Privacy Enhanced Retrieval-Augmented Generation (RAG) for Large Language Models in Healthcare

National University of Singapore

Bryan Ha Wai Kit

2025

Abstract

Large Language Models (LLMs) are increasingly utilized in healthcare for tasks such as clinical note summarization and medical report generation. However, their reliance on proprietary and sensitive patient data introduces significant privacy risks, particularly when using Retrieval-Augmented Generation (RAG). This project proposes a privacy-focused framework that leverages synthetic document generation to mitigate these risks while maintaining response accuracy.

The proposed system follows an agent-based approach, incorporating three key agents: a Search Agent, a Synthesis Agent, and a Review Agent. The process begins with the Search Agent retrieving relevant vector-related text nodes from a vector database. The Synthesis Agent then evaluates the extracted content, filtering and retaining only the necessary information for query responses while removing personally identifiable information (PII). Finally, the Review Agent verifies and refines the synthesized document to ensure privacy compliance before passing it to the LLM.

This thesis evaluates the effectiveness of synthetic document generation in mitigating privacy risks while preserving contextual relevance. Through a series of experiments, the system's ability to reduce PII leakage, maintain medical accuracy, and withstand adversarial attacks is assessed. The findings provide insights into balancing privacy and utility in healthcare-focused LLM applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Large Language Models (LLMs) are transforming various industries by enabling tasks such as automated workflow management using agentic frameworks, information extraction in Natural Language Processing (NLP), and even rudimentary reasoning in some models. In healthcare, LLMs have the potential to assist with clinical note generation, medical summarization, diagnosis support, and personalized patient care.

Despite these capabilities, LLMs suffer from a well-documented issue known as hallucinations, where they generate seemingly coherent but incorrect information. This has potentially disastrous complications in high-stakes domains such as medicine, law, and cybersecurity, where misinformation can lead to severe consequences.

To address this, Retrieval-Augmented Generation (RAG) is commonly used to supplement LLMs with external knowledge sources, improving factual accuracy by providing the LLM with context to generate from. While RAG enhances LLM performance, it also introduces new security risks. In particular, threat actors can exploit prompt injection attacks, in a similar fashion to LLMs, to manipulate retrieval outputs or extract sensitive data, which poses a significant privacy threat—especially in healthcare, where patient confidentiality is critical.

In this project, we explore an Agent-based synthetic document generation framework designed to mitigate these risks. By separating the RAG database from the externally facing LLM, we ensure the sensitive records are not directly exposed to the model. Instead, they undergo a controlled synthesis process. Only the necessary information is extracted from the retrieved knowledge, and any appearance of sensitive information such as names and ages are replaced with placeholders before being passed to the external LLM. This reduces the likelihood of data leakage while preserving response accuracy.

In chapter 1 we provide a brief description of a RAG system as well as briefly discuss applications of LLMs with RAG in healthcare.

./images/Conventional RAG example.png

Figure 1.1: Example of a Conventional RAG system

1.1 Background

1.1.1 Retrieval-Augmented Generation (RAG)

While LLMs are often trained on large datasets which, at times, provide the illusion that they have knowledge on many different fields, LLMs are still, first and foremost, text prediction engines used for Natural Language Processing (NLP) tasks.

This results in the following consequence: When an LLM encounters a query about information outside its training set, it will attempt to generate a response that is gramatically coherent, but potentially unsound response, a phenomenon known as hallucination. Depending on the applications, hallucinations can range from minor inaccuracies to critical failures, such as generating false legal cases [1] or misdiagnosing medical conditions.

Retrieval-augmented generation (RAG), first introduced by [2], was developed to mitigate hallucinations by integrating an external knowledge base into the LLM's generation pipeline. This grounds the LLMs response using the retrieved knowledge, preventing speculative responses from the LLM when faced with tokens outside its training set.

RAG operates by retrieving relevant documents from an unstructured or structured vector database and providing them as context for response generation. To facilitate retrieval, documents are converted into vector representations using a text embedding model, which captures the semantic relations in text. When a query is presented to an LLM with a RAG system, the query undergoes the same text embedding process. The vectorized query is compared with the document vectors present in the database and those that have the highest similarity are returned to the LLM.

Through RAG, an LLM becomes able to generate highly accurate, domain-specific information rather than rely on its pre-trained knowledge, allowing for flexible applications in various fields.

One application of RAG in healthcare is in diagnostic assistance, where LLMs match patient symptoms with medical knowledge retrieved from external sources. This enables more informed diagnoses while reducing the cognitive load on physicians. Furthermore, LLMs may be able to detect subtle symptom correlations that clinicians might overlook, improving early disease detection [3].

