Single LLM, Multiple Roles: A Unified Retrieval-Augmented Generation Framework Using Role-Specific Token Optimization

Yutao Zhu[1], **Jiajie Jin**[1], **Hongjin Qian**[2], **Zheng Liu**[2], **Zhicheng Dou**[1]* and **Ji-Rong Wen**[1]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]Beijing Academy of Artificial Intelligence, China
yutaozhu94@gmail.com, dou@ruc.edu.cn

Abstract

Existing studies have optimized retrieval-augmented generation (RAG) across various sub-tasks, such as query understanding and retrieval refinement, but integrating these optimizations into a unified framework remains challenging. To tackle this problem, this work proposes RoleRAG, a unified RAG framework that achieves efficient multi-task processing through role-specific token optimization. RoleRAG comprises six modules, each handling a specific sub-task within the RAG process. Additionally, we introduce a query graph to represent the decomposition of the query, which can be dynamically resolved according to the decomposing state. All modules are driven by the same underlying LLM, distinguished by task-specific role tokens that are individually optimized. This design allows RoleRAG to dynamically activate different modules within a single LLM instance, thereby streamlining deployment and reducing resource consumption. Experimental results on five open-domain question-answering datasets demonstrate the effectiveness, generalizability, and flexibility of our framework.

1 Introduction

Large language models (LLMs) have demonstrated remarkable performance across a wide range of tasks (Brown et al., 2020; OpenAI, 2023; Dubey et al., 2024; DeepSeek-AI et al., 2024). While their super power is driven by extensive parameters and large-scale training data, they still face challenges related to accuracy, reliability, and timeliness. Retrieval-augmented generation (RAG) provides an effective solution to these problems (Kim et al., 2024b; Asai et al., 2024; Chan et al., 2024). By integrating an external retriever, the LLMs can access relevant knowledge based on user input queries, thus producing more accurate and reliable responses. This approach is particularly beneficial for knowledge-intensive tasks, such as open-domain question answering (Petroni et al., 2021).

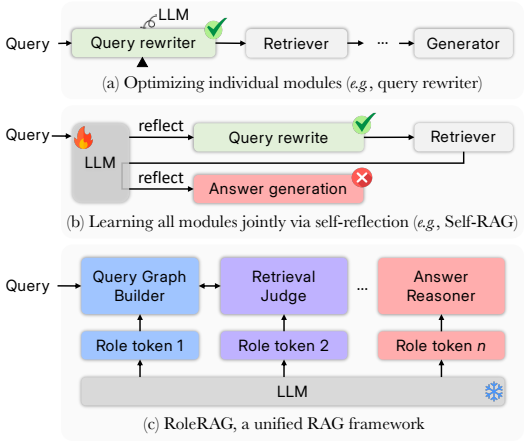


Figure 1: Comparison between existing studies and our framework.

In general, existing studies on optimizing RAG framework can be roughly categorized into two groups. The first group focuses on **improving specific modules** of the framework, as shown in Figure 1 (a). These efforts include introducing a retrieval necessity judgment module to reduce retrieval costs (Tan et al., 2024; Zhang et al., 2024b), improving query understanding to construct more effective queries for retrieving relevant knowledge (Trivedi et al., 2023; Chan et al., 2024), and refining retrieved results to extract key information that helps LLMs generate more accurate responses (Kim et al., 2024b; Jin et al., 2024b). These enhancements have demonstrated improvements in the overall performance of RAG systems. However, integrating these diverse optimizations into a single unified framework is non-trivial. The second group attempts to **consolidate multiple RAG components within a single LLM**, utilizing self-reflection mechanisms to dynamically control the response generation process (Asai et al.,

*Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 4838–4857
November 4-9, 2025

2024; Chan et al., 2024) (Figure 1 (b)). While this approach offers a straightforward framework, it faces two main challenges: (1) introducing new components, such as query rewriter, requires additional data collection and extensive re-training of the LLM, and (2) incorporating more functions increases the complexity of the model's self-reflection mechanism, which may degrade both performance and generalization capabilities.

To address these challenges, we propose **RoleRAG**, **a unified RAG framework** using role-specific token optimization (illustrated in Figure 1 (c)). RoleRAG comprises six specialized modules: query graph builder, retrieval judge, sub-answer generation, summarizer, new query generator, and answer reasoner. The workflow begins with the query graph builder, which decomposes the input query into multiple sub-queries and constructs a directed acyclic graph. For each sub-query, the retrieval judge determines whether additional knowledge retrieval is necessary or if it can be answered directly. Based on this decision, the sub-answer generator produces a response. For sub-queries requiring external knowledge, the summarizer extracts key information from the retrieved content and updates an answer memory dictionary that stores the sub-query, retrieved data, and generated answer. Once all sub-queries are processed, the new query generator examines the answer memory and the original query to determine if further sub-queries are needed. Finally, the answer reasoner synthesizes the final response.

RoleRAG employs a **role-specific token optimization strategy** to implement these modules. By introducing additional special tokens and optimizing their embeddings using task-specific data, the framework enables each module to perform its designated function effectively. Importantly, only the role tokens are tuned during training, making the training computationally efficient. During inference, a single LLM instance is deployed, with different role tokens acting as soft prompts to dynamically activate the corresponding modules. These modules collaborate in an iterative manner to generate the final response. Experimental results on five open-domain question-answering datasets demonstrate that RoleRAG achieves performance improvements of 16%-64% over state-of-the-art RAG methods in terms of exact match score. Additional experiments further confirm the generalizability and robustness of our framework.

Our contributions are three-fold:

(1) We introduce a unified RAG framework using a role-specific token optimization strategy. By integrating a frozen backbone LLM with adaptive role tokens, the model can specialize in different modules of the RAG pipeline and collaborate effectively to complete the full process.

(2) We propose a query graph construction approach to improve the handling of complex queries in RAG. Our framework dynamically refines the query graph by eliminating redundant sub-queries and generating new ones when necessary, enhancing retrieval efficiency and relevance.

(3) We release a comprehensive dataset to train different RAG modules. To our best knowledge, this is the first dataset covering the entire pipeline of a RAG system.

2 Related Work

Retrieval-Augmented Generation Retrieval-augmented generation (RAG) integrates a retrieval module that accesses external knowledge to enhance generation quality (Lewis et al., 2020; Shi et al., 2024; Jiang et al., 2023b; Trivedi et al., 2023; Shao et al., 2023). Efforts to improve RAG systems have been made in different aspects. For example, some studies have aimed to improve query understanding, thus improving retrieval accuracy and overall generation quality (Chan et al., 2024; Verma et al., 2024). Others have investigated the necessity of retrieval to minimize unnecessary retrieval calls, which in turn improves system efficiency and reduces the impact of irrelevant knowledge (Tan et al., 2024; Asai et al., 2024). Additionally, refining retrieval results has been explored to mitigate the need for processing extensive input lengths and to reduce retrieval-related noise (Jiang et al., 2023a; Xu et al., 2024). In a different vein, some studies have tried to advance the RAG pipeline, exploring strategies such as enabling LLMs themselves to determine when retrieval is beneficial during generation (Jiang et al., 2023b), interleaving retrieval with chain-of-thought reasoning (Trivedi et al., 2023), and synergizing retrieval and generation in an iterative manner (Shao et al., 2023). Our study improves the RAG pipeline by dividing sub-tasks within RAG into different modules and employing role-specific token optimization to selectively activate various LLM capabilities using designated role tokens. As all modules are carefully fine-tuned, the overall performance of our framework can be effectively improved.

