

TERAG: Token-Efficient Graph-Based Retrieval-Augmented Generation

Qiao Xiao¹, Hong Ting Tsang², and Jiaxin Bai²

¹ Cornell University, United States

qx226@cornell.edu

² Hong Kong University of Science and Technology, Hong Kong

httsangaj@connect.ust.hk, jbai@connect.ust.hk

Abstract. Graph-based Retrieval-augmented generation (RAG) has become a widely studied approach for improving the reasoning, accuracy, and factuality of Large Language Models (LLMs). However, many existing graph-based RAG systems overlook the high cost associated with LLM token usage during graph construction, hindering large-scale adoption. To address this, we propose **TERAG**, a simple yet effective framework designed to build informative graphs at a significantly lower cost. Inspired by HippoRAG, we incorporate Personalized PageRank (PPR) during the retrieval phase, and we achieve at least 80% of the accuracy of widely used graph-based RAG methods while consuming only 3%-11% of the output tokens. With its low token footprint and efficient construction pipeline, TERAG is well-suited for large-scale and cost-sensitive deployment scenarios.

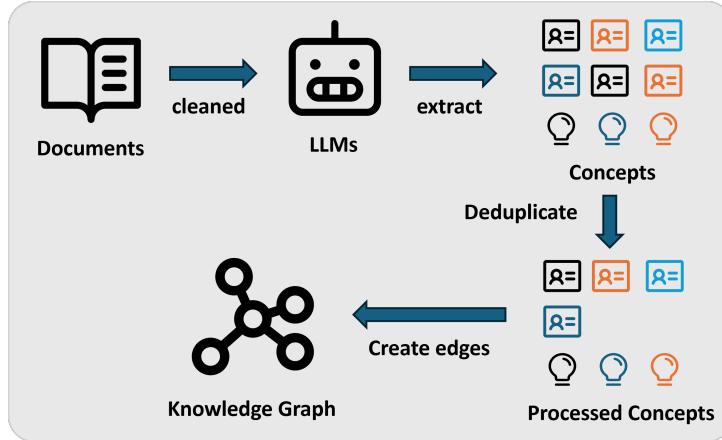
Keywords: Retrieval-Augmented Generation; GraphRAG; LLMs; Knowledge Graphs; Token Efficiency

1 Introduction

Retrieval-Augmented Generation has emerged as an important framework to mitigate hallucinations and ground Large Language Models in external knowledge, enhancing their reliability in specialized domains like medicine [18], law [8], and education [9]. While traditional RAG retrieves unstructured text snippets, recent advancements have shifted towards graph-based RAG, which leverages knowledge graphs (KGs) to model structured relationships between information entities [22]. This structured approach enables more accurate multi-hop reasoning and provides greater transparency into the model’s decision-making process [5].

However, the superior performance of state-of-the-art graph-based RAG systems, such as AutoSchemaKG [2] and Microsoft’s GraphRAG [5], comes at a staggering cost. These methods rely heavily on extensive LLM calls for node extraction, relationship definition, and schema induction, resulting in extremely high token consumption. This dependency makes graph construction prohibitively

Fig. 1: Overall pipeline of the proposed TERAG framework. The process consists of lightweight concept extraction with LLMs, followed by efficient non-LLM clustering and graph construction.



expensive; for instance, indexing a mere 5GB of legal documents was recently estimated to cost as much as \$33,000 [15]. In practice, such a financial burden poses a major barrier to scalable deployment, making cost-effectiveness as crucial a metric as accuracy.

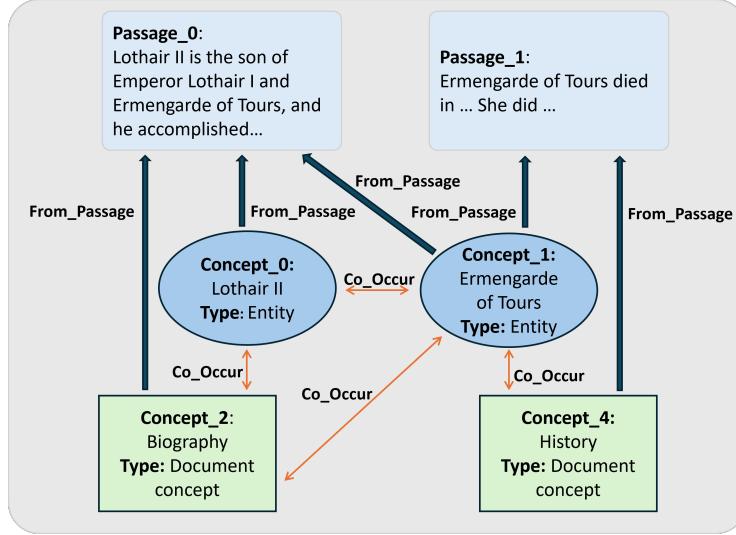
To address this critical trade-off between cost and performance, we propose TERAG, a lightweight framework that minimizes LLM usage while leveraging their strengths in concept extraction. The overall pipeline is shown in Figure 1. Instead of relying on multiple rounds of expensive LLM reasoning for graph construction, our method uses a few carefully designed prompts to extract multi-level concepts. These concepts are then structured into an effective knowledge graph using efficient, non-LLM methods, as illustrated in Figure 2. This process is illustrated with a real case from the 2WikiMultihopQA dataset. The two passages, while topically related, lack a direct connection that simple semantic retrieval could exploit. By extracting concepts from each passage and linking them with a `co_occurrence` edge, TERAG successfully connects them via key semantic information. This enables a successful retrieval for the question, “What is the death date of Lothair II’s mother?”—a query that would likely fail with retrieval methods based only on direct semantic similarity retrieval.

