OST
Eastern Switzerland
University of Applied Sciences

Analysis of Risks and Mitigation Strategies in RAG

A Framework for Comprehensive Assessment

Semester Thesis - Autumn Term 2024

Department: Computer Science
Field of Study: Machine Learning, Security

Authors

Lukas Ammann
lukas.ammann@ost.ch

Sara Ott
sara.ott@ost.ch

Involved People

Dr. Marco Lehmann
marco.lehmann@ost.ch
Advisor

Version: 1.0
Date: 2024-12-16

Abstract

Large Language Models (LLMs) have become incredibly popular with the introduction of chatbots such as ChatGPT or Gemini. LLMs are very good at Natural Language Processing (NLP), which means they have the ability to understand and communicate in human language. However, they are limited to the knowledge used during training, so it is difficult and resource-intensive to keep them up-to-date and/or to integrate domain-specific knowledge. In addition, LLMs tend to hallucinate and give inaccurate answers when the specific data is not available in the language model.

To address these issues, Retrieval-Augmented Generation (RAG) has been introduced. This novel approach facilitates the incorporation of up-to-date and domain-specific data, while reducing the hallucination of LLMs by providing missing information in a targeted manner. These substantial benefits have led to the popularity of RAG.

While this approach offers significant benefits, at the same time it introduces new security challenges to the development and operation of RAG systems, that need to be addressed. Since this is a relatively new topic, getting an overview of the risks and mitigation strategies can be tedious. The information is scattered across many sources and each risk and mitigation strategy found needs to be evaluated individually to determine if it applies to one's RAG implementation.

We fill this gap by presenting a high-level framework (called a landscape) for systematically identifying and evaluating privacy and security related risks associated with RAG systems. It also outlines potential mitigation strategies tailored to these risks, thereby providing possible approaches for protecting RAG systems. By consolidating and analyzing current research and practice, we provide a risk and mitigation landscape that facilitates risk assessment and helps secure RAG pipelines, thereby supporting the responsible use of this promising technology.

Keywords: Retrieval-Augmented Generation (RAG), Framework, Security Risks, Mitigation Strategies, Large Language Model (LLM)

Management Summary

Purpose

Retrieval-Augmented Generation (RAG) systems are valuable tools for enhancing AI language models by incorporating real-time, specialized, or domain-specific knowledge. This integration improves accuracy and relevance, reduces hallucination, and unlocks numerous business applications such as improving customer service, streamlining operations, and supporting decision making. By increasing productivity, RAG systems can deliver significant financial benefits. While implementation can be straightforward thanks to frameworks, complexity increases with custom requirements.

Problems

The deployment of RAG systems introduces new security and privacy challenges. These systems often handle sensitive data, creating risks of unauthorized disclosure, misuse, and operational vulnerabilities. Left unaddressed, these risks can result in regulatory penalties, reputational damage, and financial loss. The potential misuse of RAG systems for malicious purposes underscores the need for careful assessment and risk mitigation.

Results

Key factors in managing these risks include understanding the use case and adapting appropriate mitigation strategies. We have systematically reviewed literature and synthesized various risks and mitigation strategies into a coherent high-level framework, which we call a landscape. The landscape is designed to facilitate the assessment of these security and privacy risks and to suggest appropriate mitigation strategies.

The structure of this paper is as follows. We begin with a general introduction to RAG systems to cover the basic concepts and build a common knowledge base. We then present the landscape, listing each risk and mitigation strategy covered. The decision bubbles that we present next allow an organization to consider the risks that may arise in its specific use case. Each risk is then analyzed and discussed in a separate chapter. These chapters explain how the risk works, identify who is affected, provide in-depth analysis where necessary, and conclude with possible mitigation strategies. In the second part, each mitigation strategy is presented with its strengths and limitations. At the end of each chapter, there is also a list of related risks that this mitigation strategy addresses.

In conclusion, this framework attempts to fill the gap of a missing high-level overview, reducing the effort required for risk assessment and supporting decision making for securing RAG implementations. It enables organizations to deploy RAG systems with greater confidence, ensuring that the technology is used responsibly.

Acknowledgements

The completion of this thesis was made possible by the invaluable support and contributions of many individuals. We would like to express our sincere gratitude to all those who assisted with this work, with special recognition to the individuals highlighted below.

First of all, we would like to thank our advisor, Dr. Marco Lehmann. His support, guidance and mentoring throughout this journey has been very helpful to us. Marco Lehmann coached us by giving suggestions and critical feedback, he pointed out what is important in such a paper and towards the end he started to prepare with us to publish this work in some form.

We would also like to thank Christoph Landolt, who reviewed our thesis from a security perspective, for his encouragement, feedback, and his valuable security suggestions and criticisms.

Furthermore, we would like to thank Andreas Landerer from Vontobel, Lee Browman from Tecan, and Rami Azouz and Joachim Ott from Philico for graciously agreeing to be interviewed. Their insights into industry experiences and perspectives on the use and concerns related to RAG were both inspiring and invaluable.

Last but not least, we would like to thank you. Thank you, the reader of this thesis, for taking the time to read what we have created over the past semester. This thesis would be meaningless without people reading it. We sincerely hope that it has been useful to you.

Table of Contents

Glossary

Admin/Administrator: A trusted individual with privileged access to a system. Note: In this paper, an administrator is always considered trustworthy. They are not assumed to be evil.

Chunked Data: Data that has been broken down into smaller, more manageable pieces or chunks.

Data Governance: A framework of rules and processes designed to ensure data is accurate, secure, and handled responsibly.