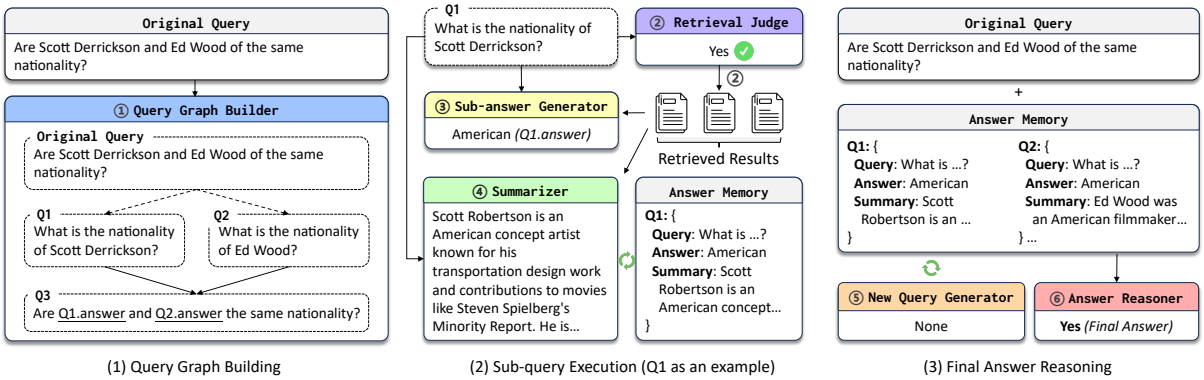


Figure 2: The illustration of our RoleRAG framework, which contains three main steps: (1) query graph building, (2) sub-query execution, and (3) final answer reasoning.

Query Understanding and Decomposition Query understanding aims at inferring the intent of a user query, which is a critical component in a retrieval system (Arens et al., 1996; Azad and Deepak, 2019; Ma et al., 2023). It involves a series of techniques such as query classification, expansion, rewriting, and suggestion. Recent studies have indicated that query understanding is also very important in RAG systems (Ma et al., 2023; Mao et al., 2024), as it determines the integration of external knowledge into the generation process. Among these studies, query decomposition has been particularly effective for handling complex queries by dividing them into more manageable sub-queries, thereby enhancing the accuracy of the retrieval process (Chen et al., 2024; Chan et al., 2024; Trivedi et al., 2023). However, these methods either rely on large-sized API-based models for query decomposition or are unable to adjust the query plan dynamically. In contrast, our RoleRAG introduces a dynamic query graph, where each sub-query can be adjusted dynamically according to the system's memory state. Besides, RoleRAG is built entirely on open-source LLMs, which significantly enhances its practical applicability and efficacy.

3 Methodology

In this work, we introduce a unified RAG framework RoleRAG, which has two characteristics: (1) We propose a role-specific token optimization strategy, in which the LLM's parameters are frozen while only the embeddings of added role tokens are tuned. By this means, all modules in our framework can share the same base LLM and address specific tasks by integrating role tokens into the input. (2) We design a query graph builder that decomposes a user query into multiple sub-queries and organizes them as a directed acyclic graph (DAG). By dynamically resolving each sub-query node, the final answer can be more accurately generated.

3.1 Module Design

As shown in Figure 2, RoleRAG divides the RAG process into three stage: (1) query graph building, (2) sub-query execution, and (3) final answer reasoning. In the first stage, a *Query Graph Builder* decomposes the user input query Q into n sub-queries $\{q_i\}_{i=1}^{n}$, forming a DAG $G(Q)$. Then, in the second stage, for each sub-query $q_i \in G(Q)$, a *Retrieval Judge* evaluates whether it can be directly answered by the LLM; if not, it will call a retriever to get relevant knowledge. An *Answer Generator* then produces an answer a_i for q_i, utilizing the retrieved knowledge if required. Simultaneously, a *Summarizer* condenses the retrieved knowledge relevant to the generated answer. Each resolved sub-query, along with its answer and summary, is stored in an answer memory M. Upon resolving all sub-queries, the framework moves into the third stage, where a *New Query Generator* examines M and Q to determine if additional knowledge is needed, potentially generating new queries handled in the same manner as the previous sub-queries. Finally, an *Answer Reasoner* synthesizes the final answer A from the accumulated data in the answer memory M. This systematic approach ensures comprehensive and accurate query resolution. Below, we introduce each module briefly.

Query Graph Builder The query graph builder constructs a representation of a user's reasoning plan as a DAG. As illustrated in the left side of Figure 2, the original query is decomposed into two sub-queries (*i.e.*, Q1 and Q2), and is ultimately represented by a final node (*i.e.*, Q3). The de-

Table 1: The input and output of each module in our framework. The content in brackets depends on the result of retrieval judgment.

Module	Input	Output	# Training samples
Query Graph Builder	Original query	Query graph	25,654
Retrieval Judge	Sub-query, answer memory	"Yes" / "No"	65,823
Sub-answer Generator	Sub-query, (retrieved result)	Answer	65,823
Summarizer	Sub-query, (retrieved result)	Summary	53,002
New Query Generator	Original query, answer memory	New query / "None"	25,654
Answer Reasoner	Original query, answer memory	Final answer	25,654

pendency among sub-queries is described using a parent-child relationship, where a node becomes a child if its resolution relies on the answers from preceding nodes. Within each node, placeholders are utilized to denote the answers from parent nodes (*e.g.*, Q1.answer and Q2.answer in Q3), which are substituted with actual values during the execution process. This DAG structure ensures that the reasoning plan follows the Markov assumption, allowing for the resolution of the final node once all preceding nodes have been addressed.

Retrieval Judge Previous studies have indicated that not all user queries require external retrieved knowledge, and in some cases, irrelevant knowledge may even hurt the LLM's performance (Yoran et al., 2024a; Tan et al., 2024). Therefore, we design a retrieval judge module to determine whether a sub-query can be directly resolved by the LLM. Only if the judgment result is "False", the retriever $\mathcal{R}$ will be called and provide relevant knowledge $K = \mathcal{R}(q_i)$. To improve the judgment accuracy, the LLM is provided with access to an answer memory (described later), which contains information from previous sub-queries. This setup enables the retrieval judge to perform a *removing* operation on the query graph, effectively minimizing unnecessary retriever activations and thereby enhancing the efficiency of the overall system.

Sub-answer Generator The sub-answer generator is tasked with producing responses based on the sub-queries and any associated retrieved knowledge. Due to the more focused nature of the sub-queries compared to the original query, the answers generated are typically more accurate. Upon generating a sub-answer, it is stored in an answer memory M, which is structured as a Python dictionary. Each sub-answer is keyed by its corresponding sub-query identifier for easy retrieval and reference. For instance, if the active sub-query is "Q1", both the sub-query content and its answer are stored in the dictionary M under the key "Q1". This method ensures organized and efficient management of generated answers throughout the processing sequence.

Summarizer When the retriever is activated, the retrieved knowledge will facilitate the answer generation process. Since the sub-queries may be dependent, it is valuable to also store the retrieved knowledge for future sub-query resolution process. However, directly storing all retrieved knowledge is ineffective. Our observations indicate that the answer to a sub-query often serves as a bridge between related sub-queries. Therefore, we introduce a summarizer that condenses the retrieved knowledge K while prioritizing the essential information about the answer.

New Query Generator Once all sub-queries have been answered, the final answer can be inferred from the answer memory. However, not all sub-queries can be ideally resolved. Therefore, we introduce a new query generator module that suggests additional sub-queries when necessary. This module takes the original query Q and the answer memory M as input, and outputs either a new sub-query q_j or a termination signal ("None"). If a new query is generated, it will be added into the graph as a child of the final node and resolved by the sub-query execution process. Functionally, this module performs an *addition* operation on the query graph, enriching it with supplementary sub-queries to incorporate additional knowledge, thereby improving the completeness of the retrieval.

