ZoomRAG: Hierarchical Random-walk Zooming across Multi-scale Information Graphs for Fast and Accurate RAG

Xianming Hu[1], **Jingyang Chen**[2], **Bin Tang**[1], **Yihe Liu**[1], **Yihong Huang**[1]
Hongbo Zhao[1], **Nuoyi Chen**[2], **Jie Zhang**[2], **Ping Li**[3], **Kai Zhang**[1,†]

[1]School of Computer Science and Technology, East China Normal University
[2]Institute of Science and Technology for Brain-inspired Intelligence, Fudan University
[3]School of Computer Science and Software Engineering, Southwest Petroleum University
Correspondence: 52285901020@stu.ecnu.edu.cn, kzhang@cs.ecnu.edu.cn

Abstract

Retrieval-Augmented Generation is a powerful tool for NLP applications. Yet, it is challenging to encode large knowledge bases as compact offline structures while simultaneously achieving accurate, low-latency online retrieval. We propose **ZoomRAG**, a coarse-to-fine, hierarchical graph inference method to tackle the challenges. ZoomRAG formulates the retrieval task as random walks across multi-scale relational graphs. *At the coarse level*, it constructs a global relational graph and performs a query-initiated random walk to quickly locate a few relevant documents over the entire corpus. *At the finer level*, it "zooms into" the selected documents to capture fine-grained semantic and temporal relations, and conducts a second random walk to pinpoint salient evidence chunks for generation. This coarse-to-fine strategy substantially reduces offline indexing costs and accelerates online retrieval. Moreover, random-walk based topological reasoning over rich, multi-scale relational structures enables ZoomRAG to effectively aggregate multi-hop evidence while suppressing noise. Finally, we address the difficulty of handling concurrent RAG queries by **algorithm-parallel ZoomRAG**. Overall, ZoomRAG avoids building expensive knowledge graphs while achieving 2.2% – 4.9% absolute gains in accuracy over SOTA RAG models, with an average online retrieval latency per-query as low as 0.019 secs by processing hundreds of queries concurrently.

1 Introduction

Retrieval-Augmented Generation (RAG) can effectively enhance question-answering capabilities of LLMs (Lewis et al., 2020; Guu et al., 2020; Gao et al., 2023; Chen et al., 2024a; Xia et al., 2025) by accessing external knowledge without requiring additional fine-tuning (Siriwardhana et al., 2023). By decoupling information retrieval from answer generation, RAG produces more reliable responses. A typical RAG pipeline involves segmenting corpus into chunks, building an index to retrieve relevant information, and feeding into an LLM to generate answers (Wang et al., 2024a; Chen et al., 2024b).

Early RAG frameworks relied on similarity-based matching, which is effective for simple queries (Lewis et al., 2020; Guu et al., 2020), but may ignore structural relationships in complex scenarios. Then various structural modeling frameworks are introduced, such as **tree-based approaches** (e.g., RAPTOR (Sarthi et al., 2024), SiReRAG (Zhang et al., 2025a)) that recursively summarize text to capture thematic associations within documents, and **knowledge-graph** based methods (Han et al., 2025; Zhu et al., 2024; Peng et al., 2024; Gutiérrez et al., 2024) that focus more on extracting entities and their relationships to support factual reasoning (see more in related work).

Despite these advances, challenges remain in RAG research. (1) Constructing compact offline indices for large knowledge bases is difficult. For example, high-quality knowledge graphs can be highly costly and resource demanding, and their entity–relation–entity triples may introduce spurious connections due to limited textual grounding (Zhuang et al., 2026). Recent lightweight models effectively improve the computational efficiency, but may affect structural expressiveness in complex reasoning scenarios (Li et al., 2025a; Zhuang et al., 2026). (2) Objects in a knowledge base, such as entities, documents and sentences are interconnected with complex high-order dependencies, making it difficult to identify the right inference pathway for reliable multi-hop reasoning. (3) Handling RAG queries in parallel received little attention, despite the practical need for high-throughput systems.

To address these challenges, we propose ZoomRAG, a hierarchical, graph-based reasoning approach for RAG tasks. ZoomRAG formulates the retrieval task as a random-walk running across

†Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 32840–32859

multi-scale relational graphs to "zoom into" relevant information adaptively and efficiently. ZoomRAG operates at two complementary levels of granularity. **At the coarse level (Doc-Zoom)**, it constructs a global tripartite graph that encodes semantic and structural connections among documents, entities, and the query. A query-initiated random walk traverses this graph to efficiently identify the top-k most relevant documents over the entire corpus. **At the fine level (Chunk-Zoom)**, it builds a local bipartite graph capturing fine-grained sentence–entity relations within the selected documents, and performs a second random walk to further narrow the search space and pinpoint salient evidence chunks for generation.

The coarse-to-fine strategy in ZoomRAG substantially improves both offline efficiency and online retrieval performance. Moreover, the flexible topological reasoning of random walks allows ZoomRAG to integrate rich global structure with fine-grained contextual information, thereby suppressing noise and supporting robust evidence aggregation. Finally, we introduce an algorithm-parallel variant to improve system throughput while incurring minimal memory overhead, thereby supporting hundreds of concurrent queries efficiently.

ZoomRAG has several important advantages:

Resource-efficient offline graph construction. ZoomRAG only requires simple entity recognition to build multi-scale information graphs as the off-line index, which is highly resource efficient and enhances scalability of RAG systems. *Empirically, ZoomRAG is up to 440× faster and 6× more memory-efficient than SOTA models requiring knowledge graph construction* (Table 3).

Time-efficient, parallelizable on-line retrieval. ZoomRAG leverages random-walks across multi-scale information graphs to quickly locate relevant objects in a coarse-to-fine manner. Besides, we devised an algorithm-parallel version to greatly improve system throughput. *Empirically, ZoomRAG can handle hundreds of queries in parallel with average latency as low as 0.019 seconds, about 6 – 147× faster than SOTA models* (Table 3).

Effective multi-hop topological reasoning. ZoomRAG effectively captures high-order dependencies by utilizing rich local/global topological contexts across multi-scale information graphs. The restarted random-walk allows dynamically refining search scope while avoiding noise accumulation in multi-step queries. *Empirically, ZoomRAG consistently yields 2.2% – 4.9% absolute gains in EM and F1 over recent SOTA models* (Table 1).

Our contributions are summarized as follows.

— We propose to encode knowledge base as multi-scale graphs, and to employ restarted random walks to systematically exploit local/global graph topology for robust and accurate retrieval.

— We introduce a coarse-to-fine retrieval strategy to locate relevant pieces of information, together with a parallel version to significantly improve the online retrieval efficiency.

— We conduct comprehensive evaluations on three benchmark datasets against several state-of-the-art approaches, and show improvements in both online/offline efficiency and generative accuracy.

2 Related work

RAG systems can effectively improve question answering and reasoning by incorporating an external knowledge sources (Zhao et al., 2024; Gupta et al., 2025; Jin et al., 2024). Early attempts relied on dense retrieval, which represents queries and chunks as dense vectors and computes semantic similarity via inner products (Fan et al., 2024; Ram et al., 2023; Xu et al., 2023; Zhang et al., 2023; Sharma et al., 2024). However, relationships between items in the database may be ignored, making complex, multi-step queries still a challenge.

To address these limitations, recent research integrated tree- and graph-based structures to enhance the reasoning capabilities for complex queries.

Tree-based methods like RAPTOR (Sarthi et al., 2024) builds a tree bottom-up by recursively embedding, clustering, and summarizing text chunks; SiReRAG (Zhang et al., 2025a) introduces a dual-tree structure to model similarity and relatedness separately; CFT-RAG (Li et al., 2025b) organizes entities in a hierarchical tree. While effective for single-document, tree structures are inherently hierarchical and less suited for capturing global, cross-document relationships in large corpora.

Graph-based methods explicitly model relations for cross-document reasoning (Cai et al., 2025; Xu et al., 2024; Chen et al., 2025a). For instance, GraphRAG (Edge et al., 2025) builds the graph to organize multi-level information through summarization, and HippoRAG 2 (Gutiérrez et al., 2025) constructs an entity-guided KG for deep knowledge integration. HopRAG (Liu et al., 2025b) constructs passage-level graphs via pseudo-queries to simulate reasoning paths. Recently, LightRAG (Guo et al., 2024) proposes a simplified KG structure

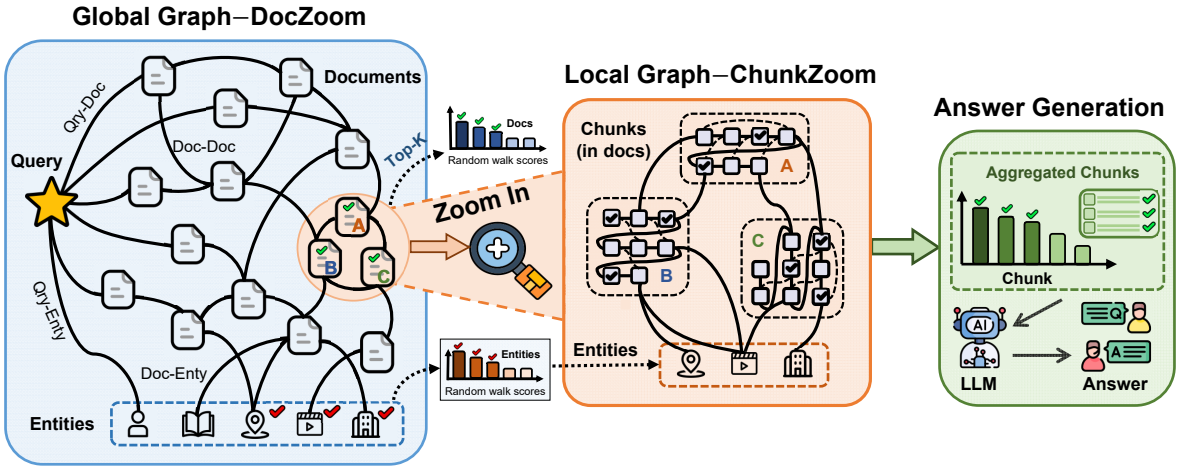

Figure 1: The coarse-to-fine graph construction and online-retrieval workflow of ZoomRAG.

with dual-level retrieval for handling both specific entities as well as broader topics for comprehensive knowledge discovery. LinearRAG (Zhuang et al., 2026) exploits heterogeneous sentence-entity and passage-entity graphs to achieve linear online and offline complexity regarding database size.

