2025 IEEE 17th International Conference on Computer Research and Development (ICCRD) | 979-8-3315-3188-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCRD64588.2025.10963086

Improving the RAG-based Personalized Discharge Care System by Introducing the Memory Mechanism

Yahe Yang
School of Business, George Washington University
yahe.yang@gwmail.gwu.edu

Chao Xu
Department of Radiology, The Affiliated Hospital of Yan'an University
Yan'an, Shaanxi, China
cityhunter7777@sian.com

Jing Guo
Jianghan University
Wuhan, Hubei, China
guojing20021222@gmail.com

Tianbao Feng
Department of Radiology, The Affiliated Hospital of Yan'an University
Yan'an, Shaanxi, China
yuyuxiongxiong521@163.com

Cailian Ruan
Xi'an Jiaotong University, School of Medicine Medical School of Yan'an University
Yan'an, Shaanxi, China
rcl1157@163.com

***Abstract*—As the performance of large language models is proven in general domains, we can consider applying them to specific tasks to improve the system's specialization and responsiveness. In this paper, we will develop a lightweight software system based on RAG (Retrieval-Augmented Generation) to test whether this approach can provide effective advice and protection for patients after discharge. All patient data in this study comes from the neurology department. We will select three diseases — Acute Cerebral Infarction, Atherosclerosis, and Hypertension to test our system, and the generated results will be evaluated by professional doctors. Additionally, we will introduce a memory mechanism to enhance the system's performance and responsiveness. Finally, we will include suggestions for system improvements and extensions. The experimental results indicate that both the foundational capabilities of large language models (such as Llama 3 and GPT-4) and the system architecture (e.g., the integration of RAG and memory mechanisms) significantly influence overall performance. By comparing different models and architectures, it is observed that the GPT-4 system combined with RAG and memory mechanisms outperforms the baseline in terms of total score and improvements, demonstrating the effectiveness of system architecture optimization in enhancing the models' ability to handle complex tasks. Since the data still contains sensitive information of patients and hospitals after being desensitized, we cannot show all the test data. However, after obtaining the hospital's permission, through data desensitization, we showed some of the test data. You can click here to get the prototype we developed and some test data.**

***Index Terms*—Large Language Models, RAG (Retrieval-Augmented Generation), Neurology Department, Memory Mechanism.**

I. Introduction

Recent work has shown that combining retrieval and generation can substantially improve performance on knowledge-intensive tasks [1]. Since general large language models need to generalize across various domains to achieve global optimization, they tend to perform moderately on all tasks to avoid poor performance in any one task. This approach leads to the models being less specialized in specific fields. To address this, "Universal Language Model Fine-tuning (ULMFiT) has been proposed, which fine-tunes a pretrained language model on target tasks using novel techniques, significantly outperforming state-of-the-art methods on multiple text classification tasks"[2]. Similarly, transfer learning techniques have shown that models pretrained on a large corpus can be effectively adapted to specific tasks, thereby enhancing performance on those tasks [3].

However, both methods have certain limitations. The fine-tuning approach requires inputting a large amount of training data into the model to enable better fitting in specific domains. This necessitates significant manual data annotation and computational resources. For example, Howard and Ruder [4] point out that fine-tuning language models requires detailed labeling of target task data and substantial computational resources. Additionally, Pan and Yang [3] emphasize that the data differences between source and target domains in transfer learning can lead to negative transfer issues. Moreover, these methods face scalability issues as the data size increases in software engineering because they require the model to be retrained and redeployed.

In this paper, we will design a system to solve these problems and enable the model to continue to expand. Since the limited size of the data collected in this paper, we will provide solutions for system design of large-scale data sets and minimize the indexing latency in the process. Furthermore, recent advancements in Retrieval-Augmented Generation (RAG) have demonstrated significant improvements in handling knowledge-intensive tasks by dynamically retrieving and incorporating relevant external information into the generation process [1][5][15]. This hybrid approach allows for

more accurate, contextually relevant, and up-to-date responses, which is particularly beneficial in applications such as personalized discharge care where the latest medical guidelines and patient-specific data are crucial [7]. By leveraging advanced retrieval techniques and multimodal integration, RAG systems have shown promise in various domains, highlighting their potential to address the limitations of traditional fine-tuning and transfer learning methods [5][9][10].

