HuixiangDou2: A Robustly Optimized GraphRAG Approach

Anonymous Author

Abstract

GraphRAG has achieved significant performance gains in question-answering systems. However, its pipeline is highly engineering-oriented, involving numerous components and parameters that require tuning. This makes it difficult to determine whether the performance gains are due to pipeline optimization or internal parameters. Additionally, with Large Language Model (LLM) training tokens approaching their limits, many public Retrieval Augmented Generation(RAG) QA datasets have been incorporated into LLM training sets. The different input prompts of LLMs can affect the generated results, and identifying which key tokens trigger appropriate outcomes is challenging for non-LLM training personnel. This uncertainty decreases the differentiation of RAG results, leading to inaccurate pipeline evaluations (for example, our tests on citation RAG accuracy improvements on certain small LLMs yielded random results). Given these challenges, this paper does not propose a new method, but selects a domain-specific dataset with low LLM scores. By integrating multiple approaches (with over 18k lines of code), we conducted parameter comparisons and ablation studies. Ultimately, we developed a GraphRAG pipeline capable of streaming responses. We release the code[1].

1 Introduction

Retrieval-Augmented Generation (RAG) is an artificial intelligence technique that integrates information retrieval with language generation models. It enhances the ability of large language models (LLMs) to handle knowledge-intensive tasks by retrieving relevant information from external knowledge bases and feeding it to the models as prompts. Typically, RAG consists of three stages: Indexing, Retrieval, and Generation. In its simplest form,

[1]https://github.com/tpoisonooo/huixiangdou2 is our implementation, while the dataset cannot be fully open-sourced due to licensing restrictions

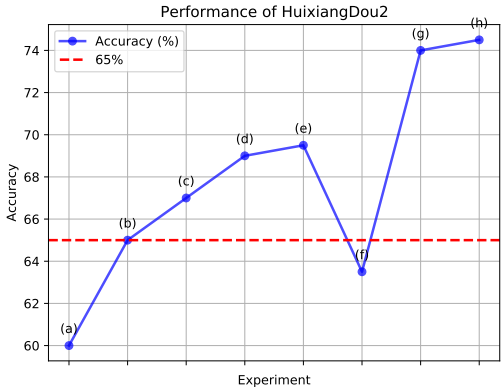

Figure 1: Performance improvements with each experiment – (a) Baseline GraphRAG, (b) Remove abundant zero-shot example during retrieval, (c) Revert LLM $rope\_scaling$ default value, (d) Use 8k length for $nodes$ and $edges$, 12k length for $chunks$, (e) Expand low level keys, (f) Exact matching method, (g) Rever to dual-level method and optimize NER prompt, (h) Fuse logic form retrieval with pre-check.

Indexing and Retrieval rely solely on dense methods (Netease, 2024; langchain, 2023; Kong et al., 2024), which match keyword features by extracting and comparing their distances. For example, the input sentences "**This board is great**" and "**This development board is terrible**" may have a similarity score as high as 0.7 due to their shared topic, despite their opposing semantics (Su, 2016).

To improve generation performance, GraphRAG methods introduce a combination of graph structures and dense methods during Indexing and Retrieval to represent the relationships between pieces of knowledge. For instance, microsoft's GraphRAG (microsoft, 2024) employs community detection algorithms during Retrieval to answer users' summarization questions. KAGs (Liang et al., 2024) expands the knowledge graph into quads and decomposes query into sub-query to enhance logical coherence. LightRAG (Guo et al., 2024), on the other hand, uses a dual-level ap-

Domain	LLM win rate	Length ratio
argriculture	0.97	9.25
art	0.91	8.20
biography	0.82	7.69
cooking	0.88	7.70
cs	0.97	10.77
fiction	0.65	6.63
fin	0.85	9.33
health	0.93	8.97

Table 1: We executed the questions from the UltraDomain (Qian et al., 2024) dataset directly using Qwen2.5-7B-Instruct (Qwen et al., 2025) then queried the Kimi API[2] for its preference between the LLM and the Ground Truth. We calculated the LLM winrate based on preferences and also measured average length ratio of LLM responses to Ground Truth. The results show that on the UltraDomain dataset, 7B model's direct response far exceeds the Ground Truth. This indicates that it is challenging to validate the RAG system effectiveness on such datasets.

proach to decompose the knowledge base and queries, thereby increasing the success rate of fuzzy matching.