3 Method

ZoomRAG works across two information granularities to hierarchically locate relevant information:

(1) Doc-Zoom: In the coarse level, a global doc-entity-query graph is constructed so that a random-walk starting from the query can quickly locate top-k most relevant documents (and entities) based on rich structural/semantic connections;

(2) Chunk-Zoom: In the finer level, a local chunk-entity graph encoding delicate temporal, structural and semantic relations within the top-K documents is used to start a second random-walk to pinpoint useful evidence chunks for generation.

3.1 Doc-Zoom: coarse-level doc discovery

The goal of DocZoom is to quickly identify the most relevant documents over the entire corpus. To achieve this, we construct a global, coarse-grained information Graph $\mathbf{G}_{global}$ as follows.

Definition of nodes: $\mathbf{G}_{global}$ has three node types:

- *Query node* q,
- *Document nodes* $\{p_1, p_2, \ldots, p_n\}$,
- *Entity nodes (unique)* $\{e_1, e_2, \ldots, e_m\}$.

Definition of edges: $\mathbf{G}_{global}$ has altogether four types of links (structural and semantic):

- *Doc-doc relation*: the similarity between two documents is defined by the normalized inner product of their Jina-embeddings (Sturua et al., 2024).
- *Doc-query relation*: the similarity between a query and a document is also computed by normalized inner product of their Jina-embeddings.
- *Doc-Entity relation*: if entity e_j appears in document p_i, their link is 1, otherwise link is 0.
- *Query-Entity relation*: if query q contains entity e_i, their link is defined as 1; otherwise link is 0.

Given one query, n documents and m entities, their relationships can be aggregated as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{(1+n+m)\times(1+n+m)}$, as below

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{A}_{\text{doc}\to\text{qry}} & \mathbf{A}_{\text{enty}\to\text{qry}} \\ \mathbf{A}_{\text{doc}\to\text{qry}}^{\top} & \mathbf{A}_{\text{doc}\to\text{doc}} & \mathbf{A}_{\text{enty}\to\text{doc}} \\ \mathbf{A}_{\text{enty}\to\text{qry}}^{\top} & \mathbf{A}_{\text{enty}\to\text{doc}}^{\top} & 0 \end{bmatrix} \quad (1)$$

Here, $\mathbf{A}$ is naturally partitioned into nine blocks, and each sub-block is formally defined as

$$\begin{cases} \mathbf{A}_{\text{doc}\to\text{qry}}(i,q) = \exp(\cos(\mathbf{q}, \mathbf{p}_i)), \\ \mathbf{A}_{\text{doc}\to\text{doc}}(i_1,i_2) = \exp(\cos(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})), \\ \mathbf{A}_{\text{enty}\to\text{qry}}(j,q) = \mathrm{I}_{[e_j \in q]}, \\ \mathbf{A}_{\text{enty}\to\text{doc}}(j,i) = \mathrm{I}_{[e_j \in p_i]}. \end{cases} \quad (2)$$

Here, q is the query index, $i, i_1, i_2 \in [1, n]$ are document indices, and $j \in [1, m]$ is entity index; I is an indicator function (true or false). Query has no self-loop and entity-nodes are not inter-connected. Both $\mathbf{A}_{\text{doc}\to\text{doc}}$ and $\mathbf{A}_{\text{doc}\to\text{qry}}$ are sparsified row-wise by retaining only entries beyond a threshold (mean plus three times the standard deviation per row) to suppress noise and reduce graph density.

On top of the coarse-level graph G_{global}, a random walk starting from the query node is used

to iteratively traverse the graph, with probability $\alpha \in [0, 1]$ to jump back to the query node

$$\mathbf{r}^{(t)} = (1 - \alpha) \cdot \widetilde{\mathbf{A}}\mathbf{r}^{(t-1)} + \alpha \cdot \mathbf{r}^{(0)}. \quad (3)$$

Here, the initial distribution $\mathbf{r}^{(0)}$ is a one-hot vector indicating the query node as the start node; $\widetilde{\mathbf{A}}$ denotes the column-normalized transition matrix, and jump-back mechanism prevents the random walk from drifting away from the query. After a predefined number of steps, the resulting distribution $\mathbf{r}^{(t)}$ can be interpreted as a *relevance ranking* over documents and entities, with the top-ranked items serving as inputs to the second-stage search.

3.2 Chunk-Zoom: fine-grained chunk filtering

After identifying the top-K most relevant documents via DocZoom, we further refine the retrieval granularity by narrowing the search space to these documents and building a fine-grained information graph G_{local} to locate relevant chunks in them.

Definition of nodes: $\mathbf{G}_{local}$ has two node types:

- *Chunk nodes* $\{c_1, c_2, \ldots, c_l\}$,
- *entity nodes* $\{e_1, e_2, \ldots, e_h\}$.

Definition of edges: $\mathbf{G}_{local}$ has two types of links (integrating temporal, structural and semantic):

- *Chunk-entity relation*: if an entity e_j appears in chunk c_i, their relation is assigned a value of 1; otherwise the link is empty (0).
- *Chunk-chunk relation*: chunk relations integrate both **temporal proximity and semantic correlations**. Temporally, proximity between two chunks (in the same document) is computed by a Gaussian decay w.r.t. their temporal distances. Semantically, similarity between two chunks is computed by the inner product of their embeddings.

Given l chunks and h entities within the top-K documents identified in the coarse level, their relations are encoded by the adjacency matrix

$$\mathbf{B}_{(l+h)\times(l+h)} = \begin{bmatrix} \mathbf{B}_{\text{chk}\to\text{chk}} & \mathbf{B}_{\text{chk}\to\text{enty}} \\ \mathbf{B}^{\top}_{\text{chk}\to\text{enty}} & 0 \end{bmatrix}, \quad (4)$$

whose blocks are formally defined as:

$$\begin{cases} \mathbf{B}_{\text{chk}\to\text{enty}}(i, j) = \mathrm{I}_{[e_j \in c_i]}, \\ \mathbf{B}_{\text{chk}\to\text{chk}} = \omega \cdot \mathrm{nm}(\mathbf{B}^{\text{t}}_{\text{c}\to\text{c}}) + (1 - \omega) \cdot \mathrm{nm}(\mathbf{B}^{\text{s}}_{\text{c}\to\text{c}}) \end{cases}$$

Here, $\mathrm{nm}(\cdot)$ is matrix column-wise normalization operator, $\mathbf{B}^{\text{t}}_{\text{c}\to\text{c}}$ and $\mathbf{B}^{\text{s}}_{\text{c}\to\text{c}}$ are the **temporal** and **semantic** relational matrix between chunks, whose ijth entry (over the ith and jth chunk) is defined as

$$\begin{aligned} \mathbf{B}^{\text{t}}_{\text{c}\to\text{c}}(i, j) &= \exp(-(i - j)^2/2\sigma^2), \\ \mathbf{B}^{\text{s}}_{\text{c}\to\text{c}}(i, j) &= \exp(\cos(\mathbf{c}_i, \mathbf{c}_j)), \end{aligned}$$

with σ controlling temporal decay and $\omega \in [0, 1]$ balancing temporal and semantic similarities. Entity node is not connected to itself or other entities.

The fine-grained local graph G_{local} encodes rich semantic, temporal, and structural dependencies within the retrieved documents. On top of it, we start a second random-walk as follows

$$\mathbf{y}^{(t)} = (1 - \beta) \cdot \widetilde{\mathbf{B}}\mathbf{y}^{(t-1)} + \beta \cdot \mathbf{y}^{(0)}. \quad (5)$$

Here, $\widetilde{\mathbf{B}}$ is the column-normalized transition matrix, and $\beta \in [0, 1]$ is the jump-back probability.

The initial score of the fine-level random walk, $\mathbf{y}^{(0)}$, is computed by inheriting the relevance scores of the entity nodes within the top-K documents from the coarse-level random walk (3), with all l chunk nodes set to zero. This setting guides the random walk to concentrate around query-relevant entities when diffusing toward informative chunks under semantic/temporal cues. In practice, we keep the top 70% chunks with the highest random-walk scores within the top-K retrieved documents.

The local graph $\mathbf{G}_{\text{local}}$ typically has a few tens of nodes, far smaller than $\mathbf{G}_{\text{global}}$ with tens of thousands of nodes, and thus incurs negligible overhead in overall online retrieval; see Appendix I for a detailed cost breakdown. Empirically, the stationary solution of the random walk on the local graph yields the best performance, which can be computed via a small matrix inverse (Appendix E).

The adjacency matrices $\mathbf{A}$ (1) and $\mathbf{B}$ (4) both have multiple sub-blocks, which are rescaled to have the same average weight. This normalization ensures the random walk has comparable probabilities of transitioning across different partitions.

