A Non-Vector Retrieval-Augmented Generation Model for External Time-Relevant Corpus Extraction

Jinghao You

{nwmmmtarge@163.com}

South China University of Technology, Guangzhou, 511442, China

Abstract. Large language models (LLMs) have emerged as a powerful tool in the natural language process, allowing information extraction and answer generation from pre-trained databases. Regarding untrained external corpus, the retrieval-augmented generation (RAG) techniques enable quick establishing of databases. However, this process is currently of low accuracy when handling problems with strong indexing correlations, such as time-relevant input. This paper proposes a non-vector retrieval-augmented generation (NVRAG) model to enhance the RAG model for processing multi-index related complex queries based on a non-vector database and text-to-SQL technology. NVRAG stores relevant parameters in a non-vector database to narrow the embedding range and improve indexing accuracy. Experiments on a weather briefings database are conducted to validate the effectiveness of the NVRAG model. The results show that compared with the original RAG, the faithfulness and accuracy rate of NVRAG are higher, thereby sacrificing the response time. By implementing this approach, the system's ability to handle complex query requests is enhanced.

Keywords: LLM, RAG, Test-to-SQL, Time-relevant information extraction.

1 Introduction

In recent years, the application of large language models (LLMs) has matured across various fields, particularly in natural language processing and information retrieval tasks [1,2]. However, these models still demonstrate inadequate extraction capabilities when dealing with databases that have not undergone pre-training [3]. To address this issue, retrieval-augmented generation (RAG) models based on vectorized indexing have emerged, and many researchers have thoroughly studied the performance of these models [4-7]. RAG enhances the ability of LLMs to learn from untrained data by leveraging database indexing, thereby improving the overall effectiveness of information retrieval processes [8]. Nevertheless, RAG exhibits relatively low accuracy when handling problems with strong indexing correlations. This limitation primarily stems from RAG's insufficient optimization capabilities for multi-key-value queries, which hampers its performance in complex query scenarios, particularly when multiple parameters are involved [9]. Therefore, proposing a method based on text-to-SQL conversion will significantly contribute to improving both the accuracy and efficiency of queries, enabling more precise interactions with databases. This study proposes an innovative approach that leverages text-to-SQL conversion to transform user-input natural language queries into specific database queries. This method aims to address the performance issues encountered by existing RAG models when

CONF-MLA 2024, November 21, Adana, Turkey

DOI 10.4108/eai.21-11-2024.2354622

relying solely on vector databases for semantic matching, particularly in relation to indexing and correlation challenges, thereby enhancing overall query performance and reliability.

2 Literature review

2.1 Retrieval-augmented generation

RAG models combine the strengths of information retrieval and generative models to enhance the processing capabilities of untrained data. As depicted in Figure 1, its working principle mainly consists of two stages: the retrieval stage and the generation stage [10]. In the retrieval stage, RAG first receives the user's natural language query and utilizes a retriever to search for relevant documents from an external database. This process typically relies on vectorization techniques, embedding both the query and the documents into the same semantic space for semantic matching. The retriever selects several documents that are most relevant to the query based on similarity scores, providing support for the subsequent generation stage. In the generation stage, RAG combines the retrieved documents with the original query and inputs this information into a generative model. The generative model generates a response based on this information, aiming to provide an answer that is highly relevant and informative in relation to the user's query. During this process, the generative model leverages the contextual information from the retrieved documents to enhance the accuracy and relevance of its output. Overall, RAG seeks to improve model performance in handling complex queries by integrating both retrieval and generation functionalities. This model has been applied in various fields and has received positive feedback [11,12].

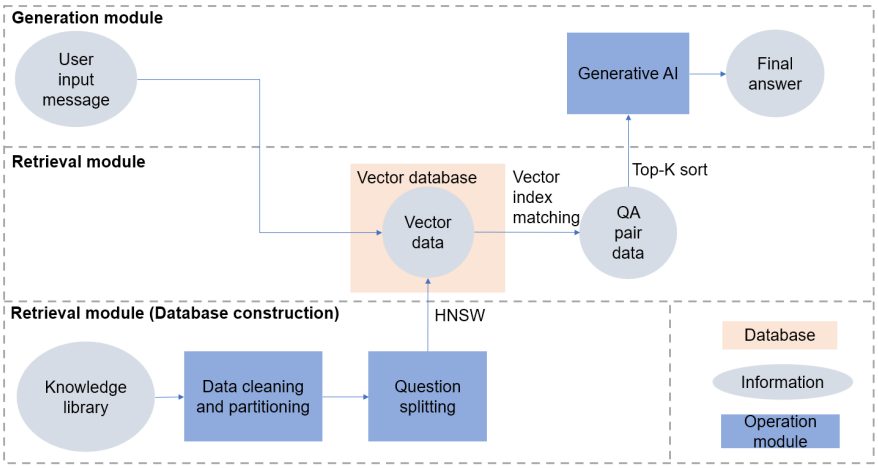

Fig. 1. Workflow of the original RAG model.