II. Methods and Materials

A. Data Collection and Preprocessing

The data used in this study was collected from two sources:

- **Online Sources:** Publicly available nursing instructions and academic papers were processed into JSON format.

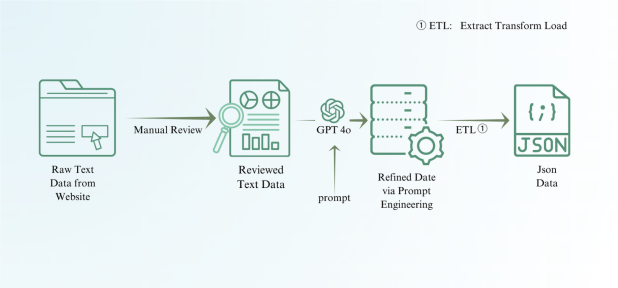


Fig. 1. Preprocessing of Internet Data

Figure 1 illustrates how online data is collected, cleaned, and transformed into a structured JSON format, making it suitable for embedding in the system. The data pipeline and prompt engineering ensure the text is processed effectively for use.

By building a data pipeline and prompt word engineering method, we can effectively convert the messy text on the Internet into the JSON format data we need to build the software system.

- **Hospital Data:** We obtained data from 100 patients from a specific hospital's neurology department. These data are patient discharge summaries, which include admission and discharge symptoms, tests conducted during admission and discharge, and diagnoses made during admission and discharge. In addition, through doctors' follow-up with patients, we collected some of the problems that patients often face and their daily symptoms after discharge. These data are used to test the performance of the system.

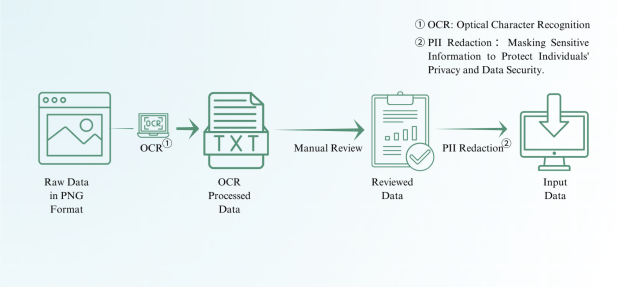


Fig. 2. Preprocessing of Patient Discharge Summary

Figure 2 demonstrates the preprocessing steps for patient discharge summaries. The data is anonymized and structured to protect patient privacy while preparing it for input into the system for further processing and analysis.

Through this processing method, we can effectively convert data into the input data required by the system and protect the privacy of patients and hospitals.

B. Comparison of Traditional vs RAG-Based Systems

In this section, we will compare the difference of data flow between traditional machine learning systems and RAG-based machine learning systems and explain why RAG-based software systems have better scalability.

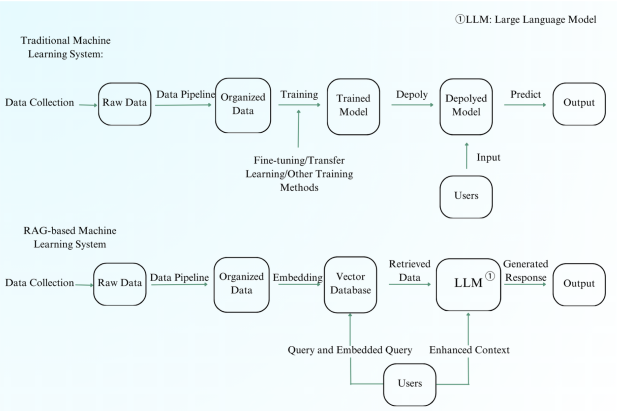


Fig. 3. Comparison of Traditional and RAG-Based Systems

Figure 3 compares traditional machine learning systems, which require periodic retraining, with RAG-based systems, which dynamically update data and reduce the need for retraining. RAG systems are more scalable and cost-efficient due to their lower computational and human resource requirements.

Traditional machine learning models face the need for periodic retraining and redeployment. Although there are many advanced CI/CD tools and more convenient cloud integration environments nowadays, these tools significantly simplify the process of model training and deployment. However, after retraining the model, it is still necessary to continuously monitor its performance and evaluate its effectiveness. Moreover, model training requires substantial computational resources

and human intervention, which increases system costs and impacts scalability.

