Clue-RAG: Towards Accurate and Cost-Efficient Graph-based RAG via Multi-Partite Graph and Query-Driven Iterative Retrieval

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

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

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

Quanqing Xu
xuquanqing.xqq@oceanbase.com
OceanBase, Ant Group

Chuanhui Yang
rizhao.ych@oceanbase.com
OceanBase, Ant Group

Abstract

Despite the remarkable progress of Large Language Models (LLMs), their performance in question answering (QA) remains limited by the lack of domain-specific and up-to-date knowledge. Retrieval-Augmented Generation (RAG) addresses this limitation by incorporating external information, often from graph-structured data. However, existing graph-based RAG methods suffer from poor graph quality due to incomplete extraction and insufficient utilization of query information during retrieval. To overcome these limitations, we propose Clue-RAG, a novel approach that introduces (1) a multi-partite graph index incorporates *text Chunk*, *knowledge unit*, and *entity* to capture semantic content at multiple levels of granularity, coupled with a hybrid extraction strategy that reduces LLM token usage while still producing accurate and disambiguated knowledge units, and (2) Q-Iter, a query-driven iterative retrieval strategy that enhances relevance through semantic search and constrained graph traversal. Experiments on three QA benchmarks show that Clue-RAG significantly outperforms state-of-the-art baselines, achieving up to 99.33% higher Accuracy and 113.51% higher F1 score while reducing indexing costs by 72.58%. Remarkably, Clue-RAG matches or outperforms baselines even without using an LLM for indexing. These results demonstrate the effectiveness and cost-efficiency of Clue-RAG in advancing graph-based RAG systems. The source code is available in https://github.com/Feesuu/ClueRAG.

Keywords

Graph-based RAG, Multi-Partite Graph, Query-Driven Iterative Retrieval

ACM Reference Format:
Yaodong Su, Yixiang Fang, Yingli Zhou, Quanqing Xu, and Chuanhui Yang. 2018. Clue-RAG: Towards Accurate and Cost-Efficient Graph-based RAG via Multi-Partite Graph and Query-Driven Iterative Retrieval. In . ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

*Yixiang Fang is the corresponding author.

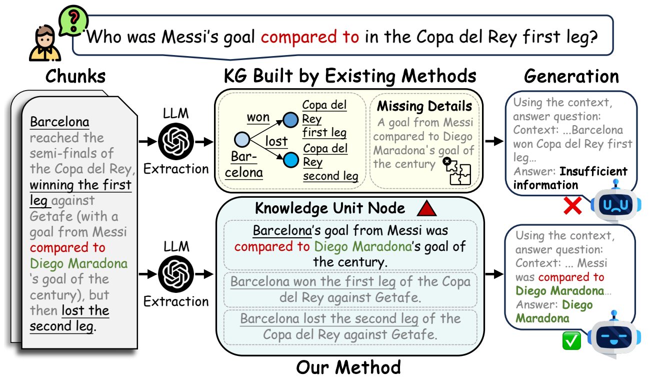
Who was Messi's goal compared to in the Copa del Rey first leg?
Chunks
Barcelona reached the semi-finals of the Copa del Rey, winning the first leg against Getafe (with a goal from Messi compared to Diego Maradona's goal of the century), but then lost the second leg.
LLM
Extraction
KG Built by Existing Methods
won
lost
Barcelona
Copa del Rey first leg
Copa del Rey second leg
Missing Details
A goal from Messi compared to Diego Maradona's goal of the century
Generation
Using the context, answer question: Context: ...Barcelona won Copa del Rey first leg... Answer: Insufficient information
Knowledge Unit Node
Barcelona's goal from Messi was compared to Diego Maradona's goal of the century.
Barcelona won the first leg of the Copa del Rey against Getafe.
Barcelona lost the second leg of the Copa del Rey against Getafe.
Our Method
Using the context, answer question: Context: ... Messi was compared to Diego Maradona... Answer: Diego Maradona

Figure 1: Comparison of triple extraction in existing Graph-based RAG methods (Upper) and ours (Below).

1 Introduction

Large Language Models (LLMs) like Qwen3 [41], DeepSeek [7], and LLaMA3.1 [12] have received tremendous attention from both industry and academia [11, 18, 20, 24, 37]. 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 [28]. To enhance the trustworthiness and interpretability of LLMs, Retrieval-Augmented Generation (RAG) methods [9, 10, 17, 18, 38, 39, 43, 48] have emerged as a core approach, which often retrieve relevant information from documents, relational data, and graph data to facilitate the QA tasks. The state-of-the-art (SOTA) approaches often use the graph to model the external data since they capture the rich semantic information and link relationships between entities.

In the literature, representative graph-based RAG methods typically follow a two-phase pipeline: (1) *Offline index construction*: They first segment the external text corpora $\mathcal{T}$ into small chunks, then extract the nodes and edges, together with their associated textual attributes (e.g., descriptions [8] or keywords [13]) from chunks using LLMs, and finally build a knowledge graph (KG) [40, 49]. (2) *Online retrieval*: Given an online query q, they either retrieve relevant nodes, edges, or subgraphs from the KG, or optionally trace these elements back to their source text chunks via associated provenance links [8, 13, 14]. Afterwards, they incorporate the relevant information from KG or chunks into a prompt template, and then feed the prompt into an LLM for answer generation. Although these methods offer improved performance, they still suffer from two major limitations.

• **Limitation 1: the indexes suffer from the incompleteness issue.** As shown in recent studies [40, 49], the constructed KGs often suffer from incompleteness, typically due to missing key nodes, edges, or the omission of information that cannot be incorporated into structured graph elements. Take the text chunk in Figure 1 as an example. Existing methods typically extract only a few triples (e.g., ⟨*Barcelona*, *won*, *Copa del Rey first legagainst Getafe*⟩), while completely ignoring valuable textual information, such as nuanced comparisons between Messi's performance and Maradona's "Goal of the Century". This incompleteness may lead to retrieval errors, i.e., when a query involving this detail is issued, the KG fails to provide sufficient information for generating an accurate answer, as shown in Figure 1.

• **Limitation 2: the retrieval methods face the issue of semantic misalignment.** In the online retrieval phase, most existing graph-based RAG methods [8, 13, 14, 19] follow two steps: (1) use the input query to identify relevant elements (e.g., nodes, edges) in the KG, and (2) retrieve additional information based on the above elements, such as text chunks or subgraphs, to augment the context for answer generation. While step (1) ensures query relevance, step (2) often fails to fully leverage the query information, leading to a semantic misalignment between the query and the retrieved context.

To address the above limitations, this paper introduces Clue-RAG, a novel graph-based RAG approach, which not only builds a high quality graph, but also well utilize the query information for answering the questions. For **Limitation 1**, we introduce a novel concept, called *knowledge unit*, which is a statement that conveys an atomic piece of information from a text chunk. Collectively, these knowledge units can fully reconstruct the semantic content of the original text. By extracting entities from the knowledge units, we can effectively address missing nodes or edges in the constructed KG. As a result, we build a multi-partite graph composed of three types of nodes, each representing distinct semantic granularities: *text Chunk*, *knowledge unit*, and *entity*. These nodes are connected through extraction and containment relations, forming a coherent multi-partite graph, as illustrated in Section 3.2 and Figure 3.

Intuitively, we can use some lightweight NLP tools [4] to extract sentences from a text chunk, and each sentence can serve as a knowledge unit. However, this approach overlooks sentence context, leading to contextual ambiguity – semantically similar sentences from different chunks may actually have distinct meanings. In Figure 2, for example, if we use an NLP tool to extract sentences from the two chunks, the results may have high cosine similarity (0.91) despite differing in meaning. To address this, we leverage LLMs to incorporate context when extracting knowledge units. As a result, the LLM-extracted units in the same example show much lower similarity (0.51), effectively capturing context-sensitive distinctions. However, directly using LLMs to extract the units for all text chunks is very time and token-consuming.

In this paper, we propose a novel hybrid extraction strategy that combines the advantages of powerful LLMs and lightweight NLP tools [4]. An ideal strategy would involve an oracle model that can identify which knowledge unit in a chunk may exhibit contextual ambiguity with units in other chunks, and then use an LLM to resolve the ambiguous unit, while relying on NLP tools for unambiguous units. However, such an oracle model does not exist in practice. In light of this, we propose to use semantic similarity between chunks as a proxy to evaluate potential contextual ambiguity, since knowledge units in highly similar chunks are intuitively more prone to ambiguity. Specifically, we assign chunks having high semantic similarity with others for LLMs processing, while the remainder use NLP tools. To balance token cost and extraction accuracy, we formulate the chunk assignment as a 0-1 knapsack problem [31], aiming to maximize the total semantic similarity of the chunks selected for LLM processing within a given token budget. As shown in our later experiments, this strategy matches the performance of end-to-end LLM-based unit extraction, but only uses 50% of the tokens. Remarkably, Clue-RAG matches or outperforms baselines even without using an LLM for indexing.