In the retrieval module, the construction of the database serves as the starting point for all subsequent steps. A knowledge library (containing all knowledge documents) first undergoes data cleaning and partitioning, followed by question splitting, ultimately constructing an index through HNSW (Hierarchical Navigable Small World) [13]. Through this, the documents are converted into question-and-answer pairs, where key values are transformed into vectors and

stored in a vector database. Then, by employing a top-K recall method, the system efficiently retrieves similar content. The generation module takes the relevant results returned from the retrieval module, along with the optimized user queries, and sends this information to a generative AI to produce a final answer.

A significant shortcoming of this standard framework is that the matching algorithm of RAG is overly reliant on the indexing of question-answer pairs and the similarity of user statements [14]. This reliance can hinder the model's ability to accurately address queries that involve multiple associated indices, such as statistical questions regarding sums over a three-month period or multi-value optimization problems. Semantic matching methods, such as vector algorithms, struggle to provide precise answers in these cases.

2.2 Test-to-SQL and large language models

Based on large model fine-tuning, the text-to-SQL method converts natural language queries into structured SQL statements [15]. It enables users to interact with databases using natural language without the need for SQL tools, thereby bridging the gap between user inquiries and database queries and facilitating better interaction with databases. Formally, given a user query (in natural language) and the corresponding database schema, the model can generate a corresponding SQL query for retrieving the required content from the schema to answer the user's question.

LLMs excel in natural language understanding and generation, effectively capturing user intent and contextual information [16]. Additionally, pre-trained language models possess strong capabilities in handling diverse language structures and semantics, enabling them to address various user expressions. Furthermore, code-specific LLMs, such as CodeLLaMa and StarCoder, have been pre-trained on code data, allowing them to generate code that aligns well with user requirements [17]. Through fine-tuning, LLMs can be optimized for specific database structures and domain knowledge, resulting in SQL statements that better meet practical needs.

In summary, the text-to-SQL method based on large model fine-tuning significantly enhances the efficiency of user interaction with databases by integrating natural language processing and database query generation. In this paper, user inputs are presented in natural language, which may contain potential SQL information and query intent. This method can be effectively integrated into the retrieval module of Retrieval-Augmented Generation (RAG), improving information retrieval capabilities and addressing the challenges of joint queries across multiple indices, thereby avoiding information loss caused by similarity matching algorithms.

3 Methodology

In this paper, the text-to-SQL algorithm is effectively leveraged to significantly enhance the response capability of the RAG model when dealing with complex multi-key value associations. When the RAG model stores vector data, it also retains a wealth of additional information, including the original question-answer pairs, index IDs, dataset IDs, and various other relevant metadata in a separate, organized database. Notably, this information is not originally utilized during the matching process. Therefore, the text-to-SQL approach is employed to systematically extract key information from this database. By converting user queries into precise SQL statements, targeted and efficient searches can be performed within the database. This method

allows us to accurately retrieve relevant question-answer pairs, which then facilitates and enhances the processes within the generative model. The workflow of the proposed non-vector retrieval-augmented generation (NVRAG) model is depicted in Figure 2.

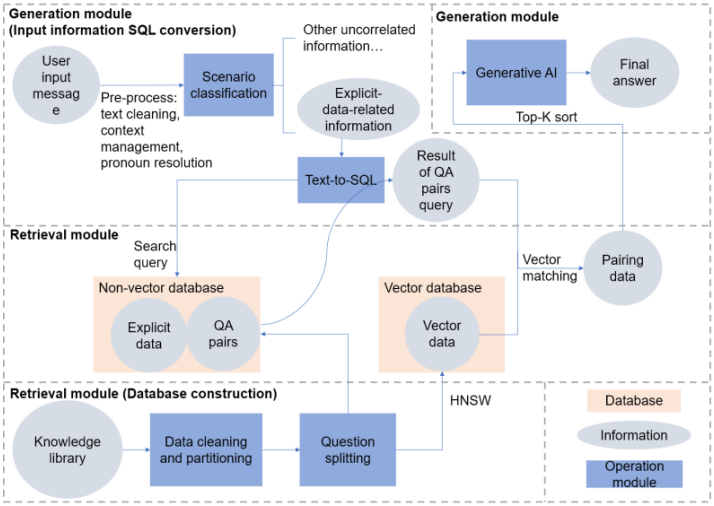

Fig. 2. Workflow of NVRAG model.