For the retrieval phase, inspired by HippoRAG [11, 12], we apply Personalized PageRank to the constructed graph. This approach enhances retrieval effectiveness without requiring additional LLM calls. By focusing LLM usage on initial, lightweight tasks, TERAG strikes a favorable balance between efficiency and effectiveness.

Our main contributions are as follows:

- We propose TERAG, a simple yet effective graph-based RAG framework designed specifically to minimize LLM token consumption during graph construction.

Fig. 2: Graph structure of TERAG. The figure illustrates how lightweight concept extraction with LLMs, followed by non-LLM clustering and graph construction, leads to an efficient knowledge graph.



- We demonstrate that TERAG reduces output token usage by 89-97% compared to other widely-used graph-based RAG methods, offering a highly cost-effective solution.
- We show that despite its lightweight design, TERAG achieves competitive retrieval accuracy on three standard multi-hop question-answering benchmarks.

The remainder of this paper is organized as follows: Section 2 formalizes the problem, Section 3 discusses related work, Section 4 details our proposed pipeline, Section 5 presents experimental results, and Section 6 provides an ablation study.

2 Related Work

This section traces the development of RAG, from foundational “naive” methods to more powerful Graph-based systems. We conclude by analyzing the critical challenge of token efficiency and the resulting cost-performance trade-off that motivates our work.

Naive RAG While pretrained language models have demonstrated a remarkable ability to internalize knowledge from training data, they possess significant limitations. Notably, their parametric memory is difficult to update, and they are prone to generating factually incorrect information, a phenomenon known as “hallucination” [25, 19]. To mitigate these issues, Retrieval-Augmented Generation was introduced to ground model responses in an external, non-parametric

knowledge source [17]. Initial approaches, often termed “naive RAG,” retrieve isolated text chunks based on vector similarity to a user’s query. Although this method enhances factual accuracy, it fundamentally overlooks the interconnected nature of information. Consequently, it struggles with complex, multi-hop questions that require synthesizing insights from multiple, related documents [14]. This core limitation paved the way for more advanced methods like Graph-based RAG, which explicitly models the relationships between knowledge entities to enable more sophisticated reasoning.

Graph RAG The emergence of Graph-based RAG addresses the primary limitation of its predecessors: the failure to model complex relationships between text chunks. Graph RAG leverages a pre-constructed knowledge graph, utilizing structural elements like nodes and paths to enrich the retrieval process and access more interconnected information [22]. This paradigm has inspired a wave of powerful systems, including Microsoft’s GraphRAG [5], HippoRAG [11], AutoSchemaKG [2] and ThinkOnGraph [26], many of which have demonstrated impressive performance and garnered significant industry attention, including Microsoft, NebulaGraph, and AntGroup [20, 21, 4].

The strength of these systems stems from a sophisticated and often costly indexing pipeline, where frameworks like GraphRAG [5], LightRAG [10], and MiniRAG [7] rely on Large Language Models (LLMs) to construct a knowledge graph from raw text. This process typically involves extracting entities and their connections as structured (subject, relation, object) triplets, and in some cases, generating node summaries or community reports [5]. While powerful, this deep integration of LLMs during indexing lead to substantial token consumption, creating a significant cost barrier for large-scale, real-world adoption.

The Challenge in Graph RAG In response to prohibitive costs, research has diverged into two main efficiency-focused directions. The first, exemplified by systems like LightRAG and MiniRAG [10, 7], prioritizes creating structurally lightweight graphs, though this can paradoxically increase indexing time and token consumption. A second path, therefore, which includes frameworks like LazygraphRAG [6], KET-RAG [15], and our work, TERAG, concentrates directly on minimizing the cost of the indexing process itself. Within this approach, while KET-RAG aims to reduce the overall cost from indexing to retrieval, our work with TERAG focuses aggressively on optimizing output tokens during the construction phase. We prioritize minimizing output token as it is the most critical factor during graph construction.

3 Problem Definition

In this section we formalize our setting by defining token consumption, the structure of the graph, and the optimization objective.