3.3 Parallel ZoomRAG for Concurrent Queries

In this section, we consider *parallel RAG queries* in practical high-throughput systems. This setting typically consist of a large query-agnostic offline structure and small query-specific components. It defies standard data-parallel strategies, as replicating the offline structures across parallel workers is prohibitively memory expensive.

For example, data-parallel version of DocZoom requires storing the large $(m+n+1)\times(m+n+1)$ transition matrix (1) for each worker, leading to substantial memory waste. So we consider running

random walks in parallel on a *shared global graph* to maximize memory and computational efficiency. A new challenge in this setting is the *cross-query interference*, i.e., a random walk initialized from one query q may traverse other query q' sharing document/entity neighbors with q. Such interactions can entangle the random-walk dynamics across queries and contaminate the result of each (Appendix H).

We solve this by **algorithm-parallel DocZoom**. It decouples query-specific computations from graph components shared across queries, while fully leveraging batched matrix operations to achieve high efficiency and low memory waste.

We first work on a single query and then extend it to support k queries $\{q_i\}_{i=1}^{k}$ in parallel. For q_i, the global graph adjacency matrix $\mathbf{A}_i$ in Eq. (1) can be written as

$$\mathbf{A}_i = \begin{bmatrix} 0 & \mathbf{H}_i \\ \mathbf{H}_i^\top & \mathbf{M} \end{bmatrix}. \tag{6}$$

Here $\mathbf{M} \in \mathbb{R}^{(n+m)\times(n+m)}$ specifies documents-entity relations shared across all queries; $\mathbf{H}_i \in \mathbb{R}^{1\times(n+m)}$ signifies connections related to the ith query and the documents/entities.

Let $\mathbf{r}_i^{(t)} \in \mathbb{R}^{(n+m+1)\times 1}$ be the random-walk distribution for the ith query at step t, initialized as $\mathbf{r}_i^{(0)}$. Define $\mathbf{H}_{i,j}$ as the jth element of $\mathbf{H}_i$ and $\mathbf{M}_{(:,j)}$ as the jth column of $\mathbf{M}$. Then the random-walk can be equivalently calculated as follows:

$$\begin{cases} \mathbf{r}_i^{(t)}[1] = \sum_{j=1}^{n+m} \mathbf{H}_{i,j} \cdot z_{ij}^{(t-1)}, \\ \mathbf{r}_i^{(t)}[2{:}m{+}n{+}1] = \widetilde{\mathbf{H}}_i^\top \cdot \mathbf{r}_i^{(t-1)}[1] + \sum_{j=1}^{n+m} \mathbf{M}_{(:,j)} \cdot z_{ij}^{(t-1)} \end{cases}$$

where $z_{ij}^{(t-1)}$ is defined as

$$z_{ij}^{(t-1)} = \frac{\mathbf{r}_i^{(t-1)}[j+1]}{\mathbf{H}_{i,j} + \sum_{k=1}^{n+m} \mathbf{M}_{k,j}}. \tag{7}$$

Here, we have computed the first $[1]$ and remaining entries $[2{:}m{+}n{+}1]$ of $\mathbf{r}_i^{(t)}$ separately, corresponding to query-specific and query-agnostic distribution component for the ith query, respectively, to fit into a globally parallel version more conveniently.

With the iteration formula of $\mathbf{r}_i^{(t)}$ above, we further define $\mathbf{H}$ by concatenating all H_i's as

$$\mathbf{H}^\top = \begin{bmatrix} \mathbf{H}_1^\top & \mathbf{H}_2^\top & \dots & \mathbf{H}_k^\top \end{bmatrix}, \tag{8}$$

and an $(n+m)\times k$ matrix $\mathbf{E}^{(t)}$ with identical rows,

$$\mathbf{E}^{(t)} = \left.\begin{bmatrix} \mathbf{r}_1^{(t)}[1] & \mathbf{r}_2^{(t)}[1] & \dots & \mathbf{r}_k^{(t)}[1] \\ \dots & \dots & \dots & \dots \\ \mathbf{r}_1^{(t)}[1] & \mathbf{r}_2^{(t)}[1] & \dots & \mathbf{r}_k^{(t)}[1] \end{bmatrix}\right\} n{+}m \text{ rows.}$$

Then, the random-walk distribution $\mathbf{r}_i^{(t)}$ for all the k queries at step t, $\mathbf{R}^{(t)} = [\mathbf{r}_1^{(t)}, \mathbf{r}_2^{(t)}, \dots, \mathbf{r}_k^{(t)}]$, can be computed in parallel as:

$$\mathbf{R}^{(t)} = \begin{bmatrix} \mathbf{1}^\top \left(\mathbf{H} \odot \mathbf{Z}^{(t-1)}\right)^\top \\ \widetilde{\mathbf{H}}^\top \odot \mathbf{E}^{(t-1)} + (\mathbf{Z}^{(t-1)} \cdot \mathbf{M})^\top \end{bmatrix}, \tag{9}$$

with each column the distribution for a single query, $\mathbf{1} \in \mathbb{R}^{n+m}$ a vector of all ones, $\odot$ for element-wise product, $\widetilde{\mathbf{H}}$ the row-normalized version of $\mathbf{H}$, and $\mathbf{Z}^{(t)} \in \mathbb{R}^{k\times(n+m)}$ a matrix having entries $z_{ij}^{(t-1)}$'s.

For ease of exposition, we omit the restart mechanism in Eq. (3) in the above derivations. When the restart component is reinstated, the final random-walk distribution can be compactly expressed as

$$\mathbf{R}_{final}^{(t)} = (1-\alpha)\mathbf{R}^{(t)} + \alpha\mathbf{R}^{(0)},$$

which is used in all practical implementations.

Appendix A gives complexity analysis to show superior space and time trade-off of algorithm-parallel DocZoom than sequential and data-parallel versions. For ChunkZoom, it only involves a few relevant documents per query, and is easy to parallelize with concurrent CPU processes.

4 Comparisons with Related Work

In this section, we discuss important differences between ZoomRAG and existing RAG models.

Graph-indexing structures. Indexing structures are critical to RAG performance. The graph indexing structure of ZoomRAG has a number of unique designs and advantages, as discussed below.

(1) *ZoomRAG is knowledge-graph-free and only requires cheap NER in building the offline-index.*

Existing RAG systems such as GraphRAG (Edge et al., 2025), GFM-RAG (Luo et al., 2025), and HippoRAG2 (Gutiérrez et al., 2025) rely on explicit knowledge graph construction, which introduces substantial computational overhead and is highly resource-intensive. In contrast, ZoomRAG bypasses this expensive step by relying solely on named entity recognition (NER) over the corpus. This design makes ZoomRAG 1-2 orders of magnitudes faster and significantly more memory-efficient than RAG models requiring knowledge graph construction (Table 3).

Despite this simplification, ZoomRAG captures comprehensive multi-scale relations among heterogeneous entities, queries, chunks, and documents through its graph-based indexing structure. This enables effective topological reasoning via random

walks. Empirically, ZoomRAG achieves 2.2–4.9% higher EM/F1 and 1.1–5.6% higher evidence recall rate compared with RAG models relying on knowledge-graph construction (Table 1 and Table 2). These results highlight the potential of knowledge-free RAG paradigms, where carefully designed relational indexing structures show clear advantage over knowledge graphs.

(2) *ZoomRAG employs augmented bi-partite (tripartite) graphs as its indexing structure*.

The graph structures used in existing methods mainly focus on cross-partite relations like chunk–keyword relation in KET-RAG (Huang et al., 2025), entity–event relation in E^{2}RAG (Zhang et al., 2025b), and sentence–entity and passage–entity relation in LinearRAG (Zhuang et al., 2026). In contrast, ZoomRAG has also introduced pairwise relations within a single partition to enrich the relational landscape, such as semantic document relations in the global graph, and temporal/semantic chunk relations in the local graph. This enables more coherent information propagation with notable performance gains (Table 4).

(3) *ZoomRAG leverages fine-grained semantic–temporal relation in the indexing stage to offload part of the reasoning process from the LLM.*

To the best of our knowledge, existing RAG systems typically **ignore temporal dependencies among document chunks at the indexing stage**, and treat retrieval primarily as a semantic matching problem, including methods based on random walks (e.g., the HippoRAG series and LinearRAG). In contrast, ZoomRAG explicitly models both semantic and temporal relations (via ChunkZoom), enabling more fine-grained reasoning and navigation over evidence pieces during retrieval stage.

This design reshapes the boundary between retrieval and generation. Conventional RAG pipelines enforce a strict separation between evidence retrieval and reasoning, placing the full burden of multi-hop inference on the LLM. In contrast, ZoomRAG partially shifts this burden to the retrieval stage by embedding fine-grained semantic–temporal structures directly into the index. This effectively transforms retrieval into a form of preconditioned reasoning, enabling tighter coordination with generation and potentially reducing the number of interaction rounds required.

Other minor points discussed in Appendix G.

Parallel query processing. Most recent RAG systems based on structured indexing perform online retrieval sequentially (in their released codes) (Gutiérrez et al., 2024, 2025; Zhuang et al., 2026). Algorithm-parallel or high-concurrency RAG is quite rare and challenging because: (1) methods using trees or knowledge-graphs rely on iterative path traversal whose parallelization often requires specialized infrastructure; (2) page-rank-based approaches like HippoRAG (Gutiérrez et al., 2024) and LinearRAG (Zhuang et al., 2026) can be formulated as matrix operations, but naive data-parallel schemes may quickly exceed memory limits by having to replicate the indexing structure shared across different queries. In contrast, our algorithm-parallel strategy allows storing query-agnostic offline structures only once, and isolates query-conditioned data/computation from those shared across queries, leading to high throughput and low memory overhead. More discussions in parallel RAG are in Appendix D.