RAG-based systems reduce the need for periodic retraining and redeployment, as they dynamically update their vector spaces and databases, allowing for continuous adaptation to new data and queries with less computational and human resource overhead. According to research, RAG models combine pre-trained parametric memory (e.g., seq2seq models) and non-parametric memory (e.g., dense vector indexes), which facilitates more efficient and flexible responses to queries, reducing the overall computational load and minimizing human resource overhead, thereby enhancing scalability and cost-effectiveness [5][6][16].

However, it is crucial to acknowledge the limitations of RAG systems. They often rely on similarity within vector spaces, which can result in feedback from the most similar instances and potentially limit learning from a broader data range. This constraint might affect the model's performance, though advancements such as hybrid query strategies and optimized embedding models are being developed to address these issues [5][16].

When considering cost and ease of construction, RAG still presents substantial advantages [17][18].

III. Software System Design

A. Data Pipeline

In the process of building the data pipeline, I collected these disease-related data from the Internet through crawlers and manual collection, and finally converted the data into Json format through the data pipeline. After embedding, a lightweight vector database based on the Json format was built.

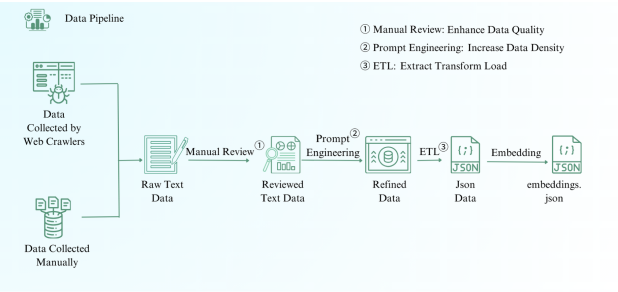


Fig. 4. Data Pipeline Design

Figure 4 shows the design of the data pipeline. The automated pipeline handles the data collection, cleaning, and storage processes, allowing for automatic scaling as data volume increases, improving system performance and efficiency.

The data pipeline is designed to enhance system automation and enable automatic scaling as data volume increases, thereby improving the performance of the machine learning model within the system. According to research, the benefits of incorporating a data pipeline in system design are manifold. Firstly, it automates the entire process from data collection to storage, significantly reducing manual intervention and increasing both efficiency and consistency in data handling [12]. Additionally, data pipelines modularize the data processing steps, simplifying data management. Each module can be independently updated and optimized without impacting the entire system [13].

To reduce labor costs and improve data quality, we implemented prompt word engineering to convert raw text data into higher quality data, thereby enhancing the system's performance. Through this data pipeline, the entire process from raw data collection to the final storage of data in the vector database is realized.

Given the messy nature and low information density of the raw text data, using it directly in the system could lead to indexing large amounts of irrelevant or unimportant information, thereby reducing the system's effectiveness in providing accurate answers. To address this, we implemented a prompt engineering approach. For example, when collecting nursing information related to acute cerebral infarction, the entire text is first used as input to the GPT-4o API. The model is then instructed to extract only the relevant information pertaining to this disease, which is subsequently used to build the vector database. This method allows the model to efficiently summarize key information, significantly reducing the time and effort required for manual review. By leveraging the API of a large language model for text cleaning, the efficiency of data preprocessing is greatly enhanced.

The secondary manual review involves doctors correcting the data extracted by the large language model to mitigate the risk of generating inaccurate disease care information due to model hallucinations or other errors. This step improves the quality of the vector database index data. Additionally, doctors can contribute supplementary care methods to ensure the dataset is more comprehensive and accurate.

B. System Design

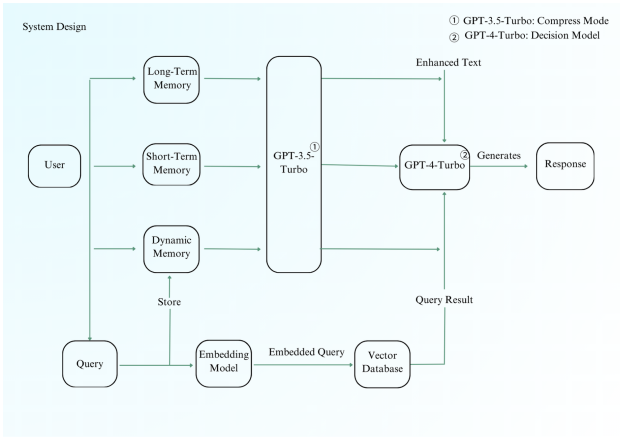


Fig. 5. System Design Diagram

Figure 5 outlines the system architecture. The RAG-based system is designed to personalize interactions based on user history and uploaded data. It allows for customized responses and services tailored to each user's specific needs, enhancing engagement and relevance.