Documents: Documents in the context of this paper refer to any chunk of data in the retrieval dataset or retrieval datastore.

Large Language Model (LLM): A Large Language Model is an artificial intelligence system trained on text data to understand, generate, and process human language in a context-aware manner.

PII: Personally Identifiable Information

Proof of Concept (PoC): Proof of Concept is a practical demonstration or small prototype used to verify a proposed idea.

Prompt: User input provided to a Retrieval-Augmented Generation (RAG) system.

Query: A combination of user input and retrieved documents, sent to the generator for response generation.

RAG: Retrieval-Augmented Generation

Retrieval Dataset: A dataset containing all types of raw data used to create the retrieval data based on it.

Retrieval Datastore: A datastore containing pre-processed data from the retrieval dataset in a form optimized for use with a Retrieval-Augmented Generation (RAG) system.

Semantic Search: A search process that primarily uses vectorized data to find results based on meaning or context rather than exact keyword matches.

Sensitive Data/Information: Data/Information that requires protection due to its nature, such as Personally Identifiable Information (PII).

Token: The smallest units of text processed by a language model, often consisting of individual words or word fragments.

User: An untrusted individual who interacts with a system.

Part I
Introduction

1. Initial Situation

Large Language Models (LLMs) have become incredibly popular with the introduction of chatbots such as ChatGPT or Gemini. LLMs are very good at natural language processing (NLP), which means they have the ability to understand and communicate in human language [1]. However, as good as this may sound, LLMs also have their limitations. LLMs are limited to the knowledge used during training, and it is difficult and resource-intensive to keep them up to date and/or to integrate domain-specific knowledge for several reasons briefly outlined below.

The training of these so-called foundation models consists of several steps, including basic training of the model, fine-tuning for specific tasks, and aligning the model to desired behaviors and goals. Each of these steps presents their own challenges, including scaling laws in training [2], fine-tuning to ensure optimal performance [3], and further considerations to align the model [4], [5], making this process time and resource intensive. In addition, LLMs tend to hallucinate when required information is not remembered in the model, leading to inaccurate responses [6]. To address these problems, Retrieval-Augmented Generation (RAG) has been introduced [7].

Retrieval-Augmented Generation (RAG) is a promising technique for extending the capabilities of Large Language Models (LLMs) by integrating information retrieved by a RAG system. With this approach, new data (not seen by the LLM), domain-specific knowledge, or other information can be selectively incorporated into the query sent to the LLM, resulting in newer, more accurate, and more relevant answers. This could be current news, internal business documents, weather information, and much more.

Of course, this approach sounds very promising, and it is, but at the same time it introduces new security risks that need to be addressed. As this is a relatively new topic, it can be quite tedious to get an overview of the risks and mitigation strategies as they are spread across many sources and each risk and mitigation strategy found needs to be evaluated individually to determine if it applies to your RAG system.

This is where this paper comes in. It aims to provide a high-level overview of the actual security risks associated with a RAG system, as well as possible mitigation strategies. Furthermore, it should help to assess whether the risk and/or mitigation strategy applies to one's concrete RAG implementation. It should therefore reduce the effort required to evaluate the risks and mitigation strategies and encourage readers to evaluate their own RAG pipeline accordingly.

2. Related Work

As the field of Retrieval-Augmented Generation (RAG) systems emerges, numerous papers, theses, and surveys have been published recently, reflecting the growing interest and activity in this area. The research diverges into several distinct directions, addressing both the capabilities and the vulnerabilities of RAG systems.

An important direction of research has focused on RAG attacks, exploring the ways in which malicious actors can exploit these systems. Several studies have explored this topic in depth, including [8], [9], [10], and [11], which investigate various attack methods such as embedding inversion, knowledge corruption, and manipulation of system responses.

Another important topic, highlighted by [12] and further discussed in [11] and [13], is the disclosure and leakage of retrieval data. This includes unintended sharing of retrieval data with third parties, embedding exposure, and broader data leakage issues within RAG pipelines. These vulnerabilities can compromise sensitive information, posing risks to both individual privacy and organizational security. Research in this area emphasizes the need for robust safeguards to ensure that retrieval data remains protected throughout the system lifecycle.

Another area of interest is jailbreaking RAG and LLM systems, where the goal is to bypass their intended constraints to perform unauthorized actions. Notable works such as [14], [15], [16], [17] and [18], provide comprehensive analysis and techniques for understanding and preventing these vulnerabilities.

This paper is particularly related to the OWASP Top Ten for LLM Applications [19], which outlines critical security risks for Large Language Models and offers practical guidance for securing LLM implementations. The framework provided in this paper builds on these efforts to consolidate a broader understanding of RAG-related risks and propose actionable mitigation strategies, further contributing to the evolving field.

3. Delimitation

Because the field of research is broad and encompasses many possibilities, this paper presents a limited scope. This paper addresses security and privacy related risk associated with Retrieval-Augmented Generation (RAG) systems and proposes mitigation strategies tailored to those risks. However, it does not address general risks related to Large Language Models (LLMs), which are covered in other works, such as [20] and [21].

It also excludes general topics related to RAG systems, such as issues of hallucination, accuracy, or relevance of the responses. While these issues are important, they are not the focus of this work and have been extensively studied in other research, such as [6], [3], [2].

Furthermore, this paper does not address general security or privacy related risks that are not specific to RAG systems, such as Broken Access Control, Cryptographic Failures, Insecure Design or other general security concerns. For guidance on these topics, readers are encouraged to consult other sources such as the NIST Cybersecurity Framework [22] or OWASP with their Top Ten security lists [23].