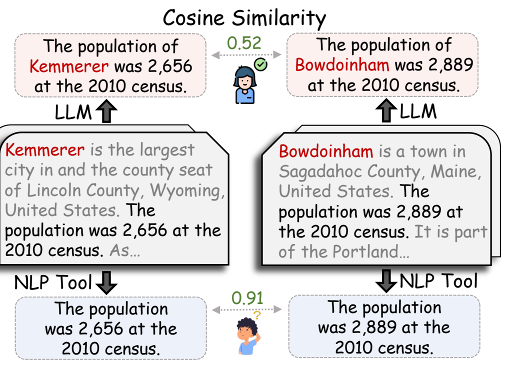
Cosine Similarity
The population of Kemmerer was 2,656 at the 2010 census.
0.52
The population of Bowdoinham was 2,889 at the 2010 census.
LLM
LLM
Kemmerer is the largest city in and the county seat of Lincoln County, Wyoming, United States. The population was 2,656 at the 2010 census. As...
Bowdoinham is a town in Sagadahoc County, Maine, United States. The population was 2,889 at the 2010 census. It is part of the Portland...
NLP Tool
NLP Tool
The population was 2,656 at the 2010 census.
0.91
The population was 2,889 at the 2010 census.

Figure 2: Comparing the knowledge units extracted by NLP tool and LLM.

To address **Limitation 2**, we propose Query-driven iterative retrieval (Q-Iter), an iterative retrieval strategy that alternately traverses knowledge units and entities. The process begins by extracting entities from the query to activate related entities, while concurrently using semantic search to anchor associated knowledge units. These units, in turn, expand the entity set by introducing entities referenced in their content. The strategy then performs constrained multihop traversal over the multi-partite graph to retrieve more relevant units. To ensure alignment with the query, each candidate unit is scored using a lightweight re-ranker specialized in query-context relevance. Finally, the collected knowledge units are mapped to their originating text chunks, which are subsequently re-ranked to yield the final context for answer generation.

In our experiments, we test 13 solutions across three datasets, focusing on two key aspects: (1) *Cost efficiency*, based on token usage during offline indexing and online retrieval; and (2) *QA performance*, measured by F1 score and Accuracy. Remarkably, Clue-RAG demonstrates significant improvements over the SOTA baselines, delivering up to 99.33% higher Accuracy and 113.51% greater F1 score while simultaneously reducing indexing costs by 72.58%. Besides, the zero-token variant of Clue-RAG achieves comparable or superior performance to baselines. This highlights Clue-RAG's superior graph indexing and inherent retrieval capabilities.

In summary, we make the following principal contributions:

- We propose a novel multi-partite graph index that integrates text chunks, knowledge units, and entities to enable semantically coherent cross-granularity retrieval, coupled with a hybrid extraction strategy for knowledge units that strategically balances the benefits of LLM processing against token usage.
- We propose `Q-Iter`, an iterative retrieval strategy that incrementally expands relevant knowledge units through semantic search and constrained graph traversal, significantly improving RAG performance.
- We conduct extensive experiments and demonstrate the superior performance of Clue-RAG over baselines.

Outline. We provide the preliminaries and problem formulation in Section 2. Section 3.1 presents an overview of Clue-RAG, Section 3.2 details the construction of the offline index `Clue-Index`, and Section 3.3 describes the online retrieval algorithm `Q-Iter`. The experimental results are then reported in Section 4. We review related work in Section 5 and conclude the paper in Section 6.

2 Preliminary

Let $\mathcal{T}$ be a text corpora whose element $t_i \in \mathcal{T}$ is a text chunk whose token length is at most L. The embedding of each text chunk t_i is denoted by $\boldsymbol{\phi}(t_i)$. We denote a graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges connecting pairs of nodes. In addition, we use calligraphic uppercase letters (e.g., $\mathcal{S}$) to represent a set of elements. Table 1 provides a summary of frequently used notations.

Table 1: Notations and meanings.

Category	Notation	Meaning
Text Processing	$\mathcal{R}(\cdot,\cdot)$	The re-ranking function
	$\boldsymbol{\phi}(t_i)$	The embedding of text t_i
	$\oplus(\mathcal{S})$	Concatenation of texts in set $\mathcal{S}$
	$\mathcal{T}, t_i, L$	Text $t_i \in \mathcal{T}$ whose length is $\leq L$
Retrieval	M	Beam size M
	D	Search depth D
	N	Returned chunks size N
	K	Top-K relevant results K
	α	Token constraint coefficient α
Index		Edges set $\mathcal{E}_c \subseteq \mathcal{V}_\mathcal{T} \times \mathcal{V}_\mathcal{K}$
	$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Edges set $\mathcal{E}_e \subseteq \mathcal{V}_e \times \mathcal{V}_\mathcal{K}$
	$\mathcal{E} = \mathcal{E}_c \cup \mathcal{E}_e$	Entity nodes set $\mathcal{V}_e$
	$\mathcal{V} = \mathcal{V}_\mathcal{T} \cup \mathcal{V}_\mathcal{K} \cup \mathcal{V}_e$	Chunk nodes set $\mathcal{V}_\mathcal{T}$
		Knowledge unit nodes set $\mathcal{V}_\mathcal{K}$

Problem 1 (Graph-based RAG). *Given an LLM, a text corpora $\mathcal{T}$, and a set of questions Q, find the relevant information for Q from the KG built on $\mathcal{T}$, which can be used to augment Q so that the LLM can generate high-quality answers for Q.*

3 Our Proposed Approach: Clue-RAG

In this section, we develop a novel graph-based RAG, called Clue-RAG, which addresses the two major limitations in existing methods by manipulating *text Chunk*, *knowLedge Unit*, and *Entity*. Specifically, we design a multi-partite graph index that supports semantically coherent retrieval across multiple levels of granularity, coupled with a hybrid strategy for knowledge unit extraction that balances LLM utility with computational cost. Besides, we propose `Q-Iter`, an iterative retrieval strategy that improves retrieval accuracy. We first provide an overview of Clue-RAG in Section 3.1 and then describe its two core modules in Sections 3.2 and 3.3.

3.1 Overview

At a high level, Clue-RAG consists of two modules: offline indexing `Clue-Index` and online retrieval `Q-Iter`. As shown in Figure 3, the offline `Clue-Index` has two stages:

- **Stage-1: hybrid extraction.** Given a text corpora $\mathcal{T}$ and a token constraint coefficient $\alpha \in [0, 1]$, a subset of core text chunks $\mathcal{T}_c \subseteq \mathcal{T}$ is selected based on relevance metrics. For each $t \in \mathcal{T}_c$, an LLM is employed to extract disambiguated knowledge units. For $t \in \mathcal{T} \setminus \mathcal{T}_c$, we use an NLP tool to segment the text into sentence-level units. Finally, entities are extracted from knowledge units using an NLP tool.
- **Stage-2: graph construction.** We construct a multi-partite graph index $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, including three types of nodes: chunk, knowledge unit, and entity. In $\mathcal{G}$, edges only connect nodes of different types: specifically, chunk nodes are linked to their corresponding knowledge unit nodes, and knowledge unit nodes are further connected to the entities they reference.