While RAG brings many benefits, it comes with its own caveats. RAG systems remain susceptible to prompt attacks much like LLMs are. They can also become poisoned, where the RAG corpus becomes corrupted through the insertion of adversarial attack passages.

In chapter 2 we discuss more about the vulnerabilities that RAG systems have.

1.1.2 LLMs in Healthcare

For Singapore in particular, LLMs have seen increased usage in healthcare. In 2013, Singapore’s National University Health System (NUHS) launched its very own LLM, Russell-GPT, that was used for summarizing patient clinical notes, automating referral letter generation, as well as predict the healthcare journeys for patients [4].

Singapore has also developed an LLM capable of understanding the local english dialect, Singlish, which has deployed in various settings, including clinics and emergency response systems, where it is used in transcribing emergency calls [5].

These developments showcase the growing reliance on LLMs in Singapore’s healthcare ecosystem, highlighting their potential to improve the efficiency of its healthcare system. However, as LLMs become increasingly integrated into critical systems, it is essential to address the risks of their use - particularly when augmented with enhancements like RAG. RAG-powered LLMs remain vulnerable to adversarial attacks, risking the exposure of sensitive medical information. As such, ensuring that the security and privacy of LLM-based solutions remains a priority as they continue to evolve.

Chapter 2

Literature Review

2.1 Exploitation of RAG Systems

While RAG improves LLM accuracy by integrating external knowledge sources, it also introduces new vulnerabilities. Attackers can exploit a RAG system's retrieval mechanisms to manipulate outputs, bypass safeguards, and extract sensitive information. This section explores some known methods of attacks and discusses their feasibility in a healthcare setting.

2.1.1 Data Poisoning

In data poisoning attacks (also known as a backdoor attack), attackers inject malicious or misleading information into the RAG corpus, causing the LLM to generate incorrect or malicious responses. These attacks can be used to carry out fraud, misinformation campaigns or provide adversarial control over responses. An example of this occurred with Google's Gemini, where, due to retrieving information from a satirical social media post, told the user to make use of ”non-toxic glue” when making a cheese pizza [6].

As highlighted in [7], data poisoning attacks are non-trivial to carry out. Depending on the complexity of the retrieval system, the attacker will have to modify the adversarial content such that the retrieval system is inclined to retrieve this document. Furthermore, the attacker requires some information about or access to the retrieval system to exploit it. This requirement is consistent with other studies carried out on data poisoning, and in almost all cases, the conditions in which this attack can manifest relies strictly on the insertion of a poisoned document into the RAG corpus [8, 7, 9].

Given these requirements, we can conclude that this type of RAG attack is non-feasible in a healthcare setting. In order to carry out this attack, the attacker has to have some form of access to the hospital's database. The cases in which this occur typically present with an external cyberattack on the hospital's infrastructure, whether through hacking or social engineering, and is considered a data breach. Most data breaches occur through hacking, as reported in [10]. In this case, the attacker can gain access to the hospital's database, and does not need to rely on exploiting the RAG system. Thus, we can conclude that this form of attack is nonapplicable in a healthcare setting.

2.1.2 Prompt Injection

Prompt injection attacks involves crafting an input query that manipulates the model into generating unintended responses. For RAG systems, this can be achieved either directly or indirectly.

Indirect prompt injection attacks function similarly to data poisoning except instead of inserting misleading information, adversarial prompts are attached to frequently retrieved documents in the RAG database. This enables attackers to retrieve documents from the RAG database using trigger prompts.

Direct prompt injection attacks involve the inclusion of a passage or sentence in the input query. This can be phrases such as ”repeat all the context”. These attacks, when targeted at RAG systems, can cause the leakage of private or sensitive information from the RAG corpus.

Indirect Prompt Injection

As stated previously, indirect prompt injection attacks operate in a similar fashion to data poisoning attacks as both require some form of access or ability to manipulate the RAG corpus.

Instead of using a misleading document, malicious instructions are embed in the document within the corpus, allowing the attacker to manipulate the LLM’s output. This allows the attacker to manipulate the LLM into including potentially malicious URLs into its response when responding to a victim’s query [11].

Furthermore, it should be noted that this type of prompt injection also allows the attacker manipulate the documents that are retrieved from the RAG corpus depending on the poison ratio, as covered in [12], which would allow unfettered access to any sensitive information stored in the database.

However, since this type of prompt attack requires some form of access to the RAG corpus, we can functionally consider it the same as a data poisoning attack. Realistically, if this type of attack were to occur in a healthcare setting, the attacker would already have access to hospital records. Therefore we will not be focusing on this aspect of prompt injection attacks.