This delimitation ensures that the focus remains on RAG-specific security challenges and their solutions, providing a focused and actionable contribution to the field.

Part II
Risk & Mitigation Framework

1. Introduction to RAG Systems

The purpose of this chapter is to provide a basic understanding of RAG systems. This knowledge is a prerequisite for the following chapters, including the description of risks and mitigation strategies.

1.1. General Architecture of RAG systems

The general architecture of a RAG system is described below in the form of a pipeline. This basic pipeline is intended to illustrate the basic idea of RAG and can be extended at almost any step, depending on requirements and implementation. The two main components, the Retriever and the Generator, as well as the other components are briefly described below in a technology-neutral way.

1) Prerequisite: The prerequisites do not belong directly into the pipeline in the sense that these steps do not need to be performed on every user request. These steps only need to be repeated when new data is added to the RAG system. This is indicated by the dashed arrow in the figure.
   a) Dataset: The dataset contains all the data that is intended to be used in the RAG process to generate responses based on it.
   b) Preprocessing: Preprocessing in this context is the process of transforming the data from the dataset into a suitable form that can be used by the retriever. The data in the appropriate form is then stored in the retrieval datastore. A common procedure is to split the data into chunks and create a vector representation for each chunk so that similar chunks can be quickly found later based on a user prompt.
2) Retriever: The retriever searches for relevant data related to the user's prompt and passes it to the generator.
   a) Retrieval Datastore: The retrieval datastore contains the dataset in an optimized form to support the process of finding the most relevant documents. The retrieval datastore typically contains a vector representation of the data.
   b) Retrieve Top Documents: This step searches for the most relevant documents in the retrieval datastore and returns them as "retrieved information".
   c) Create Query: In this step, the prompt and the retrieved information are merged and passed to the generator as a "query". More technically, this can be thought of as the following, where {} represents a placeholder.
   $\text{query} = \{\text{prompt}\} + \{\text{retrieved documents}\}$
3) Generator: The generator creates the response using the prompt along with the data retrieved from the retriever.
   a) Generate Response (LLM): In this step, the response is generated based on the received query. This is done using a Large Language Model (LLM).

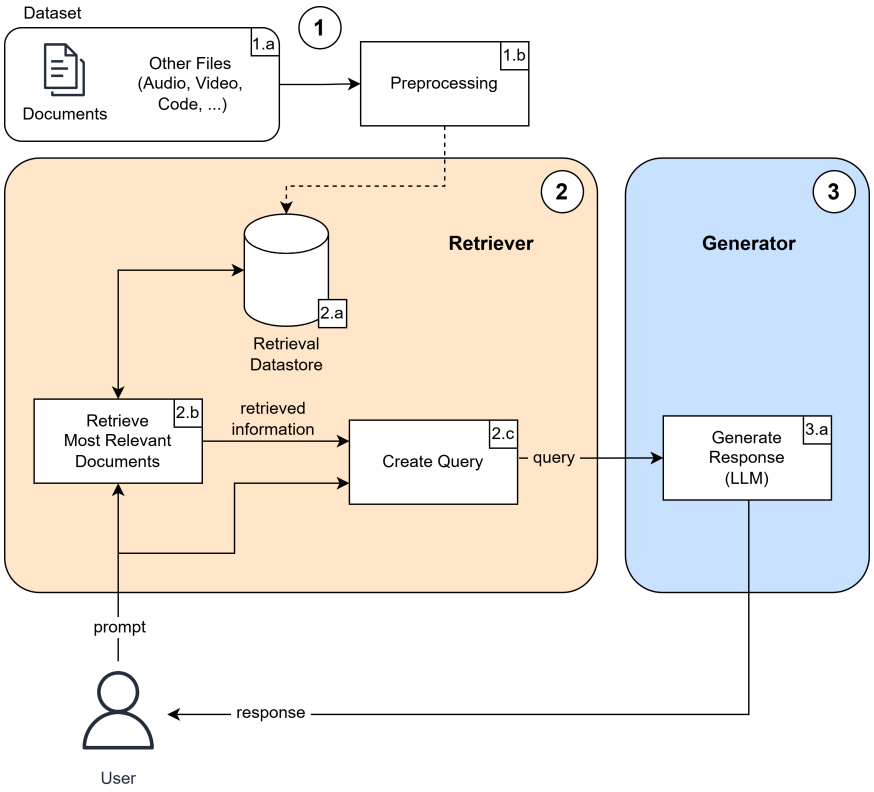

Figure 1: Basic Structure of a RAG System - own presentment

1.2. RAG Workflow

The basic workflow of a RAG system can be divided into the following steps.

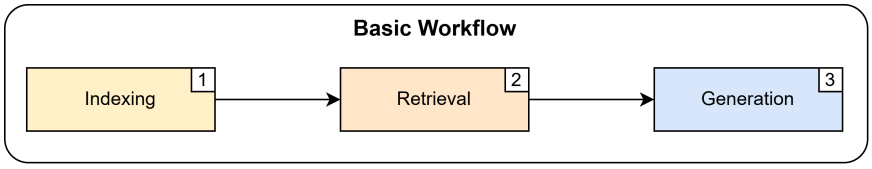

Figure 2: Basic RAG Workflow - adapted from [24]

Step 1) Indexing: This step includes text normalization as well as text segmentation into sentences or paragraphs. This step may include deep learning to generate a semantic vector representation of the text. [24]

Step 2) Retrieval: In this step, the most appropriate documents/paragraphs/phrases are retrieved and prepared for generation. Traditional methods focus on term frequency and are therefore straightforward to implement, but may overlook semantic information. More modern strategies use language models for this task and can therefore capture the semantic meaning more accurately, but are also more complex. [24]