During the online retrieval, `Q-Iter` begins by extracting entities from the input query q to activate related entities, while semantic search anchors initial knowledge units. These units expand the entity set via referenced entities, enabling constrained multihop traversal over $\mathcal{G}$ to retrieve additional relevant units. The resulting units are mapped to their source chunks and re-ranked to form the final context for answer generation.

```
Algorithm 1: Clue-Index ($\mathcal{T}, \alpha$)
Input: Text corpora $\mathcal{T} = \{t_1, \ldots, t_n\}$, constraint $\alpha \in [0, 1]$
Output: A multi-partite graph index $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
$\mathcal{T}_c \leftarrow$ Chunk-Selection ($\mathcal{T}, \alpha$) in Algorithm 2;
$\mathcal{V}_\mathcal{T} \leftarrow \{t_i \mid t_i \in \mathcal{T}\}$, $\mathcal{V}_\mathcal{K} \leftarrow \emptyset$, $\mathcal{V}_e \leftarrow \emptyset$;
$\mathcal{E}_c \leftarrow \emptyset$, $\mathcal{E}_e \leftarrow \emptyset$;
foreach $t_i \in \mathcal{V}_\mathcal{T}$ do
    if $t_i \in \mathcal{T}_c$ then
        $\mathcal{K}_i \leftarrow$ extracted by LLM from $t_i$;
    else
        $\mathcal{K}_i \leftarrow$ split by NLP tool from $t_i$;
    $\mathcal{V}_\mathcal{K} \leftarrow \mathcal{V}_\mathcal{K} \cup \mathcal{K}_i$; $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup \{(t_i, k) \mid k \in \mathcal{K}_i\}$;
foreach $k \in \mathcal{V}_\mathcal{K}$ do
    $\mathcal{V}_k \leftarrow$ NER($k$); $\mathcal{V}_e \leftarrow \mathcal{V}_e \cup \mathcal{V}_k$;
    $\mathcal{E}_e \leftarrow \mathcal{E}_e \cup \{(v, k) \mid v \in \mathcal{V}_k\}$;
return $\mathcal{G} = (\mathcal{V}_\mathcal{T} \cup \mathcal{V}_\mathcal{K} \cup \mathcal{V}_e, \mathcal{E}_c \cup \mathcal{E}_e)$;
```

3.2 Offline index: `Clue-Index`

We illustrate the two key stages of indexing as follows.

Stage-1: hybrid extraction. As shown in Algorithm 1, the hybrid extraction process begins by selecting a subset of core text chunks

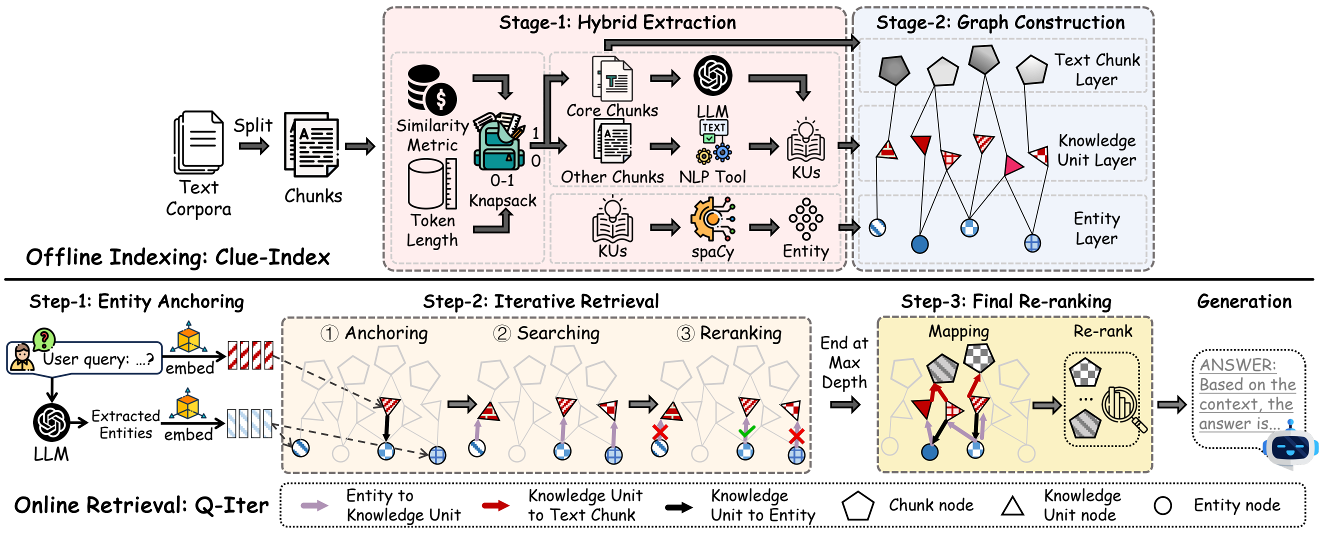

Figure 3: Clue-RAG consists of two phases: offline indexing and online retrieval.

Algorithm 2: Chunk-Selection $(\mathcal{T}, \alpha)$

```
1  Input: Text corpora $\mathcal{T} = \{t_1, \ldots, t_n\}$, constraint $\alpha \in [0, 1]$
2  Output: a core subset of text chunks $\mathcal{T}_c \in \mathcal{T}$
3  $W_{\text{total}} \leftarrow 0$, $\mathcal{S} \leftarrow \emptyset$;
4  foreach $t_i \in \mathcal{T}$ do
5      $w_i \leftarrow \text{TokenLength}(t_i)$;
6      $v_i \leftarrow \text{BLEU}(t_i, \mathcal{T} \setminus \{t_i\})$;        ▷ Standard BLEU calculation
7      $W_{\text{total}} \leftarrow W_{\text{total}} + w_i$; $\mathcal{S} \leftarrow \mathcal{S} \cup \{(w_i, v_i)\}$;
8  $W_{\max} \leftarrow \lceil \alpha \cdot W_{\text{total}} \rceil$;
9  $\mathcal{T}_c \leftarrow \text{Knapsack}(\mathcal{S}, W_{\max})$;
10 return $\mathcal{T}_c$;
```

$\mathcal{T}_c \subseteq \mathcal{T}$ using Algorithm 2, based on a token constraint coefficient $\alpha \in [0, 1]$. In Algorithm 2, we formulate the selection stage as a 0-1 knapsack problem, where each chunk's relevance score determines its assigned value (line 5) and its token length corresponds to its weight (line 6), with the goal of selecting subsets to maximize total relevance under the token budget (line 9) — a classic optimization paradigm identical to 0-1 knapsack problem [31].

In Algorithm 1, once $\mathcal{T}_c$ is identified (line 3), each text chunk t_i is processed according to its selection status (lines 6–11). For $t_i \in \mathcal{T}_c$, an LLM extracts disambiguated knowledge units $\mathcal{K}_i$ that capture core semantics. For $t_i \in \mathcal{T} \setminus \mathcal{T}_c$, a lightweight NLP tool [4] is used to segment the text into sentences as knowledge units $\mathcal{K}_i$ to ensure broad coverage at lower computational cost. All text chunks and extracted knowledge units are instantiated as text nodes $\mathcal{V}_\mathcal{T}$ and knowledge unit nodes $\mathcal{V}_\mathcal{K}$, respectively, forming the upper and intermediate layers of the multi-partite graph. Finally, the Named Entity Recognition (NER) component in spaCy [16] is utilized to extract entities $\mathcal{V}_k$ for each knowledge unit k (lines 12-13). All these extracted entities $\mathcal{V}_k$ are collected and form entity nodes $\mathcal{V}_e$, acting as anchors for precise information retrieval.

Stage-2: graph construction. As shown in Figure 3, Clue-Index is a multi-partite graph $\mathcal{G}$ built from a text corpora $\mathcal{T}$, organizing information across three semantic levels. In the upper layer, chunk nodes $\mathcal{V}_\mathcal{T}$ represent coarse-grained content by encoding text chunks. The intermediate layer contains knowledge unit nodes $\mathcal{V}_\mathcal{K}$, which capture semantically richer information distilled from the text. In the lower layer, entity nodes $\mathcal{V}_e$ encode fine-grained factual elements extracted from knowledge units. The graph is then connected by two types of edges: $\mathcal{E}_c$, linking each chunk node to its corresponding knowledge units, and $\mathcal{E}_e$, connecting knowledge units to the entities they mention. We present the construction algorithm in Algorithm 1.