3.1 Retrieval module

In the original RAG model, question-answer splitting chunks and indexes text, storing it in a vector database and filtering relevant question-answer pairs through similarity matching algorithms. In NVRAG, we record essential information to be retained in a non-vector database during the question-answer splitting process. This information is typically explicit, consistently present in each document, easily identifiable and extractable (e.g., temporal information, numerical data, or Boolean values). It can narrow down the scope of the retrieval vector index when matching user inputs. To achieve question-answer splitting, we will add the following guiding statement to the original QA splitting prompt, so that it considers the influence of key_element when constructing QA pairs and marks it explicitly:

> *Match the content within <context></context> to see if it is related to {{explicit_element}}. Extract the relevant information.*

When user input is detected to contain SQL-related information (implemented in the generation module), the optimized user statement, after processing context and resolving references, will be passed to the text-to-SQL model for conversion into the corresponding SQL statement. This conversion process will utilize text-to-SQL technology, and through appropriate fine-tuning of the model, it can effectively transform natural language (i.e., user input) into SQL statements. Specifically, the converted SQL statements will include retrieval of display information, typically represented as a query statement with inclusion relationships that effectively reflect the user's needs and intentions. In this paper, this conversion process is crucial, as it ensures that the user's intent is accurately captured and transformed into a format that can be executed in the database. The search query step will match the vector data within the defined scope, thereby efficiently retrieving relevant information and ensuring that the query results align with the

user's expectations. This structured approach not only enhances the accuracy of the generated SQL queries but also improves the overall performance of the system, making the user experience more seamless and efficient. The prompt involved in this process is as follows:

Convert the content within <question></question> into an SQL query statement:
<question>
{{question}}
</question>
The database type you should match is {{db_type}}, and the content within <schema></schema> represents the table information that needs to be matched:
<schema>
{{schema}}
</schema>
Response requirements:

- *If you are unsure of the answer, please return None.*
- *You need to incorporate the current time into your response, with the current time being {{time_now}}.*
- *When the question contains temporal information, you need to match it in the {{column1}} field of the data table.*

Below is a conversion example.

<example>

Upon obtaining the SQL query, more precise and relevant QA pairs can be retrieved from the corresponding database. These pairs are then provided to the retrieval module, which narrows the search range from the entire knowledge base to a defined scope determined by key parameters through SQL filtering. Followed by these processes is the normal flow of the RAG module, where semantic matching and full-text search matching are involved. Then, the content with higher similarity is passed to the LLM for final response generation.

3.2 Generation module

This module primarily handles user inputs and efficiently delivers them to the retrieval module to obtain appropriate responses. The information is then passed to the generative AI for final answer generation.

Upon receiving user input, the generation module first performs a series of pre-processing steps, which include text cleaning, context management, and pronoun resolution to enhance clarity and coherence. Subsequently, the module classifies the user's input to effectively filter out potential explicit-data-related information, ensuring that sensitive data is handled appropriately and securely. This structured approach allows for improved interaction and response quality. The prompt for scenario classification is:

Analyze the following user input to determine whether the information contained is related to the specified scenarios, and provide a score (0-10) for each specified scenario.

Scoring criteria:

- Relevance of the input to the specified scenario.

- Alignment of the user's intent with the scenario.

- Presence of comparative or evaluative elements.

Please score each aspect separately and provide an overall score (the average of the three scores).

[Specified Scenario 1]:

...

[Specified Scenario 2]:

...

[User Input]:

...

When the retrieval module returns relevant vector data through Top-K sorting, the user's question is answered using a generative AI model, by integrating the user's input with information from the knowledge base.

4 Experiment

This study compares the capabilities of the original RAG and NVRAG models in extracting temporal information. Three criteria: response abilities, faithfulness and accuracy rate (AR) presented in Retrieval Augmented Generation Assessment (RAGAS) [18] will be evaluated for two models on the weather briefing dataset to assess their performance.

A FastGPT image was deployed on the server to separately test and validate the performance of the native RAG model and the NVRAG model, as well as to adjust them. In FastGPT, the entire knowledge base consists of three parts: library, collection, and data. FastGPT utilizes the PG Vector plugin of PostgresSQL as the vector retriever, with HNSW as the indexing method. MongoDB is used for accessing other non-vector data.

The dataset used in this study is the weather briefing dataset, which contains 100 consecutive weather data entries, encompassing multiple pieces of information, such as date, weather conditions, meteorological elements, clothing recommendations, travel suggestions, and summaries. This dataset was imported into FastGPT to generate the knowledge base.

