges to generate better clustering results. In the KNN method, we set the K value as the average degree of nodes in the KG.

B.3 More experiments

• **More Details on the Efficiency of C-HNSW.** To evaluate the efficiency of C-HNSW, we conduct experiments on a synthetic hierarchical dataset comprising 11 layers. The bottom layer (Layer 0) contains 10 million nodes, and the number of nodes decreases progressively across higher layers by randomly dividing each layer's size by 3 or 4. The top layer (Layer 10), for example, contains only 74 nodes. This hierarchical structure simulates the process of LLM-based hierarchical clustering. Each node is assigned a randomly generated 3072-dimensional vector, simulating high-dimensional embeddings such as those produced by text or image encoders (e.g., `text-embedding-3-large`, used in ChatGPT, can generate 3072-dimensional vectors, and text-embedding-v3 can generate 1024-dimensional vectors). For each layer, we generate 200 random queries and compute the top-5 nearest neighbors for each query. Both C-HNSW and Base-HNSW are configured with identical parameters: $M = 32$, $efSearch = 100$, and $efConstruction = 100$. Importantly, our method maintains comparable retrieval accuracy to Base-HNSW, with recall of 0.5537 and 0.6058, respectively.

We also conducted experiments on a synthetic dataset of 1024-dimensional vectors, keeping all other parameters unchanged. The results are shown in Figure 9. As 1024 dimensions are more representative of commonly used high-dimensional embeddings, our method still achieves over 5× speedup in the best case and an average speedup of 3× compared to the baseline.

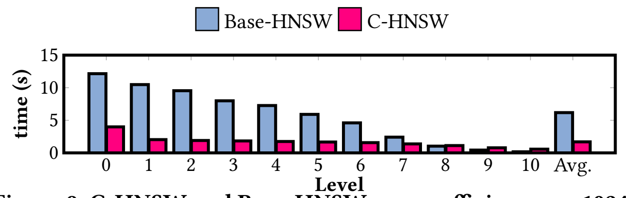


Figure 9: C-HNSW and Base-HNSW query efficiency on 1024-dimensional vector dataset.

To further demonstrate the effectiveness of ArchRAG, we conduct the following experiments:

• **Effect of LLM backbones.** Given the limited budget, we restrict our evaluation to GPT-4o-mini and GPT-3.5-turbo as the LLM backbones, and compare a representative subset of strong RAG methods on the HotpotQA and Multihop-RAG datasets. As strong LLMs with hundreds of billions of parameters (e.g., GPT-3.5-turbo) possess enhanced capabilities, our proposed ArchRAG may also benefit from performance improvement. As shown in Table 5, the results of Llama 3.1-8B are similar to those of GPT-3.5-turbo, as Llama3.1's capabilities are comparable to those of GPT-3.5-turbo [11]. GPT-4o-mini performs better than other LLM backbones because of its exceptional reasoning capabilities.

Besides, we have compared several strong RAG baselines under different LLM backbones. As LLMs' parameters and reasoning capabilities increase, all RAG approaches benefit from performance gains, especially HippoRAG. ArchRAG consistently achieves state-of-the-art performance across most settings.

• **Effect of k values.** We compare the performance of ArchRAG under different retrieved elements. As shown in Figure 10, the performance of ArchRAG shows little variation when selecting different retrieval elements (i.e., communities and entities in each layer). This suggests that the adaptive filtering process can reliably extract the most relevant information from the retrieval elements and integrate it to generate the answer.

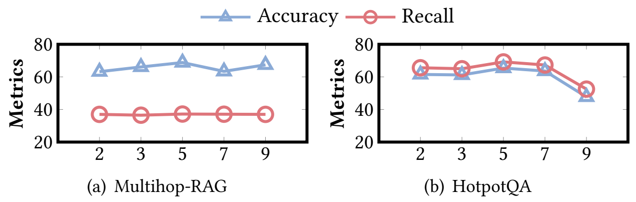


Figure 10: Comparative analysis of the different numbers of retrieval elements in ArchRAG.

• **More experiments on the additional dataset.** We further conduct experiments on the RAG-QA Arena dataset [22], a high-quality, multi-domain benchmark featuring human-annotated, coherent long-form answers. To the best of our capability, we use publicly available data from five domains (including lifestyle, recreation, science, technology, and writing), selecting 200 questions per domain. Following prior work, we employ LLMs as evaluators to compare the RAG-generated responses with ground-truth answers in terms of win ratio and win + tie ratio. We evaluate the top-performing methods (including Vanilla RAG, HippoRAG, RAPTOR, and our ArchRAG) and present the results in terms of the win ratio and win + tie ratio against the ground-truth annotations in the table below. As shown in Table 6, ArchRAG consistently achieves state-of-the-art performance across all evaluated settings.

• **Effectiveness of our attributed clustering algorithm.** To further demonstrate the effectiveness of our attributed clustering algorithm, we evaluate the quality of communities (i.e., CHI and Cosine Similarity) generated by our attributed clustering algorithm compared to those produced by the Leiden algorithm, which is used in GraphRAG for structural clustering. As shown in Table 7, our attribute-based clustering consistently yields higher-quality communities.