Among these three stages, Indexing involves constructing a knowledge graph, often relying on Named Entity Recognition (NER) and requiring the repeated input of the entire knowledge base. This process leads to significant token consumption in LLMs. Therefore, analyzing the Indexing process to clarify its cost-effectiveness is necessary. The Retrieval stage depends on the accuracy of LLMs, such as decomposing queries into low-level and high-level representations to capture entities and relationships, or breaking down query for step-by-step execution. We need to verify the robustness of Retrieval methods against LLM inaccuracies, ensuring that the retrieval results remain unaffected even if LLMs fail to correctly identify entities (e.g., "Zhefu 802").

However, LLM training tokens have covered most publicly available RAG evaluation datasets. Due to the nature of LLMs, input prompts influence the probability distribution of next-token predictions (von Oswald et al., 2023), it is difficult to determine whether performance improvements stem from the GraphRAG method itself or from training data, as shown in table 1.

To clarify the effectiveness of each component of the GraphRAG pipeline, we selected three popular GraphRAG implementations (LightRAG, KAG, and DB-GPT (Xue et al., 2024), with a combined GitHub star count of 40k) and merged their codebases for ablation studies[3]. To avoid issues related to LLMs being too powerful or having seen the RAG test sets, we manually selected domain-specific dataset where LLMs performed average.

2 Background

In this section, we give a brief overview of GraphRAG approach and some of the choices that we will examine experimentally in the following section.

2.1 Architecture

Let the corpus of GraphRAG be denoted as C and the query as Q. A GraphRAG framework, Framew $= (f, g, d)$, consists of three functions:

- f operates during the indexing stage to extract representations of C, resulting in R_c
- g operates during the retrieval stage to obtain representations of Q, resulting in R_q
- d calculates the distance between R_c and R_q

Formally, the indexing and retrieval process can be described as follows, e is an element that could reduce $\text{dist}(S_k, R_q)$.

$$
\begin{aligned}
&R_c = f(C) \\
&R_q = g(Q) \\
&\text{Initialize } S_0 \subseteq R_c \\
&\text{For each iteration } k = 1, 2, \ldots \\
&\quad e_k^+ = \arg \min_{e \in R_c \setminus S_{k-1}} \text{dist}(S_{k-1} \cup \{e\}, R_q) \\
&\quad e_k^- = \arg \min_{e \in S_{k-1}} \text{dist}(S_{k-1} \setminus \{e\}, R_q) \\
&\quad S_k = S_{k-1} \cup \{e_k^+\} \setminus \{e_k^-\} \\
&\quad \text{Stop if } \text{dist}(S_k, R_q) \text{ has no improvement}
\end{aligned}
$$

2.2 Objectives

In the GraphRAG framework, the following subtasks influence the final performance: **Indexing**, **Retrieval**, and **Generation**.

[3]Due to the immense workload (with over 18k lines of code), we could only evaluate the impact of methodological changes and could not ensure identical results before and after codebase integration. However, this does not affect the conclusions of this paper. Meanwhile, the high cost of manually selecting poorly performing LLM datasets limits our findings to a single domain

Graph-Based Indexing The function f in section 2.1 is typically a serial pipeline defined as $f = \text{dump}(\text{ner}(\text{preproc}(C)))$, where each component performs the following operations:

- **Preprocessing (preproc)**: This step involves converting the file format, slicing the corpus C into manageable segments, and generating chunks.
- **Entity & Rel Extraction (ner)**: This step extracts unique ⟨entity, relation, description ⟩tuples from the preprocessed chunks to form a graph structure.
- **Persistence (dump)**: This step establishes persistent connections between entities, relations, and embeddings of chunks, storing them in a database.