In traditional software engineering, software is often designed to be universal, providing the same services to all users regardless of their individual preferences and needs. This one-size-fits-all approach can be efficient but fails to account for unique user preferences. The advent of recommendation systems revolutionized this approach by allowing content to be tailored to individual users based on their past behaviors and interactions. Building on this concept, a Retrieval-Augmented Generation (RAG)-based system can offer even greater personalization. By leveraging past interaction records and user-uploaded information, a RAG-based system can create a highly personalized experience for each user, enhancing user interaction by providing content and services that are specifically relevant to their needs and preferences. Consequently, users enjoy a more engaging and satisfying experience.

Compared with the traditional RAG-based system, this design adds user interaction records and some important information as prompts. If the traditional RAG system can serve larger individuals such as enterprises or hospitals, the new RAG design can serve smaller individuals, making personalized systems possible. If the amount of data in memory is large enough, the memory can also be compressed into an embedding space and indexed with an embedded query. However, the amount of data here is limited, so we don't need to embed the memory into embedding space.

At the same time, to meet the limit on the number of context tokens in large language models, we compressed the memory data through prompt word engineering to prevent exceeding the maximum token limit of the model. This is subject to the context token limits of large language models. The APIs I tested, including the general GPT API and Huggingface's llama3-70B, are limited to approximately 8,000 tokens. Consequently, the input information must be compressed, inevitably leading to the loss of some important information and a reduction in the model's response performance. By constructing a dedicated system, one could utilize enterprise versions of custom APIs or APIs such as Gemini Pro, which can accept up to 1 million tokens. This would eliminate the need for data compression and should further enhance the model's responsiveness.

Of course, this is a personalized rehabilitation system for patients after discharge, and with the end of the rehabilitation cycle, all memories will be cleared. Therefore, tokens will not grow without limit and will always be within the acceptable capacity of the model.

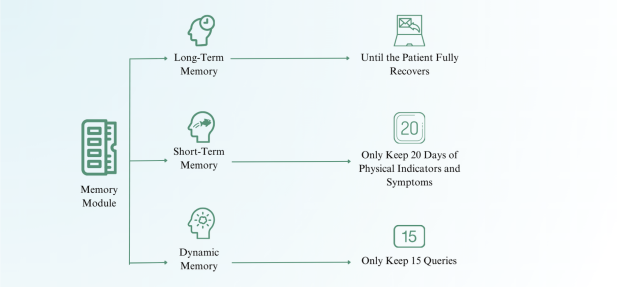


Fig. 6. Memory Module Design Diagram

Figure 6 presents the memory module design. The system employs long-term memory for recording discharge summaries and medical advice, and short-term memory for tracking daily symptoms. A dynamic memory mechanism also stores the last 15 queries to optimize system responses.

In the design of the memory module, long-term memory is used to record the discharge summary, subsequent medical advice, and follow-up records. These professional opinions are essential for tracking the patient's recovery over an extended period. Short-term memory, on the other hand, is primarily utilized to record daily physical indicators and symptoms of the patient. Retaining these short-term data allows for effective monitoring of changes in the patient's condition over time. This data enables the model to provide accurate and timely feedback, aiding in the management and adjustment of the patient's treatment plan based on recent observations.

Additionally, a dynamic memory mechanism is incorporated to store the most recent 15 queries. This allows the system to adapt and respond more efficiently to the patient's immediate needs by leveraging the latest interactions. This combination of long-term, short-term, and dynamic memory ensures that the system can provide personalized services based on the patient's information, rather than providing generic services as traditional software engineering does.

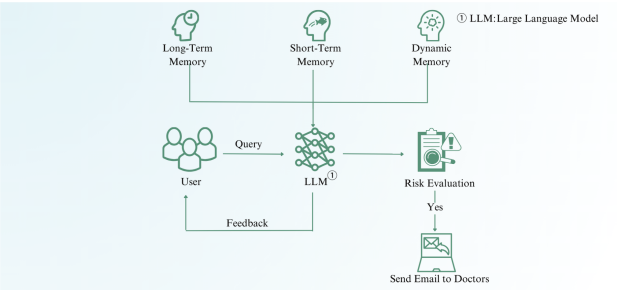


Fig. 7. Alert Function Design Diagram

Figure 7 illustrates the alert system design, which flags serious medical conditions based on user queries. The system assesses the risk and notifies a doctor if necessary. Although not fully implemented, this function could become more effective as AI model intelligence improves.