Furthermore, we propose a new variant, *GraphRAG+AC*, which replaces the original Leiden-based communities in GraphRAG with our attributed communities, while preserving the original Global Search pipeline. As shown in Table 8, this variant results in a significant performance improvement compared to the original approach. Specifically, on the HotpotQA dataset, GraphRAG+AC improves accuracy by 51% compared to GraphRAG-Global.

• **Case study.** We present an additional case study from Multihop-RAG. As shown in Figure 11, only our method generates the correct answer, while others either provide incorrect or irrelevant information. We only show the core output for brevity, with the remaining marked as "</>". We also show the retrieval and adaptive filtering process of ArchRAG in Figure 12. The results demonstrate that ArchRAG effectively retrieves relevant information and filters out noise, leading to a correct final answer.

• **Discussion of the performance of other graph-based RAG methods.** On the Multihop-RAG dataset, HippoRAG performs worse than retrieval-only methods while outperforming retrieval-based methods on the HotpotQA dataset. This is mainly because, on the HotpotQA dataset, passages are segmented by the expert annotators; that is, passages can provide more concise information, whereas on Multihop-RAG, passages are segmented based on chunk size, which may cause the LLM to lose context and produce incorrect answers. Besides, HippoRAG also suffers from inaccurate entity recognition. In addition, we also provide a brief analysis of why

Table 5: Comparing ArchRAG with other RAG methods on the specific QA tasks under different LLM backbone models.

LLM backbone	Methods	Multihop-RAG		HotpotQA	
		(Accuracy)	(Recall)	(Accuracy)	(Recall)
Llama3.1-8B	Vanilla RAG	58.6	31.4	50.6	56.1
	HippoRAG	38.9	19.1	51.3	56.8
	RAPTOR	59.1	34.1	N/A	N/A
	ArchRAG	**68.8**	**37.2**	**65.4**	**69.2**
GPT-3.5-turbo	Vanilla RAG	65.9	32.8	60.7	**65.9**
	HippoRAG	68.9	31.4	58.0	62.3
	RAPTOR	64.4	**34.6**	N/A	N/A
	ArchRAG	**67.2**	31.5	**62.8**	65.0
GPT-4o-mini	Vanilla RAG	71.4	32.8	68.2	70.1
	HippoRAG	70.5	31.6	65.0	68.5
	RAPTOR	70.1	32.6	N/A	N/A
	ArchRAG	**77.3**	**33.8**	**69.9**	**73.8**

Table 6: Comparing ArchRAG with other RAG methods on the RAG-QA Area dataset. Each entry denotes the win ratio and win + tie ratio of the corresponding method against the ground-truth annotations, based on LLM evaluation.

Method	Lifestyle	Recreation	Science	Technology	Writing
Vanilla RAG	17.5 / 20.5	17.0 / 25.0	32.5 / 37.0	28.5 / 34.0	15.0 / 16.5
HippoRAG	26.5 / 26.5	29.5 / 30.5	49.5 / 49.5	42.0 / 42.5	21.0 / 21.5
RAPTOR	17.0 / 19.5	18.5 / 24.5	39.0 / 45.0	33.0 / 35.5	25.0 / 26.5
ArchRAG	**49.5 / 50.0**	**41.5 / 41.5**	**56.0 / 56.0**	**59.0 / 59.5**	**45.0 / 45.0**

Table 7: Comparison of Community Quality between Our Attributed Clustering Method and Leiden

Method	Multihop-RAG		HotpotQA	
	(CHI)	(Sim)	(CHI)	(Sim)
Leiden	3.02	0.71	3.42	0.71
Ours	4.68	0.89	4.82	0.88

Table 8: Results of GraphRAG variants using our attributed clustering methods

Method	Multihop-RAG		HotpotQA	
	(Accuracy)	(Recall)	(Accuracy)	(Recall)
GraphRAG-Global	45.9	28.4	33.5	42.6
GraphRAG+AC	49.3 (7.4% ↑)	31.4	50.6 (51.0% ↑)	52.8
ArchRAG (Ours)	68.8	37.2	65.4	69.2

some graph-based RAG methods underperform. Specifically, they often fail to retrieve relevant information during the search process. For example:

- *GraphRAG:* The suboptimal performance of GraphRAG has been widely observed in research. For instance, Zhang et al. [79] find that GraphRAG tends to respond with "I don't know" when retrieving irrelevant content, indicating that it prefers not to give a concrete answer. PIKE-RAG [64] identifies that GraphRAG tends to echo the query and include meta-information about answers within its graph structure. Through manual verification, we confirm the presence of similar issues in our experimental results. In particular, GraphRAG-Local Search also frequently returns irrelevant content.
- *HippoRAG/LightRAG:* We attribute part of HippoRAG's suboptimal performance to entity recognition issues. HippoRAG first extracts relevant entities from questions and then retrieves chunks based on these entities. However, it sometimes fails to identify appropriate entities (Entity Recognition Errors, ER Errors), and we conduct corresponding experiments, as shown in Table 9. For example, on HotpotQA, HippoRAG fails to detect entities in 5.0% of questions and identifies low-quality ones in 15.8% (e.g., less than 3 characters, numeric-only terms, or sentence entities with more than 5 words). These issues make it difficult to match accurate entities in the KG, resulting in irrelevant retrieval and suboptimal performance. Similar issues also impair LightRAG's performance because it requires accurate topic identification from questions. We also observe that as the LLM's capability improves, the performance of these methods increases significantly, as shown in the experiment **Effect of LLM backbone**.