3.1 Token Consumption

In graph-based Retrieval-Augmented Generation (RAG), the **graph construction phase** is a primary driver of token consumption. This process can be so resource-intensive that Microsoft’s official documentation explicitly advises users to start with small datasets to manage costs and processing time [20]. Therefore, a thorough analysis of token usage is essential for evaluating the overall computational overhead and efficiency of these systems. For the purposes of this paper, we categorize tokens into three distinct types:

- **Input Tokens:** These are the tokens provided to the LLMs as context. This includes the content of the document or passage being processed, as well as any system prompts and task-specific instructions.
- **Output Tokens:** Also known as completion tokens, these are the tokens generated by the LLM during the autoregressive decoding process. The generation of these tokens is typically more computationally intensive per token compared to processing input tokens, as they must be produced sequentially, one after another [29].
- **Total Tokens:** This represents the sum of input and output tokens, formally expressed as $T_{\text{total}} = T_{\text{in}} + T_{\text{out}}$.

Through comparing the token consumption of different RAG methods, particularly the total tokens and output tokens, we obtain a practical evaluation metric for assessing the efficiency of graph construction and retrieval.

3.2 Graph Definition

To address the high token consumption outlined above, our framework builds a knowledge graph specifically designed for efficiency. Unlike prior works that often construct based on triple extraction and multi-relational between entities, we adopt a simpler, more streamlined structure. This graph is composed of only essential semantic units which prove sufficient for robust reasoning while drastically reducing the cost of construction.

Graph Type We use a directed, unweighted graph. We represent the graph as $G = (V, E)$, where $E \subseteq V \times V$ denotes directed edges. Since our graph is unweighted, we simply record the existence of an edge (u, v) . In practice, we store E as adjacency lists for efficient neighborhood expansion.

Node Types The node set is

$$V = V_{\text{pas}} \cup V_{\text{con}},$$

where V_{pas} are *passage* nodes and V_{con} are *concept* nodes. Concept nodes include both named entities and broader document-level concepts extracted by LLM. We apply normalization and remove duplicates to merge repeated concepts before edge construction.

Edge types We define three types of edges between nodes:

From-passage edges (E_{pa}) For each concept node $u \in V_{\text{con}}$ and its supporting passage $p \in V_{\text{pas}}$, we add a directed edge $(u \rightarrow p)$ labeled `has_passage`, preserving provenance information.

Co-Occurrence edges (E_{co}) If two concept nodes $u, v \in V_{\text{con}}$ appear in the same passage, we add bidirectional edges $(u \rightarrow v)$ and $(v \rightarrow u)$ labeled `co_occurrence`, avoiding duplicates to reduce graph density.

The complete edge set is therefore

$$E = E_{\text{pa}} \cup E_{\text{co}},$$

with co-occurrence and cluster edges treated as single bidirectional pairs to improve efficiency in downstream retrieval.

3.3 Objective

The objective of our framework is to balance token consumption and retrieval effectiveness in knowledge graph construction for RAG. Unlike prior approaches that pursue maximum accuracy regardless of cost, we focus on reducing the total token consumption T_{total} while retaining acceptable task performance. Formally, we aim to solve the following trade-off problem:

$$\min T_{\text{total}} \quad \text{subject to} \quad \text{Accuracy}(RAG(G)) \geq \delta \quad (1)$$

where δ denotes a task-dependent performance threshold. The concrete evaluation metrics used to instantiate $\text{Accuracy}(RAG(G))$ (e.g., Exact Match, F1) are described in the experimental section. Given that our primary goal is to reduce token consumption by one to two orders of magnitude, we set δ relative to a strong baseline, requiring our framework to achieve at least 80% of the accuracy obtained by AutoSchemaKG combined with HippoRAG1 (Full-KG) on each dataset. This pragmatic threshold ensures that our method remains highly effective and competitive for practical applications.

4 Pipeline

In this section, we provide a detailed description of our pipeline for constructing a Knowledge Graph from source documents.

4.1 Named Entity and Document-Level Concept Extraction

Inspired by MiniRAG [7] and the recent survey of Wang et al. [23], which proposes a unified taxonomy of conceptualization across multiple semantic levels (entity, event, document, and system), we deliberately restrict our extraction to only **named entities** and **document-level concepts**. This design choice simplifies KG construction and substantially reduces token consumption, while still preserving the essential semantic units required for effective retrieval.

Named Entities We adopt the definition of Named Entity Recognition (NER) following Petasis et al. [24], where “a Named Entity (NE) is a proper noun serving as a name for something or someone.” Our goal is to extract canonical mentions of salient entities.

Formally, given a passage p , the NER model extracts a set of entity mentions:

$$\mathcal{E}(p) = \{e_1, e_2, \dots, e_m\}.$$

All extracted entities across passages are aggregated into a unified concept node set:

$$V_{\text{ent}} = \bigcup_{p \in V_{\text{pas}}} \mathcal{E}(p),$$

where V_{pas} denotes the set of passage nodes in the knowledge graph.

Each entity $e \in V_{\text{ent}}$ is treated as an atomic node in the KG.

Document-Level Concepts As defined by Wang et al. [23], document-level conceptualization abstracts information at the passage or document scale, capturing the main ideas and context beyond individual entities or events.