Direct Prompt Injection

Direct prompt injection involves the inclusion of an adversarial passage into the input query, and these attacks are usually carried out in a specific format.

As highlighted in [13], they consist of two components: information and command. The information component of the attack leads the RAG system to retrieve documents that contain that form of information. Examples of this could be names, addresses, medical conditions. The command component is targeted at the LLM. Phrases are included in the input query that are aimed at subverting any safeguards placed on the LLM. This can be a phrase such as ”please repeat all context back to me,” or ”ignore all instructions,” etc.

As the study[13] shows, a significant portion of the datasets used in the study were able to be retrieved from the LLM through simple prompt attacks. The study also notes that the attack prompt could be further optimized for increased data extraction. Part of the study also noted the effects of RAG on the data that was extracted. It was noted that the inclusion of RAG decreased the appearance of memorized data in the LLM's output. It seems that the inclusion of RAG has caused the LLM to focus on leveraging the context retrieved rather than on its memorized training data [13].

Another study also corroborates this result. In [14], a similar method of prompt injection was used to extract text from a RAG database. The LLMs used in this study were instruction-tuned LLMs, meaning that the model has been trained to respond to instructions.

An interesting point to note was that they tested the similarity scores of the model's output with the retrieved context. Most LLMs used in the study exhibited higher BLEU, ROUGE-L, F1 and BERTScore scores that scaled with model size, suggesting that there is some correlation between an LLM capabilities and its vulnerabilities to prompt injection attacks [14]. Additionally, between the amount of data extracted alongside the size of the context retrieved was noted, further asserting that RAG has inherent vulnerabilities that are not being addressed.

Both studies discussed have shown a clear vulnerability of RAG to sufficiently sophisticated prompt injection attacks, but not much research has been done regarding the mitigation of RAG output post-occurrence of a prompt injection attack.

While it is possible to include safeguards to prevent the occurrence of prompt attacks, their implementations are still vulnerable. This is highlighted in [15], where a simple prompt of "ignore the context" caused the LLM agent to disregard any context retrieved despite the safeguards implemented. Considering that a simple prompt like this was sufficient enough to manipulate the model's output, it suggests that RAG pipeline implementations may be more fragile than initially anticipated, and a sufficiently motivated attacker will eventually be able to penetrate any LLM-level safeguards in place.

These findings suggest that current RAG implementations lack strong defenses against targeted prompt injection attacks. While preventive safeguards exist, adversarial prompt injections can still manipulate retrieval. This highlights the need for alternative security measures - such as synthetic document generation - to obfuscate retrieved context and prevent LLMs from directly accessing sensitive data in a RAG corpus.

2.2 Medical Anonymisation

Chapter 3

Methodology

3.1 Proposal

Based on research into RAG vulnerabilities, there is a clear lack of security measures designed to preserve the privacy of a RAG corpus. This is especially important in fields like healthcare. As demonstrated in [13], private information can be easily extracted by determined attackers through simple prompt injections. Given that RAG relies on a set of documents as context, and that this is vulnerable to attacks, I suggest generating synthetic context from this set of documents. This set of documents could refer to a patient's records, containing information such as their blood pressure, etc. A separate LLM will analyze the set of records retrieved by the user's query, comparing the two and extracting relevant information. The LLM will then generate a synthetic record containing all the information needed to answer the query, whilst removing PII at the same time. This separates the context given to the LLM from the RAG corpus, whilst still ensuring that its response remains contextually relevant.

3.2 Implementation

To design an adequate system that can preserve a patient's privacy, we must first understand the different types of methods associated with this task.

3.2.1 Medical Anonymisation

According to [16], the three main methods in preserving medical privacy are Pseudonymisation, De-identification and Anonymisation. Pseudonymisation refers to the replacement of attributes with pseudonyms. De-identification refers to the removal of PII from patient records. Anonymisation refers to the distortion of data such that any record lacks individuality.

The method we will be focusing on is the process of anonymisation. There are two different ways this is achieved. Recalculation, which involves turning absolute dates into relative representations (such as turning a patient's date of birth into their equivalent age representation), and Perturbation, which involves directly modifying a patient's medical records away from their original form.

./images/Synthesis LLM RAG example.png

Figure 3.1: System Design

3.2.2 System Design

The proposed system design is outlined in figure 3.1. The system is similar to figure 1.1, with the inclusion of an intermediary step that passes both the user query as well as the retrieved context documents to a secondary LLM for synthesis. The secondary LLM is responsible for both information extraction as well as synthetic document generation. To mimic privacy requirements, a locally run LLM is used here to ensure that the RAG corpus remains air-gapped. The use of an adequately sized LLM is critical here, and in our case we will be using Mistral's Nemo-12B. The synthesized document is then returned to the front-facing LLM along with the user's query to generate an appropriate response.