3.3 Online retrieval algorithm: Q-Iter

Given a query q and a multi-partite graph index $\mathcal{G}$, our retrieval algorithm Q-Iter, outlined in Algorithm 3, identifies relevant information of q from the corpora. It has four input parameters, i.e., the number of most relevant results K, maximum search depth D, beam size M per depth level, and returned chunks size N. Q-Iter proceeds through three carefully designed steps as follows:
Step-1: entity anchoring. The goal is to identify query-relevant seed nodes $\mathcal{V}_q^{(0)}$. Inspired by cognitive theories of spreading activation [1, 6], we first extract entities from the input query using an LLM and identify their top-K most similar entity nodes in $\mathcal{G}$ to form the initial query set $\mathcal{V}_q^{(0)}$. In parallel, we retrieve the top-K knowledge units that are most semantically similar to the query and expand $\mathcal{V}_q^{(0)}$ by including all entities referenced in these units. This dual expansion incorporates both directly matched entities and entities introduced via semantically aligned knowledge units, enhancing the coverage and specificity of subsequent multihop retrieval. We refer to this combined process of Spreading Activation and Knowledge Anchoring as Entity Anchoring shown in Algorithm 4, which yields the initial seed nodes $\mathcal{V}_q^{(0)}$.
Step-2: iterative retrieval. The key idea is to progressively gather highly query-relevant and non-redundant knowledge units at each

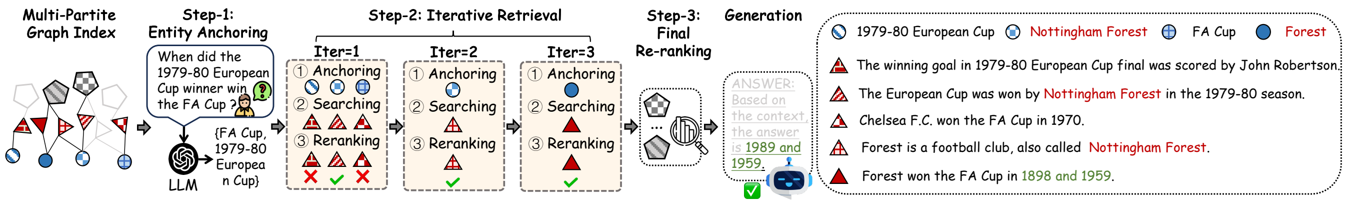
Multi-Partite Graph Index
Step-1: Entity Anchoring
When did the 1979-80 European Cup winner win the FA Cup ?
LLM
{FA Cup, 1979-80 European Cup}
Step-2: Iterative Retrieval
Iter=1
① Anchoring
② Searching
③ Reranking
Iter=2
① Anchoring
② Searching
③ Reranking
Iter=3
① Anchoring
② Searching
③ Reranking
Step-3: Final Re-ranking
Generation
ANSWER: Based on the context, the answer is 1989 and 1959.
1979-80 European Cup
Nottingham Forest
FA Cup
Forest
The winning goal in 1979-80 European Cup final was scored by John Robertson.
The European Cup was won by Nottingham Forest in the 1979-80 season.
Chelsea F.C. won the FA Cup in 1970.
Forest is a football club, also called Nottingham Forest.
Forest won the FA Cup in 1898 and 1959.

Figure 4: A toy example illustrating the online retrieval.

Algorithm 3: `Q-Iter`$(q, \mathcal{G}, K, D, M, N)$

```
1  Input: $q$, $\mathcal{G} = (\mathcal{V}_\mathcal{T} \cup \mathcal{V}_\mathcal{K} \cup \mathcal{V}_e, \mathcal{E}_c \cup \mathcal{E}_e)$, $K, D, M, N$
2  Output: top-$N$ relevant text chunks $\mathcal{T}^*$
3  $\mathcal{V}_q^{(0)} \leftarrow$ Entity Anchoring$(q, K, \mathcal{G})$ in Algorithm 4;
4  # Iterative Retrieval
5  $Q^{(0)} \leftarrow \{(v, \phi(q), \emptyset) \mid v \in \mathcal{V}_q^{(0)}\}$ ; $C^* \leftarrow \emptyset$;
6  foreach $d = 1, \ldots, D$ do
7      $C \leftarrow \emptyset$;
8      foreach $(v, \phi_v, \mathcal{S}_v) \in Q^{(d-1)}$ do
9          $\mathcal{K}_v \leftarrow \underset{k \in \mathcal{V}_\mathcal{K} \setminus \mathcal{S}_v \wedge (v,k) \in \mathcal{E}_e \wedge |\mathcal{K}_v| = K}{\arg\max} \cos(\phi(k), \phi_v)$;
10         foreach $k \in \mathcal{K}_v$ do
11             $\phi'_v \leftarrow \phi_v - \phi(k)$;                ▷ Update query embedding
12             $\mathcal{S}'_v \leftarrow \text{sort}(\mathcal{S}_v \cup \{k\})$;          ▷ Dictionary sorting
13             $score \leftarrow \mathcal{R}(q, \oplus(\mathcal{S}'_v))$;                ▷ Re-rank
14             $\mathcal{V}' \leftarrow \{v \mid (v, k) \in \mathcal{E}_e\}$;
15             $C \leftarrow C \cup \{(\mathcal{V}', \phi'_v, \mathcal{S}'_v, score)\}$;
16     $C' \leftarrow \underset{(\mathcal{V},\phi,\mathcal{S},score) \in C \wedge |C'| = M}{\arg\max} score$;          ▷ Pruning
17     $C^* \leftarrow C^* \cup C'$;
18     $Q^{(d)} \leftarrow \bigcup_{(\mathcal{V},\phi,\mathcal{S},score) \in C'} \{(v, \phi, \mathcal{S}) \mid v \in \mathcal{V}\}$;
19 # Final Re-ranking
20 $\mathcal{T}' \leftarrow \emptyset$;
21 foreach $(\mathcal{V}, \phi, \mathcal{S}, score) \in C^*$ do
22     $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{t \mid k \in \mathcal{S} \wedge (t, k) \in \mathcal{E}_c\}$;
23 $\mathcal{T}^* \leftarrow \underset{t \in \mathcal{T}' \wedge |\mathcal{T}^*| = N}{\arg\max} \mathcal{R}(q, t)$;
24 return $\mathcal{T}^*$;
```

depth. Given the seed nodes $\mathcal{V}_q^{(0)}$, Algorithm 3 first initializes $Q^{(0)}$ with tuples $(v, \phi(q), \emptyset)$ for each anchor $v \in \mathcal{V}_q^{(0)}$ (line 5), where $\phi(q)$ is the original query embedding that guides the semantic search and $\emptyset$ will accumulate the retrieved knowledge units. For each depth d from 1 to D, it processes the query queue $Q^{(d-1)}$ (lines 6-18). For each $(v, \phi_v, \mathcal{S}_v) \in Q^{(d-1)}$, it retrieves top-$K$ new knowledge units $\mathcal{K}_v \subseteq \mathcal{V}_\mathcal{K} \setminus \mathcal{S}_v$ using ϕ_v, ensuring relevance while avoiding redundancy (lines 8-9). For each $k \in \mathcal{K}_v$, it first updates the query embedding by subtracting k's embedding (lines 10-15), dynamically shifting focus to uncovered information and avoiding redundancy (This mechanism is termed Query Updating, as demonstrated in Experiment 4.4.). Next, the algorithm adds k to $\mathcal{S}_v$, sorts it canonically, then uses re-ranker $\mathcal{R}$ to score $\oplus(\mathcal{S}'_v)$ for coherence with q. Unlike semantic similarity measures, $\mathcal{R}$ specializes in query-context relevance scoring, allowing a more accurate assessment of knowledge units' relevance to q. Then, the algorithm: (1) finds adjacent entities $\mathcal{V}'$ via k's edges as next-depth anchors, and (2) updates C with the new state tuple and its re-ranked *score*. Finally, the algorithm prunes C to the top-M tuples C' based on their re-ranked *scores*, merges them into C^*, and initializes $Q^{(d)}$ for the next iteration (lines 16-18), where the re-ranker $\mathcal{R}$ guarantees C^* contain maximal evidential support for answering q while maintaining computational efficiency.

Step-3: final re-ranking. The key idea is to retrieve the most query-relevant text chunks from C^* via re-ranking for augmented generation. The algorithm first retrieves text nodes $\mathcal{T}'$ citing $\mathcal{S}$ via $\mathcal{E}_c$, and then selects top-N by their re-ranked *scores* to form final chunks $\mathcal{T}^*$ for augmented generation (lines 19-23).

Example 3.1. Given a query q = "When did the 1979-80 European Cup winner win the FA Cup?", then `Q-Iter` executes the three steps:

Step-1 (entity anchoring): The LLM first extracts entities $\mathcal{V}_q^{(0)}$ = {*1979-80 European Cup, FA Cup*} from q. Simultaneously, it retrieves the most semantically similar knowledge unit *"The European Cup was won by Nottingham Forest in the 1979-80 season"* using the query embedding, expanding $\mathcal{V}_q^{(0)}$ to {*1979-80 European Cup, FA Cup, Nottingham Forest*}.