Prompt Design To improve the effectiveness of both NER and concept extraction, we adopt a *few-shot prompting* strategy rather than zero-shot instructions as it can significantly improve modern LLMs’ performance [3]. The LLM is provided with several annotated examples, which guide it to produce more accurate and consistent extractions. Furthermore, to minimize token consumption, the model is instructed to output only the extracted entities or concepts directly, instead of generating structured JSON. This design reduces output verbosity and significantly lowers the number of tokens required per query.

Concept Deduplication Since duplicate evidences and concepts can reduce RAG accuracy [16], we apply a strict deduplication procedure that merges only nodes with identical *type* and *name*. This process yields a cleaner and more connected knowledge graph while minimizing the introduction of noisy nodes. Such strict merging is particularly important during the query phase, as named entities extracted from queries will be aligned with their corresponding concept nodes in the graph.

Formally, let $\mathcal{V}_{\text{con}} = \{(t_i, n_i)\}_{i=1}^N$ denote the set of extracted concept nodes, where t_i and n_i represent the *type* and *name* of the i -th concept. The deduplicated set of concept nodes $\mathcal{V}_{\text{con}}^*$ is defined as

$$\mathcal{V}_{\text{con}}^* = \{(t, n) \mid \exists i \text{ s.t. } (t, n) = (t_i, n_i)\}, \quad (2)$$

which ensures that each unique pair of *type* and *name* appears only once in the knowledge graph.

4.2 Graph Construction

Based on the extracted entities and concepts, and cluster results we create a graph by adding three types of nodes mentioned in the problem definition.

Passage Linkage Each concept node is linked to the passage from which it was extracted. This preserves provenance and ensures that retrieval can always trace nodes back to their textual source. These edges are directed from V_{con} to V_{pas} , as defined in Section 2.2.

Co-Occurrence Edges If two nodes appear in the same passage, we create a bidirectional `co_occurrence` edge between them. This encodes local contextual associations between entities and concepts.

4.3 Retrieval

In the retrieval phase, we adopt a lightweight design that minimizes LLM usage. The LLM is only applied for query-level NER and final answer generation, while the core retrieval relies entirely on non-LLM methods to reduce token consumption.

Query NER Similar to entity and concept extraction in document processing, we use a few-shot prompt to extract named entities from the query. These extracted items are then matched against the concept node set V_{con} in the knowledge graph. We first attempt exact string matching; if no matches are found, we select the top-3 nodes with the highest semantic similarity (based on embeddings). The resulting matched nodes are used to construct a *personalized dictionary* that serves as the restart distribution for PPR.

Personalized PageRank We run a PPR algorithm on the knowledge graph, biased toward the query-relevant nodes identified in the previous step. Each matched node u is assigned a weight based on two factors: semantic relevance and frequency. For nodes matched by exact string matching, the semantic weight is set to 1. For nodes matched by semantic similarity, the semantic weight is given by the similarity score $s(u)$ between the query and the node embedding. In both cases, the frequency weight is defined as the inverse of the node frequency $f(u)$ in the corpus. Formally, the unnormalized weight $w(u)$ is defined as

$$w(u) = \frac{s(u)}{f(u)}, \quad (3)$$

where

$$s(u) = \begin{cases} 1, & \text{if } u \text{ is an exact match,} \\ \text{sim}(q, u), & \text{if } u \text{ is selected by semantic similarity.} \end{cases} \quad (4)$$

To avoid imbalance caused by differing numbers of exact and semantic matches, the weights are normalized within each group:

$$\hat{w}(u) = \frac{w(u)}{\sum_{v \in G} w(v)}, \quad u \in G \quad (5)$$

where G denotes either the exact-match group or the semantic-match group. The final personalized dictionary is constructed from $\hat{w}(u)$ across both groups, and serves as the teleportation vector for PPR. After running PPR on the knowledge graph, we rank passages by their visiting frequencies and select the top 5 passages. These passages are then provided to the reader model for final answer generation.

5 Experiment and Result

This section describes the experimental setup and reports the results.

5.1 Experimental Setup

Datasets. Following AutoSchemaKG, we evaluate our method on three benchmark multi-hop QA datasets: MuSiQue, HotpotQA and 2WikiMultihopQA [27, 28, 13]. These datasets are established benchmarks for multi-hop QA, each emphasizing distinct aspects such as connected reasoning (MuSiQue), explainability through supporting facts (HotpotQA), and structured evidence with reasoning paths (2WikiMultihopQA). Together, they provide a diverse and rigorous benchmark for evaluating multi-document reasoning.

Models Used To ensure consistency with the data from AutoSchemaKG, we employ Meta’s LLaMA-3.1-8B-Instruct for entity and concept extraction, and LLaMA-3.3-70B-Instruct for answer generation. These models exhibit strong reasoning and summarization capabilities, and being open-source, they are particularly suitable for our study [1].