Answer Reasoner As the final step, the answer reasoner leverages the original query Q and the updated answer memory M to drive the final answer A. Since the answer memory retains all sub-queries, their corresponding answers, and relevant knowledge summary, the answer reasoner can synthesize such information to generate a well-grounded response to the original query.

3.2 Data Collection

Manually annotating data for training each component in `RoleRAG` requires extensive human re-

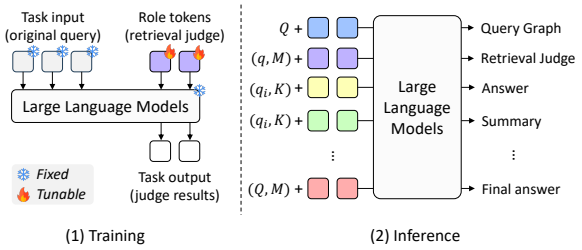


Figure 3: Illustration of the training and inference processes in our framework.

sources, which is impractical for our research. Consequently, we employ an expert LLM, specifically `Llama-3.1-70B-Instruct`, to generate training data automatically. Specifically, we conduct the RAG process following our designed workflow, construct the input of each component using different prompts (provided in Appendix B), and record the corresponding output as raw data. By this means, we can automatically collect amounts of data without human intervention.

However, the absence of golden annotations for evaluating the quality of each component's output poses a challenge. To address this, we borrow the idea of outcome reward models from reinforcement learning (Uesato et al., 2022; Yu et al., 2024) and use the *final answer quality* as a delegate for evaluation. Concretely, we compare the final answer A generated by the expert model with the golden answer $\hat{A}$ provided by the dataset and compute the metric as $s = g(A, \hat{A})$.[1] Data samples only where the final answer score s exceeds a predetermined threshold α are retained. Recent studies (Uesato et al., 2022) have shown that the outcome reward model can provide effective signals for model performance verification, so we believe our strategy can ensure the high-quality of the generated data. Table 1 shows the statistics of our collected data.

3.3 Training Strategy

`RoleRAG` contains six interconnected modules, making it challenging for LLMs to learn and balance their abilities across different tasks. An ideal training strategy should meet three requirements: (1) It should be parameter-efficient as tuning LLMs is often expensive; (2) It should maintain the LLM's general ability as it may serve for different purposes in practice (including non-RAG scenarios); and (3) It should help the LLM understand various tasks in different components while facilitating seamless extension to new tasks.

To tackle these challenges, we propose a role-specific token optimization strategy, illustrated in Figure 3. The core idea is to use specialized role tokens to facilitate task-specific behavior in LLMs. We implement this by expanding the LLM's vocabulary with new special tokens designated for optimization, thereby preserving the integrity of the LLM's parameters. During training, only these newly added tokens' embeddings are tuned, ensuring parameter efficiency while preserving the original LLM weights. This naturally satisfies the first two requirements. For the third requirement, since the added tokens are role-specific, they can be tailored for the task and not affect each other in training. Besides, it is easy to extend our framework with new modules by adding new role tokens. Specifically, for a specific task T and a sample input X^T, we add several new tokens $[t_1, \ldots, t_n]$ and reformulate the input as $[X; t_1; \ldots; t_n]$, where $[;]$ is the concatenation operation. The next-token prediction objective can be defined as:

$$p = \prod_{i=1}^{m} p_{\theta,\delta}(y_i^T | X^T; \underbrace{t_1; \ldots; t_n}_{trainable}; y_{<i}^T), \quad (1)$$

where $Y^T = [y_1^T, \ldots, y_m^T]$ is the target output, $\delta \in \mathbb{R}^{n \times d}$ represents the trainable parameters of the role tokens (*i.e.*, their embeddings), and d is the embedding size of the LLM. θ denotes the parameters of the backbone LLM, which are *frozen* during training. Given that $|\delta| \ll |\theta|$, this method is *highly efficient*. For example, with the Llama-3-8b model (where $d = 4,096$), introducing $n = 30$ tokens results in only 0.1M parameters.

The inference stage is shown in the right side of Figure 3. `RoleRAG` only deploys a single LLM, where task-specific role tokens are appended to the input to guide the LLM in performing different tasks effectively.

4 Experiments

4.1 Datasets and Evaluation Metrics

We conduct our experiments on five question-answering (QA) datasets: HotpotQA (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), 2WikimultihopQA (Ho et al., 2020), Bamboogle (Press et al., 2023), and PopQA (Mallen et al., 2023). The details of these datasets are presented in Appendix C. We mix the training set of HotpotQA, MuSiQue, and 2WikimultihopQA for constructing

[1]The metrics can be exact matching score or F1 score that are commonly used in answer evaluation.

our training set, and the remaining two datasets are used as head-out datasets. Among these datasets, PopQA (Mallen et al., 2023) is the only dataset consisting merely of single-hop queries, which allows us to evaluate the generalizability of our approach to simpler queries. For evaluation, we primarily use the test sets provided by each dataset. If a test set is unavailable, we substitute it with the development set. Importantly, although some datasets have provided golden reference passages for the answer, we choose to use only the passages retrieved from the retrieval sets in both training and inference stages, which aligns with practical applications. Exact match (EM) and F1 score are employed as evaluation metrics.

4.2 Baselines

In addition to comparing with direct generation, we consider two kinds of RAG methods as baselines:

(1) **Sequential pipeline**: These methods follow a standard *retrieve-then-read* flow and focus on improving specific RAG components (*e.g.*, query rewriting). Six representative methods are selected, including Standard RAG, SKR (Wang et al., 2023), SuRe (Kim et al., 2024b), Trace (Fang et al., 2024), Adaptive-RAG (Jeong et al., 2024), and BlendFilter (Wang et al., 2024).

(2) **Iterative pipeline**: This kind of method adjusts the sequential RAG pipeline by involving multiple cycles of retrieval and generation to refine outputs iteratively. We select five typical methods as baselines, including Self-RAG (Asai et al., 2024), IRCoT (Trivedi et al., 2023), Iter-Retgen (Shao et al., 2023), RetRobust (Yoran et al., 2024b), and RQ-RAG (Chan et al., 2024).

Notably, some recent API-based models (Verma et al., 2024; Kim et al., 2024a) are not selected as baselines, because they do not integrate seamlessly with open-source LLMs. Our code is available on GitHub,[2] and the implementation details are provided in Appendix D.

4.3 Experimental Results

The experimental results are shown in Table 2. It is evident to see that our RoleRAG significantly outperforms other baseline methods on all five datasets. This clearly demonstrates the superiority of our method. We have further observations as:

(1) RAG methods generally outperform direct generation by a large margin, highlighting the advantage of integrating external knowledge for knowledge-intensive tasks. Specifically, iterative pipeline methods perform better than sequential pipeline methods. This is particularly evident in scenarios involving multi-hop queries, where the complexity often hinders the retriever's ability to gather all relevant information, leading to suboptimal generation performance. (2) Our RoleRAG achieves the best performance in both in-domain and out-of-domain evaluations. This indicates that our proposed dynamic query graphs and multi-task prompt tuning effectively enhance RAG performance and exhibit strong generalizability. (3) On the single-hop QA dataset (PopQA), some iterative pipeline methods (*e.g.*, RQ-RAG) underperform compared to sequential pipelines. This can be potentially attributed to the overly complex processing applied to relatively simple queries, which introduces unnecessary noise. In contrast, RoleRAG can construct graphs with fewer nodes to represent simpler queries, which is more accurate and efficient. (4) We notice a poor performance of Self-RAG on several datasets, which has also been reported by recent studies (Zhang et al., 2024a). By carefully checking its output, we find that Self-RAG tends to generate long reasoning paths that eventually mislead itself to generate incorrect answers. This may stem from its training strategy, which integrates all modules into a single generation process. Conversely, RoleRAG employs independent training for each module using role tokens, which clarifies and simplifies the tasks each module must learn, thereby improving overall performance.