Question & Ground-truth

Question: Who is the individual associated with generative AI technology that was reportedly ousted from a leading AI organization, but is recognized for brilliance and generosity in the industry, and is currently planning to launch a new venture according to reports from TechCrunch and Fortune?
Ground-truth: Sam Altman

Responses of Different Methods

Zero-shot
Jesse Schell.
He is the CTO of Calm and was the lead designer on the “Star Wars" theme park ride at Disney.

Vanilla RAG
It seems like there are multiple articles and topics discussed in this text. Here’s a brief summary of each section:
1.**Robotics predictions.** The article discusses the future of robotics, with predictions made by Brian Heater. </>
2.**TechCrunch’s Week in Review.** This section summarizes various tech industry happenings. </>
3.**Bumble gets a new CEO.** Bumble announces a change in leadership. </>...

HippoRAG
There is **no question** in the provided text. It appears to be a collection of news articles and updates related to OpenAI’s ChatGPT. </>

GraphRAG-Global
Key Points and Implications
The individual associated with generative AI technology who was reportedly ousted is **Andrew Ng**. </>
According to multiple analysts, **Andrew Ng** [Data: Reports (5, 6)] </>
Implications of the New Venture </>

ArchRAG
Sam Altman.
This information is supported by multiple analysts’ reports, which highlight **Sam Altman’s** involvement in various AI-related projects. The direct answer is simply **"Sam Altman"**. </>
Note: I have not included any information from Analyst 2, 3, or 7 as their reports are not relevant. </>

Figure 11: Case study of responses by different RAG methods on a question from the Multihop-RAG dataset. RED TEXT indicates incorrect content, CYAN TEXT denotes correct content, and </> marks omitted irrelevant parts.

```
Top Retrieved Entities:
 - Entity 1058: OPENAI
   Description: OpenAI is a tech company founded by Sam Altman </>

 - Community 4097: Microsoft, OpenAI, and AI Regulation
   Summary: Discussion between Satya Nadella and Sam Altman </>

Adaptive Filtering Result:
 - Sam Altman is among the backers of an AI startup.
 - Score: 80.0 (Reports: 1) </>
```

Figure 12: ArchRAG Retrieval & Filtering Output. </> marks omitted irrelevant parts.

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 13: The prompt for generating abstract questions.

Table 9: Distribution of HippoRAG's ER Errors

Datasets	Null Entity Rate	Low-Quality Entity Rate
Multihop-RAG	1.3%	11.9%
HotpotQA	5.0%	15.8%

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 14: The prompt for the evaluation of abstract QA.

Prompt for Adaptive filtering-based generation

Filger Prompt:
# Role
You are a helpful assistant responding to questions about data in the tables provided.
# Goal
Generate a response consisting of a list of key points that respond to the user's question, summarizing all relevant information in the input data tables. You should use the data provided in the data tables below as the primary context for generating the response. If you don't know the answer or if the input data tables do not contain sufficient information to provide an answer, just say so. Do not make anything up.
Each key point in the response should have the following element:

- Description: A comprehensive description of the point.
- Importance Score: An integer score between 0-100 that indicates how important the point is in answering the user's question. An 'I don't know' type of response should have a score of 0.

The response should be JSON formatted as follows:

```
{
    "points": [
       {"description": "Description of point 1 [Data: Reports (report ids)]", "score": score_value},
       {"description": "Description of point 2 [Data: Reports (report ids)]", "score": score_value},
    ]
}
```

# User Question

```
    {user_query}
```

# Data tables

```
    {context_data}
```

Output:

Merge Prompt:
# Role
You are a helpful assistant responding to questions and may use the provided data as a reference.
# Goal
You should incorporate insights from all the reports from multiple analysts who focused on different parts of the dataset to support your answer. Please note that the provided information may contain inaccuracies or be unrelated. If the provided information does not address the question, please respond using what you know. namely,

- A response that utilizes the provided information, ensuring that all irrelevant details from the analysts' reports are removed.
- A response to the user's query based on your existing knowledge when <Analyst Reports> is empty.

The final response should merge the relevant information into a comprehensive answer that clearly explains all key points and implications, tailored to the appropriate response length and format. Note that the analysts' reports provided below are ranked in the *descending order of importance*. Do not include information where the supporting evidence for it is not provided.
# Target response length and format

```
    {response_format}
```

# User Question

```
    {user_query}
```

# Analyst Reports

```
    {report_data}
```

Output:

Figure 15: The prompt for adaptive filtering-based generation.