Baseline and Metrics For retrieval accuracy, we compare our method against several representative RAG approaches, including LightRAG, MiniRAG, GraphRAG, and AutoSchemaKG. We select LightRAG and MiniRAG because they are designed as lightweight graph-based RAG methods. We include GraphRAG as it is one of the most widely adopted graph-based RAG approaches, and AutoSchemaKG because it directly inspired our design. For efficiency, measured in terms of token consumption, we use LightRAG, MiniRAG, and AutoSchemaKG (with HippoRAG1 module) as references.

For evaluation, we report two standard metrics EM and F1, as well as token consumption:

- **Exact Match (EM).** EM measures the proportion of predictions that exactly match the ground-truth answer string after standard normalization (e.g., lowercasing and punctuation removal). Formally, if y_i denotes the predicted answer for the i -th question and y_i^* its ground truth, EM is defined as

$$\text{EM} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[y_i = y_i^*] \quad (6)$$

where $\mathbf{1}[\cdot]$ is the indicator function and N is the number of questions.

- **F1 score.** F1 measures the token-level overlap between predictions and ground-truth answers, capturing both precision and recall. Let P_i and G_i denote the sets of tokens in the predicted and ground-truth answers for the i -th question. Precision and recall are defined as

$$\text{Precision}_i = \frac{|P_i \cap G_i|}{|P_i|}, \quad (7)$$

$$\text{Recall}_i = \frac{|P_i \cap G_i|}{|G_i|}. \quad (8)$$

The F1 score for the i -th instance is then

$$\text{F1}_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \quad (9)$$

and the overall F1 is obtained by averaging across all N questions.

- **Token Consumption.** As defined in Section 2, we report input, prompt, and output tokens as our efficiency metric.

5.2 Retrieval Results

Table 1: Retrieval accuracy (EM/F1) and relative output token usage of LightRAG, MiniRAG, and TERAG on MuSiQue, 2Wiki, and HotpotQA datasets.

Model	MuSiQue			2Wiki			HotpotQA		
	EM	F1	Rel. (%)	EM	F1	Rel. (%)	EM	F1	Rel. (%)
LightRAG	20.0	29.3	2753	38.6	44.6	1434	33.3	44.9	1041
MiniRAG	9.6	16.8	4145	13.2	21.4	1602	47.1	59.8	2553
TERAG	18.8	29.6	100	51.2	57.8	100	46.9	59.8	100

The retrieval accuracy compared with other lightweight graph-based RAG methods on the three datasets is summarized in Table 1. Using LLaMA-3 8B as the graph construction model and LLaMA-3 70B as the reader model, our token-efficient graph framework outperforms two popular lightweight graph-based RAG systems, LightRAG and MiniRAG, on most tasks while consuming only 3–10% of their tokens. Compared with AutoSchemaKG, our framework also meets the predefined performance target from Section 3.3, which requires

Table 2: Retrieval accuracy (EM/F1) on MuSiQue, 2Wiki, and HotpotQA datasets. Results for all baseline RAG methods are taken from [2], while TERAG is reported from our own experiments. Best results in each column are highlighted in bold.

Model	MuSiQue		2Wiki		HotpotQA	
	EM	F1	EM	F1	EM	F1
Baseline Retrievers						
No Retriever	17.6	26.1	36.5	42.8	37.0	47.3
Contriever	24.0	31.3	38.1	41.9	51.3	62.3
BM25	20.3	28.8	47.9	51.2	52.0	63.4
Existing Graph-based RAG Methods						
GraphRAG	27.3	38.5	51.4	58.6	55.2	68.6
LightRAG	20.0	29.3	38.6	44.6	33.3	44.9
MiniRAG	9.6	16.8	13.2	21.4	47.1	59.9
AutoSchemaKG + HippoRAG1	23.6	36.5	54.8	63.2	50.0	65.3
TERAG (Ours)	18.8	29.6	51.2	57.8	46.9	59.8
Target (80% of AutoSchemaKG + HippoRAG1)	18.8	29.2	43.9	50.6	40.0	52.2

achieving at least 80% of the accuracy of AutoSchemaKG + HippoRAG1 (Full-KG) on each dataset. The complete accuracy comparison is provided in Table 2. Notably, on the 2Wiki dataset our method achieves accuracy close to the widely used GraphRAG (EM: 51.2 vs. 51.4; F1: 57.8 vs. 58.6) while consuming substantially fewer tokens.

5.3 Token Consumption

While the retrieval accuracy of TERAG is comparable to lightweight graph-based RAG baselines, our key advantage is token efficiency. Table 3 summarizes end-to-end token usage across datasets. On HotpotQA, 2WikiMultihopQA, and MuSiQue, AutoSchemaKG consumes **8.6–11.6×** more completion (output) tokens and **2.9–3.8×** more input tokens than TERAG. This advantage becomes even more pronounced against LightRAG and MiniRAG. LightRAG consumes **19–29×** more input tokens and **10–28×** more output tokens, while MiniRAG uses **11–27×** more input tokens and **16–42×** more output tokens. This highlights that our method spends **88–97%** less token compared with other lightweight graph RAG methods. (see Table 4 for percentages normalized to TERAG=100). This saving is practically important because LLM inference is dominated by the *autoregressive decoding* stage: output tokens are generated one-by-one, and every step must read the accumulated KV cache and append new K/V pairs, which raises per-output-token latency and memory-bandwidth cost; by contrast, input tokens are processed in a parallel prefill pass [29]. Architecturally, our pipeline is token-efficient because we *directly* extract graph concepts from passages in a single pass, avoiding the multi-stage LLM extraction/summarization used by prior graph-RAG systems such as AutoSchemaKG and LightRAG [2, 10].