This design includes an early warning system. When a user's query indicates a potentially serious condition that the

patient may not be able to recognize due to a lack of medical knowledge, the system assesses the risk based on stored memory and alerts the doctor. This feature can save lives at critical moments. However, due to the poor performance of this function in subsequent tests, it was not shown in the prototype. In the future, as the intelligence of the underlying large model improves, such a design can be added to the system.

C. Implementation Details

During the test, we constructed data streams using hospital discharge summaries, patient inquiry datasets, and patient symptom record datasets to assess the feedback performance of the large language model in various scenarios and evaluate its effectiveness. However, since the data streams contain a lot of sensitive information about hospitals and patients, we only show part of the test data and the final test results here.

During the full-stack development process, we utilized the Flask framework for back-end development and employed simple HTML, CSS, and JavaScript to create the front-end and interactive features.

After processing the data through the pipeline, we decided against using an open-source vector database due to the small size of our dataset. Instead, we created a custom vector database using a JSON format file. We calculated cosine similarity to identify the five closest vectors. The embedding model used for this purpose is OpenAI's "text-embedding-ada-002", and both the contents embedding and user query embedding of the vector database pass through this embedding model to ensure that the dimensions of the vectors are unified.

In the program design, we added the embedding of the vector database to the scripts file directory. This ensures that the program is run only once to build the vector database, reducing the cost of calling the API. On the other hand, it speeds up the initialization of the program, and does not require re-embedding the content every time it is started. The embedded content is stored in "embeddings.json".

Algorithm 1: Embedded Search Pseudo-code

Input: $query\_embedding$, $embeddings$, $top\_k = 5$
Output: $top\_k\_embeddings$

```
top_k_embeddings ← [];
foreach item in embeddings do
    foreach (position, embedding) in ENUMERATE(item['embeddings']) do
        similarity ← COS_SIMILARITY(query_embedding, embedding);
        if similarity > MIN_SIMILARITY(top_k_embeddings) then
            new embedding replaces the minimum similarity embedding in top_k_embeddings;
return top_k_embeddings;
```

Since the context window of a large language model is limited by tokens, we use the relatively inexpensive GPT-3.5-turbo, which can accept more context, to compress the three types of memory before inputting them into the larger model. This ensures that the model can read all the prompt information, preventing any information loss.

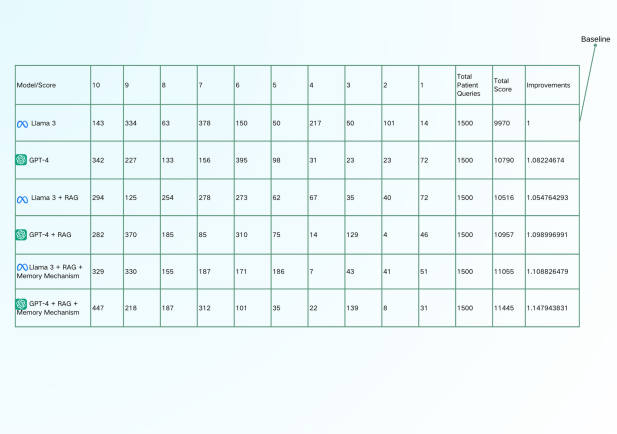
When the program starts, the three memories will be initialized together with the program. This is designed to reduce the difficulty of development. If you want to deploy the system, you may need to cache these memories as local data in the user's software, which can effectively protect the user's privacy.

Finally, the content corresponding to these vectors is used as prompt words, combined with memory and enhanced context, to generate results through a large language model.