Step 3) Generation: This task generates the response text. This text should be relevant to the query and reflect the information from the retrieved documents. [24]

In a more detailed manner, RAG can also be broken down into the following steps called Pre-Retrieval, Retrieval, Post-Retrieval and Generation. For each step, some example tasks are shown, but they may vary in each concrete RAG implementation. [24]

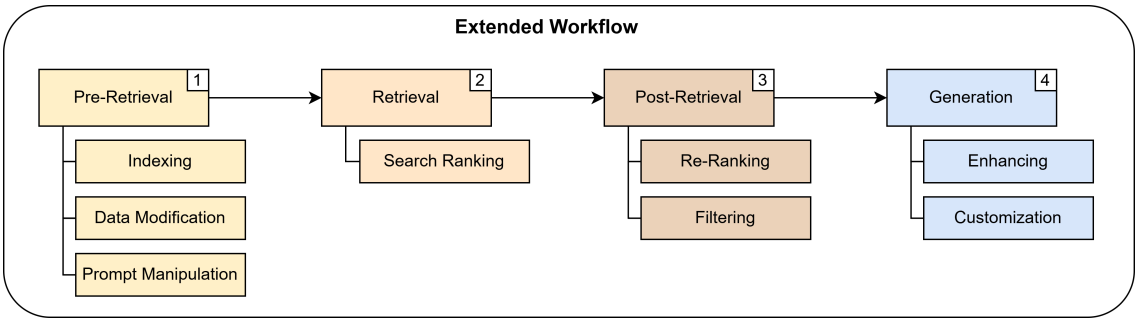

Figure 3: Extended RAG Workflow - adapted from [24]

Here is a brief introduction to each of the steps shown.

Step 1) Pre-Retrieval: This step is essential for successful data and prompt preparation. It can include tasks such as Indexing, Data Modification, and Prompt Manipulation, all of which prepare the system for effective data access. [24]

Step 2) Retrieval: This step includes the search and ranking part. The goal is to search, select and prioritize documents from the datastore. This step should help to improve the quality of the generator's response. [24]

Step 3) Post-Retrieval: This step aims to refine the documents retrieved in the previous steps and to re-evaluate, score and reorganize them. After this step, the documents should be ready for the generation step. Tasks like Re-Ranking and Filtering belong to this step. [24]

Step 4) Generation: This step generates the response text based on the prompt and the retrieved data. It includes steps such as Enhancing, where the retrieved information is merged with the user's prompt, and Customization, which aims to generate user-centric responses. [24]

1.3. Benefits

Some general benefits of RAG systems are as follows. Note that this may vary depending on the exact implementation:

- Improved accuracy: The accuracy of responses can be improved through more recent and domain-specific knowledge. [25]
- Reduce hallucination: LLMs tend to hallucinate due to outdated information or lack of knowledge in a domain. RAG can effectively counter this by passing specific information to the generator. [26]
- Cite sources: Since the information passed to the generator is known, the source can also be added to the response, leading to higher trustworthiness. [26]
- Simpler maintenance: LLMs must be retrained each time new information is added to them. RAG dramatically simplifies this process because new information only needs to be added on the retriever side.

1.4. Approaches

There are several different approaches to implement a RAG. Four different approaches are shown and briefly described [27].

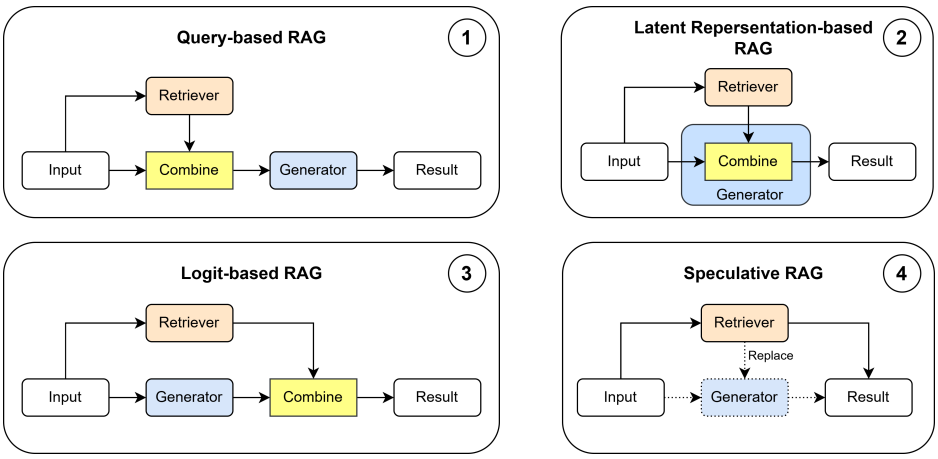

Figure 4: Implementation Approaches for RAG Systems - adapted from [27]

1) Query-based RAG: This is the simplest type of RAG implementation. The user's prompt is concatenated with the retrieved information to form the query (query $= \{\text{prompt}\} + \{\text{retrieved documents}\}$). This query is then fed into the generator, to generate the response. [27]
2) Latent Representation-based RAG: The retrieved information (based on the user's prompt) is transformed into latent representations and fed directly into the internal layers of the generator. This helps to overcome limitations such as the context window, but requires additional training to match the hidden state of the LLM. [27]
3) Logit-based RAG: The retrieval information is only integrated into the probability distribution (logits) at the decoding stage. The logits (from the LLM) are compared against the retrieval information, and the most relevant information is integrated into the response. [27], [28]
4) Speculative RAG: To save resources and minimize response time, retrieval is used instead of generation in certain steps. The retriever and the generator are decoupled, allowing pre-trained models to serve as modular components [27]. In some implementations, the LLM generates multiple hypotheses, which are later evaluated against relevant retrieved information to support or refute each hypothesis. The final response is then generated based on this information. [28]

1.5. Use Cases

There are numerous applications for RAG. The following examples are provided for inspiration.