Step-2 (iterative retrieval): In the first iteration, the anchoring stage initializes with the seed nodes from Step 1. For each entity in $\mathcal{V}_q^{(0)}$, the algorithm retrieves the top-K ($K = 1$) semantically similar knowledge unit linked to it via the query embedding. For instance, *FA Cup* retrieves *Chelsea F.C. won the FA Cup in 1970.* After each iteration, Step 2 applies re-ranker $\mathcal{R}$ to score knowledge units against q. In this toy example, only the top-M ($M = 1$) relevant units are retained per iteration. The process terminates at maximum depth $D = 3$. The complete retrieval path originates from *Nottingham Forest* to *Forest won the FA Cup in 1898 and 1959.*

Step-3 (final re-ranking): The algorithm maps the retrieved knowledge units back to their source text chunks, then re-ranks them based on q, and finally uses the most relevant chunks for RAG.

4 Experiments

In this section, we assess the effectiveness of Clue-RAG by addressing the following research questions:

- **Q1**: How does Clue-RAG compare to existing baselines in terms of QA performance and cost efficiency?
- **Q2**: How does our hybrid extraction strategy compare to alternatives and full LLM-based extraction?
- **Q3**: What are the contributions of each step of `Q-Iter`?
- **Q4**: How sensitive is Clue-RAG to hyperparameter settings?

4.1 Experimental Setup

Datasets. We evaluate Clue-RAG on three multihop QA benchmarks: MuSiQue [35], HotpotQA [42], and 2WikiMultiHopQA (2Wiki

Algorithm 4: `Entity Anchoring`$(q, \mathcal{G}, K)$

```
**Input:** $q$, $K$, $\mathcal{G} = (\mathcal{V}_\mathcal{T} \cup \mathcal{V}_\mathcal{K} \cup \mathcal{V}_e, \mathcal{E}_c \cup \mathcal{E}_e)$
**Output:** initial seed nodes $\mathcal{V}_q^{(0)}$
# Spreading Activation
$\mathcal{V}_q \leftarrow \text{NER}(q)$; $\mathcal{V}_q^{(0)} \leftarrow \emptyset$;
**foreach** $v \in \mathcal{V}_q$ **do**
    $\mathcal{N}_v \leftarrow \underset{v' \in \mathcal{V}_e \wedge |\mathcal{N}_v| = K}{\arg\max} \cos(\phi(v'), \phi(v))$;
    $\mathcal{V}_q^{(0)} \leftarrow \mathcal{V}_q^{(0)} \cup \mathcal{N}_v$;
# Knowledge Anchoring
$\mathcal{K}_q \leftarrow \underset{k \in \mathcal{V}_\mathcal{K} \wedge |\mathcal{K}_q| = K}{\arg\max} \cos(\phi(k), \phi(q))$;
$\mathcal{V}_q^{(0)} \leftarrow \mathcal{V}_q^{(0)} \cup \bigcup_{k \in \mathcal{K}_q} \{v \mid (v, k) \in \mathcal{E}_e\}$;
**return** $\mathcal{V}_q^{(0)}$
```

Below) [15]. Following prior work [14, 29, 34, 45], we use 1,000 validation questions per dataset, with corresponding paragraphs preprocessed into $\mathcal{T}$. See Appendix Table 5 for statistics.

Metrics. We evaluate each solution's performance in terms of QA performance and cost efficiency. For QA performance, we report Accuracy (Acc. below) and F1 score (F1 below), where the former one measures whether the golden answers are included in the generation answer rather than requiring exact match [2, 22], and the latter one is computed via token-level overlap between golden and generated answers, balancing precision and recall to assess both answer completeness and correctness [14, 45]. For cost efficiency, we quantify two metrics: total token expenditure for offline indexing and average token consumption per online query q.

Baselines. We test the following 13 solutions in 4 categories.

- **Simple baselines:** We consider 2 straightforward approaches: (1) the Zero-shot method, which directly applies an LLM for QA without any retrieval, and (2) VanillaRAG, which retrieves relevant passages through query embedding similarity before using these passages as context for generation.
- **Graph-based baselines:** We evaluate 6 graph-based baselines that construct text-attribute KG using LLMs for content retrieval: HippoRAG [14], KETRAG [19], and local search of GraphRAG (LGraphRAG) [8], and 3 versions of LightRAG (LLightRAG for local search, GLightRAG for global search, and HLightRAG for hybrid search) [13], excluding GraphRAG's global version as it targets abstract QA rather than multi-hop QA.
- **Tree-based baselines:** We evaluate 2 tree-based baselines: RAPTOR [32] and SIRERAG [45], both constructing multi-level trees for passage organization and retrieval.
- **Our proposed solutions:** Our approach supports three operational modes, determined by a hyperparameter $\alpha \in [0, 1]$, which controls the proportion of tokens in $\mathcal{T}$ processed by LLMs during offline indexing. Clue-RAG-0.0 ($\alpha = 0.0$) demonstrates the strong performance of our LLM-free graph index and retrieval strategy. Clue-RAG-0.5 ($\alpha = 0.5$) highlights the effectiveness of our hybrid extraction strategy, achieving strong performance while reducing 50% indexing cost. Clue-RAG-1.0 ($\alpha = 1.0$) represents the full-capacity configuration, where all text chunks are processed by LLMs, enabling the system to achieve SOTA performance.

Settings. All experiments are conducted on a Linux machine with an Intel Xeon 2.0 GHz CPU, 1024 GB of memory, and 8 NVIDIA GeForce RTX A5000 GPUs (each with 24 GB of memory), capable of running both LLaMA3.0-8B [12] and Qwen2.5-32B [3] LLM models. To ensure a fair comparison, all methods are implemented under a unified framework [49] using their default configurations. Additional experimental details are provided in Appendix A.1.

4.2 Overall performance evaluation (Q1)

In the first set of experiments, we compare Clue-RAG against ten existing solutions in terms of QA performance and cost efficiency. As shown in Table 2, Clue-RAG consistently achieves superior QA performance across multiple benchmark datasets when evaluated with different LLMs, with Clue-RAG-1.0 establishing SOTA performance in both average F1 and Acc. Specifically, it outperforms the strong baseline KETRAG by 21.53% in average F1, while also surpassing another competitive baseline SIRERAG by 5.75% in average Acc. Besides, Clue-RAG-0.5 demonstrates competitive QA performance relative to KETRAG, and achieves improvements of 19.29%/27.19% in average F1/Acc. Notably, Clue-RAG-0.0, which entirely avoids LLM usage in indexing, still matches or exceeds the capabilities of two strong baselines, achieving a 13.01%/52.50% average F1 improvement over KETRAG and SIRERAG, respectively. Beyond outperforming these two competitive baselines, Clue-RAG demonstrates superior QA performance across all evaluated datasets compared to the other eight alternatives. This consistent advantage highlights the effectiveness of its innovative indexing and retrieval strategies in multi-hop QA scenery.

Meanwhile, compared to KETRAG, Clue-RAG-0.5 reduces token costs by up to 9.41%/83.87% during offline indexing and online retrieval, as shown in Figure 5 and 6. Remarkably, during offline indexing, Clue-RAG-0.0 reduces token costs by 100% compared to KETRAG and SIRERAG. During online retrieval, it reduces up to 82.92%/56.06% token costs compared to these two strong baselines. Hence, Clue-RAG is token-cost efficient.

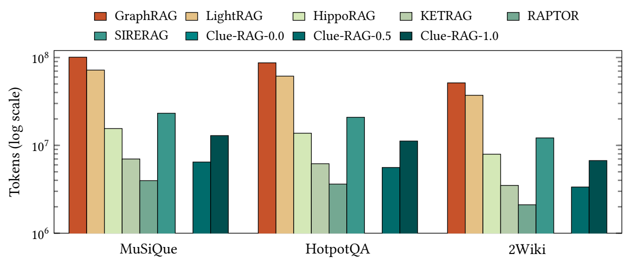

Figure 5: Token cost in offline indexing using LLaMA3.0-8B (Clue-RAG-0.0 requires no LLM).