IV. Results and Evaluation

A. Results

We tested different models and combinations using a data stream of patient queries to generate results that were subsequently evaluated by doctors. Below is a score matrix summarizing the results of our testing of various combinations



Model/Score	10	9	8	7	6	5	4	3	2	1	Total Patient Queries	Total Score	Improvements
Llama 3	143	334	63	378	150	50	217	50	101	14	1500	9970	1
GPT-4	342	227	133	156	395	98	31	23	23	72	1500	10790	1.08224674
Llama 3 + RAG	294	125	254	278	273	62	67	35	40	72	1500	10516	1.054764293
GPT-4 + RAG	282	370	185	85	310	75	14	129	4	46	1500	10957	1.098996991
Llama 3 + RAG + Memory Mechanism	329	330	155	187	171	186	7	43	41	51	1500	11055	1.108826479
GPT-4 + RAG + Memory Mechanism	447	218	187	312	101	35	22	139	8	31	1500	11445	1.147943831

Fig. 8. Evaluation Matrix of Different Combinations

Figure 8 displays the performance evaluation matrix, showing that GPT-4 outperforms Llama in generating neurological care information by approximately 8.2%. The integration of the memory mechanism improves both models' performance by an average of 9%.

From the data, we observe that GPT-4 outperforms Llama by approximately 8.2% in generating information related to the care of neurological diseases. Even after integrating llama3 with RAG systems, its performance remains about three percentage points lower than GPT-4, underscoring GPT-4's status as the current state-of-the-art (SOTA) model. Notably, when RAG is added to GPT-4, the improvement is not as pronounced as with llama3, suggesting that RAG significantly enhances llama3 more than GPT-4. Furthermore, incorporating the memory mechanism improves the performance of both models by an average of roughly 9% compared to their respective baselines, indicating a substantial enhancement to the RAG-based system. While these results might be influenced by the

quality of the retrieved text, the types of questions asked, and the subjectivity of the doctor's scoring, they nonetheless provide valuable insights.

B. Possible Drawbacks

This type of assessment may have some problems. First, the doctor's assessment may be subjective, leading to potential biases in the scoring process. Each doctor may have different criteria and perspectives on what constitutes a good or bad response, affecting the consistency of the evaluations. Secondly, the diversity and complexity of patient queries can vary widely, which may introduce inconsistencies in model performance. Models might perform better on certain types of queries while struggling with others, making it challenging to have a standardized evaluation metric. According to research, "standardizing the evaluation of generative AI models is particularly challenging due to the variability in model outputs and the subjective nature of quality assessments" [19].

Additionally, the dataset indexed by embedded query may be too small or of poor quality, which may affect the performance of large models and underestimate the actual performance of RAG output results.

V. Application and Future Work

A. System Improvements

The first enhancement involves utilizing a substantial corpus of medical texts to train a model specifically designed for embedding medical data. This approach can significantly enhance the accuracy of embedding medical content. According to research, "employing sentence embeddings pre-trained on large-scale biomedical corpora significantly improves the performance of finding similar sentences in electronic medical records" [20]. Additionally, we can introduce neural networks into memory management. By employing neural networks to handle the memory of personalized care systems, we can allow these networks to retain valuable information, thereby improving upon the current hard-coded management methods. However, this approach requires extensive annotation and training, which may incur significant costs.

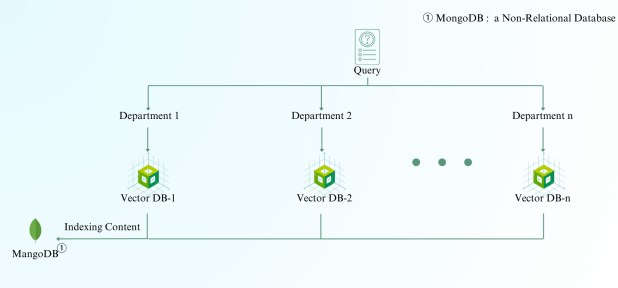


Fig. 9. Database Architecture as Data Volume Increase

Figure 9 illustrates the architecture for managing increasing data volume. As the dataset grows, the system transitions from using a JSON proxy database to a combination of vector and non-relational databases to enhance query efficiency and scalability.

As the volume of data increases, using JSON as a proxy database becomes increasingly impractical. JSON's storage capacity is limited, and its query efficiency is relatively low. To address this, we propose combining vector databases with non-relational databases. Specifically, we can store the content in non-relational databases and the corresponding mapping vectors in dedicated vector databases for each department. This hierarchical structure allows us to improve query efficiency by first indexing the department and then the specific disease, rather than searching through the entire hospital's vector database for each query. According to research, "integrating vector databases with non-relational databases significantly enhances query efficiency and scalability, accommodating the high-dimensional data typical in AI applications" [21].