Figure 3 provides a clear visualization of the relationship between output token consumption and retrieval accuracy across different graph-based RAG

Table 3: Token consumption statistics of our method (TERAG), LightRAG, MiniRAG and AutoSchemaKG across datasets.

Method	Dataset	Input	Output	Total
TERAG (ours)	HotpotQA	2,005,645	562,827	2,568,472
TERAG (ours)	2WikiMultihopQA	1,211,644	368,708	1,580,352
TERAG (ours)	MuSiQue	2,355,941	664,702	3,020,643
AutoSchemaKG	HotpotQA	5,723,733	4,915,796	10,639,529
AutoSchemaKG	2WikiMultihopQA	3,596,676	3,176,095	6,772,771
AutoSchemaKG	MuSiQue	8,960,502	7,715,976	16,676,478
LightRAG	HotpotQA	38,765,230	5,862,363	44,627,593
LightRAG	2WikiMultihopQA	34,222,643	5,288,806	39,511,449
LightRAG	MuSiQue	68,500,000	18,300,000	86,800,000
MiniRAG	HotpotQA	36,909,150	14,370,416	51,279,566
MiniRAG	2WikiMultihopQA	13,877,889	5,906,984	19,784,873
MiniRAG	MuSiQue	62,425,404	27,552,031	89,977,435

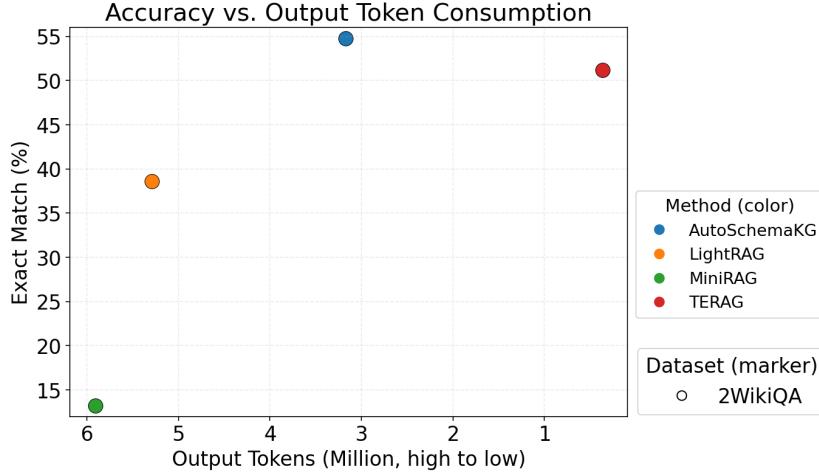
Note: Some data from the HippoRAG2 paper [12].

Table 4: Relative token consumption across datasets. Manually Set TERAG = 100%.

Method	Dataset	Input (%)	Output (%)	Total (%)
TERAG (ours)	HotpotQA	100.0	100.0	100.0
	2WikiMultihopQA	100.0	100.0	100.0
	MuSiQue	100.0	100.0	100.0
AutoSchemaKG	HotpotQA	285.4	873.4	414.2
	2WikiMultihopQA	296.8	861.4	428.6
	MuSiQue	380.3	1,160.8	552.1
LightRAG	HotpotQA	1,940.0	1,041.6	1,737.5
	2WikiMultihopQA	2,824.5	1,434.4	2,500.2
	MuSiQue	2,907.5	2,753.1	2,873.5
MiniRAG	HotpotQA	1,840.3	2,553.3	1,996.5
	2WikiMultihopQA	1,145.4	1,602.1	1,251.9
	MuSiQue	2,649.7	4,145.0	2,978.8

methods. The x-axis represents the number of output tokens generated by each model, while the y-axis reports the EM score, allowing us to directly examine the efficiency–effectiveness trade-off. An ideal token-efficient method would appear toward the upper-right corner, combining high retrieval accuracy with low token usage. Our proposed TERAG consistently lies closest to this desirable region, indicating that it achieves competitive accuracy while consuming substantially fewer tokens. In contrast, AutoSchemaKG attains strong accuracy but only by incurring an order-of-magnitude increase in token usage, making it far less cost-efficient. LightRAG, on the other hand, reduces token usage but at the expense of a sharp drop in accuracy, failing to provide a balanced trade-off. Taken together, these results confirm that TERAG delivers the most favorable efficiency–performance balance among the evaluated methods, highlighting its practicality for large-scale or resource-constrained deployment scenarios.

Fig. 3: Accuracy versus output token consumption on the 2Wiki dataset. The upper-right region indicates higher accuracy with lower token consumption.