4.3 Effectiveness of hybrid extraction (Q2)

In this experiment, we evaluate the effectiveness of our hybrid extraction strategy for knowledge units, as described in Section 3.2. Table 3 reports the QA performance of four methods, each constrained to use only 50% of the total token budget for LLM processing ($\alpha = 0.5$). The methods differ in their chunk selection strategies: RANDOM (random selection), COS (our `Chunk-Selection` using cosine similarity), and BLEU (our `Chunk-Selection` using BLEU score). For direct comparison, we also include the performance of Clue-RAG-1.0 (BLEU), which uses 100% of the token budget. The results show that both COS and BLEU significantly outperform

Table 2: Overall performance of RAG solutions. The best and second-best results among the ten competitor solutions (excluding the Clue-RAG series) in each column are highlighted in bold and underlined, respectively.

	LLaMA3.0-8B						Qwen2.5-32B							
	MuSiQue		HotpotQA		2Wiki		MuSiQue		HotpotQA		2Wiki		Avg	
Method	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.
Simple Baselines														
Zero-shot	7.14	5.11	24.34	25.30	18.88	33.78	9.62	5.71	25.51	25.90	26.44	27.10	18.66	20.48
Vanilla	12.28	14.21	40.62	52.50	18.22	34.00	20.97	21.02	53.55	60.50	30.94	44.80	29.43	37.84
Graph-Structured Baselines														
LGraphRAG	9.23	14.21	24.81	43.90	15.11	42.20	13.12	19.12	34.04	53.20	21.57	46.20	19.65	36.47
LLightRAG	7.94	6.91	25.43	36.30	17.35	33.60	13.82	18.52	34.09	49.10	22.53	48.40	20.19	32.14
GLightRAG	5.34	5.14	17.80	27.40	11.82	23.20	9.31	11.51	21.69	34.30	8.09	35.70	12.34	22.88
HLightRAG	7.87	9.41	25.82	37.40	13.16	28.40	14.57	22.62	35.89	55.60	19.16	52.30	19.41	34.29
HippoRAG	11.58	11.81	39.63	44.40	24.94	<u>48.20</u>	15.00	20.42	40.36	52.20	26.95	53.70	26.41	38.46
KETRAG	<u>16.25</u>	12.51	<u>48.75</u>	47.10	<u>27.90</u>	41.40	21.08	16.52	<u>61.94</u>	57.70	<u>50.02</u>	49.90	<u>37.66</u>	37.52
Tree-Structured Baselines														
RAPTOR	10.58	16.22	34.18	<u>56.50</u>	16.92	43.30	17.31	21.12	47.09	63.70	28.88	47.20	25.83	41.34
SIRERAG	11.99	<u>21.22</u>	34.02	55.70	18.34	43.90	<u>22.46</u>	<u>31.93</u>	46.75	**65.10**	33.88	<u>57.10</u>	27.91	<u>45.83</u>
Clue-RAG-0.0	24.10	21.02	53.18	57.20	32.19	46.60	34.26	31.33	62.14	63.50	49.47	54.70	42.56	45.73
Clue-RAG-0.5	**25.60**	22.12	55.80	**59.50**	35.52	48.90	**36.68**	**32.93**	63.44	64.10	52.49	58.80	44.92	47.73
Clue-RAG-1.0	25.48	**22.32**	**55.97**	59.20	**37.41**	**50.70**	36.50	32.83	**64.02**	<u>64.70</u>	**55.20**	**61.00**	**45.76**	**48.46**

53.04 59.10

Table 3: Performance comparison of Clue-RAG-0.5 using different core chunk selection strategies. The table highlights the best and second-best results in each column with bold and underlined formatting, respectively.

	LLaMA3.0-8B						Qwen2.5-32B							
	MuSiQue		HotpotQA		2Wiki		MuSiQue		HotpotQA		2Wiki		Avg	
Method	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.
Clue-RAG-0.5 (RANDOM)	<u>25.18</u>	21.32	54.44	57.60	34.00	46.90	<u>36.61</u>	**33.93**	61.59	63.40	49.49	54.70	43.55	46.31
Clue-RAG-0.5 (COS)	24.64	<u>21.42</u>	<u>55.80</u>	<u>59.10</u>	<u>35.18</u>	**49.10**	36.19	32.13	**63.65**	**64.90**	**52.83**	**59.10**	<u>44.72</u>	<u>47.63</u>
Clue-RAG-0.5 (BLEU)	**25.60**	**22.12**	<u>55.80</u>	**59.50**	**35.52**	<u>48.90</u>	**36.68**	<u>32.93</u>	<u>63.44</u>	<u>64.10</u>	<u>52.49</u>	<u>58.80</u>	**44.92**	**47.73**
Clue-RAG-1.0 (BLEU)	25.48	22.32	55.97	59.20	37.41	50.70	36.50	32.83	64.02	64.70	55.20	61.00	45.76	48.46

RANDOM on average F1 and Acc. Particularly, the BLEU-based strategy achieves the highest performance, exceeding RANDOM and COS by 3.14% and 0.46% in average F1, and by 3.05% and 0.21% in average Acc, respectively. These gains underscore the effectiveness of our hybrid extraction approach in the offline indexing stage.

While Clue-RAG-1.0 (BLEU) yields stronger results, Clue-RAG-0.5 (BLEU) achieves 94.95%–100.51% of its performance, with half of the token usage. An interesting phenomenon emerges on the MuSiQue dataset with LLaMA3.0-8B and Qwen2.5-32B, where Clue-RAG-0.5 (BLEU) occasionally outperforms its full-token counterpart. This suggests that certain questions in MuSiQue exhibit a stronger lexical alignment with original sentences than with LLM-extracted knowledge units, as the paraphrasing process may substitute critical terms with less relevant alternatives. Overall, these results demonstrate that hybrid extraction strategy can maintain robust QA performance while significantly reducing the token cost.

4.4 Ablation study (Q3)

In this experiment, we evaluate the key components of Q-Iter, i.e., Spreading Activation (S-A below) and Knowledge Anchoring

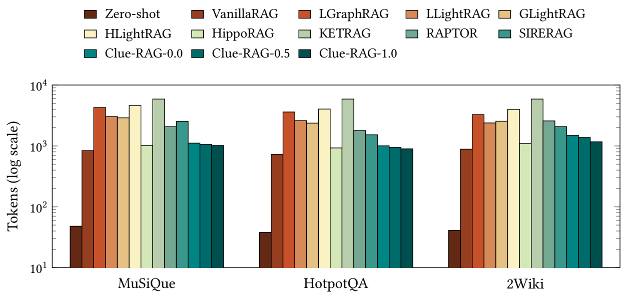

Figure 6: Token cost in online retrieval using LLaMA3.0-8B.

(K-A below) of Step-1, and the Query Updating (Q-U below) of Step-2. Table 4 shows the performance of removing each component. We observe that S-A enhances the average F1 and Acc. by 9.5% and 9.95%, respectively, while K-A provides improvement of 5.34% and 4.83%, and Q-U improves the average F1 and Acc. by 1.63% and 1.32%. The superior performance of S-A can be attributed to its alignment in semantic granularity: The entities extracted from the query by the LLM are matched against entity nodes in the multi-partite graph, ensuring consistency. In contrast, K-A relies

Table 4: Ablation study of Clue-RAG-1.0, evaluating the contributions of three key components (K-A, S-A, and Q-U). The best performance for each metric is highlighted in bold, while the second-best result is underlined.

	LLaMA3.0-8B						Qwen2.5-32B							
	MuSiQue		HotpotQA		2Wiki		MuSiQue		HotpotQA		2Wiki		Avg	
Method	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.
`Clue-RAG-1.0`	**25.48**	**22.32**	**55.97**	**59.20**	37.41	50.70	**36.50**	**32.83**	**64.02**	**64.70**	55.20	61.00	**45.76**	**48.46**
`w/o K-A`	25.26	22.02	52.93	55.50	36.75	50.10	32.64	29.13	60.50	62.00	52.58	58.60	43.44	46.23
`w/o S-A`	23.18	19.52	53.48	55.70	34.76	45.60	31.64	28.03	59.12	59.90	48.57	55.70	41.79	44.08
`w/o Q-U`	24.51	20.72	54.51	56.80	**38.00**	**51.50**	35.36	32.03	62.52	64.10	**55.27**	**61.80**	45.03	47.83
`w/o Q-U or K-A`	22.53	18.82	52.00	53.90	36.52	49.50	31.93	28.33	60.14	62.00	52.72	58.90	42.64	45.24
`w/o Q-U or S-A`	21.57	17.42	51.22	53.50	33.43	46.10	31.84	27.93	56.71	58.30	48.34	55.80	40.52	43.18

on semantic embeddings of the query to align with knowledge unit nodes in the graph, which may introduce noise due to potential inconsistencies in semantic granularity. Q-U improves F1 by up to 3.96% and Acc by 7.72% on MuSiQue, and by 2.68% and 4.23% on HotpotQA, but slightly degrades on 2Wiki. The performance decline on 2Wiki stems from its frequent semantically similar entities (e.g., family relations or homonyms like Elizabeth I/II), which challenge disambiguation. Nevertheless, they collectively enhance the retrieval performance of Clue-RAG.