Coarse-to-fine strategy. ZoomRAG adopts a coarse-to-fine strategy, with the coarse stage comprising document-level information propagation and fine stage restricted to chunk-level reasoning. This hierarchical design differs from flat retrieval structures in existing methods. Although LinearRAG also uses a two-stage pipeline, its first stage already operates on fine-grained sentence-level units across the entire corpus. In contrast, our approach defers fine-grained modeling to the 2nd stage for only a few selected documents, making our runtime 45%-85% lower than LinearRAG (offline and online) while delivering 8% gains in accuracy.

5 Experiments

Datasets. We use three widely used multi-hop QA benchmarks: 2WikiMultiHopQA (Ho et al., 2020), HotpotQA (Yang et al., 2018), and MuSiQue (Trivedi et al., 2022). Following standard settings in (Gutiérrez et al., 2024; Zhang et al., 2025a; Liu et al., 2025b), we used the same corpus and question set as in their study to ensure a fair comparison.

Baselines. (1) Zero-shot LLM: Llama-3-8B, Qwen2.5-7B, GPT-4o-mini. (2) Basic RAG: RAG using top-10 passages from semantic-based retrieval. (3) Tree-based methods: RAPTOR (Sarthi et al., 2024) and SiReRAG (Zhang et al., 2025a). (4) Graph-based methods: GraphRAG (Peng et al., 2024), HippoRAG (Gutiérrez et al., 2024), HippoRAG2 (Gutiérrez et al., 2025), HopRAG (Liu et al., 2025b), LightRAG (Li et al., 2025a), and LinearRAG (Zhuang et al., 2026). **All baselines rigorously reproduced by their official codes.**

Method	2Wiki		HotpotQA		MuSiQue		Average	
	EM	F1	EM	F1	EM	F1	EM	F1
Zero-shot LLM Inference								
Llama-3-8B	20.00	23.83	28.40	35.75	5.00	10.80	17.80	23.46
Qwen2.5-7B	15.40	18.47	21.40	28.92	2.60	7.69	13.13	18.36
GPT-4o mini	27.50	31.66	37.90	46.45	10.30	17.75	25.23	31.95
Basic version RAG								
NaiveRAG (Top-10)	37.90	41.28	49.90	61.54	21.40	29.75	36.40	44.19
Tree-based RAG Methods								
RAPTOR (ICLR 2024)	45.70	49.51	48.10	60.34	16.20	24.85	36.66	44.90
SiReRAG (ICLR 2025)	54.10	61.05	57.00	70.97	34.40	44.80	48.50	58.94
Graph-based RAG Methods								
GraphRAG (Arxiv 2024)	45.60	52.07	54.60	66.92	28.40	40.08	42.86	53.02
HippoRAG (NIPS 2024)	61.30	69.20	51.40	64.33	28.60	36.40	47.10	56.64
HopRAG (ACL 2025)	41.90	45.50	50.40	63.12	23.50	31.64	38.60	46.75
LinearRAG (ICLR 2026)	<u>63.30</u>	69.71	59.50	73.35	28.50	38.86	50.43	60.64
LightRAG (EMNLP 2025)	58.10	64.42	60.30	72.56	35.50	47.30	51.30	61.42
HippoRAG 2 (ICML 2025)	61.80	<u>70.65</u>	<u>62.30</u>	<u>74.43</u>	<u>37.70</u>	<u>49.96</u>	<u>53.93</u>	<u>65.01</u>
ZoomRAG (Ours)	**66.70**	**74.33**	**65.00**	**76.60**	**42.50**	**53.12**	**58.06**	**68.01**

Table 1: QA performance (EM and F1 scores) of 12 methods on three benchmark datasets. GPT-4o-mini used to generate answers for all methods in the evaluation. The 1st/2nd highest score marked by bold/underlines.

Evaluation metrics. We used three key metrics widely used in the literature (Gutiérrez et al., 2024; Zhang et al., 2025a): (1) Exact Match (EM), the percentage of predictions that exactly match ground truth after normalization; (2) F1 score, the harmonic mean of token-level precision and recall; and (3) Retrieval time per query (RTPQ), average time for generating the answer for each question.

All methods employ jina-embedding-v3 (Sturua et al., 2024) and a unified top-k retrieval setting (k=10). Following current practices (Gutiérrez et al., 2024; Zhuang et al., 2026; Guo et al., 2024), we used GPT-4o-mini for entity extraction and answer generation, with a simple COT-reasoning prompt for all methods. *This setup offers a practical cost–performance trade-off such that a rigorous, side-by-side comparison can be made for all competing methods reported*. Our codes ran on one NVIDIA A6000 GPU with 3.2GHz CPU.

5.1 Evaluation Results

Accuracy. Table 1 shows that ZoomRAG achieves strong performance on three multi-hop question answering tasks. Compared with zero-shot LLM inference, ZoomRAG yields substantial improvements in EM and F1, with absolute gains of up to 44.93% and 49.65%, respectively. Relative to NaiveRAG, ZoomRAG improves average F1 by 23.82%. When compared with tree-based methods, ZoomRAG achieves 9.07%–23.11% absolute gains in average F1. Even against graph-based retrieval approaches, including the current strongest baseline HippoRAG 2 and more recently proposed structured RAG methods such as LinearRAG, ZoomRAG shows promising results, with average F1 improvements ranging from 3% to 21.26%. These results show that the topological reasoning over multi-scale graphs allows ZoomRAG to effectively integrate global structural signals with fine-grained contextual information for robust multi-hop retrieval generation.

Evidence Recall. Table 2 reports the evidence retrieval recall rate of all competing methods, measuring the proportion of gold supporting evidence successfully retrieved across all questions. As shown, ZoomRAG consistently achieves the highest recall across all datasets, outperforming the strongest baseline, HippoRAG 2, by margins ranging from 1.09% to 5.55%. These gains are particularly notable considering that multi-stage retrieval pipelines are more prone to error accumulation: once relevant evidence is missed in the first stage, it cannot be recovered in subsequent steps. ZoomRAG's two-step retrieval strategy explicitly mitigates this issue by tightly coordinating coarse- and fine-grained retrieval to balance coverage and precision. Specifically, the coarse stage leverages global semantic and structural signals to ensure

Method	2Wiki	HotpotQA	MuSiQue	Average
Basic RAG				
NaiveRAG (Top-10)	49.22	71.11	40.59	53.64
Tree-based RAG Methods				
RAPTOR (ICLR 2024)	55.23	70.05	34.29	53.19
SiReRAG (ICLR 2025)	60.92	71.96	41.54	58.14
Graph-based RAG Methods				
GraphRAG (arXiv 2024)	58.69	74.27	35.34	56.10
HippoRAG (NeurIPS 2024)	91.63	84.15	56.30	77.36
LinearRAG (ICLR 2026)	92.81	95.76	50.94	79.83
LightRAG (EMNLP 2025)	74.10	92.88	64.90	77.29
HippoRAG 2 (ICML 2025)	93.83	97.95	83.35	91.71
ZoomRAG (Ours)	**94.92**	**99.15**	**88.90**	**94.32**

Table 2: Evidence retrieval recall of different RAG methods on three benchmark datasets.

Method	Offline Indexing Cost		Online Retrieval	
	Time (s) ↓	**Mem** (GB) ↓	**RTPQ** (s) ↓	**F1** (%) ↑
RAPTOR	5,446	4.68	1.41	44.90
SiReRAG	15,633	13.82	4.53	61.79
GraphRAG	10,630	**2.57**	15.35	53.02
HippoRAG	7,275	19.90	2.17	56.64
HippoRAG_2	3,343	16.70	2.80	65.01
LightRAG	153,474	5.97	6.90	61.42
LinearRAG	637	5.37	0.13	60.64
ZoomRAG	**346**	3.36	**0.019**	**68.01**

Table 3: **Off-line indexing cost** and **online retrieval performance** for 8 competing methods averaged across 3 datasets. Best/second-best results in bold/underline.

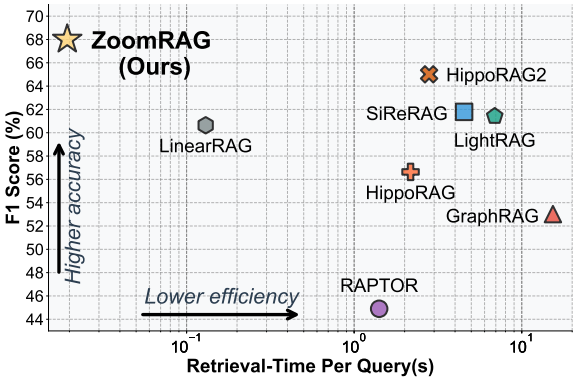

Figure 2: Averaged Answer-quality (F1) versus online-efficiency (retrieval time per query) for all methods.

broad yet targeted coverage of relevant documents, thereby minimizing early-stage errors. The fine stage then exploits local relational structures to precisely identify key evidence chunks within selected documents. Detailed case studies are in Table 15.

In Appendix F, we report empirical results where GPT-4o is used for answer generation. ZoomRAG achieves 13.13% and 5.47% gains in EM and F1, respectively, over the best performing baseline in Table 1. This result shows the broad adaptability of ZoomRAG to different base language models.

Offline Indexing Cost. ZoomRAG Off-line cost involves document entity extraction, embedding, and construction of document and entity adjacency matrix M (6). In Table 3, ZoomRAG takes 346 secs averagely in offline stage, roughly 10 – 443.56× faster than KG based methods (GraphRAG and HippoRAG 2) and twice as fast as LinearRAG. ZoomRAG requires 3.36 GB on average — up to 6× less than other graph-based methods. While GraphRAG takes 25% less memory, it is 30× slower. Moreover, ZoomRAG is highly scalable: new documents can be incorporated by appending new rows to adjacency matrix without re-indexing the entire graph.