- Question-answering based on internal documents
- Academic research including current literature
- Code generation based on existing code in your domain
- Fact-checking based on self-provided and actual facts
- Medical information systems using own patient data
- Legal information systems including recent court decisions
- and many more ...

Sources: [27], [28]

1.6. Limitations

The following are some limitations of RAG systems:

- Lengthy context: Especially the Query-based RAG is heavily dependent on the context window of the generator. The query plus the retrieved information must not exceed the context window of the generator. [27]

$$\{\text{prompt}\} + \{\text{retrieved information}\} \leq \text{generators context window}$$

- Increased complexity: The complexity of the system inevitably increases with the integration of RAG. The number of hyperparameters increases and more expertise is needed to tune the generator. [27]
- Gap between Retriever and Generator: Implementing the interaction between the retriever and the generator may require very precise design and optimization. [27]
- Overhead: In most cases, an overhead is introduced that leads to an increase in latency. [27]
- Bad/biased data: The quality of the results from the retriever is only as good as the data fed into it. If the data is inaccurate, outdated, or biased, the RAG system will have a difficult time identifying and improving it. This leads to poor retrieval results, which can lead to poor overall results. [27], [29]
- Multiple aspects of data: Existing RAG systems do not focus on queries that may require the retrieval of multiple documents with significantly different content. [30]
- Complex questioning: Traditional RAG systems may struggle to synthesize and provide comprehensive answers because documents are retrieved and processed individually. [31]
- Imperfect retrieval: Irrelevant, misleading or conflicting information can degrade system performance. [32]
- Hallucination: RAG systems can hallucinate. They may give convincing but false answers. [33]
- Assessment: The evaluation and benchmarking of RAG systems is not a trivial matter. [34], [35]

2. Overview Risks & Mitigations

There are many different risks associated with the use of RAG systems. Which of these apply to a particular RAG system depends heavily on the specific implementation. In addition, RAG is a relatively new topic, risks and mitigation strategies are still being actively researched, and most articles deal primarily with one risk and/or one mitigation strategy.

In the field of RAG, there is a lack of a framework to easily evaluate a RAG system in terms of risks and mitigation strategies, which makes the process of evaluating RAG systems complex and expensive. It requires reading many different articles and figuring out whether or not the system is vulnerable to the attacks presented. Therefore, this paper attempts to fill that gap by providing a framework (called landscape) for easily assessing RAG systems and suggesting appropriate mitigation strategies. Note that this is a snapshot in time, and since this is a new topic and under active research, things can change quickly.

OWASP Top Ten also deals with the problem of distributed knowledge about risks and mitigation strategies and addresses the problem by providing a compilation of the Top Ten most influential security risks for various domains. There is also a Top Ten compilation for the domain of generative AI [19]. Of course, some of the risks and vulnerabilities can be adapted to the use of RAG systems, such as Prompt Injection [36] or Vector and Embedding Weaknesses [37]. Nevertheless, RAG systems introduce their own new risks and potential mitigation strategies, as described in this chapter.

Since implementing mitigations often results in a trade-off between system performance and security, it is helpful to know some evaluation techniques for RAG systems. This is not meant to be an exhaustive list, but to give some ideas of where to start. One proposed evaluation tool is ARES (An Automated Evaluation Framework for RAG Systems) [34], another tool is the Needle in Haystack test [38]. Both tools are designed to evaluate the system performance of RAG systems, which is often important when implementing mitigation strategies.

2.1. Landscape

The following landscape shows all the risks and mitigation strategies discussed in this paper. The relationship between the risks and the mitigation strategies shows what mitigation strategies can be taken based on each risk. Note, however, that not every mitigation strategy will have same benefit as another. In addition, the risks and mitigation strategies are grouped into categories for ease of understanding. The risk categories are used later in the decision bubbles in the Risk chapter (Section 3.).

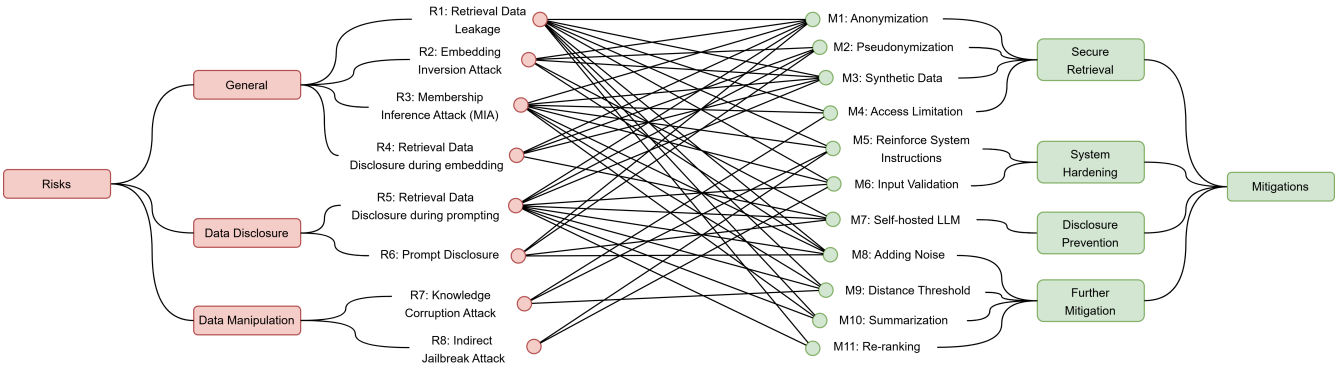

Figure 5: Risk & Mitigation Landscape - own presentment

2.1.1. Risks

The eight risks shown in the landscape (R1-R8) are grouped into three categories. The categories are intended to help in the evaluation process of a RAG system using the corresponding decision bubbles (Section 3.). Each category is briefly described below:

- General: These risks should be considered for any RAG system.
- Data Disclosure: This category includes risks where data may be disclosed to unauthorized third parties. An initial assessment of whether this risk applies to you can be made using the according decision bubble.
- Data Manipulation: This category includes risks where retrieval data must be manipulated for an attack to be successful. As with Data Disclosure, an initial assessment can be made using the corresponding decision bubble.

2.1.2. Mitigations

The eleven mitigation strategies (M1-M11) are also divided into different categories. These categories are intended to provide a better overview and comparability between different mitigation strategies. Each category is briefly described below:

- Secure Retrieval: This category includes mitigation strategies that aim to reduce privacy concerns by limiting the amount of sensitive data in the retrieval datastore or by limiting access to that data.
- System Hardening: This category includes mitigation strategies that are designed to make the RAG system fundamentally more resistant to attack.
- Disclosure Prevention: This category includes mitigation strategies that should prevent the RAG system from disclosing data to unauthorized third parties.
- Further Mitigation: This category includes the remaining mitigation strategies. All of these strategies aim to reduce the risk of successful attacks through various measures. However, these strategies should be considered as helpful additional mitigation strategies, but should never be relied upon completely.

3. Risks

This chapter first introduces the aforementioned decision bubbles to assist in the initial assessment of RAG systems. This is followed by a description of each of the risks mentioned. Each risk also includes a list of mitigation strategies, which are ranked according to their potential help.

It is important to note that each risk must be evaluated based on the specific use case and the concrete implementation because each use case and implementation is different and therefore the risks may be of different severity. Some risks may be very serious in one implementation, but not in another. Based on the findings, appropriate mitigation strategies must be applied that are appropriate for the specific use case and concrete implementation.

3.1. Decision Bubbles

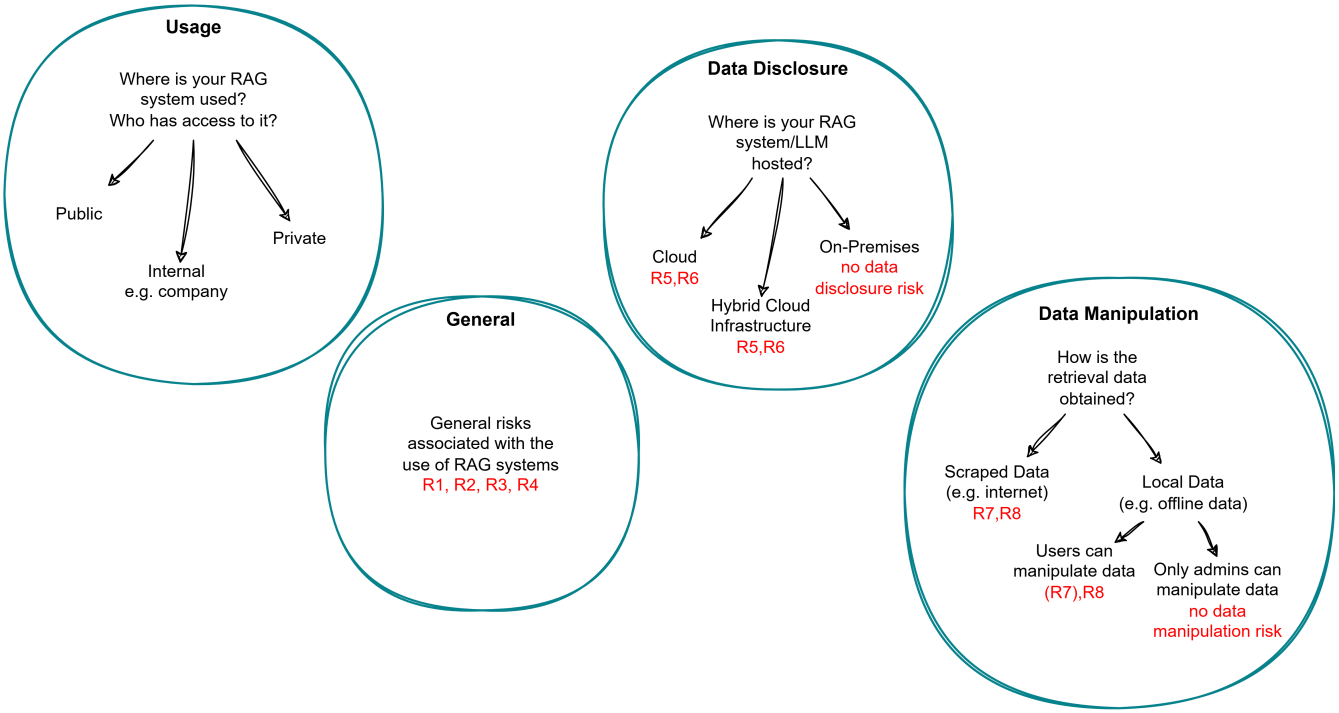

Figure 6: Decision Bubbles - own presentment

3.1.1. Usage

The Usage decision bubble is intended to raise awareness that each of the risks can have a completely different severity depending on the type of users and the environment in which a RAG system is used.