4.5 Parameter sensitivity (Q4)

In this experiment, we analyze the sensitivity of Clue-RAG using Qwen2.5-32B w.r.t. key hyperparameters: the number K of top-K retrieved results, beam size M, search depth D, and token constraint coefficient α, where $K \in \{3, 5, 7, 9\}$, $M \in \{3, 5, 7, 9\}$, $D \in \{1, 2, 3, 4\}$, and $\alpha \in \{0, 0.25, 0.5, 0.75, 1.0\}$.

First, as shown in Figure 7 (a), Clue-RAG does not exhibit a consistent improvement in generation quality with increasing K across all three datasets. Instead, the results suggest that $K = 3$ provides sufficient retrieval results, while larger values introduce noise without meaningful gains. Next, Figure 7 (b) reveals that generation quality generally improves with larger beam sizes M. However, the gains plateau beyond $M = 5$, and a slight decline occurs at $M = 7, 9$. This indicates that $M = 5$ strikes an optimal balance between generation quality and computational efficiency, as excessively large beam sizes may introduce noise with diminishing returns. Furthermore, Figure 7 (c) demonstrates that deeper search depth D leads to better performance, which aligns with the nature of the datasets, where most questions require 2 hops while fewer necessitate 3-4 hops. Nevertheless, when $D = 4$, generation quality slightly deteriorates, likely due to over-retrieval of irrelevant information for most 2-hops questions. Finally, Figure 7 (d) shows a positive correlation between the token constraint coefficient α and generation quality. This is expected because higher α values allow the LLM to process more chunks, thereby improving the quality of knowledge units and enhancing retrieval accuracy. Consistent results are observed across other datasets in Appendix Figure 8.

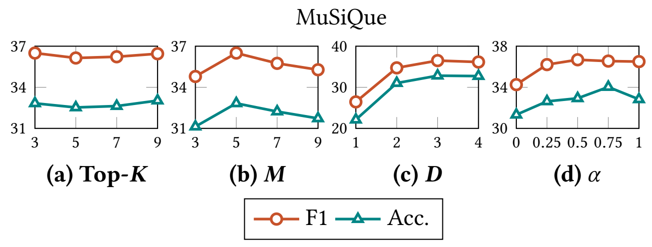

Figure 7: Performance analysis under different hyperparameter settings on MuSiQue dataset.

5 Related Works

In this section, we review the representative RAG methods based on vector database (VDB) [26, 47], trees, and graphs. For more details, please see recent surveys [9, 28, 46] and empirical study [49].

• **VDB-based RAG.** Vallina RAG is a basic RAG solution that (1) splits the corpora into text chunks, (2) encodes them into embeddings via an embedding model, and (3) stores embeddings in a VDB. During online retrieval, the same model embeds the query and then retrieves top-K relevant text chunks for augmented generation.

• **Tree-based RAG.** RAPTOR [32] introduces a tree-based index, which can be built in a bottom-up tree manner. Specifically, it begins with raw text chunks as leaf nodes, clusters their embeddings via Gaussian Mixture Models [23], then recursively summarizes clusters using LLMs and re-clusters them to build the tree bottom-up. During online retrieval, all tree nodes are indexed in a VDB, enabling a fast semantic similarity search with query embeddings. Building on RAPTOR, SIRERAG [45] introduces a dual tree framework, which first extracts propositions and entities from texts using LLMs, regrouping propositions linked to the same entity into new passages, and then organize these passages into a relatedness tree alongside the RAPTOR similarity tree, with both trees jointly indexed in the same vector database for retrieval. This hybrid approach aims to balance semantic similarity and relatedness coherence. EraRAG [44] is a novel hierarchical tree-based RAG method that focuses on the dynamic scenario, allowing for efficient and incremental index updates as the corpus evolves.

• **Graph-based RAG.** Compared to trees, graphs are more effective for modeling complicated relationships. Microsoft's GraphRAG [8] first constructs a text-attributed KG by extracting entities, relationships, and other detailed contextual features from text fragments through LLMs. Then, it employs Leiden algorithm [33] to cluster the KG into some hierarchical clusters, where each cluster is associated with a community report. For retrieval, it uses the local search to retrieve relevant context from entities, relationships, community reports, and text chunks to augment response.

ArchRAG [36] organizes attributed communities hierarchically, and augments the question using summaries of attributed communities. LightRAG [13] constructs a KG from text chunks while using LLMs to augment both entities and relations with extracted keywords and leverages LLMs to extract query-relevant low-/high-level keywords for retrieval. KETRAG [19] first selects core text chunks from a KNN graph built using both semantic and lexical similarity, and then constructs a KG. It supports online retrieval by using the standard local search but extracting ego networks to improve LLM generation. HippoRAG [14] constructs a KG by extracting triplets from the text chunks using LLMs and enhances its quality by linking similar entities via semantic embeddings. For multi-hop reasoning, it employs PPR to retrieve relevant KG entities and augments the response with entities' associated text chunks for LLM generation.

6 Conclusion

In this work, we propose Clue-RAG, a graph-based RAG approach that features a multi-partite graph index integrating chunks, knowledge units, and entities. To efficiently build this index, we introduce a hybrid extraction strategy for knowledge unit that maximizes LLM processing benefits while minimizing token usage. For online retrieval, we design a query-driven iterative retrieval (`Q-Iter`) that ensures relevant retrieval results. These designs enhance graph quality and retrieval relevance while reducing LLM token consumption. Experimental results on three QA benchmarks demonstrate that Clue-RAG significantly outperforms SOTA baselines in both QA performance and cost efficiency. Notably, its zero-token variant achieves comparable or superior performance, highlighting the robustness and effectiveness of our approach. In the future, we will extend our Clue-RAG for processing multimodal data.

References

[1] John R Anderson. 1983. A spreading activation theory of memory. *Journal of verbal learning and verbal behavior* 22, 3 (1983), 261–295.

[2] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

[3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[4] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc. https://www.nltk.org/book/

[5] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. arXiv:2402.03216 [cs.CL]

[6] Allan M Collins and Elizabeth F Loftus. 1975. A spreading-activation theory of semantic processing. *Psychological review* 82, 6 (1975), 407.

[7] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

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

[9] 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.

[10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2, 1 (2023).

[11] Aashish Ghimire, James Pather, and John Edwards. 2024. Generative AI in education: A study of Educators' awareness, sentiments, and influencing factors. In *2024 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[13] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. (2024).

[14] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[15] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060* (2020).

[16] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python. (2020). doi:10.5281/zenodo.1212303

[17] 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).

[18] Yizheng Huang and Jimmy Huang. 2024. A survey on retrieval-augmented text generation for large language models. *arXiv preprint arXiv:2404.10981* (2024).

[19] 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).

[20] 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).

[21] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).

[22] 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).

[23] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[24] 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).

[25] OpenAI. [n. d.]. tiktoken. https://github.com/openai/tiktoken

[26] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[28] 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).

[29] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350* (2022).

[30] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics* 11 (2023), 1316–1331.

[31] Sartaj Sahni. 1975. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)* 22, 1 (1975), 115–124.

[32] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.

[33] 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.

[34] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509* (2022).

[35] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics* 10 (2022), 539–554.

[36] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. 2025. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation. arXiv:2502.09891 [cs.IR] https://arxiv.org/abs/2502.09891

[37] 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).

[38] 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).

[39] 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).

[40] Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2025. When to use Graphs in RAG: A Comprehensive Analysis for Graph Retrieval-Augmented Generation. *arXiv preprint arXiv:2506.05690* (2025).

[41] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[42] 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).

[43] Hao Yu, Aoran Gan, Kai Zhang, Shiwei Tong, Qi Liu, and Zhaofeng Liu. 2024. Evaluation of retrieval-augmented generation: A survey. In *CCF Conference on Big Data*. Springer, 102–120.

[44] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. 2025. EraRAG: Efficient and Incremental Retrieval Augmented Generation for Growing Corpora. arXiv:2506.20963 [cs.IR] https://arxiv.org/abs/2506.20963

[45] 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).