Online Retrieval Efficiency. ZoomRAG online cost involves query entity resolution, query embedding, query link construction, global random walk, chunk–entity graph construction and local random walk. In Table 3, ZoomRAG takes only 0.019 seconds for each online query, 114 – 807× faster than KG-based methods while attaining the highest accuracy. The combined advantage in efficiency and accuracy is further illustrated in Figure 2. See a comprehensive latency analysis in Appendix J.

5.2 Ablation Studies

Table 4 analyzes impact of key design choices in ZoomRAG. (1) Removing the fine-level Chunk-Zoom stage reduces average F1 by 2%, highlighting the importance of combining global exploration with local refinement. (2) Eliminating within-partite linkages degrades performance sub-

stantially: removing document–document edges in the first stage results in a 14% drop, while removing chunk–chunk semantic and temporal links in the second stage causes a further 10% decrease. This verifies that within-partite connections effectively enrich graph structure and strengthen random-walk-based reasoning. (3) At the coarse Doc-Zoom level, removing entity nodes reduces average F1 by 16%, and removing query–entity links leads to a 5% drop, underscoring the integral role of the document–query–entity tripartite graph in ZoomRAG.

Method	2Wiki	HotpotQA	MuSiQue	Avg
ZoomRAG	**74.33**	**76.60**	**53.12**	**68.01**
Hierarchical Search Strategy				
w/o Fine-Grained Search	71.81	75.02	51.02	65.95
Graph Construction(in Doc-Zoom)				
w/o entity node	57.79	66.79	32.58	52.38
w/o query-entity link	69.58	73.90	45.67	63.05
w/o doc-doc link	63.58	66.61	33.72	54.63
Graph Construction(in Chunk-Zoom)				
w/o chk-chk link (semantic)	69.56	73.91	49.41	64.29
w/o chk-chk link (temporal)	72.90	75.34	51.68	66.64
w/o chk-chk link (overall)	64.82	68.14	42.43	58.46

Table 4: Ablation studies of ZoomRAG.

Appendix C (Fig.4) reports ZoomRAG's accuracy over key hyper-parameters, including random-walk iterations T, restart probabilities α and β, and the number of retrieved documents K. The performance curves exhibit stable and regular trends with a clear saturation point, making these hyper-parameters easy to select in practice. In all experiments, we simply fixed $T = 5$, $\alpha = 0.8, \beta = 0.9$, and $K = 10$. More hyper-parameter studies are also provided in Appendix C.

5.3 Scalability Analysis

Figure 3 shows ZoomRAG parallel behavior. (a) as the inference batch size increases from 1 to 256, the memory cost rises by **less than 1%**, demonstrating substantial memory efficiency. In contrast, a naive data-parallel strategy leads to out-of-memory errors quickly on processing 27 concurrent queries. (b) shows that the average processing time per query decreases steadily with batch size. Empirically, the efficiency reaches a plateau when processing more than 512 concurrent queries, with an average processing time of 0.019 seconds per query, which we largely attribute to I/O bandwidth limitations.

Table 5 compares the offline scalability profile of ZoomRAG with LinearRAG, a recently released and highly efficient RAG model. When the processed token-subset increases from 5M to 50M,

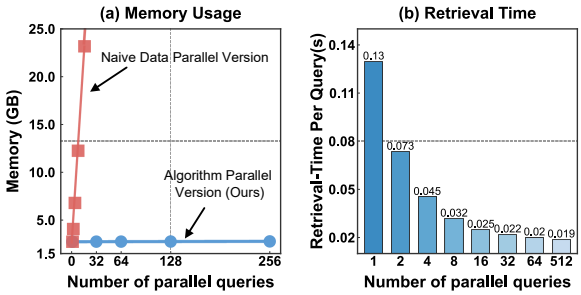

Figure 3: ZoomRAG behavior in parallel setting. (a) Memory usage of algorithm-parallel vs. data-parallel versions. (b) Retrieval time per query across batch size.

Dataset	Method	Indexing Time (s)	Memory (MB)
5M	LinearRAG	12802.57	13426.01
	ZoomRAG (Ours)	**552.61**	**3669.44**
10M	LinearRAG	24386.06	22401.48
	ZoomRAG (Ours)	**1177.97**	**6103.04**
15M	LinearRAG	44051.11	41979.59
	ZoomRAG (Ours)	**1846.31**	**10451.02**
20M	LinearRAG	57544.17	50271.88
	ZoomRAG (Ours)	**2425.69**	**13914.83**
50M	LinearRAG	136426.43	124866.48
	ZoomRAG (Ours)	**5753.42**	**32530.78**

Table 5: Index construction time and memory consumption of LinearRAG and ZoomRAG on large-scale ATLAS-Wiki subsets ranging from 5M to 50M tokens.

ZoomRAG shows (sub-)linear scaling in both time and memory. It is about **23× faster** than LinearRAG and takes about **73% less memory**. This demonstrates ZoomRAG's strong offline scalability and efficient resource usage.

6 Conclusion

We have proposed ZoomRAG, a hierarchical retrieval method on multi-scale information graphs to improve practical efficiency, accuracy and scalability of RAG systems. ZoomRAG avoids expensive knowledge graph construction and introduces algorithm-parallel random-walk retrieval so that RAG can be more friendly to large-scale and high-concurrency applications. Besides, encouraging results on popular multi-hop QA benchmarks are observed. In the future, we will study how to build more informative off-line graph structures to further improve reasoning and efficiency.

7 Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (with Grants No. 62276099).

8 Limitations

Despite the encouraging results, our approach has a few limitations. First, the empirical evaluations were primarily conducted on three widely used multi-hop QA datasets 2Wiki, HotpotQA, and MuSiQue. To ensure broader applicability and a more comprehensive assessment, future evaluations should include a more diverse set of datasets and question types. Second, all the datasets used in our experiments are English-language datasets. This lack of multilingual evaluation limits our understanding of the model's generalizability and robustness across different languages. Future work should include datasets in other languages to thoroughly assess the effectiveness and adaptability of ZoomRAG in multilingual/cross-lingual scenarios.

References

Yuzheng Cai, Zhenyue Guo, YiWen Pei, WanRui Bian, and Weiguo Zheng. 2025. SimGRAG: Leveraging similar subgraphs for knowledge graphs driven retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. 2025a. Pathrag: Pruning graph-based retrieval augmented generation with relational paths. *Preprint*, arXiv:2502.14902.

Guanzheng Chen, Qilong Feng, Jinjie Ni, Xin Li, and Michael Qizhe Shieh. 2025b. RAPID: Long-context inference with retrieval-augmented speculative decoding. In *Forty-second International Conference on Machine Learning*.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024a. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. 2024b. Dense x retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15159–15177.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From local to global: A graph rag approach to query-focused summarization. *Preprint*, arXiv:2404.16130.

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*, pages 6491–6501.

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).

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*.

Shubham Gupta, Zichao Li, Tianyi Chen, Cem Subakan, Siva Reddy, Perouz Taslakian, and Valentina Zantedeschi. 2025. Retreever: Tree-based coarse-to-fine representations for retrieval. *Preprint*, arXiv:2502.07971.

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*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From RAG to memory: Non-parametric continual learning for large language models. In *Forty-second International Conference on Machine Learning*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. 2025. Retrieval-augmented generation with graphs (graphrag). *Preprint*, arXiv:2501.00309.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. 2025. Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 1003–1012, New York, NY, USA. Association for Computing Machinery.

Wenqi Jiang, Shuai Zhang, Boran Han, Jie Wang, Bernie Wang, and Tim Kraska. 2024. Piperag: Fast retrieval-augmented generation via algorithm-system co-design. *arXiv preprint arXiv:2403.05676*.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. 2024. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*.

Junkyum Kim and Divya Mahajan. 2025. An adaptive vector index partitioning scheme for low-latency rag pipeline. *arXiv preprint arXiv:2504.08930*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Mufei Li, Siqi Miao, and Pan Li. 2025a. Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. In *The Thirteenth International Conference on Learning Representations*.

Zihang Li, Yangdong Ruan, Wenjun Liu, Zhengyang Wang, and Tong Yang. 2025b. Cft-rag: An entity tree based retrieval augmented generation algorithm with cuckoo filter. *Preprint*, arXiv:2501.15098.

Chien-Yu Lin, Keisuke Kamahori, Yiyu Liu, Xiaoxiang Shi, Madhav Kashyap, Yile Gu, Rulin Shao, Zihao Ye, Kan Zhu, Rohan Kadekodi, and 1 others. 2025. Telerag: Efficient retrieval-augmented generation inference with lookahead retrieval. *arXiv preprint arXiv:2502.20969*.

Chaoqiang Liu, Haifeng Liu, Dan Chen, Yu Huang, Yi Zhang, Wenjing Xiao, Xiaofei Liao, and Hai Jin. 2025a. Heterrag: Heterogeneous processing-in-memory acceleration for retrieval-augmented generation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pages 884–898.

Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025b. Hoprag: Multi-hop reasoning for logic-aware retrieval-augmented generation. *Preprint*, arXiv:2502.12442.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2025. Gfm-rag: Graph foundation model for retrieval augmented generation. *Preprint*, arXiv:2502.01113.

Jie Ou, Jinyu Guo, Shuaihong Jiang, Zhaokun Wang, Libo Qin, Shunyu Yao, and Wenhong Tian. 2025. Accelerating adaptive retrieval augmented generation via instruction-driven representation reduction of retrieval overlaps. *arXiv preprint arXiv:2505.12731*.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. 2024. Graph retrieval-augmented generation: A survey. *Preprint*, arXiv:2408.08921.