4.4 Further Analysis

Ablation Study We conduct comprehensive experiments to explore the contribution of each module in our framework, with results shown in Table 3. Our analysis first focuses on the query graph builder by examining two variants: complete removal (#2) and replacement with a prompt-based query decomposition approach (#3). The results indicate that query decomposition is crucial for handling complex queries, and that LLMs struggle to perform this task effectively through direct prompting, highlighting the significance of our graph-based approach. The retrieval judge component demonstrates an interesting trade-off: while it causes a marginal decrease in performance due to reduced knowledge incorporation (#4), it substantially reduces retrieval costs, thereby improving system efficiency. To evaluate the summarizer's

[2] https://github.com/DaoD/RoleRAG

Table 2: Experimental results of all methods using `LLaMA-3.1-8b` as the backbone model. The left three datasets are used for training `RoleRAG`, representing in-domain evaluation, while the right two datasets are used for out-of-domain evaluation. The best and second-best results are highlighted in **bold** and underlined, respectively.

	HotpotQA		MuSiQue		2WikiMultihopQA		Bamboogle		PopQA	
Method	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
Direct Generation	16.20	25.15	3.30	9.30	16.50	26.30	9.60	16.13	11.10	20.65
Sequential pipeline										
Standard RAG	29.50	40.00	4.30	10.28	15.20	25.40	18.40	24.55	25.80	41.34
SKR	24.20	34.85	3.40	9.67	15.70	26.50	12.80	19.43	19.40	32.04
SuRe	23.80	36.24	5.20	10.05	10.20	18.00	16.80	25.96	27.60	44.94
Trace	26.00	35.30	5.60	11.30	9.50	15.80	13.60	19.60	26.60	39.29
Adaptive-RAG	31.70	43.45	9.50	15.57	25.20	36.40	25.60	35.39	26.10	36.14
BlendFilter	34.90	45.56	7.70	13.54	24.30	33.19	22.40	31.04	25.40	41.01
Iterative pipeline										
Self-RAG	9.00	18.46	0.90	4.80	3.70	17.32	4.00	9.07	6.50	16.75
IRCoT	30.50	40.62	9.70	15.42	27.60	36.20	30.40	41.10	29.00	35.64
Iter-Retgen	32.00	42.43	6.50	12.34	16.80	27.14	20.00	26.84	26.50	40.87
RetRobust	27.20	30.10	12.10	14.70	32.20	33.50	32.80	36.00	32.80	37.00
RQ-RAG	26.30	33.94	10.20	16.04	28.70	37.64	24.80	32.18	15.60	31.37
`RoleRAG` (ours)	**37.40**	**49.17**	**18.20**	**27.30**	**47.00**	**53.87**	**44.00**	**54.47**	**33.70**	**45.42**

Table 3: Performance (F1 score) of `RoleRAG` with specific components removed.

		HotpotQA		MuSiQue	
#	**Variant**	EM	F1	EM	F1
1	Full	37.40	49.17	18.20	27.30
2	↪ *w/o* Q. Graph	31.30	42.60	5.80	12.54
3	↪ *w* Decompose prompt	29.40	39.98	6.90	13.31
4	↪ *w/o* Retrieval judge	38.40	50.31	18.50	27.58
	Save retrieval	*22.56%*		*9.2%*	
5	↪ *w/o* Summarizer	31.20	42.47	5.70	12.44
6	↪ *w/o* New Q. gen	37.20	49.07	18.00	27.30
	Need new query	*5.9%*		*14.10%*	

Table 4: Performance (F1 score) of `RoleRAG` on two datasets using different LLMs.

#	**Query graph**	**Other modules**	**HotpotQA**	**MuSiQue**
		`RoleRAG` *Default setting*		
1	Llama-8B	Llama-8B	49.17	27.30
		Using various LLMs as backbones		
2	Llama-3B	Llama-3B	41.50	20.94
3	Mistral-7B	Mistral-7B	47.62	25.51
4	Llama-70B	Llama-70B	53.46	29.86
		Using various LLMs for different modules		
5	Llama-70B	Llama-8B	50.65	27.53
6	Llama-8B	Llama-70B	54.23	28.19

impact, we implement a variant that simply uses the first retrieved passage for length control. The observed performance degradation confirms the important role of the summarizer. Finally, the new query generator improves performance by introducing additional useful knowledge, despite being activated in fewer than 15% of queries, highlighting its effectiveness.

Impact of Model Size The size of LLMs often determines their performance. Therefore, we investigate the impact of model sizes from two perspectives: (1) by using different backbone LLMs to drive the entire `RoleRAG` framework, and (2) by replacing the core module (*i.e.*, query graph builder) with various LLMs. We conduct experiments using `Llama-3.2-3B-instruct` (Llama-3B), `Llama-3.1-8B-instruct` (Llama-8B), `Llama-3.1-70B-instruct` (Llama-70B), and `Mistral-7B-Instruct-v0.3` (Mistral-7B). Notably, only the Llama-70B is applied using few-shot examples, while the others are fine-tuned on our training set. The results are shown in Table 4. First, we can observe that `RoleRAG` consistently achieves promising results across all settings, demonstrating the method's robust versatility. Second, using larger LLMs generally leads to better performance (#1-4). This is reasonable as larger models have strong abilities in language understanding and generation. Intriguingly, the Llama-70B model plays a better role in resolving queries (#6) than in building the query graph (#5). This suggests that while query decomposition is a complex task, it can be effectively learned with sufficient model training. Conversely, the ability to resolve queries appears to be more closely tied to the intrinsic performance of the model itself.

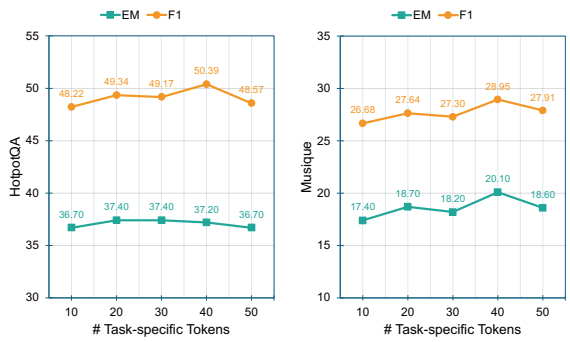


Figure 4: Performance with various numbers of tokens.

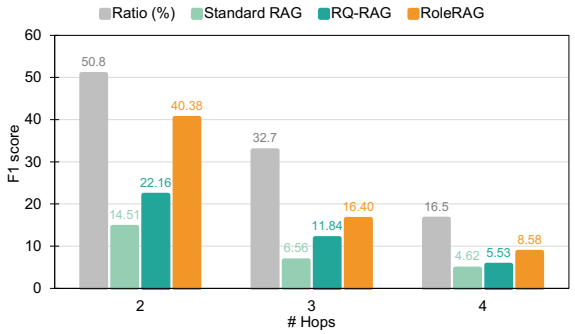


Figure 5: Performance of various models on questions of different complexity (MuSiQue). "Ratio" indicates the proportion of a certain category to the entire data.