[46] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Junnan Dong, Hao Chen, Yi Chang, and Xiao Huang. 2025. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. *arXiv preprint arXiv:2501.13958* (2025).

[47] Yunan Zhang, Shige Liu, and Jianguo Wang. 2024. Are there fundamental limitations in supporting vector data management in relational databases? A case study of PostgreSQL. *2024 IEEE 40th International Conference on Data Engineering (ICDE)* (2024), 3640–3653.

[48] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. 2024. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473* (2024).

[49] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. *arXiv preprint arXiv:2503.04338* (2025).

A Supplementary Details

A.1 Experimental Settings

For token cost comparison between offline indexing and online retrieval using LLaMA3.0-8B shown in Figure 5 and 6, we account for both prompt and completion tokens. For methods requiring LLM-based query preprocessing (e.g., keyword or entity extraction), these additional token costs are included in our calculations. During the LLM generation phase, we employ a strategy by duplicating the input query to the end of the original prompt. This design explicitly mitigates the "lost in the middle" effect [21] observed in approaches like LightRAG, where placing the question before lengthy context can cause the model to overlook the task objective [21, 30]. Furthermore, inspired by the prompt designs of SIRERAG [45] and KETRAG [19], which include phrases like "Answer this question as short as possible" to prevent verbose or irrelevant outputs, we uniformly add this instruction to the final prompt of each method. This ensures concise responses, maintaining evaluation fairness and highlighting our method's effectiveness. For other hyperparameters of each method, we follow the original settings in their available codes. For NLP pre-processing, we employ the following tools: cl100k_base from tiktoken library [25] for word tokenization and counting, NLTK's sent_tokenize [4] for sentence segmentation, and spaCy [16] for named entity recognition. For LLM inference, we use LLaMA3.0-8B [12] and Qwen2.5-32B [3], with a maximum token limit of 8,000. In top-K selection tasks, we set $K = 5$ to accommodate token constraints. The embedding model for vector search is BGE-M3 [5], while re-ranking is performed using the lightweight yet powerful BGE-Reranker-v2-M3 [5]. As for our proposed approach, Clue-RAG employs the following default configurations: the number of most relevant results K is set to 3, search depth D to 3, beam size M per depth to 5, and returned candidate chunks N to 5, ensuring that all methods ultimately retrieve the same number of query-relevant chunks (i.e., 5) for fair comparison. By default, we set the value of α to 1 if it is not explicitly specified. Additionally, BLEU score [27] serves as the core selection metric for relevance scoring.

A.2 Dataset Details

Table 5 presents the key statistics for the three benchmark datasets used in our study: MuSiQue, HotpotQA, and 2Wiki, including corpora size, number of questions, and total token count. Table 6 reports the node counts for the Clue-RAG-1.0 graph index, detailing the quantities of text nodes, knowledge unit nodes, and entity nodes for each dataset. Figure 8 presents the performance under different hyperparameter settings on HotpotQA and 2Wiki datasets.

Table 5: Details of three benchmark datasets.

Dataset	MuSiQue	HotpotQA	2Wiki
Questions	1,000	1,000	1,000
Passages	11,656	9,221	6,119
Tokens	1,281,422	1,239,838	640,205

Table 6: Statistics of Clue-RAG-1.0's graph index in LLaMA3.0-8B and Qwen2.5-32B.

Model	Nodes	MuSiQue	HotpotQA	2Wiki
LLaMA 3.0-8B	Texts	11,656	9,221	6,119
	Knowledge	108,513	100,789	52,149
	Entities	222,940	209,869	123,116
Qwen 2.5-32B	Texts	11,656	9,221	6,119
	Knowledge	114,256	108,957	57,442
	Entities	238,954	229,762	136,966

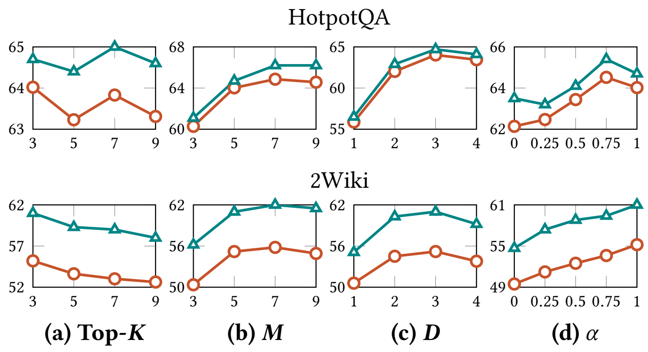

Figure 8: Performance analysis under different hyperparameter settings on HotpotQA and 2Wiki datasets.

A.3 Prompt Templates

Figure 9 and 10 illustrate the LLM instruction prompts used for extracting knowledge units and answer generations, respectively.

Prompt for generating knowledge units

Prompt:
Decompose the content into clear knowledge units, ensuring they are interpretable independently of their original context:

(1) Sentence Simplification: Break compound sentences into simpler, individual sentences. Whenever possible, retain the original phrasing from the input text.
(2) Entity and Description Separation: For named entities that are accompanied by descriptive information, separate the descriptive details into a distinct knowledge unit. Ensure each knowledge unit represents a single, clear fact.
(3) Pronoun Resolution: Replace all pronouns (e.g., "it", "they", "this") with explicit references, using full taxonomic names or clear identifiers. Always use "[entity]'s [property]" construction.
(4) Output Format: Present the resulting knowledge units as a list of strings, formatted in JSON.

EXAMPLE-1:
Input:
Jesúfas Aranguren. His 13-year professional career was solely associated with Athletic Bilbao, with which he played in nearly 400 official games, winning two Copa del Rey trophies.

```
Output:
{
    "knowledge units": [
        "Jesús Aranguren had a 13-year professional career.",
        "Jesús Aranguren's professional career was solely associated with Athletic Bilbao.",
        "Athletic Bilbao is a football club.",
        "Jesús Aranguren played for Athletic Bilbao in nearly 400 official games.",
        "Jesús Aranguren won two Copa del Rey trophies with Athletic Bilbao.",
    ]
}
```

EXAMPLE-2:
Input:
Ophrys apifera. Ophrys apifera grows to a height of 15 – 50 centimetres (6 – 20 in). This hardy orchid develops small rosettes of leaves in autumn. They continue to grow slowly during winter. Basal leaves are ovate or oblong - lanceolate, upper leaves and bracts are ovate - lanceolate and sheathing. The plant blooms from mid-April to July producing a spike composed from one to twelve flowers. The flowers have large sepals, with a central green rib and their colour varies from white to pink, while petals are short, pubescent, yellow to greenish. The labellum is trilobed, with two pronounced humps on the hairy lateral lobes, the median lobe is hairy and similar to the abdomen of a bee. It is quite variable in the pattern of coloration, but usually brownish - red with yellow markings. The gynostegium is at right angles, with an elongated apex.

```
Output:
{
    "knowledge units": [
        "Ophrys apifera grows to a height of 15-50 centimetres (6-20 in)",
        "Ophrys apifera is a hardy orchid",
        "Ophrys apifera develops small rosettes of leaves in autumn",
        "The leaves of Ophrys apifera continue to grow slowly during winter",
        "The basal leaves of Ophrys apifera are ovate or oblong-lanceolate",
        "The upper leaves and bracts of Ophrys apifera are ovate-lanceolate and sheathing",
        "Ophrys apifera blooms from mid-April to July",
        "Ophrys apifera produces a spike composed of one to twelve flowers",
        "The flowers of Ophrys apifera have large sepals with a central green rib",
        "The flowers of Ophrys apifera vary in colour from white to pink",
        "The petals of Ophrys apifera are short, pubescent, and yellow to greenish",
        "The labellum of Ophrys apifera is trilobed with two pronounced humps on the hairy lateral lobes",
        "The median lobe of Ophrys apifera's labellum is hairy and resembles a bee's abdomen",
       "The coloration pattern of Ophrys apifera is variable but usually brownish-red with yellow markings",
        "The gynostegium of Ophrys apifera is at right angles with an elongated apex" ,
    ]
}
```

JUST OUTPUT THE RESULTS IN JSON FORMAT!
Input: {passage}
Output:

Figure 9: The prompt for generating knowledge units.

```
Prompt for answer generation

Prompt:
Your goal is to give the best full answer to question the user input according to the given context below.
Given Context: {context}
Give the best full answer to question :{question}
Answer this question in as fewer number of words as possible.
```

Figure 10: The prompt for generating answer.