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*, 11:1316–1331.

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*.

Sanat Sharma, David Seunghyun Yoon, Franck Dernoncourt, Dewang Sultania, Karishma Bagga, Mengjiao Zhang, Trung Bui, and Varun Kotte. 2024. Retrieval augmented generation for domain-specific question answering. *Preprint*, arXiv:2404.14760.

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*, 11:1–17.

Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Nan Wang, and Han Xiao. 2024. jina-embeddings-v3: Multilingual embeddings with task lora. *Preprint*, arXiv:2409.10173.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, and 1 others. 2024a. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736.

Zilong Wang, Zifeng Wang, Long Le, Huaixiu Steven Zheng, Swaroop Mishra, Vincent Perot, Yuwei Zhang, Anush Mattapalli, Ankur Taly, Jingbo Shang, and 1 others. 2024b. Speculative rag: Enhancing retrieval augmented generation through drafting. *arXiv preprint arXiv:2407.08223*.

Yuan Xia, Jingbo Zhou, Zhenhui Shi, Jun Chen, and Haifeng Huang. 2025. Improving retrieval augmented language model with self-reasoning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pages 25534–25542.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval meets long context large language models. In *The Twelfth International Conference on Learning Representations*.

Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2905–2909.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. 2025a. SireRAG: Indexing similar and related information for multihop reasoning. In *The Thirteenth International Conference on Learning Representations*.

Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. 2023. Retrieve anything to augment large language models. *Preprint*, arXiv:2310.07554.

Ze Yu Zhang, Zitao Li, Yaliang Li, Bolin Ding, and Bryan Kian Hsiang Low. 2025b. Respecting temporal-causal consistency: Entity-event knowledge graphs for retrieval-augmented generation. *Preprint*, arXiv:2506.05939.

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. *Preprint*, arXiv:2402.19473.

Xishi Zhu, Xiaoming Guo, Shengting Cao, Shenglin Li, and Jiaqi Gong. 2024. Structugraphrag: Structured document-informed knowledge graphs for retrieval-augmented generation. In *Proceedings of the AAAI Symposium Series*, volume 4, pages 242–251.

Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2026. LinearRAG: Linear graph retrieval augmented generation on large-scale corpora. In *The Fourteenth International Conference on Learning Representations*.

A Time and Space Complexity Analysis of Algorithm-Parallel Random Walk

We analyze the time and space complexity of random-walk-based retrieval under three versions: the sequential version, the naive data-parallel version, and the proposed algorithm-parallel version. The comparison is summarized in Table 6.

Method	Time complexity	Space complexity
Sequential	$O(k\|\mathcal{V}\| + k\gamma(m+n))$	$O(\|\mathcal{V}\| + \gamma(m+n))$
Parallel (naive)	$O(k\|\mathcal{V}\| + k\gamma(m+n))/k$	$O(k\|\mathcal{V}\| + k\gamma(m+n))$
Parallel (ours)	$O(\|\mathcal{V}\| + k\gamma(m+n))/k$	$O(\|\mathcal{V}\| + k\gamma(m+n))$

m: num. documents, n: num. entities, k: num. parallel queries;
$|\mathcal{V}|$: num. non-zeros in sparse matrix $\mathbf{M}_{(n+m)\times(n+m)}$ - Eq. (6)
γ: sparsity ratio of query-dependent links

Table 6: Time and space complexity of sequential, data-parallel and algorithm-parallel random-walks.

In sequential version, each query independently performs random walk propagation over the same document–entity graph. For k queries, the total time scales linearly as $O(k|\mathcal{V}|+k\gamma(m+n))$, where $|\mathcal{V}|$ denotes the number of non-zero entries in the shared graph matrix and $\gamma(m+n)$ accounts for sparse query-dependent links. The space complexity is $O(|\mathcal{V}| + \gamma(m+n))$, dominated by storing the global graph and query-specific vectors.

The naive data-parallel version reduces wall-clock time by processing k queries concurrently, but requires replicating the entire graph structure for each query. This results in a prohibitive space complexity of $O(k|\mathcal{V}| + k\gamma(m+n))$, making it impractical for high-throughput settings.

In contrast, the proposed algorithm-parallel version decouples query-independent graph components from query-specific computations. The document–entity graph M is shared across all queries, while only lightweight query-dependent vectors are maintained separately. Consequently, the space complexity is reduced to $O(|\mathcal{V}| + k\gamma(m+n))$, avoiding redundant storage of the global graph. Meanwhile, batched matrix operations enable parallel propagation for all queries, yielding a per-query time complexity of $O(|\mathcal{V}| + k\gamma(m+n))/k$.

Overall, this analysis demonstrates that the algorithm-parallel version achieves a more favorable trade-off between computational efficiency and memory consumption, enabling scalable retrieval under a large number of concurrent queries.

B Datasets Statistics

Table 7 summarizes the key statistics of the datasets used in our experiments: 2Wiki, HotpotQA, and MuSiQue. Each dataset contains 1000 queries, with varying numbers of documents and supporting facts. The average document word count also differs across datasets, reflecting their unique characteristics. Specifically, 2Wiki has 6,119 documents with an average token count of 111.63, HotpotQA includes 9,221 documents averaging 132.1 tokens, and MuSiQue contains 11,656 documents with an average token count of 116.94.

We used the datasets strictly in accordance with their license requirements and solely for research purposes. More specifically, 2Wiki is released under the Apache 2.0 license, HotpotQA under CC-BY-SA 4.0, and MuSiQue under CC-BY 4.0.

Statistics	**2Wiki**	**HotpotQA**	**MuSiQue**
#Queries	1000	1000	1000
#Documents	6119	9221	11656
#Supporting Facts	2471	2461	2648
#Avg. Doc. Tokens	111.63	132.10	116.94
#Max. Sent. Tokens	290	492	600

Table 7: Dataset statistics

C Ablation Studies

We analyze the sensitivity of ZoomRAG to core random-walk hyperparameters at both stages, including the iteration count (t), restart probabilities (α, β), and the document cut-off rank (K).

Figure 4(a) shows F1 scores versus the number of iterations t (1 to 9) in the coarse-grained Doc-Zoom stage, with $\alpha = 0.8$. As noted, increasing iterations expands coverage to retrieve more relevant documents, improving answer generation, but excessive steps introduce noise, causing performance to plateau or slightly drop.

Figure 4(b) illustrates F1 scores versus restart probability α used in the coarse-level random-walk (0.1–0.9, with five steps). As shown, the F1 score generally increases with α, peaking at $\alpha = 0.8$, and then slightly decreasing. This indicates that a relatively larger restart probability enhances semantic focus and answer quality.

Figure 4(c) studies the effect of the restart probability β in the fine-grained Chunk-Zoom stage (with $\alpha = 0.8$, five iterations, and top-$K = 10$). As β increases, the F1 score consistently improves, reaching its peak at $\beta = 0.9$. This suggests that

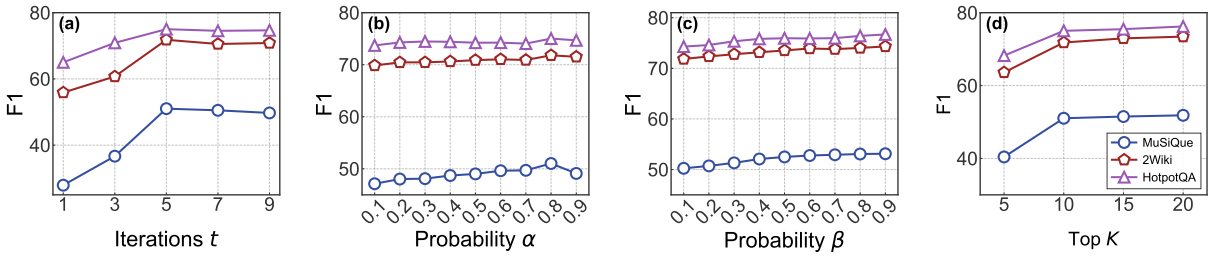

Figure 4: ZoomRAG accuracy w.r.t. random-walk steps t (a), coarse-grained restart probability α (b), fine-grained restart probability β (c) and cut-off rank k (d).

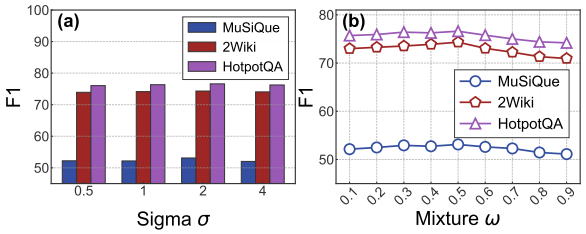

Figure 5: ZoomRAG performance w.r.t. gaussian kernel bandwidth (a) and mixture coefficient (b).

a high restart probability is beneficial in the fine-grained retrieval stage, which helps concentrate the random walk on the most salient local evidence while reducing unnecessary exploration.

Figure 4(d) shows F1 scores versus cut-off rank K. Although increasing K slightly improves F1, the gains are marginal, so we select top 10 documents to balance accuracy and computational cost.

Based on these observations, we fix $t = 5$, $\alpha = 0.8$, $K = 10$ for Doc-Zoom, and $\beta = 0.9$ for Chunk-Zoom in all our experiments for evaluation.

We further study the temporal–semantic hyperparameters in the fine-grained retrieval stage, including the temporal kernel bandwidth σ and the semantic–temporal mixture coefficient ω, with all other hyperparameters fixed to isolate their effects. As shown in Figures 5(a)–(b), performance remains stable across a wide range of σ and ω, with only minor fluctuations at extreme values, indicating that the fine-grained temporal–semantic random walk is largely insensitive to these settings.