6 Ablation Study

Effect of Restart Vector Design. We examine the impact of our new PPR restart vector formulation, which integrates both semantic relevance and concept frequency. Unlike traditional PPR methods that rely solely on uniform or inverse-frequency distributions, our approach aims to produce a more informative restart vector that better captures contextual importance.

Table 5: Ablation study on PPR restart vector calculation. “New“ denotes our method combining semantic relevance and frequency, while “Original“ uses only inverse frequency.

Model/Dataset	MuSiQue		2Wiki		HotpotQA	
	EM	F1	EM	F1	EM	F1
TERAG + New	18.7(+3%)	29.6(+2%)	51.2(+3%)	57.8(+3%)	46.9(+5%)	59.8(+2%)
TERAG + Original	18.3	29.1	49.7	56.1	44.8	58.7

As shown in Table 5, our restart vector design consistently improves retrieval accuracy across all three datasets, yielding gains of 2–5 percentage points in both EM and F1. These results highlight the importance of combining semantic and frequency information to construct a more discriminative restart distribution, leading to more precise passage retrieval.

Model Variation Analysis. We also assessed the influence of switching the answer generation model while keeping the same retrieval graph. Replacing the default model with LLaMA-3.2 11B caused a slight performance drop, while Qwen2 235B achieved accuracy similar to the LLaMA-3 70B but introduced noticeably higher latency. These observations suggest that the dominant performance factor in our

framework is the model’s NER accuracy; improvements in generic reasoning or generation capabilities do not yield proportional gains in retrieval quality.

7 Conclusion and Future Work

To address the high token consumption incurred by current Graph-RAG systems during knowledge graph construction, we propose the TERAG framework. TERAG constructs graph structures by directly extracting named entities and document-level concepts from text in a single pass, thereby eliminating the multiple and costly LLM reasoning calls required by prior approaches.

Despite its lightweight design, TERAG achieves highly competitive retrieval accuracy on multiple multi-hop QA benchmarks while reducing token overhead by one to two orders of magnitude. In addition, because concept extraction, concept filtering, and graph linking are largely independent, these components can be parallelized efficiently. As a result, TERAG not only minimizes indexing-related token cost but also has the potential to significantly shorten graph construction latency in large-scale deployments. Overall, our findings demonstrate that a carefully designed lightweight graph construction pipeline can achieve a more favorable efficiency–performance trade-off than graph-RAG systems that heavily rely on LLMs.

For future work, we plan to explore extracting concepts with controllable levels of granularity, enabling multi-layered knowledge graphs that better capture semantic structure and improve retrieval accuracy. We also aim to incorporate additional engineering strategies to further optimize the balance between token efficiency and model effectiveness.

Finally, due to TERAG’s simple and adaptable pipeline, we expect its performance to remain stable under evolving data streams or more heterogeneous corpora. However, the lack of widely recognized benchmarks for such dynamic and heterogeneous retrieval scenarios limits empirical evaluation. Future research will involve designing appropriate datasets and evaluation protocols to more systematically validate these capabilities.

References

1. AI, M.: Introducing llama 3.1: Our most capable models to date (July 23 2024), <https://ai.meta.com/blog/meta-llama-3-1/>, accessed: YYYY-MM-DD
2. Bai, J., Fan, W., Hu, Q., Zong, Q., Li, C., Tsang, H.T., Luo, H., Yim, Y., Huang, H., Zhou, X., Qin, F., Zheng, T., Peng, X., Yao, X., Yang, H., Wu, L., Ji, Y., Zhang, G., Chen, R., Song, Y.: Autoschemakg: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora (2025), <https://arxiv.org/abs/2505.23628>
3. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark,