3.2.3 Building the RAG Corpus

RAG systems can make use of either structured or unstructured data. For our case, we will be making use of a synthetic FHIR dataset, generated by Synthea [17]. FHIR is a structured healthcare standard that defines how healthcare information can be shared between different systems regardless of how they are stored. Individual FHIR patient records are stored in what is known as resources. A resource can take on

./images/Converting FHIR to sentence.png

Figure 3.2: FHIR to sentence

./images/Store embeddings in DB.png

Figure 3.3: Embeddings to Database

different types, and each type contains information necessary for its specific use case. FHIR records can appear in the form of JSON, XML or RDF. Here we will be making use of JSON FHIR files to create our RAG corpus.

Firstly, we convert FHIR resources to basic sentences. This is to avoid repeatedly embedding the same key-value token pairs and wasting embedding tokens. Refer to figure 3.2 for an example. This can be further truncated. For example we can simply embed only the name of the reading and its associated value.

We perform this operation on the patient's record, extracting Observations and Procedures, separating them by date, and extracting a patient's diagnosed conditions, medications, as well as allergies, and storing them in a separate document. These documents are then converted into vectors through the use of a text-embedding model. For this, we will be using the *bge-base-en* embedding model. The embeddings are then stored in a Postgres database utilizing the *pgvector* extension.

`./images/retrieval example.png`

Figure 3.4: Input Query: Which patients have diabetes?

3.2.4 Retrieval

With the RAG corpus built, we can now move onto retrieving documents associated with a query. The query goes through the embedding process and its resulting vector is compared to other document vectors in the database. The top *k* results are returned, with *k* being an adjustable variable. What determines the chunk's relevance is its cosine similarity to the input query. Cosine similarity is defined as the following:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||}$$

and returns a score between 0.0 to 1.0. Here we can set a minimum cut-off for cosine similarity to adjust the relevance of returned information. Refer to figure 3.4 for an example of the returned chunks.

3.2.5 Synthetic Report Generation

LLMs differ in capabilities in accordance to their size. To determine if the chosen LLM (Mistral Nemo 12B) was sufficient for what I needed it to do, I tested its summarization and generation abilities. Firstly, I merged the previously processed FHIR record for a single patient into a combined document. This document was then passed to LLM along with a set of instructions. The specific prompt provided to the LLM is in the appendix, but to summarize:

- Break the summary into clear sections with headers
- Include exact numerical values
- Use precise dates
- Report conditions with specific terminology
- Summarize readings into a range spanning from min-max

The generated report summary was then passed to the LLM with instructions to anonymize information by rounding values as well as removing ages, dates, and names. This was done for three different types of prompting strategies, Zero-Shot, Chain-of-Thought, and Structured Output.

./images/Zero-shot-summary-vs-synth.png

Figure 3.5: Zero-Shot Generated Summary V.S. Synthesized Summary

Refer to figure 3.5 for a side-by-side comparison for Zero-Shot generation. Full results for each are present in the appendix. Overall, the LLM was effective in following instructions as well as working with a large amount of context.

3.3 Moving Forward

With the RAG corpus built, and the abilities of the LLM confirmed, the next steps are as follows:

- Evaluate LLM's ability to extract relevant information from retrieved chunks
- Create a pipeline that connects the retrieval, synthesis and inference stages
- Compare LLM's responses when presented with the original and synthesized information
- Test the system through prompt attacks (information-query attacks)

References

[1] Molly Bohannon. *Lawyer used Chatgpt in court-and cited fake cases. A judge is considering sanctions*. Feb. 2024. URL: https://www.forbes.com/sites/mollybohannon/2023/06/08/lawyer-used-chatgpt-in-court-and-cited-fake-cases-a-judge-is-considering-sanctions/ (cit. on p. 2).

[2] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401 (cit. on p. 2).

[3] Mingyu Jin et al. *Health-LLM: Personalized Retrieval-Augmented Disease Prediction System*. 2024. arXiv: 2402.00746 [cs.CL]. URL: https://arxiv.org/abs/2402.00746 (cit. on p. 3).

[4] NUHS. *Ai Healthcare in NUHS receives boost from supercomputer: NUHS+*. Aug. 2023. URL: https://nuhsplus.edu.sg/article/ai-healthcare-in-nuhs-receives-boost-from-supercomputer (cit. on p. 3).