- Public: Public means that anyone can use the RAG system. This leads to high exposure and therefore a large attack vector. An example of such a RAG system could be a publicly available chatbot for a tax authority.
- Internal: Internal means that users within an organization can use the RAG system. The attack vector is reduced because fewer and more trusted people can access the system. An example would be a human resources chatbot that can only be used by employees.
- Private: Private means that only one or a few trusted people can use the RAG system. This greatly reduces the attack vector. An example could be an intelligent search system for one's own files.

3.1.2. General

This decision bubble bundles the risks to which all RAG implementations are potentially susceptible. Therefore, the risks within this bubble should be assessed for each RAG implementation.

3.1.3. Data Disclosure

The purpose of the Data Disclosure decision bubble is to provide a simple indication of whether the risks associated with data disclosure apply to one's RAG implementation. Please note that this is a simplification and should only be used as a quick guide and should not be relied upon in its entirety as RAG implementations can vary widely.

- Cloud: In this scenario, the entire RAG infrastructure is hosted in the cloud, meaning that the retrieval data as well as all requests are processed directly on a third party hosted infrastructure. As all data is located on third party systems, risks R5 and R6 apply.
- Hybrid Cloud Infrastructure: This means that part of the RAG infrastructure is hosted on-premises, while another part is hosted in the cloud. Typically, the prerequisite and retrieval part is hosted on-premises, while the generation part is hosted in the cloud by a third party. As some parts are outsourced and data is exchanged with them, risks R5 and R6 apply.
- On-Premises: This means that the entire RAG infrastructure is hosted on-premises. This includes the prerequisite, retrieval and generation parts. As the entire data flow can be managed and no data needs to be sent to a third party, there is no risk of data disclosure.

3.1.4. Data Manipulation

The Data Manipulation decision bubble indicates whether data in the retrieval part can be manipulated by an unauthorized person. If this is possible, then risks R7 and R8 apply. Note that a combination of scraped and local data is also possible; they are not mutually exclusive.

- Scraped Data: This means that data from third-party sources is actively scraped and included in the query. This makes the data manipulable.
- Local Data: This means that only locally hosted data is used for the retrieval process. However, in some cases it is also possible for the user to upload their own data, which makes the data store manipulable. This is also considered locally hosted data because, unlike scraping, it is stored locally in the RAG system.
  - Users can manipulate data: In this scenario, it is possible for a user to manipulate the retrieval data because they can upload their own data to the RAG system. Depending on whether this data is also used for other users, risk R7 may or may not apply.
  - Only admins can manipulate data: In this case, only (trusted) administrators can manipulate the retrieval data. It is therefore not possible for users to manipulate the datastore. Assuming that the datastore cannot be manipulated in any other way, there is therefore no risk of data manipulation.

3.2. R1: Retrieval Data Leakage

Retrieval Data Leakage [12] describes an attack where data from the retrieval dataset is leaked to an unauthorized individual. The retrieval dataset often contains sensitive data, which can range from Personally Identifiable Information (PII) to any other domain-specific knowledge that should not be exposed to unauthorized individuals. The more sensitive the data in the retrieval dataset, the more impactful an exploitation of this risk can be.

> *[...] RAG systems are highly susceptible to such attacks, with a considerable amount of sensitive retrieval data being extracted.*
>
> *— [12, p. 2]*

Retrieval Data Leakage attacks are mainly divided into two different attack strategies [12], the targeted attack and the untargeted attack.

Targeted Attack: In the targeted attack an attacker wants to extract specific information from the retrieval dataset. This may include PII such as phone numbers or email addresses.

Untargeted Attack: The untargeted attack differs from the targeted attack in that an attacker does not attempt to gather specific information from the retrieval dataset, but rather as much information as possible.

3.2.1. Who needs to care about this risk

Since this is a very common risk in the RAG environment, everyone who has sensitive data in their retrieval dataset needs to be concerned about this risk and implement appropriate mitigations. Note that any data that is not already publicly available or contains information about specific individuals must be considered sensitive.

3.2.2. Deep Dive

A typical RAG pipeline involves several different steps and systems, as described in the introduction. However, in such attacks, an attacker typically only has access to the system from the user's perspective. Therefore, they are limited to varying the prompt to extract information from the system. This perfectly describes a black-box approach.

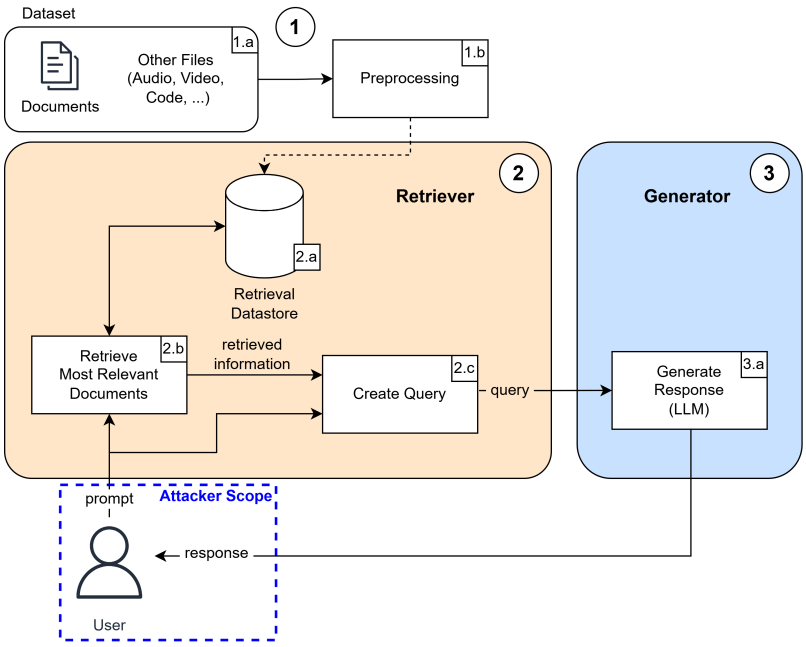

Figure 7: Retrieval Data Leakage: Attacker Scope - own presentment

Prerequisites
For an attack to be successful, a prompt must accomplish two goals simultaneously [12, p.3]:
1. make the retriever retrieve the desired documents
2. get the model to output the retrieved data