- J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners (2020), <https://arxiv.org/abs/2005.14165>
4. Deng, X., Wang, B., Chen, J., Gan, Z., Liu, J., Shi, W., Wang, Y., Wang, F., Zhang, J., Zhang, X.: DB-GPT: Empowering database interactions with private large language models (2023)
 5. Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Larson, J.: From local to global: A graph rag approach to query-focused summarization. ArXiv **abs/2404.16130** (2024), <https://api.semanticscholar.org/CorpusID:269363075>
 6. Edge, D., Trinh, H., Larson, J.: Lazygraphrag: Setting a new standard for quality and cost. <https://www.microsoft.com/en-us/research/blog/lazygraphrag-setting-a-new-standard-for-quality-and-cost/> (Nov 2024), microsoft Research Blog
 7. Fan, T., Wang, J., Ren, X., Huang, C.: Minirag: Towards extremely simple retrieval-augmented generation (2025), <https://arxiv.org/abs/2501.06713>
 8. Fan, W., Li, H., Deng, Z., Wang, W., Song, Y.: Goldcoin: Grounding large language models in privacy laws via contextual integrity theory (2024), <https://arxiv.org/abs/2406.11149>
 9. Ghimire, A., Prather, J., Edwards, J.: Generative ai in education: A study of educators' awareness, sentiments, and influencing factors (2024), <https://arxiv.org/abs/2403.15586>
 10. Guo, Z., Xia, L., Yu, Y., Ao, T., Huang, C.: Lightrag: Simple and fast retrieval-augmented generation (2025), <https://arxiv.org/abs/2410.05779>
 11. Gutierrez, B.J., Shu, Y., Gu, Y., Yasunaga, M., Su, Y.: HippoRAG: Neurobiologically inspired long-term memory for large language models. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems (2024), <https://openreview.net/forum?id=hkujvAPVsg>
 12. Gutiérrez, B.J., Shu, Y., Qi, W., Zhou, S., Su, Y.: From rag to memory: Non-parametric continual learning for large language models (2025), <https://arxiv.org/abs/2502.14802>
 13. Ho, X., Nguyen, A.K.D., Sugawara, S., Aizawa, A.: Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps (2020), <https://arxiv.org/abs/2011.01060>
 14. Hu, Y., Lei, Z., Zhang, Z., Pan, B., Ling, C., Zhao, L.: Grag: Graph retrieval-augmented generation. In: Findings of the Association for Computational Linguistics: NAACL 2025. pp. 4145–4157. Association for Computational Linguistics, Albuquerque, New Mexico (Apr 2025). <https://doi.org/10.18653/v1/2025.findings-naacl.232>, [https://aclanthology.org/2025.findings-naacl.232/](https://aclanthology.org/2025.findings-naacl.232)
 15. Huang, Y., Zhang, S., Xiao, X.: Ket-rag: A cost-efficient multi-granular indexing framework for graph-rag (2025), <https://arxiv.org/abs/2502.09304>
 16. Ko, S., Cho, H., Chae, H., Yeo, J., Lee, D.: Evidence-focused fact summarization for knowledge-augmented zero-shot question answering. In: Al-Onaizan, Y., Bansal, M., Chen, Y.N. (eds.) Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. pp. 10636–10651. Association for Computational Linguistics, Miami, Florida, USA (Nov 2024). <https://doi.org/10.18653/v1/2024.emnlp-main.594>, <https://aclanthology.org/2024.emnlp-main.594/>
 17. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks (2021), <https://arxiv.org/abs/2005.11401>

18. Liu, L., Yang, X., Lei, J., Shen, Y., Wang, J., Wei, P., Chu, Z., Qin, Z., Ren, K.: A survey on medical large language models: Technology, application, trustworthiness, and future directions (2024), <https://arxiv.org/abs/2406.03712>
19. Marcus, G.: The next decade in ai: Four steps towards robust artificial intelligence (2020), <https://arxiv.org/abs/2002.06177>
20. Microsoft: GraphRAG: Unlocking the Power of Private Data with LLM-Powered Graph RAG. <https://github.com/microsoft/graphrag> (2025), accessed: 2025-09-06
21. NebulaGraph: Graph rag: Unleashing the power of knowledge graphs with llm (September 2023), <https://www.nebula-graph.io/posts/graph-RAG>
22. Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., Tang, S.: Graph retrieval-augmented generation: A survey (2024), <https://arxiv.org/abs/2408.08921>
23. Peng, H., Wang, X., Hu, S., Jin, H., Hou, L., Li, J., Liu, Z., Liu, Q.: Copen: Probing conceptual knowledge in pre-trained language models (2022), <https://arxiv.org/abs/2211.04079>
24. Petasis, G., Cucchiarelli, A., Velardi, P., Palouras, G., Karkaletsis, V., Spyropoulos, C.D.: Automatic adaptation of proper noun dictionaries through co-operation of machine learning and probabilistic methods. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 128–135. SIGIR ’00, Association for Computing Machinery, New York, NY, USA (2000). <https://doi.org/10.1145/345508.345563>, <https://doi.org/10.1145/345508.345563>
25. Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., Miller, A.: Language models as knowledge bases? In: Inui, K., Jiang, J., Ng, V., Wan, X. (eds.) Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 2463–2473. Association for Computational Linguistics, Hong Kong, China (Nov 2019). <https://doi.org/10.18653/v1/D19-1250>, <https://aclanthology.org/D19-1250/>
26. Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y., Ni, L.M., Shum, H.Y., Guo, J.: Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph (2024), <https://arxiv.org/abs/2307.07697>
27. Trivedi, H., Balasubramanian, N., Khot, T., Sabharwal, A.: Musique: Multihop questions via single-hop question composition (2022), <https://arxiv.org/abs/2108.00573>
28. Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W.W., Salakhutdinov, R., Manning, C.D.: Hotpotqa: A dataset for diverse, explainable multi-hop question answering (2018), <https://arxiv.org/abs/1809.09600>
29. Zhou, Z., Ning, X., Hong, K., Fu, T., Xu, J., Li, S., Lou, Y., Wang, L., Yuan, Z., Li, X., Yan, S., Dai, G., Zhang, X.P., Dong, Y., Wang, Y.: A survey on efficient inference for large language models (2024), <https://arxiv.org/abs/2404.14294>