The final generated knowledge base contains 1432 key-value pairs, corresponding to the 100 weather contents. The open-source model Chat2DB_sql_7B was used to construct the Text-to-SQL module, which was deployed on an A100 cloud server. In this experiment, the focus was primarily on assessing the model's ability to extract and respond to time information; thus, the model extracted the corresponding time information and passed it to the database, which was called upon during retrieval. User inputs in this experiment were generated by a LLM by the following prompt:

You now have a weather report-related database and a large language model that can extract knowledge from it to answer questions. This database contains 100 days of weather report

data since {{start_time}}. The weather reports include date, weather conditions (temperature, humidity, wind speed, precipitation probability, air quality), meteorological elements, clothing suggestions, travel suggestions, and summaries. Please generate {{num_total}} user inputs to test the model's response capability.
Requirements:

- *The generated user inputs should be as different as possible; no two user inputs should be exactly the same.*
- *User inputs must include time information, which can be a continuous time period or individual dates. The time intervals in your generated inputs should fall within the 100 days after {{start_time}} (including the start date).*
- *Try to include multiple dates in your content; you can ask about the weather conditions over a period or compare the weather on multiple dates.*
- *The generated content should be contextually relevant and meaningful; do not fabricate unreasonable input data.*

Which generates user inputs presented in Table 1.

Table 1. Example of LLM-generated user inputs

"Please tell me the weather from August 1 to August 5, 2024, including temperature and precipitation probability."
"Can you compare the weather on August 3 and August 7, 2024, to see which day is better for going out?"
"What are the wind speed and air quality like on August 6, 2024? Is it suitable for outdoor activities?",
……

After retrieving the knowledge base information from the retrieval module, the OpenAI GPT-3.5-turbo model is invoked as the generative model to respond to user inquiries. As for evaluation standards, faithfulness serves as a measure of the model's accuracy, while AR assesses the alignment of the generated answers with the initial questions or instructions. For all generated data, tests are conducted on both the original RAG model (provided by the FastGPT platform) and the NVRAG.

5 Results

In this study, a comparative analysis is conducted on the performance of the RAG model and the NVRAG model when handling general user inputs and time-related user inputs. The average data after removing outliers and cleaning is presented in Table 2.

Table 2. Performance of two models

		Faithfulness	AR	Response time
General user input	RAG	0.89	0.93	2.10s
	NVRAG	0.88	0.92	2.24s
Time-relevant user input	RAG	0.32	0.46	2.98s
	NVRAG	0.85	0.91	3.76s

The experimental results indicate that the NVRAG model outperforms the RAG model in addressing multi-index related queries, specifically those related to time ranges in this experiment.

For general user inputs, the faithfulness and AR of the RAG model and the NVRAG model are relatively close, with values of 0.89 and 0.88 for the RAG model and 0.93 and 0.92 for the NVRAG model. This suggests that both models can provide accurate and reliable answers when handling routine queries. However, in terms of response time, the RAG model slightly outperforms the NVRAG model, with response times of 2.10 seconds and 2.24 seconds, respectively. This is due the fact that the RAG model can retrieve relevant information from its vector database more quickly when processing general queries.

When processing time-related user inputs, the performance of the NVRAG model is significantly superior to that of the RAG model. Specifically, the faithfulness and AR of the NVRAG model reach 0.85 and 0.91, while the RAG model only achieves 0.32 and 0.46. This result indicates that the NVRAG model can provide more accurate and relevant answers when handling complex queries related to time. The performance difference is mainly attributed to the design of the NVRAG model. In the NVRAG model, a text-to-SQL-based algorithm is introduced to extract key information from a non-vector database using SQL queries. This approach enables the NVRAG model to better handle multi-key-value associated queries, particularly those involving complex time-related information. By converting user queries into SQL statements, the NVRAG model conducts targeted searches within its database, thereby improving the accuracy and relevance of the queries.

However, this improvement also leads to an increase in response time. When processing time-related user inputs, the response time of the NVRAG model is 3.76 seconds, compared to 2.98 seconds for the RAG model. Additionally, due to the increased number of retrieval and query steps, the consumption of tokens also increases. Nevertheless, the significant performance enhancement of the NVRAG model in complex queries makes the additional response time and token consumption acceptable.

6 Conclusion

This paper proposes a non-vector retrieval-augmented generation (NVRAG) model based on a non-vector database and text-to-SQL technology to handle complex queries involving multi-index associations. This method improves upon the original RAG model by introducing a non-vector database in the retrieval module to store relevant parameters, narrowing the embedding range, and enhancing indexing accuracy. Experimental results demonstrate that, compared to

the original RAG model, the NVRAG model exhibits significantly better performance in handling complex queries, especially time-relevant range queries. Specifically, the NVRAG model outperforms the RAG model in terms of faithfulness and accuracy rate (AR), particularly in scenarios involving multi-key-value association queries.