This indexing processing is also shown in fig. 2.

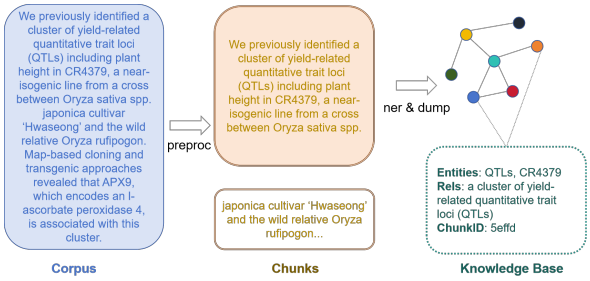

Figure 2: This is the GraphRAG indexing process. We first clean and split the raw corpus into chunks, then extract entities, keywords, relationships, and descriptions, and finally link the graph nodes and edges to the chunks.

Graph-Guided Retrieval The retrieval stage corresponds to g and iterative part in section 2.1. When $k = 1$, the dual-level method is used to decompose the query; when $k > 1$, we choose logic form method.

- The Dual-level method is similar to LightRAG, as shown in fig. 3, it decomposes the query into low-level entities and high-level relationships during retrieval. Entities are fuzz match with nodes in the graph to retrieve associated edges, while relationships are matched with edges to search for related nodes. The results are merged and redundant edges, nodes, and ChunkIDs are removed to obtain the retrieval context. We will demonstrate in the analysis section that this method essentially performs fuzzy retrieval.
- The Logic Form method, inspired by inference framework and KAG, first defines the functionality and parameters of operators. It then leverages an LLM to decompose the query into appropriate operator sequences, solving sub-problems step-by-step to ultimately obtain the retrieval context. The pseudocode is shown in algorithm 1.

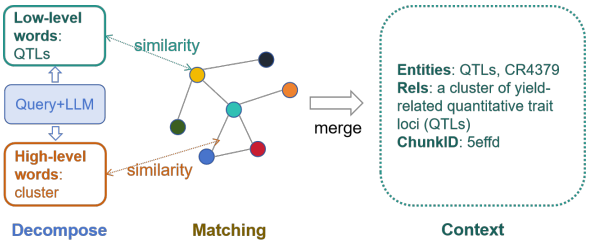

Figure 3: Dual-level retrieval method. User query would be decomposed into low-level and high-level keywords, then match with knowledge graph.

Graph-Enhanced Generation Most of RAG utilize LLMs for both generation and evaluation. For example, input formatting and prompt engineering are used to optimize the generation process, while the LLM assesses whether the generated response answers the question and whether the context is relevant to the query.

2.3 data

UltraDomain and CMedQA are common benchmark data for GraphRAG.

3 Experimental Setup

In this section, we describe the experimental setup for our replication study of GraphRAG.

3.1 Implementation

We reused the conclusions from our previous work on refusal-to-answer task. For text splitting, we employed `ChineseRecursiveTextSplitter` for Chinese text, which considers both maximum length and punctuation positions. For English text, we use `RecursiveCharacterTextSplitter` with an overlap of 32. Both methods had a default chunk size of 768.

To enable the codebase to switch between multiple knowledge bases, we chose TuGraph (TuGraph,

2023) for knowledge graph storage. Since the training data for bce-embedding is from a commercial company, we used bce to extract features for entities and relations in the knowledge graph, with a maximum token size of 512. The extracted features were then saved to Faiss (Douze et al., 2024).

To ensure that our approach does not require high computational resources, the pipeline works on Qwen2.5-7B-Instruct, which is relatively small model.

3.2 Data and Evaluation

We choose the XBench[4] dataset, a domain-specific dataset synthesized by human experts. In XBench, the 1-1 subset consists of single-choice questions and the evaluation metric is accuracy. The questions are presented in one-shot and zero-shot styles.