Impact of Task-specific Token Amounts In RoleRAG, we use task-specific tokens in multi-task prompt tuning to learn different tasks in RAG. We explore the correlation between the number of added tokens and the final performance, as shown in Figure 4. We can observe: (1) It is surprising that using only ten tokens per task can provide significant performance improvement, highlighting the efficiency of our approach. (2) The performance generally improves when more tokens are used, with optimal results occurring when 30-40 tokens are used per task (varies slightly across different datasets). Taking adding 30 tokens as an example, our method adds 0.86M parameters in total, which is only about 0.01% of the full model, validating again its parameter efficiency. (3) However, further increasing the token amount does not improve performance; a decline is noted when 50 tokens are used per task, implying potential overfitting issues.

Impact of Query Complexity An advantage of our framework lies in its ability to decompose complex queries into sub-queries and leverage multiple modules to resolve them. We employ the MuSiQue dataset, which contains human-annotated decomposition labels, to investigate the performance across

Table 5: Performance (F1 score) of RoleRAG with different numbers of passages per query (# P. / Q.). RoleRAG-x denotes using x passages per sub-query.

#	Method	HotpotQA		MuSiQue	
		# P. / Q.	F1	# P. / Q.	F1
1	Standard RAG	5.00	40.00	5.00	10.28
2	RQ-RAG	7.35	33.94	7.43	16.04
3	RoleRAG-1	2.27	43.58	2.35	20.73
4	RoleRAG-2	4.54	46.73	4.70	25.08
5	RoleRAG-3	6.81	47.80	7.05	26.24
6	RoleRAG-4	9.08	48.66	9.40	27.07
7	RoleRAG-5	11.35	51.03	11.75	27.30

query complexities. We analyze the performance of different models, including RoleRAG, RQ-RAG, and Standard RAG, on questions categorized by the number of intermediate steps (hops) required. The experimental results are shown in Figure 5. It is evident to see that the standard RAG method struggles with complex multi-hop queries, because the retriever cannot effectively gather comprehensive information that spans all facets of a query. In contrast, both RQ-RAG and RoleRAG can iteratively resolve the sub-queries, which significantly improves the performance. Unfortunately, RQ-RAG learns both the query decomposing and query resolving tasks by a single model, making it challenging for the LLM to learn different abilities required by these tasks. Notably, our framework achieves over 60% alignment with human-annotated decomposition results, while RQ-RAG reaches only 18%. This highlights again the superiority of our RoleRAG, which distinctively separates these tasks to optimize performance.

Impact of Retrieval Since RoleRAG decomposes original queries into multiple sub-queries, its superior performance may be benefited from more sufficient external knowledge. To examine this, we conduct experiments by adjusting the number of retrieved passages per sub-query, and the results are illustrated in Table 5. We can observe that RoleRAG can significantly outperform the standard RAG method, with fewer than half the retrieved passages (#3 vs. #1). This shows that query decomposition can indeed improve retrieval accuracy, which in turn enhances the overall performance of the RAG model. Compared with another iterative RAG pipeline RQ-RAG, RoleRAG still has better performance, suggesting that it can construct sub-queries more accurately.

To provide a more intuitive understanding of our

framework, we include a case study in Appendix F.

5 Conclusion

In this paper, we introduced RoleRAG, a unified RAG framework that comprises six modules that collaborate to accomplish the full RAG process. To efficiently optimize these modules, we proposed a role-specific optimization strategy, which enhances the LLM's ability across diverse tasks by tuning only a small set of role tokens, while keeping the backbone model parameters frozen. Additionally, we structured the RAG process as a query graph resolution process, where dynamic sub-query resolution efficiently retrieves and supplements relevant knowledge. Through extensive experiments on multiple datasets, we demonstrated the effectiveness, generalizability, and flexibility of our method.

Limitations

This study introduces a unified RAG framework using role-specific token optimization. While our approach is highly effective, it has some limitations. First, the RAG process follows a predefined workflow, where all modules are activated in a fixed sequence. This restricts the framework's flexibility, as an ideal solution would allow the LLM to autonomously determine the workflow. Recent reinforcement learning methods could potentially enable such adaptive decision-making; however, collecting high-quality processing paths is challenging, and reinforcement learning itself is often unstable. Investigating an automatic workflow optimization remains an important direction for future work. Second, our framework processes each query by iteratively activating different modules, which may introduce efficiency overhead compared to directly feeding retrieved results and user queries into an LLM. Fortunately, when deployed as an online service, this efficiency issue can be mitigated. Since our framework leverages role tokens to modulate the LLM's functionality, it enables the batching of multiple LLM requests, significantly improving inference efficiency.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 62402497 and 62272467). The work was partially done at the Beijing Key Laboratory of Research on Large Models and Intelligent Governance.

References

Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. 1996. Query reformulation for dynamic information integration. *J. Intell. Inf. Syst.*, 6(2/3):99–130.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Hiteshwar Kumar Azad and Akshay Deepak. 2019. Query expansion techniques for information retrieval: A survey. *Inf. Process. Manag.*, 56(5):1698–1735.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Claudio Carpineto and Giovanni Romano. 2012. A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.*, 44(1):1:1–1:50.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. RQ-RAG: learning to refine queries for retrieval augmented generation. *CoRR*, abs/2404.00610.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Jiangning Liu, Wenwei Zhang, Kai Chen, and Feng Zhao. 2024. Mindsearch: Mimicking human minds elicits deep AI searcher. *CoRR*, abs/2407.20183.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu,

Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, and Wangding Zeng. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Jinyuan Fang, Zaiqiao Meng, and Craig Macdonald. 2024. TRACE the evidence: Constructing knowledge-grounded reasoning chains for retrieval-augmented generation. *CoRR*, abs/2406.11460.

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, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 6609–6625. International Committee on Computational Linguistics.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 7036–7050. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 13358–13376. Association for Computational Linguistics.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7969–7992. Association for Computational Linguistics.

Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024a. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *CoRR*, abs/2405.13576.

Jiajie Jin, Yutao Zhu, Yujia Zhou, and Zhicheng Dou. 2024b. BIDER: bridging knowledge inconsistency for efficient retrieval-augmented llms via key supporting evidence. *CoRR*, abs/2402.12174.

Dongkyu Kim, Byoungwook Kim, Donggeon Han, and Matous Eibich. 2024a. Autorag: Automated framework for optimization of retrieval augmented generation pipeline. *CoRR*, abs/2410.20878.

Jaehyung Kim, Jaehyun Nam, Sangwoo Mo, Jongjin Park, Sang-Woo Lee, Minjoon Seo, Jung-Woo Ha, and Jinwoo Shin. 2024b. Sure: Summarizing retrievals using answer candidates for open-domain QA of llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5303–5315, Singapore. Association for Computational Linguistics.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada,*

July 9-14, 2023, pages 9802–9822. Association for Computational Linguistics.

Shengyu Mao, Yong Jiang, Boli Chen, Xiao Li, Peng Wang, Xinyu Wang, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2024. Rafe: Ranking feedback improves query rewriting for RAG. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 884–901. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick S. H. Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. 2021. KILT: a benchmark for knowledge intensive language tasks. In *NAACL-HLT*, pages 2523–2544. Association for Computational Linguistics.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 5687–5711. Association for Computational Linguistics.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 9248–9274. Association for Computational Linguistics.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. REPLUG: retrieval-augmented black-box language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 8371–8384. Association for Computational Linguistics.

Jiejun Tan, Zhicheng Dou, Yutao Zhu, Peidong Guo, Kun Fang, and Ji-Rong Wen. 2024. Small models, big insights: Leveraging slim proxy models to decide when and what to retrieve for llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 4420–4436. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Trans. Assoc. Comput. Linguistics*, 10:539–554.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10014–10037. Association for Computational Linguistics.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275.

Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. 2024. Planxrag: Planning-guided retrieval augmented generation. *CoRR*, abs/2410.20753.