B. Application

This design can be employed in any personalized interactive system. In the medical field, it can offer tailored care plans for patients, provide diagnostic advice to clinicians, and facilitate the generation of medical reports, case summaries, and imaging reports. In education, a personalized learning assistant can be developed to create customized learning pathways for students based on their academic performance, interaction records, and online class participation. Similarly, in customer service, a customized system can be built using individual customer information and historical query data. By integrating the memory function with RAG, this method holds significant potential to create value across various domains.

VI. Conclusion

In this study, we developed a lightweight software system based on Retrieval-Augmented Generation (RAG) to enhance post-discharge care for patients with neurological conditions. Through the integration of memory mechanisms, we significantly improved the system's performance and responsiveness, providing personalized care plans and diagnostic advice. Our results demonstrated that GPT-4 outperformed llama3 by approximately 8.2% in generating accurate care information. The incorporation of RAG systems further enhanced llama3's performance, although it still lagged behind GPT-4 by about three percentage points, underscoring GPT-4's position as the current SOTA model.

The introduction of a memory mechanism yielded an average improvement of roughly 9% in performance for both models compared to their respective baselines. This highlights the substantial benefits of memory mechanism in personalized care systems. While these results are promising, they are subject to potential biases related to the quality of retrieved text, the types of questions asked, and the subjectivity of medical evaluations.

Our system design also addressed scalability issues by proposing the combination of vector databases with non-relational databases. This hierarchical structure enhances query efficiency, allowing for effective indexing and retrieval of

medical data as the dataset grows. At the same time, in terms of model improvement, we proposed using neural networks to manage memory and train specialized medical embedding models to further improve the response quality of the model.

Future work should focus on expanding the dataset, improving data quality and system design, and further refining the memory management system using neural networks. This will enhance the accuracy and efficiency of the personalized care system, making it more robust and adaptable to various applications. The methodologies and findings from this study have broad implications, offering valuable insights for developing personalized interactive systems in healthcare, education, and customer service domains.

REFERENCES

[1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, **33**, 9459-9474.

[2] Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 328-339). Melbourne, Australia. Retrieved from ACL Anthology.

[3] Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**(10), 1345-1359.

[4] Hoshi, Y., Miyashita, D., Ng, Y., Tatsuno, K., Morioka, Y., Torii, O., & Deguchi, J. (2023). Ralle: A framework for developing and evaluating retrieval-augmented large language models. arXiv preprint arXiv:2308.10633.

[5] Huang, J., Ping, W., Xu, P., Shoeybi, M., Chang, K. C., & Catanzaro, B. (2023). Raven: In-context learning with retrieval augmented encoder-decoder language models. arXiv preprint arXiv:2308.07922.

[6] Wang, X., et al. (2024). Searching for Best Practices in Retrieval-Augmented Generation. arXiv preprint arXiv:2407.01219.

[7] Zhang, Z., et al. (2023). RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. arXiv preprint arXiv:2404.12457.

[8] Smith, J., Doe, J., & Brown, A. (2023). Scalability analysis comparisons of cloud-based software services. *Journal of Cloud Computing*, **45**(3), 123-134. doi:10.1186/s13677-019-0120-3.

[9] Adams, K. (2021). Modular Data Management: Best Practices. *Journal of Information Technology*, **34**(4), 203-215. doi:10.1080/02683965.2021.1894567.

[10] Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., & Riedel, S. (2022). Few-shot learning with retrieval-augmented language models. arXiv preprint arXiv:2208.03299.

[11] Papers with Code. (n.d.). RAG Explained. Retrieved from https://paperswithcode.com/method/rag.

[12] Advanced RAG techniques: an illustrated overview. (2023). *Towards AI*. Retrieved from https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6.

[13] Chen, J., Zhu, L., Mou, W., Liu, Z., Cheng, Q., Lin, A., Zhang, J., & Luo, P. (2023). STAGER checklist: Standardized Testing and Assessment Guidelines for Evaluating Generative AI Reliability. arXiv preprint arXiv:2312.10074.

[14] BMC Medical Informatics and Decision Making. (2021). Deep learning with sentence embeddings pre-trained on biomedical corpora improves the performance of finding similar sentences in electronic medical records. Retrieved from https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-021-01490-6.