Table 2 is our partial benchmark result. It proves that Qwen2.5-7B-Instruct is not well-suited for this task and subsequent optimization should be effective.

Subset	Metric	Score
1-1	acc	0.57
1-2	f1	0.71
1-3	rouge	0.16
1-4	rouge	0.35

Table 2: Baseline results of Qwen2.5-7B-Instruct on our domain-specific data. The poor performance of LLM on this data can demonstrate that the subsequent optimization of GraphRAG is effective.

To reduce costs, we only use the one-shot questions from the 1-1 subset of SeedBench for ablation studies.

4 Indexing Procedure Analysis

In this section, we examine the impact of different parameters at GraphRAG indexing stage.

4.1 How Different Loop NER Methods Affect the Precision?

To maximize the recall of the NER task, GraphRAG tends to iteratively call the LLM NER to extract as many entities as possible. The common stopping condition is based on the LLM's judgment of whether any entities have been missed.

We compared two implementations of iterative NER, as shown in algorithm 2. Trial version (trial) follows standard programming logic: it performs NER first, then asks the LLM whether further extraction is needed, and only continues if the LLM responds affirmatively. Baseline version (base) deviates from this logic: it performs NER, informs the LLM that more entities might exist, and requests additional answers before entering the if-condition.

As shown in table 3, the volume of data in the knowledge graph is positively correlated with its precision, and the erroneous entities generated by LLM do not affect the precision. Our insights are as follows:

- More entity results improve precision, thus we can optimize the prompt by specifying entity types and splitting examples. The optimized prompt is provided in the code[5].
- Smaller *chunksize* may yield better results.

Version	Nodes	Edges	Acc
trial	20739	19857	0.61
base	21838	26847	0.69
optimize prompt	29086	35750	0.74

Table 3: The relationship between the number of nodes and edges in a Graph and the precision under different loop NER implementations. **Increasing the quantity can improve precision.** Since erroneous entity words generated by LLMs become isolated nodes, they do not affect precision.

4.2 Impact of LLM Max Context Length on Precision

LLMs typically perform well in needle-in-a-haystack experiments, but RAG systems often need to focus on implicit expressions in the corpus. We modified the parameter $rope\_scaling$ (which can increase the maximum length of the LLM in YaRN (Peng et al., 2023)) and compared the precision differences between the maximum lengths of 32k and 64k. Our empirical conclusion is that a 64k maximum length results in 2% precision decrease. Therefore, **smaller rope_scaling is better**, as shown in table 4.

5 Retrieval & Generation Analysis

In this section, we first validate the parameters that affect the performance. We then compare different retrieval methods and ultimately integrate the two approaches to achieve the optimal effect.

[4]XBench would be introduced in other work.

[5]See huixiangdou/service/prompt/kag.py

Max LLM context length	Acc
32k	0.67
64k	0.65

Table 4: The impact of different maximum lengths of LLM on accuracy, when the model's maximum length is sufficient, we prefer a smaller $rope\_scaling$.

5.1 Impact of representation length on precision

Based on the former conclusions, we assume that the essence of the dual-level method is approximate matching. **The richer the representations parsed from the corpus and query, the higher the precision.** We conducted experiments along two opposite directions.

Increasing the Maximum Length of R_q and Low-Level Keys Since LLMs often struggle with domain-specific data (e.g., mistaking entities for relationships), we expanded the maximum length of the query representation (R_q in section 2.1) and increased the number of low-level keys. This approach aims to provide more context for the query and improve the matching process.

Switching to Exact Matching Instead of concatenating entity list, we independently stored the features of each entity during the indexing phase and matched individual entities during query time. This method aims to improve precision by avoiding the noise introduced by approximate matching.

Our hypotheses are validated in table 5. Increasing the maximum length of R_q to 28k resulted in an improvement in precision 3% compared to baseline. Expanding the number of low-level keys helped fix some rare bad cases. In contrast, switching to exact matching led to a decrease in precision. This suggests that **fuzzy matching is essential for capturing the nuances of the query**, as exact matching fails to account for the implicit keywords derived from the query. This aligns with common sense, as exact matching is too rigid for complex queries.