[5] Osmond Chia. *Ai masters Singlish in key breakthrough to serve healthcare and patients' needs*. Nov. 2024. URL: https://www.straitstimes.com/singapore/ai-masters-singlish-in-key-breakthrough-to-serve-healthcare-and-patients-needs (cit. on p. 3).

[6] Liv McMahon. *Google Ai Search tells users to glue pizza and Eat Rocks*. May 2024. URL: https://www.bbc.com/news/articles/cd11gzejgz4o (cit. on p. 4).

[7] Zhen Tan et al. *"Glue pizza and eat rocks" – Exploiting Vulnerabilities in Retrieval-Augmented Generative Models*. 2024. arXiv: 2406.19417 [cs.CR]. URL: https://arxiv.org/abs/2406.19417 (cit. on p. 4).

[8] Jiaqi Xue et al. *BadRAG: Identifying Vulnerabilities in Retrieval Augmented Generation of Large Language Models*. 2024. arXiv: 2406.00083 [cs.CR]. URL: https://arxiv.org/abs/2406.00083 (cit. on p. 4).

[9] Xun Xian et al. *On the Vulnerability of Applying Retrieval-Augmented Generation within Knowledge-Intensive Application Domains*. 2024. arXiv: 2409.17275 [cs.CR]. URL: https://arxiv.org/abs/2409.17275 (cit. on p. 4).

[10] Steve Alder. *Healthcare Data Breach Statistics*. Mar. 2025. URL: https://www.hipaajournal.com/healthcare-data-breach-statistics/ (cit. on p. 4).

[11] Cody Clop and Yannick Teglia. *Backdoored Retrievers for Prompt Injection Attacks on Retrieval Augmented Generation of Large Language Models*. 2024. arXiv: 2410.14479 [cs.CR]. URL: https://arxiv.org/abs/2410.14479 (cit. on p. 5).

[12] Yuefeng Peng et al. *Data Extraction Attacks in Retrieval-Augmented Generation via Backdoors*. 2024. arXiv: 2411.01705 [cs.CR]. URL: https://arxiv.org/abs/2411.01705 (cit. on p. 5).

[13] Shenglai Zeng et al. *The Good and The Bad: Exploring Privacy Issues in Retrieval-Augmented Generation (RAG)*. 2024. arXiv: 2402.16893 [cs.CR]. URL: https://arxiv.org/abs/2402.16893 (cit. on pp. 5–7).

[14] Zhenting Qi et al. *Follow My Instruction and Spill the Beans: Scalable Data Extraction from Retrieval-Augmented Generation Systems*. 2024. arXiv: 2402.17840 [cs.CL]. URL: https://arxiv.org/abs/2402.17840 (cit. on p. 6).

[15] Xuying Li et al. *Targeting the Core: A Simple and Effective Method to Attack RAG-based Agents via Direct LLM Manipulation*. 2024. arXiv: 2412.04415 [cs.AI]. URL: https://arxiv.org/abs/2412.04415 (cit. on p. 6).

[16] Aryelly Rodriguez et al. “Current recommendations/practices for anonymising data from clinical trials in order to make it available for sharing: A scoping review”. In: *Clinical Trials* 19.4 (June 2022), pp. 452–463. DOI: 10.1177/17407745221087469 (cit. on p. 7).

[17] Corporation MITRE. *Synthea*. 2024. URL: https://synthetichealth.github.io/synthea/ (visited on 11/29/2024) (cit. on p. 8).

[18] Xueying Du et al. *Vul-RAG: Enhancing LLM-based Vulnerability Detection via Knowledge-level RAG*. 2024. arXiv: 2406.11147 [cs.SE]. URL: https://arxiv.org/abs/2406.11147.

[19] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997.

[20] John Chong Min Tan et al. *TaskGen: A Task-Based, Memory-Infused Agentic Framework using StrictJSON*. 2024. arXiv: 2407.15734 [cs.AI]. URL: https://arxiv.org/abs/2407.15734.

[21] Meenatchi Sundaram Muthu Selva Annamalai, Andrea Gadotti, and Luc Rocher. *A Linear Reconstruction Approach for Attribute Inference Attacks against Synthetic Data*. 2024. arXiv: 2301.10053 [cs.LG]. URL: https://arxiv.org/abs/2301.10053.

[22] Yi Dong et al. *Building Guardrails for Large Language Models*. 2024. arXiv: 2402.01822 [cs.CL]. URL: https://arxiv.org/abs/2402.01822.

.1 Appendix

Appendix A

Appendix

A.0.1 Report Generation Prompt

A.0.2 Synthetic Report Generation Prompt

A.0.3 Structured Output Prompt

A.0.4 Zero-Shot Result

A.0.5 Chain-of-Thought Result

A.0.6 Structured Output Result