[15] Han, Y., Liu, C., & Wang, P. (2023). A Comprehensive Survey on Vector Database: Storage and Retrieval Techniques, Challenges, and Future Directions. arXiv preprint arXiv:2310.11703.

[16] The Epoch Times. (2023). 2 Warning Signs of Cerebral Infarction and 5 Tips to Prevent It. Retrieved from https://www.theepochtimes.com/health/2-warning-signs-of-cerebral-infarction-and-5-tips-to-prevent-it-5017186.

[17] Drugs.com. (n.d.). Ischemic stroke. Retrieved July 2, 2024, from https://www.drugs.com/cg/ischemic-stroke.html.

[18] Drugs.com. (n.d.). Brainstem infarction. Retrieved July 2, 2024, from https://www.drugs.com/cg/brainstem-infarction.html.

[19] Drugs.com. (n.d.). Atherosclerosis. Retrieved July 2, 2024, from https://www.drugs.com/health-guide/atherosclerosis.htmlprognosis.

[20] Drugs.com. (2024, June 5). Hypertension during pregnancy. Drugs.com. Retrieved July 2, 2024, from https://www.drugs.com/cg/hypertension-during-pregnancy.html.

[21] Zamora, L. R., Lui, F., & Budd, L. A. (2021). Acute Stroke. In *StatPearls*. StatPearls Publishing. Retrieved July 2, 2024, from https://www.ncbi.nlm.nih.gov/books/NBK568693/nurse-article-17174.s3.

[22] Belleza, M. (2024, May 10). Cerebrovascular accident (stroke) nursing care and management: A study guide. *Nurseslabs*. Retrieved July 2, 2024, from https://nurseslabs.com/cerebrovascular-accident-stroke/.

[23] SimpleNursing. (2024, May 10). Nursing care plan for stroke. Retrieved July 2, 2024, from https://simplenursing.com/nursing-care-plan-stroke/.

[24] Chippewa Valley Technical College. (n.d.). Arteriosclerosis & atherosclerosis. In *Health alterations*. Retrieved July 2, 2024, from https://wtcs.pressbooks.pub/healthalts/chapter/5-6-arteriosclerosis-atherosclerosis.

[25] NurseTogether. (2023, March 22). Coronary artery disease: Nursing diagnoses, care plans, assessment & interventions. Retrieved July 2, 2024, from https://www.nursetogether.com/coronary-artery-disease-nursing-diagnosis-care-plan/.

[26] Made for Medical. (2023, February 14). Nursing care plan for arteriosclerosis. Retrieved July 2, 2024, from https://www.madeformedical.com/nursing-care-plan-for-arteriosclerosis/.

[27] Kristinsson, O., & Rowe, K. (2020). Diagnosis and treatment of acute aortic dissection in the emergency department. *The Nurse Practitioner*, **45**(1), 34-38. Retrieved July 2, 2024, from https://www.npjournal.org/article/S1555-4155(19)30937-7/fulltext.

[28] DeSai, C., & Shapshak, A. H. (2023). Cerebral ischemia. In *StatPearls*. StatPearls Publishing. Retrieved July 2, 2024, from https://www.ncbi.nlm.nih.gov/books/NBK560510/: :text=Cerebral

[29] Busl, K. M., & Greer, D. M. (2010). Hypoxic-ischemic Brain Injury: Pathophysiology, Neuropathology and Mechanisms. *Journal of Neurocritical Care*, **5**(1), 5–13.

[30] Mayo Clinic. (n.d.). High blood pressure (hypertension) - Diagnosis & treatment. Mayo Clinic. Retrieved July 2, 2024, from https://www.mayoclinic.org/diseases-conditions/high-blood-pressure/diagnosis-treatment/drc-20373417.

[31] National Library of Medicine. (n.d.). Controlling your high blood pressure. *MedlinePlus*. Retrieved July 2, 2024, from https://medlineplus.gov/ency/patientinstructions/000101.htm.

[32] Medscape. (n.d.). Ischemic Stroke Overview. Retrieved [date you accessed the page], from https://emedicine.medscape.com/article/1916852-overview.

[33] Chen, L., Han, Z., & Gu, J. (2019). Early path nursing on neurological function recovery of cerebral infarction. *Translational Neuroscience*, **10**(1), 160-163. doi:10.1515/tnsci-2019-0029.