Haoyu Wang, Ruirui Li, Haoming Jiang, Jinjin Tian, Zhengyang Wang, Chen Luo, Xianfeng Tang, Monica Xiao Cheng, Tuo Zhao, and Jing Gao. 2024. Blendfilter: Advancing retrieval-augmented large language models via query generation blending and knowledge filtering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 1009–1025. Association for Computational Linguistics.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *CoRR*, abs/2212.03533.

Yile Wang, Peng Li, Maosong Sun, and Yang Liu. 2023. Self-knowledge guided retrieval augmentation for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 10303–10315. Association for Computational Linguistics.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jinxi Xu and W. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'96, August 18-22, 1996, Zurich, Switzerland (Special Issue of the SIGIR Forum)*, pages 4–11. ACM.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2369–2380. Association for Computational Linguistics.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024a. Making retrieval-augmented language models robust to irrelevant context. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2024b. Making retrieval-augmented language models robust to irrelevant context. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2024. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 858–875. Association for Computational Linguistics.

Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, Xinyu Dai, Shikun Zhang, and Qingsong Wen. 2024a. RAGLAB: A modular and research-oriented unified framework for retrieval-augmented generation. *CoRR*, abs/2408.11381.

Zhen Zhang, Xinyu Wang, Yong Jiang, Zhuo Chen, Feiteng Mu, Mengting Hu, Pengjun Xie, and Fei Huang. 2024b. Exploring knowledge boundaries in large language models for retrieval judgment. *CoRR*, abs/2411.06207.

A Discussion about Query Graph Builder

In our framework, the query graph builder constructs a query graph before resolving each subquery. Indeed, pre-constructing the query graph may constrain dynamic reasoning paths to some extent. However, during our preliminary experiments, we observe significant issues with purely iterative methods such as IRCoT (Trivedi et al., 2023) and Self-RAG (Asai et al., 2024), which tend to become overly reliant on intermediate retrieved results, thus propagating generation errors through subsequent reasoning steps. This may be because current LLMs cannot accurately coordinate internal and external knowledge. As a result, we choose to build a query graph before resolving the query, and our experiments demonstrate that this strategy is effective for complex queries like the multi-hop questions in QA tasks. Additionally, our new query generator serves as a complementary iterative mechanism to dynamically enhance query exploration when necessary. We believe further exploring a hybrid approach that integrates planning-based and iterative methods is a promising direction for future work.

B Prompt for Data Collection

As illustrated in Figure 6–11, we manually craft different prompts and employ an expert LLM to generate training data. Each prompt consists of five key components: (1) Task description, providing context to help the LLM understand the task; (2) Output requirements, specifying the expected format and structure; (3) Guidelines, highlighting rules for data generation; (4) Demonstration examples, serving as in-context learning references; and (5) Task input, representing the specific instance to be processed.

C Details of Datasets

We conduct our experiments on five QA datasets, which are all provided by FlashRAG (Jin et al., 2024a) under the license of CC-BY-SA-4.0.[3]

HotpotQA (Yang et al., 2018) is a large-scale QA dataset comprising Wikipedia-based question-answer pairs. Designed to facilitate complex reasoning, it features questions that require synthesizing information from multiple supporting documents. The dataset is diverse, unconstrained by pre-existing knowledge bases or schema. Additionally, HotpotQA introduces factoid comparison questions to assess a system's ability to extract and compare relevant information.

MuSiQue (Trivedi et al., 2022) is a multi-hop QA dataset designed to require genuine multi-hop reasoning. Each question necessitates 2 to 4 reasoning steps (hops). The dataset is constructed by systematically selecting and composing pairs of single-hop questions that are connected, ensuring

[3] https://creativecommons.org/licenses/by-sa/4.0/

that one reasoning step critically relies on information from another. This bottom-up methodology provides fine-grained control over the construction process and the properties of the resulting multi-hop questions.

2WikimultihopQA (Ho et al., 2020) is a multi-hop QA dataset designed to evaluate complex reasoning across both structured and unstructured data. It comprises questions that require models to perform multiple reasoning steps, utilizing information from different Wikipedia articles.

Bamboogle (Press et al., 2023) is a curated dataset designed to assess the compositional reasoning abilities of language models. It comprises 125 questions that are intentionally challenging for standard search engines, like Google, to answer correctly. Each question requires the model to integrate information from multiple sources or perform multi-step reasoning to arrive at the correct answer. The dataset covers a wide range of topics and question formats.

PopQA (Mallen et al., 2023) is a large-scale, open-domain QA dataset comprising entity-centric single-hop question-answer pairs. Each question is generated by converting a knowledge tuple from Wikidata into a natural language format using predefined templates. The dataset includes detailed annotations such as the subject entity, object entity, relationship type, and corresponding Wikidata identifiers. PopQA is designed to evaluate language models' abilities to recall factual knowledge, particularly focusing on less popular long-tail entities.

D Implementation Details

We use PyTorch (Paszke et al., 2019) and Huggingface Accelerate library to implement our method. The learning rate is set as 5e-5 with a warm-up ratio of 0.02. Our method is trained for three epochs, with a training batch size of 32. The maximum sequence length is set as 2,048 tokens. We use eight NVIDIA A800 GPUs for training. For the datasets and baseline methods, we use the version provided by FlashRAG (Jin et al., 2024a), where `Llama-3.1-8B-instruct` is used as the default backbone LLM. For the retrieval sets, we follow previous studies (Yoran et al., 2024a) and use Wikipedia as the retrieval corpus. `E5-base-v2` (Wang et al., 2022) is used as the retriever.

All of the methods in our experiments use the same retrieval corpus, retriever, and backbone LLM. Specifically:

(1) Standard RAG, SKR, SuRe, Trace, Blender-Filter, IRCoT, and Iter-Retgen rely solely on prompt engineering strategies without additional model training.

(2) Adpative-RAG involves training a query classifier on data sampled from SQuAD, NQ, TriviaQA, MuSiQue, HotpotQA, and 2WikiMultihopQA. We use the classifier provided by the original authors.

(3) Self-RAG and RQ-RAG are trained using the same dataset as ours, utilizing the publicly available code provided by their authors.

E Efficiency Analysis

For sequential pipeline methods, they only conduct retrieval once, so their computational costs are lower but their performance is also relatively worse. For iterative pipeline methods (including ours), we theoretically analyze the computational costs of IRCoT, RQ-RAG, and our `RoleRAG`. For clarity, we assume:

(1) All queries/sub-queries have equal length m; answers/sub-answers have length t; each sub-query retrieves k passages; and they have the same length l.

(2) The summarizer in `RoleRAG` produces a summary with the same length of a single passage, *i.e.*, its length is also l.

(3) Retrieval is assumed for each sub-query to simplify analysis.

From the results, we can see IRCoT has the highest computational cost due to iterative processing and repeated input of all previous sub-results. Our `RoleRAG` and RQ-RAG have a similar input token complexity ($O(nkl)$), but `RoleRAG` produces additional output tokens due to the summarization step. Nevertheless, in practical scenarios, the retrieval is selective, thus reducing real-world overhead.

In summary, while iterative methods naturally incur higher computational costs compared to sequential methods, our `RoleRAG`'s additional costs remain moderate relative to its significantly improved performance. Furthermore, `RoleRAG` is inherently parallelizable due to its modular design driven by role tokens, making it feasible for practical deployments.