5.2 Which retrieval method is better ? Dual-level vs. Logic Form

We compare the results of Dual-level and Logic Form in table 6. Although the Dual-level achieves higher precision values, it fails to provide convincing answers to questions that require calculations (e.g., "How much taller is Zhefu 802 than its parent?"). In our real-world scenario tests, human domain experts prefer unambiguous answers, which Logic Form provides. Logic form responses are more concise and contain more logical progression words.

Method	Acc
dual-level	0.65
+28k context length	0.69
+expand low level keys	0.695
exact matching method	0.635

Table 5: We hypothesize that the length of R_q and R_c improve the accuracy. Conducting validations in two opposite directions, the results supported this hypothesis.

Method	Avg output size	Acc
Optimized dual-level	9863	0.74
Logic form	1699	0.55

Table 6: We have tallied average lengths of the contexts outputted by two methods, and then used these retrieval results to answer XBench questions. Despite the Logic Form retrieval method's suboptimal precision performance, it yields a higher information density.

If we had a verifier with 100% accuracy, we could combine the strengths of both methods to achieve an accuracy of 0.77.

5.3 How to build a verifier: Checking Arguments vs. Checking Results

RAG systems often use LLM to verify the correctness of results. We compared two approaches: Checking Arguments (pre-check) and Checking Results (post-check). Pre-check verifies whether the context arguments can answer the question before generating the final response, while post-check examines the question, context, and response together for overall coherence. Pseudocode is shown in algorithm 3.

Our results are show in table 7 which prefer Checking Arguments. From the inference perspective, the response prompt diverts some of the LLM's attention, reducing its ability to focus on the core question; from model's view, Qwen-2.5-7B-Instruct is causal model, correct reasoning in the context implies correct results. Therefore, checking result is redundant.

6 HuixiangDou2

Based on the ablation and analysis, we ultimately constructed the HuixiangDou2 retrieval pipeline,

Method	Acc
post-check	0.715
pre-check	0.745

Table 7: Different check strategy on logic form retrieval method.

as shown in fig. 4.

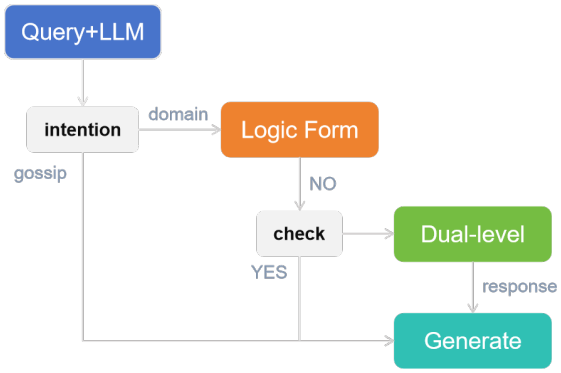

Figure 4: Multi-stage retrieval.

Its precision results are shown in table 8 and features include:

Multi-stage Prioritizing query decomposition, and degrading to fuzzy matching if decomposition fails or verification does not pass. This mechanism ensures that the pipeline always supports streaming responses.

Version compatibility HuixiangDou2 retrain the useful refusual-to-answer and intent slots from previous generation.

Subset	LLM Baseline	HuixiangDou2
1-1	0.57	**0.745**
1-2	0.71	**0.79**
1-3	0.16	**0.36**
1-4	0.35	**0.38**

Table 8: After ablation studies on the baseline, we present some results tested with the new version on XBench.

7 Conclusion

Based on our engineering practice, we find that **Logic Form, with its step-by-step approach, is more readily accepted by human domain experts**. This method aligns well with human reasoning processes and provides clear, logical answers that are easier to validate.