Overall, ZoomRAG demonstrates robust performance across all parameter settings.

D Current Practices in Parallel RAG

Recent work has explored various forms of parallelism to improve the efficiency of retrieval-augmented generation (RAG). Most existing approaches focus on accelerating the processing of a single query, for example through pipeline parallelism (Jiang et al., 2024; Gao et al., 2023) between the retrieval and generation stages, speculative decoding of multiple candidate responses (Wang et al., 2024b; Chen et al., 2025b), or batched and hardware-accelerated retrieval (Lin et al., 2025; Liu et al., 2025a). Besides, various engineering techniques—such as index sharding(Kim and Mahajan, 2025), caching(Jin et al., 2024; Ou et al., 2025), and batched approximate nearest neighbor search—are commonly adopted to improve system throughput. These approaches primarily operate at the systems level and do not fundamentally alter the underlying algorithmic structure of RAG.

In contrast, we explicitly target the challenge of supporting high-concurrency RAG workloads in a memory-efficient manner through *algorithm-parallel RAG*. In this setting, a large query-agnostic offline structure is stored only once and shared across queries, while query-specific states and computations are carefully isolated. This design enables efficient concurrent execution without duplicating large indexes and fundamentally differs from prior parallelization strategies, which emphasize intra-query acceleration rather than inter-query concurrency.

E Convergent Solution of Restarted Random-Walk in ChunkZoom

We observe that in the ChunkZoom step, using an effectively infinite number of restart random walks—corresponding to the stationary distribution—yields the best empirical performance. The stationary distribution of a restarted random walk, given in Eq. (10), can be computed in closed form as

$$\mathbf{y}^{(\infty)} = \beta\,(I - (1-\beta)\widetilde{\mathbf{B}})^{-1}\mathbf{y}^{(0)}. \qquad (10)$$

Since the local graph is relatively small with only a few tens of nodes, this analytical solution can be efficiently obtained via matrix inversion.

Method	2Wiki	HotpotQA	MuSiQue	Avg
Iterative Random-Walk Solution				
iter = 1	71.02	75.87	52.75	66.54
iter = 3	71.82	76.14	52.78	66.91
iter = 5	72.86	76.10	53.00	67.32
iter = 7	72.93	75.86	52.35	67.04
iter = 9	73.78	75.62	52.17	67.19
Analytical Solution (small matrix inverse)				
iter = ∞	**74.33**	**76.60**	**53.12**	**68.01**

Table 8: ZoomRAG accuracy versus the number of fine-scale random-walks in the ChunkZoom stage.

As shown in Table 8, when comparing a finite number of random-walk iterations $(1, 3, 5, 7, 9)$, the analytical solution consistently achieves the highest average F1 scores across all datasets. This is likely because the local chunk graph has already been filtered by the coarse-grained retrieval stage and thus contains highly relevant nodes, making a fully converged random walk more effective. In contrast, for coarse-grained retrieval in Doc-Zoom, using a small and fixed number of iterations is more appropriate, as it helps preserve query relevance signals and prevents excessive propagation.

F Using GPT-4o for Answer Generation

To assess the generalization capability of ZoomRAG, we perform answer generation using the stronger LLM, GPT-4o, while keeping the retrieval process unchanged. Due to API cost, we select only HippoRAG 2 - the strongest baseline from our main experiments - for this empirical comparison.

As shown in Table 9, ZoomRAG consistently has a better performance across all datasets in terms of EM and F1 scores. This demonstrates that our mixed-granularity retrieval strategy effectively generalizes to stronger LLMs for answer generation. The notable improvements on 2Wiki and HotpotQA highlight the capability of our coarse-to-fine retrieval mechanism to capture key evidence in multi-hop QA scenarios. Overall, these results further validate the robustness and generality of our retrieval design.

G Starting Nodes in Random-Walks

Beyond differences in graph construction, another key distinction lies in whether the query node is explicitly retained during random-walk propagation. In HippoRAG 2 and LinearRAG, random walks are initialized from the first-order neighbors of the query (e.g., related entities or chunks), which excludes the query node itself from subsequent propagation. In contrast, our approach explicitly incorporates the query node at every iteration (as the restart node) of the random walk, thereby maintaining continuous query awareness throughout the entire propagation process.

Method	2Wiki		HotpotQA		MuSiQue		Average	
	EM	F1	EM	F1	EM	F1	EM	F1
HippoRAG 2	45.90	63.57	53.20	72.67	35.40	50.95	44.83	62.39
ZoomRAG	**64.50**	**71.26**	**64.80**	**76.91**	**44.60**	**55.41**	**57.96**	**67.86**

Table 9: QA performance (EM and F1 scores) of ZoomRAG and HippoRAG 2 when GPT-4o is used solely for answer generation.

H Importance of Query Decoupling in Parallel ZoomRAG

Here we provide empirical evidence that when accommodating multiple random-walk in parallel on a shared global graph, avoiding the interference across different queries is crucial to preserving the performance as in independent random-walks.

Suppose we are given m documents, n entities, and k queries. Table 10 compares two strategies for executing concurrent random walks. (1) Coupled random walks naively run k queries in parallel via batched matrix-matrix multiplication by concatenating the random-walk distributions of all queries into a matrix and multiplying it with a unified transition matrix (encompassing all queries, documents and entities). This formulation leads to cross-query interference during propagation. (2) Decoupled random walks execute k queries in parallel by explicitly separating query-specific distributions from shared graph components. All queries share a global $(n+m) \times (n+m)$ document–entity transition matrix $\mathbf{M}$ (Eq. 6), while each query maintains its own lightweight $(n+m+1)$-dimensional random-walk distribution that interacts with the shared graph only through query-specific node connections. This prevents cross-query interference while still fully leveraging batched matrix operations for efficiency.

Table 10 shows that decoupled random walks consistently outperform the coupled formulation, yielding a 17.43% average improvement in F1 score across datasets. This highlights the importance of isolating query-specific propagation states under high concurrency to avoid contaminating

random-walk retrievals.

Method	2Wiki	HotpotQA	MuSiQue	Avg
Uncoupled Random-walk	48.67	66.27	30.62	48.52
Decoupled Random-walk	**71.81**	**75.02**	**51.02**	**65.95**

Table 10: F1 scores of decoupled (algorithm-parallel) versus uncoupled random walks.

I ChunkZoom Cost Breakdown

To better understand the computational overhead of fine-grained graph construction, we provide a detailed analysis of the ChunkZoom sub-step performance for documents of varying lengths. On the 2WikiMultihopQA dataset, we fixed the retrieval to 10 documents and measured the average time of each sub-step across different document lengths (measured by the number of chunks), including chunk embedding, chunk similarity computation, random walk computation, and chunk ranking. The results are shown in Table 11.

Chunks	Embed(s)	Sim(s)	RW(s)	Rank(s)	Total(s)
16–40	0.0006	0.0003	0.0002	0.0002	0.0013
41–65	0.0007	0.0003	0.0002	0.0002	0.0014
66–90	0.0008	0.0003	0.0002	0.0002	0.0015
91–115	0.0008	0.0003	0.0002	0.0002	0.0015
115–144	0.0010	0.0003	0.0002	0.0002	0.0017

Table 11: Latency breakdown of ChunkZoom across varying document lengths.

Observation. The ChunkZoom stage introduces minimal computational overhead, with total latency only 1.3–1.7 ms, accounting for approximately 10% of the overall retrieval cost. Among all components, embedding construction dominates the runtime, while similarity computation, random walk, and ranking incur negligible overhead. As the number of chunks increases from 16–40 to 115–144 (approximately 4×), the total latency grows only from 1.3 ms to 1.7 ms (approximately 1.3×). This clear sub-linear scaling behavior indicates that ZoomRAG is well-suited for large-scale corpora.

J Latency Performance Analysis

To provide a more comprehensive evaluation of latency performance, we conduct experiments on the 2Wiki dataset using 1,000 queries. We further report single-query latency as well as tail latency (P95 and P99) under a unified concurrency setting to ensure fair comparisons across methods.The results are summarized as follows:

J.1 Single-Query Latency

Since all baselines are executed in a serial manner (batch size = 1), we measured the average and tail latency of each method under the same batch size for a fair comparison, as shown in Table 12.

Method	Batch	Avg. (s)	P95 (s)	P99 (s)
RAPTOR	1	1.397	1.852	2.410
SiReRAG	1	4.420	5.921	7.633
GraphRAG	1	13.931	18.450	25.120
HippoRAG	1	1.569	2.103	2.843
HippoRAG 2	1	2.161	3.012	4.342
LightRAG	1	6.309	9.569	20.350
LinearRAG	1	0.137	0.179	0.235
ZoomRAG	1	**0.129**	**0.157**	**0.168**

Table 12: Single-query latency performance comparison across methods.

Observation. KG-based methods such as GraphRAG and LightRAG exhibit significantly higher tail latency (e.g., P99 reaching 25.120 s and 20.350 s), indicating substantial variance in response time. Although LinearRAG achieves relatively low average latency, its tail latency remains noticeably higher than ZoomRAG. In contrast, ZoomRAG achieves the lowest average latency (0.129 s) and maintains a stable P99 latency (0.168 s), demonstrating its suitability for real-time single-query scenarios.

J.2 Batch Processing and Scalability

We further evaluated ZoomRAG's latency performance under different batch sizes, as shown in Table 13.

Method	Batch	Avg. (s)	P95 (s)	P99 (s)
ZoomRAG	1	0.129	0.157	0.168
ZoomRAG	4	0.045	0.051	0.068
ZoomRAG	16	0.025	0.027	0.029
ZoomRAG	64	0.019	0.020	0.021
ZoomRAG	256	0.018	0.019	0.019
ZoomRAG	512	0.018	0.019	0.019
ZoomRAG	1000	0.018	0.018	0.018

Table 13: Latency performance of ZoomRAG under different batch sizes.