F Case Study

To further evaluate our framework qualitatively, we conduct a case study and present three representa-

Method	Module	Input	Input tokens	Output	Output tokens
RoleRAG	Query graph builder	Original query	m	Sub-queries	$n*m$
	Retrieval judge	Sub-query	$n*m$	Judge result (Yes or No)	n
	Sub-answer generator	Sub-query, retrieved passages	$n*m+n*k*l$	(Sub-)answer	$n*t$
	Summarizer	Retrieved passages	$n*k*l$	Summary	$n*l$
	New query generator	Answer memory	$n*(m+t+l)$	New sub-query	m
	Answer reasoner	Answer memory	$n*(m+t+l)$	Answer	t
	Total		$n*(4m+2kl+2t+2l)+m$		$n*(m+t+l+1)+m+t$
IRCoT	Sub-query and Sub-answer generation	(previous) Sub-query, (previous) Sub-answer retrieved passages	$n*m+n*(k*l)+(n-1)*(k*l+t)+(n-2)*(k*l+t)...+k*l+t$	Sub-queries and Sub-answers	$n*(m+t)$
	Final answer generation	Original query, all retrieved passages	$m+n*k*l$	Answer	t
	Total		$n*(m+\frac{n+3}{2}*k*l+\frac{n-1}{2}*t)$		$n*(m+t)+t$
RQ-RAG	Sub-query generation	Original query	m		$n*m$
	Answer generation	(Sub-)query, retrieved passages	$(n+1)*m+n*k*l+n*t$	Answer	$n*t$
	Total		$n*(m+kl+t)+m$		$n*(m+t)$

tive examples in Table 12 and Table 13. In the first case, our framework successfully decomposes the query into three sub-queries, where the third sub-query depends on the answers to the first two. By iteratively resolving the first two sub-queries, the answer to the third can be inferred directly without requiring additional retrieval. In the second case, our framework only rewrites the query, and the corresponding answer is incomplete. Fortunately, the new query generator successfully adds an effective sub-query to provide supplemental information. In contrast, the third case highlights a failure scenario. Although the original user query is split into three sequential sub-queries, the first two should be dependent, yet the model incorrectly treats them as independent. This suggests that accurately decomposing complex queries remains a challenging problem. Besides, while the second and third sub-queries are correctly formulated, the third sub-query fails to retrieve useful knowledge, leading to an incorrect final answer. In this scenario, the new query generator attempts to repeat the third sub-query. However, due to the limitations of the retrieval repository, the necessary information remains unavailable, resulting in an incorrect response. This case demonstrates that even when individual model components function correctly, external factors such as retrieval limitations can still prevent the system from generating the correct answer.

	HotpotQA	MuSiQue	2WikiQA
Full	49.17	27.30	53.87
+ Rewrite Ori. Q	50.04	27.19	53.49
+ Rewrite All	48.88	27.47	53.79

Table 6: Performance (F1 score) of RoleRAG with query rewrite module.

G Impact of Query Rewrite

Query rewriting (Xu and Croft, 1996; Carpineto and Romano, 2012) addresses the problem of users' ambiguity and inaccurate queries by rewriting the user's original query, which is helpful in RAG systems (Mao et al., 2024). Recent studies have demonstrated that LLMs are capable of understanding user intents and providing more informational rewritten queries (Ma et al., 2023). Motivated by these findings, we consider incorporating a query rewriting module in our framework and evaluate its impact under two settings: (1) applying query rewriting only to the original query and (2) applying it to all sub-queries. The experimental results are shown in Table 6. Generally, we can observe that query rewriting does not consistently improve performance. When applied to the original query, it influences the query graph construction, leading to mixed results. Notably, improvements are observed

only on the HotpotQA dataset. A closer inspection of the data reveals that HotpotQA queries are relatively well-formed, making query rewriting beneficial in this case. However, applying query rewriting to all sub-queries also yields unstable performance, likely because the sub-queries in our query graph are already simple and do not require further refinement. Given these findings, we exclude the query rewriting module from our final framework, as it introduces additional computational overhead without providing consistent benefits.

Prompt for Query Graph Builder

Task Description: You are tasked with constructing reasoning directed acyclic graphs (DAGs) for complex queries. Start by breaking down a complex main query into smaller, manageable subqueries. Each subquery should independently contribute to solving the main query. Utilize these queries and their respective answers to form a DAG starting from the root node (main query) and branching out without forming any cycles. The final node or the leaf node will closely resemble the main query but will include placeholders filled by answers from preceding nodes, adhering to the Markovian property, which requires answers from parent nodes to resolve child subqueries.

Output Requirement: Produce a Python list of tuples where each tuple represents a connection in the DAG. The tuples should contain pairs linking a query to its subquery. Format your output as a list of tuples without additional text, comments, or line breaks.

Note: Ensure that the DAG is connected and forms a rooted tree structure. For straightforward queries that do not require decomposition, simply return the original query within a list.

Examples:

Query: {Example query }
DAG: {Example DAG}
...
Query: {Input query}
DAG: {Output DAG}

Figure 6: Prompt using for generating data for query graph builder.

Prompt for Retrieval Judge

Task Description: As a language model, you are tasked to act as a critic determining whether a given question can be answered directly based on the provided information or the semantics of the question itself. When no information is provided, you should rely solely on the literal meaning of the question to make your judgment. You will receive a question along with any corresponding information. Your job is to assess if the question can be answered based solely on the provided information without requiring additional context.

Output Requirement: Return a JSON object with a key "Response" and a value "True" or "False". Here, "True" indicates that the question can be answered with the given information, while "False" indicates that additional information is needed. Always represent the values as strings.

Note: Your output should consist solely of the JSON response without any explanatory text.

Examples:

Query: {Example query}
Provided information: {Example information}
Response: {Example response}
...
Query: {Input query}
Provided information: {Memory dict}
Response: {Output}

Figure 7: Prompt using for generating data for retrieval judge.

Prompt for Sub-answer Generator

Task Description: You are a concise answering assistant. You need to answer the query based on the retrieved materials and your own knowledge. If the retrieved materials are not useful, directly ignore it.

Output Requirement: Return a JSON with a single key "Response" and a value that is a short phrase or a few words. In this JSON, you need to always put each value as a string, not float.

Note: Generate only JSON without any explanation.

Examples:

Query: {Example query}
Retrieved materials: {Example materials}
Answer: {Example answer}
...
Query: {Input query}
Retrieved materials: {Retrieved knowledge}
Answer: {Output}

Figure 8: Prompt using for generating data for sub-answer generator.

Prompt for Summarizer

Task Description: You are a relevant information aggregator. You need to extract and summarize information relevant to the keyword from several retrieved materials.

Output Requirement: Return a JSON with a single key "Summary" and a value that is its content. Always represent the values as strings.

Note: Generate only JSON without any explanation.

Examples:

Keyword: {Example keyword}
Retrieved materials: {Example materials}
Response: {Example summary}
...
Keyword: {Answer}
Retrieved materials: {Retrieved knowledge}
Response: {Output}

Figure 9: Prompt using for generating data for summarizer.

Prompt for New Query Generator

Task Description: You are to act as a final answer reasoner tasked with evaluating if a specific query can be answered directly by the provided question-answer pairs. If the query is answerable based on the given data, respond with "Yes". If information is lacking, formulate a new query that would request the necessary missing information.

Output Requirement: Produce a JSON object with a single key "Response". The value should either be the word "Yes" or a new query, both formatted as a string. Ensure all numbers are converted to string format.

Note: Generate only JSON without any explanation.

Examples:

Final question: {Example keyword}
Reference question-answer pairs: {Example materials}
Response: {Example response}
...
Final question: {Original query}
Reference question-answer pairs: {Memory dict}
Response: {Output}

Figure 10: Prompt using for generating data for new query generator.

Prompt for Answer Reasoner

Task Description: As a final answer generator, synthesize information from provided question-answer pairs to respond to a final query.