However, several challenges remain. First, constructing effective supervision signals, such as a high-accuracy Verifier, is difficult but crucial for further improving precision. Second, the initial steps decomposed by LLMs often deviate significantly from the knowledge graph, leading to failures in execution and result retrieval. These issues highlight the gap between LLM-generated plans and practical knowledge graph operations.

Future work will focus on addressing these challenges. We plan to explore more robust methods for building verifiers and improving the alignment between LLM-generated steps and knowledge graph structures. Additionally, we aim to integrate hybrid approaches that combine the strengths of both Logic Form and Dual Level methods to achieve better overall performance.

References

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

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

Huanjun Kong, Songyang Zhang, Jiaying Li, Min Xiao, Jun Xu, and Kai Chen. 2024. Huixiangdou: Overcoming group chat scenarios with llm-based technical assistance.

langchain. 2023. Langchain: Building applications with llms through composability.

Lei Liang, Mengshu Sun, Zhengke Gui, Zhongshu Zhu, Zhouyu Jiang, Ling Zhong, Yuan Qu, Peilong Zhao, Zhongpu Bo, Jin Yang, Huaidong Xiong, Lin Yuan, Jun Xu, Zaoyang Wang, Zhiqiang Zhang, Wen Zhang, Huajun Chen, Wenguang Chen, and Jun Zhou. 2024. Kag: Boosting llms in professional domains via knowledge augmented generation.

microsoft. 2024. A modular graph-based retrieval-augmented generation (rag) system.

Netease. 2024. Netease youdao's open-source embedding and reranker models for rag products.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2023. Yarn: Efficient context window extension of large language models.

Hongjin Qian, Peitian Zhang, Zheng Liu, Kelong Mao, and Zhicheng Dou. 2024. Memorag: Moving towards next-gen rag via memory-inspired knowledge discovery.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

Jianlin Su. 2016. What's the deal with word vectors and embeddings?

TuGraph. 2023. Tugraph: A high performance graph database.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers learn in-context by gradient descent.

Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, Wang Zhao, Fan Zhou, Danrui Qi, Hong Yi, Shaodong Liu, and Faqiang Chen. 2024. Db-gpt: Empowering database interactions with private large language models.

A Logic Form Retrieval

Algorithm 1 Logic form retrieval

Input: Operator set $O = \{o_1, o_2, \ldots, o_n\}$, where each $o_i = (\text{operator}_i, \text{function}_i)$
Input: User query Q
Output: History
Decompose query Q into a list of subqueries L using LLM
$L \leftarrow \{(q_1, a_1), (q_2, a_2), \ldots, (q_m, a_m)\}$, where each $a_j \in O$
for $(q_j, a_j) \in L$ **do**
    Identify the corresponding operator o_j for a_j
    Execute $a_j \leftarrow o_j(q_j)$
end for
Concatenate all sub-queries and sub-answers a_j into history
return History

B Loop NER Implementation

Algorithm 2 Loop NER

Initialize input text T
Initialize maximum attempts MAX
Initialize history $H \leftarrow NER\_init(T)$

function TRIAL
    for $i = 0$ **to** MAX **do**
        $continue \leftarrow LLM\_judge(H)$
        if $continue ==$ "no" **then**
            break
        end if
        $result \leftarrow NER\_continue(T, H)$
        $H \leftarrow H \cup result$
    end for
end function

function BASE
    for $i = 0$ **to** MAX **do**
        $result \leftarrow NER\_continue(T, H)$
        $H \leftarrow H \cup result$
        $continue \leftarrow LLM\_judge(H)$
        if $continue ==$ "no" **then**
            break
        end if
    end for
end function

C Verifier Methods

Algorithm 3 Retrieval Verifier

Initialize: `context` $\leftarrow$ `logic_form_retrieve()`

function PRE-CHECK
    if $LLM\_judge(query, context) ==$ "support" **then**
        return $LLM(query, context)$
    end if
end function

function POST-CHECK
    $reply \leftarrow LLM(query, context)$
    if $LLM\_judge(query, context, reply) ==$ "support" **then**
        return $reply$
    end if
end function