Observation. As the batch size increases, both the average latency and tail latency (P95/P99) of ZoomRAG steadily decrease and eventually stabilize.

Overall, ZoomRAG achieves lower latency than all baselines in single-query settings, reducing average latency by 5.8% and P99 latency by 28.5%

compared to the best-performing baseline, LinearRAG. Under concurrent workloads, larger batch size further reduces both average and tail latencies, fully leveraging parallel retrieval capabilities.

K Case Studies

To better understand ZoomRAG's retrieval behavior, we present a qualitative case study in Table 15 on a relatively difficult multi-hop question from the 2WikiMultihopQA dataset, comparing it with two strong baselines: HippoRAG 2 and LinearRAG.

We can see that for the two baseline methods, they may only retrieve a limited amount of useful evidence, or the retrieved contexts contain a mixture of relevant and irrelevant information, which can make it difficult for the model to identify the key supporting evidence. In comparison, ZoomRAG shows a stronger capability in identifying relevant information from the knowledge base by employing a coarse-to-fine retrieval strategy. For example, in the coarse-level DocZoom stage, two relevant documents have already been identified, providing a solid set of candidate documents, which is better than the other two methods; in the fine-grained ChunkZoom stage, the retrieval further pinpoints the two chunks within these documents that are most relevant to the query, effectively refining the evidence and focusing on the information most critical for answering the question.

L Hyperparameters in Some Baselines

We summarize the main hyperparameters of ZoomRAG and two representative SOTA RAG methods, HippoRAG 2 and LinearRAG, in Table 14. As can be seen, they all require 4 to 5 key parameters in order to properly control the behaviour of the system. Therefore, the presence of multiple hyperparameters in ZoomRAG follows common practice in current RAG systems and does not introduce additional challenges in terms of robustness or practical deployment compared to existing methods.

It is worthwhile to note that, through systematic ablation studies in Appendix C, ZoomRAG have exhibited highly regular and robust performance profiles, making hyper-parameters easy to choose in practice (see Figure 4).

M LLM Prompts

In choosing a base LLM for NER and answer generation, recent works have gradually shifted towards the lightweight GPT-4o-mini due to its efficiency and lower computational cost.

Method	Hyperparameter	Value
HippoRAG 2	Top-n triples	5
	Top-k passages	10
	Synonym threshold	0.8
	Damping factor	0.5
	Synonymy edge top-n	2047
LinearRAG	Dynamic pruning threshold	4
	Damping factor	0.5
	Top-k passages	10
	Passage node weight	0.05
	Passage ratio	1.5
ZoomRAG	Number of iterations (t)	5
	Restart probability (α)	0.8
	Restart probability (β)	0.9
	Top-k documents (K)	10

Table 14: Main hyperparameters of ZoomRAG, HippoRAG 2 and LinearRAG.

Following these literature, we therefore employed GPT-4o-mini for two core tasks in our system: Named Entity Recognition (NER) and answer generation. For NER, we adopted an instruction-based prompting strategy designed to extract entities from both the document content and the user query. The prompt provides a clear and concise instruction (see Figure 6), allowing the model to consistently identify relevant entities, which are subsequently used to build the information graph.

For answer generation, we follow the experimental settings of prior state-of-the-art RAG methods, including HippoRAG 2 and LinearRAG, and also adopt a chain-of-thought (CoT) prompting strategy to guide the generation process. The detailed reasoning prompt is illustrated in Figure 7.

Question	Who is the maternal grandmother of Eleanor Of Brittany (Abbess)?
Support Facts	[“Eleanor of Brittany” → “mother” → “Beatrice of England”]
	[“Beatrice of England” → “mother” → “Eleanor of Provence”]
Ground Truth	Eleanor of Provence

Case Study

HippoRAG2

Retrieved Context:

✓ (1) “Eleanor of Brittany (abbess)”: ...She was born in England to John II, Duke of Brittany and Beatrice of England...

✗ (2) “Blanche of Brittany”: Blanche of Brittany(1271–1327) was a daughter of John II, Duke of Brittany...

✗ (3) “Guy of Thouars”: Guy of Thouars (died 13 April 1213) was the third husband of Constance, Duchess of...

✗ (4) “Éléonore Desmier d’Olbreuse”: Éléonore Marie Desmier d’Olbreuse (3 January 1639 – 5 February 1722) was the...

✗ (5) “Marie of Brittany, Countess of Saint-Pole”: Marie of Brittany(1268–1339) was the daughter of John II, Duke...

...

***Prediction:* ✗ Beatrice of England.**

LinearRAG

Retrieved Context:

✗ (1) “Adelaide I, Abbess of Quedlinburg”: Adelaide I (973/74 - 14 January 1044 or 1045), a member of...

✗ (2) “Adelaide II, Abbess of Quedlinburg”: Adelaide II (1045 - 11 January 1096), a member of...

✗ (3) “John Ernest II, Duke of Saxe-Weimar”: John Ernest II (11 September 1627, in Weimar - 15 May 1683, in Weimar), was a duke of Saxe-Weimar...

✗ (4) “Anna Amalia, Abbess of Quedlinburg”: Princess Anna Amalia of Prussia (9 November 1723 - 30 March 1787) was Princess-Abbess...

✗ (5) “Matilda, Abbess of Quedlinburg”: Matilda (December 955 - 999), also known as Mathilda and Mathilde, was...

...

***Prediction:* ✗ Unknown.**

ZoomRAG

Stage I (Coarse-grained Retrieval):
Retrieved Documents:

✓ (1) “Eleanor of Brittany (abbess)”: ...She was born in England to John II, Duke of Brittany and Beatrice of England...

✓ (2) “Beatrice of England”: Beatrice of England (25 June 1242 - 24 March 1275) was a member of the House of Plantagenet, the daughter of Henry III of England and Eleanor of Provence.

✗ (3) “Marie of Brittany, Countess of Saint-Pol”: Marie of Brittany (1268 - 1339) was the daughter of John II, Duke of Brittany, and...

✗ (4) “Charles the Child”: Charles the Child (Latin "Karolus puer", from the "Annales Bertinian"...

✗ (5) “Russell High School (Ontario)”: Russell High School is a secondary school in Russell located...

...

Stage II (Fine-grained Retrieval):
Retrieved Chunks:

✓ (1) “Eleanor of Brittany (abbess)”: She was born in England to John II, Duke of Brittany and Beatrice of England.

✓ (2) “Beatrice of England”: Beatrice of England (25 June 1242 - 24 March 1275) was a member of the House of Plantagenet, the daughter of Henry III of England and Eleanor of Provence.

✗ (3) “Eleanor of Brittany (abbess)”: Eleanor of Brittany (1275 - 2013 16 May 1342) was the sixteenth abbess of Fontevrault.

...

***Prediction:* ✓ Eleanor of Provence.**

Table 15: Qualitative comparison of retrieval behaviors across different RAG methods.

Prompt for Performing Name Entity Recognition

Prompt: You are a professional Named Entity Recognition system. Please analyze the text, perform Named Entity Recognition, extract the key entities, and minimize the extraction of irrelevant ones. Follow the rules below when analyzing and extracting entities from the input text. Do not output in markdown codeblock.

Extraction Rules:

1. Return strictly formatted JSON
2. Extract only proper nouns and named entities, avoiding generic or common terms
3. Perform entity disambiguation:
- If multiple mentions refer to the same real-world entity, recognize them as referring to the same entity and ensure that they are represented by the same standardized entity name
- If one entity has a commonly used abbreviation, make sure to use both the full name and the abbreviation
- For example, if the text mentions "United Nations" and "UN", recognize that both refer to the same entity and extract only "United Nations (UN)"
- Similarly, if the text contains "Apple" and "Apple Inc.", recognize that both refer to the same entity and standardize to "Apple Inc."
4. Handle compound names appropriately:
- Treat meaningful multi-part entity names as a single entity (e.g., "Theodred II (Bishop of Elmham)" should be treated as one entity)
- Remove non-essential qualifiers when they do not contribute to the identity of the entity (e.g., "Lothair I of the Franks" $\rightarrow$ "Lothair I")
5. Handle nested entities (e.g., "University of California, Berkeley" contains "California" and "Berkeley" but should be treated as a single organization)
6. Do not extract:
- Common nouns that aren't proper names
- Generic time references (unless they're specific named periods, e.g., "Renaissance")
- Pronouns and other referring expressions
7. For entities with different names/spellings in the text, standardize to the most formal or complete version

Output Format:

```json{"entities": [{"entity": "exact_text"}]}```

Passage: {text}

Figure 6: Prompt for Performing Name Entity Recognition.
```

Prompt for LLM answer generation

System: You are an advanced reading comprehension assistant.
When answering, follow this exact structure:
1. Start with "Thought:"
  - In this section, you must perform full reasoning and analysis.
  - You may explain how the context leads to the answer.
  - This part can be long or short, but must reflect your reasoning.
2. Then output "Answer:"
  - This MUST follow strict rules:
    • Use ONLY the provided context.
    • NO extra words, NO explanation, NO prefixes besides "Answer:".
    • If the question asks for name/date/place → return the exact item only.
    • If yes/no → return "yes" or "no".
    • If comparison → output only the correct entity.
  - The answer must be concise and minimal, exactly like the examples.
The "Thought:" section is for reasoning.
The "Answer:" section must be concise and precise.
User: Context: {context}
Question: {question}
Thought:

Figure 7: Prompt for LLM answer generation