Output Requirement: Produce a JSON object with a single key "Response". The value should be a concise phrase or a few words that directly answer the question, formatted as a string. Ensure all numerical values are converted to strings.

Note: Generate only JSON without any explanation.

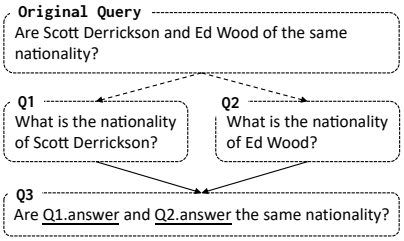
Examples:

Final question: {Example keyword}
Reference question-answer pairs: {Example materials}
Answer: {Example response}
...
Final question: {Original query}
Reference question-answer pairs: {Memory dict}
Answer: {Output}

Figure 11: Prompt using for generating data for answer reasoner.

Original query: Are Scott Derrickson and Ed Wood of the same nationality?

Query graph: [["Q: Are Scott Derrickson and Ed Wood of the same nationality?", "Q1: What is the nationality of Scott Derrickson?"], ["Q: Are Scott Derrickson and Ed Wood of the same nationality?", "Q2: What is the nationality of Ed Wood?"], ["Q1: What is the nationality of Scott Derrickson?", "Q3: Are <A1> and <A2> the same nationality?"], ["Q2: What is the nationality of Ed Wood?", "Q3: Are <A1> and <A2> the same nationality?"]]



Q1: What is the nationality of Scott Derrickson?

Retrieval judge: False (need retrieval)

Retrieved knowledge: [["Scott Derrickson directed the film "Doctor Strange", based on the Marvel Comics property and part of the Marvel Cinematic Universe. It was released in November 2016. The film was both a commercial and critical success. In February 2018, it was announced that Derrickson will executive produce the television series "You Bury Me" for Blumhouse Productions. …]]

Answer: American

Summary: Scott Robertson is an American concept artist, known for his transportation design work and contributions to movies like Steven Spielberg's Minority Report. He is a graduate of Art Center College of Design in Pasadena, CA and was the program director of Entertainment Design. Robertson was born in Minnesota and grew up in Plymouth, where he was taught to draw and design by his artist father. He is also known for his educational DVDs with the Gnomon Workshop.

Q2: What is the nationality of Ed Wood?

Retrieval judge: False (need retrieval)

Retrieved knowledge: [["Ed Wood Grey's 1992 oral biography "Nightmare of Ecstasy: The Life and Art of Edward D. Wood Jr.", the public took a greater interest in Wood, leading up to director Tim Burton's biopic of Wood's life, "Ed Wood" (1994), a critically acclaimed film starring Johnny Depp as Wood that earned two Academy Awards. …]]

Answer: American

Summary: Ed Wood was an American filmmaker, actor, and author born on October 10, 1924, in Poughkeepsie, New York. He made low-budget films in various genres, including science fiction, crime, and horror, and later made sexploitation movies and wrote pulp novels. Wood's career and camp approach earned him a cult following, and he was awarded a Golden Turkey Award as Worst Director of All Time in 1975. He passed away on December 10, 1978.

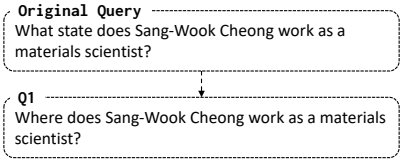
Q3: Are American and American the same nationality?

Retrieval judge: True (no need for retrieval)

Answer: Yes

Original query: What state does Sang-Wook Cheong work as a materials scientist?

Query graph: [["Q: What state does Sang-Wook Cheong work as a materials scientist?", "Q1: Where does Sang-Wook Cheong work as a materials scientist?"]]



Q1: Where does Sang-Wook Cheong work as a materials scientist?

Retrieval judge: False (need retrieval)

Retrieved knowledge: [["Stephen Pearton is an American materials scientist and engineer and Distinguished Professor at University of Florida.", " Pearton's work focuses on the use of advanced materials in areas such as laser diodes and nanomaterial applications and similar applications.", " He is a recipient of the J.J…]]

Answer: Rutgers University

Summary: Sang Wook Cheong is a Korean American materials scientist at Rutgers University. He has made ground-breaking contributions to the research field of enhanced physical functionalities in complex materials originating from collective correlations and collective phase transitions such as colossal magnetoresistive and colossal magnetoelectric effects in complex oxides.

New query: In which state is Rutgers University located?

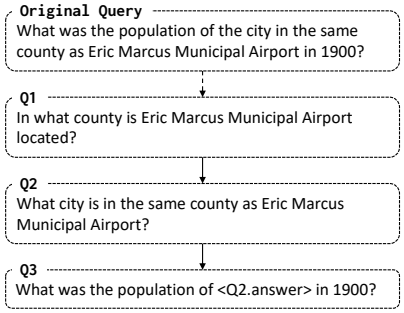
Retrieval judge: True (no need for retrieval)

Answer: New Jersey

Figure 12: An example of RoleRAG execution process from HotpotQA.

Original query: What was the population of the city in the same county as Eric Marcus Municipal Airport in 1900?

Query graph: [["Q: What was the population of the city in the same county as Eric Marcus Municipal Airport in 1900?", "Q1: In what county is Eric Marcus Municipal Airport located?"], ["Q1: In what county is Eric Marcus Municipal Airport located?", "Q2: What city is in the same county as Eric Marcus Municipal Airport?"], ["Q2: What city is in the same county as Eric Marcus Municipal Airport?", "Q3: What was the population of <A2> in 1900?"]]



Q1: In what county is Eric Marcus Municipal Airport located?

Retrieval judge: False (need retrieval)

Retrieved knowledge: [["Eric Marcus Municipal Airport" Eric Marcus Municipal Airport Eric Marcus Municipal Airport is a county-owned, public-use airport in Pima County, Arizona, United States. It is located north of the central business district of Ajo and is about southwest of Phoenix. …]]

Answer: Pima County

Summary: Pima County in Arizona is home to the Eric Marcus Municipal Airport, a county-owned, public-use airport located north of Ajo's central business district. The airport was formerly known as Ajo Municipal Airport and was renamed on February 11, 2006. It has one serviceable runway and is included in the FAA's National Plan of Integrated Airport Systems as a general aviation facility. The airport was originally established in 1941 as part of the Army Air Corps expansion prior to World War II and was later inactivated in 1946, with the land being transferred to Pima County in 1949.

Q2: What city is in the same county as Eric Marcus Municipal Airport?

Retrieval judge: True (no need for retrieval)

Answer: Ajo

Q3: What was the population of Ajo in 1900?

Retrieval judge: False (need retrieval)

Answer: Not found

Retrieved knowledge: [["Ajo, Cantabria" Ajo, Cantabria Ajo is the capital of Bareyo municipality in Cantabria, Spain. The town is from Santander. The first historical written reference to Ajo (Asio) is in the "Liber Testamentarum" of Oviedo Cathedral from 923, in which the King Ordo\u00f1o II of Le\u00f3n donated the church of San Juan de Asio. …]]

Summary: (None)

Q4: What was the population of Ajo in 1900? (Generated new query)

Answer: Not found

Retrieved knowledge: [["Ajo, Cantabria" Ajo, Cantabria Ajo is the capital of Bareyo municipality in Cantabria, Spain. The town is from Santander. The first historical written reference to Ajo (Asio) is in the "Liber Testamentarum" of Oviedo Cathedral from 923, in which the King Ordo\u00f1o II of Le\u00f3n donated the church of San Juan de Asio. …]]

Summary: (None)

Figure 13: An example of RoleRAG execution process from MuSiQue.