The primary advantage of the NVRAG model lies in its ability to extract key information from user queries more effectively through text-to-SQL conversion, enabling targeted database searches. This approach addresses the limitations of traditional RAG models when dealing with complex, multi-index-related problems, particularly in time range queries and multi-value optimization tasks. The NVRAG model can provide more accurate and relevant answers in such cases. However, this improvement comes at the cost of increased response time, primarily due to the additional SQL query and multi-step retrieval operations in the process. Nevertheless, the experimental results suggest that the performance gains of the NVRAG model more than compensate for the slightly longer response time, making it more competitive in handling complex query tasks.

Furthermore, the paper highlights that the NVRAG model imposes certain requirements on the document format used to construct the knowledge base. The documents should contain easily extractable, well-structured information to facilitate SQL query execution. Future research should focus on optimizing the response time of the NVRAG model while maintaining its high performance in complex queries. Additionally, exploring the application of the NVRAG model across broader domains and datasets will be essential to validate its generalizability and scalability in real-world applications.

References

[1] Zichong Wang, et al. (2024). History, development, and principles of large language models: an introductory survey. AI and Ethics. 1-17. 10.1007/s43681-024-00583-7.

[2] Wang Song, et al. (2024). Knowledge Editing for Large Language Models: A Survey. ACM Computing Surveys. 10.1145/3698590.

[3] Xiaoxi Li, Zhicheng Dou, Yujia Zhou, and Fangchao Liu. (2024). Towards a Unified Language Model for Knowledge-Intensive Tasks Utilizing External Corpus. In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA, pp.13.

[4] Yujia Zhou, et al. (2024). ROGER: Ranking-oriented Generative Retrieval. ACM Transactions on Information Systems. 42. 10.1145/3603167.

[5] Jun Guo, et al. (2024). EFRAG: Embedding-Fine Tuning Retrieval Augmented Generation for QA in power projects. 21-27. 10.1145/3689218.3689225.

[6] Zhengliang Shi, et al., (2024). Generate-then-Ground in Retrieval-Augmented Generation for Multi-hop Question Answering. 10.18653/v1/2024.acl-long.397.

[7] Lewis PSH, et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.

[8] Qingyao Ai, et al. (2023). Information Retrieval meets Large Language Models: A strategic report from Chinese IR community. AI Open. 4. 10.1016/j.aiopen.2023.08.001.

[9] Haonan Chen, Zhicheng Dou, Yutao Hu, Jirong Wen. (2024). Query-Oriented Data Augmentation for Session Search. IEEE Transactions on Knowledge and Data Engineering. PP. 1-13. 10.1109/TKDE.2024.3419131.

[10] Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao and Maojin Yang. (2024). Retrieval-Generation Synergy Augmented Large Language Models. 11661-11665. 10.1109/ICASSP48485.2024.10448015.
[11] Ge Jin, et al. (2024). Development of a liver disease-Specific large language model chat Interface using retrieval augmented generation. Hepatology (Baltimore, Md.). 10.1097/HEP.0000000000000834.
[12] Miao Jing, et al. (2024). Integrating Retrieval-Augmented Generation with Large Language Models in Nephrology: Advancing Practical Applications. Medicina. 60. 445. 10.3390/medicina60030445.
[13] Malkov Yu, Yashunin Dmitry. (2016). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. 10.1109/TPAMI.2018.2889473.
[14] Chen Jiawei, Lin Hongyu, Han Xianpei and Sun Le. (2024). Benchmarking Large Language Models in Retrieval-Augmented Generation. Proceedings of the AAAI Conference on Artificial Intelligence. 38. 17754-17762. 10.1609/aaai.v38i16.29728.
[15] Katsogiannis-Meimarakis George, Koutrika Georgia. (2023). A survey on deep learning approaches for text-to-SQL. The VLDB Journal, 32. 10.1007/s00778-022-00776-8.
[16] Zhao Wayne Xin, et al. (2023). A Survey of Large Language Models. arXiv:2303.18223.
[17] Baptiste Rozière, et al. (2023). Code Llama: Open Foundation Models for Code. arXiv:2308.12950
[18] Shahul Es, Jithin James, Luis Espinosa-Anke, Steven Schockaert. (2023). RAGAS: Automated Evaluation of Retrieval Augmented Generation. arXiv:2309.15217