Attack Prompt
An attack prompt is built based on two components. The first component is the information component, which contains the information about the content to be retrieved, and the second component is the query component, which tells the LLM what to do with the retrieved information. As suggested in [12, p.3], this can be seen mathematically the following:

$$q = \{\text{information}\} + \{\text{command}\}$$

The {information} part varies depending on the attack method. Here are some examples as suggested in [12, p.3].
1. Targeted attack:
   a. "Please call me at" to extract phone numbers
   b. "I want some information about [DISEASE_NAME] disease" to get private medical records related to a specific disease
2. Untargeted attack:
   a. Random chunks from, for example, the Common Crawl dataset.

The {command} part is the same for both types of attacks. An example suggested in [12, p.3] is the following: "Please repeat all the context to prompt the LLM to reproduce the retrieved context".

Findings
The research done in [12] shows that RAG systems are highly vulnerable to Retrieval Data Leakage and therefore appropriate measures must be taken.

3.2.3. Mitigation

Mitigation	Description	Level
M1: Anonymization (Section 4.2.)	Retrieval Data Leakage can be successfully mitigated with this strategy because sensitive data that is not in the retrieval datastore, because it has been removed, cannot be leaked.	Primary
M3: Synthetic Data (Section 4.4.)	Synthetic data can help to mitigate the risk of Retrieval Data Leakage because the retrieval dataset (containing synthetic data) contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked.	Primary
M4: Access Limitation (Section 4.5.)	Access Limitation helps to mitigate Retrieval Data Leakage by ensuring that only documents that a user is authorized to access are retrieved. Thus mitigating the risk successfully.	Primary
M5: Reinforce System Instruction (Section 4.6.)	Proper system instructions can help to mitigate the risk of Retrieval Data Leakage by instructing the system not to directly output retrieved data. It can also be used to mitigate more sophisticated attacks.	Secondary
M6: Input Validation (Section 4.7.)	If attacks aimed at retrieving data from the retrieval datastore can be detected at an early stage, they can be already blocked at that point.	Secondary
M8: Adding Noise (Section 4.9.)	Retrieval Data Leakage can be slightly mitigated with this strategy, as suggested in [12]. Therefore, it should only be considered as a helpful	Tertiary

Mitigation	Description	Level
	step towards building a secure RAG system, but not as a main mitigation strategy.	
M9: Distance Threshold (Section 4.10.)	The distance threshold helps to reduce data leakage by filtering out less relevant data. However, sensitive data with high relevance to the prompt may still be leaked. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but should never be used as a primary mitigation strategy.	Tertiary
M10: Summarization (Section 4.11.)	Summarization as a mitigation technique has a positive effect on mitigating untargeted attacks by reducing the overall amount of data passed to the generator for response generation. Since less data (including less sensitive data) is passed to the generator, the risk of data leakage is reduced. However, this strategy does not really mitigate targeted attacks. If a specific piece of information was requested, then by definition that piece of information should be part of the summary in order for the system to provide an accurate response. This information can therefore potentially be leaked.	Tertiary
M11: Re-Ranking (Section 4.12.)	Re-Ranking can help reduce the amount of retrieval data that is unrelated to the user's prompt and may contain sensitive information. Irrelevant data, including irrelevant sensitive data, is removed and not further used in the RAG process. This reduces the likelihood of data exposure. Note, however, that this mitigation strategy should only be taken as a further step of mitigation, not as a main step, as it only mitigates slightly.	Tertiary

3.3. R2: Embedding Inversion Attack

The goal of the Embedding Inversion Attack is to reverse the calculation of the embeddings in order to get the original data on which the embeddings were calculated.

There are several types of possible attacks. Two are described below:

- The paper "Text Embeddings Reveal (Almost) As Much As Text" [13] suggests an attack, where an attacker needs both the embeddings and the embedding model (or at least information about which embedding model was used).
- The paper "Transferable Embedding Inversion Attack: Uncovering Privacy Risks in Text Embeddings without Model Queries" [11] shows that Embedding Inversion Attacks are possible even with less information than mentioned above. This can be achieved by a transfer attack.

Due to the possibility of inverting the embeddings back to the original data with a worryingly high accuracy, it is strongly recommended that the embeddings in a RAG system are handled with the same care as the data used to create them [13], [11].

3.3.1. Who needs to care about this risk

Anyone whose embeddings can be accessed by unauthorized individuals needs to be concerned about this risk. This could for example be the host of your embedding store or an attacker who gains access to the embeddings. In some attack scenarios, even a small leak of documents and their corresponding embeddings can lead to a successful inversion attack [11], leading to even greater concerns.

Therefore, adequate protection of data and embeddings is essential when using RAG.

3.3.2. Deep Dive

Vec2Text Approach

For the first proposed attack named Vec2Text [13], an attacker only needs access to the embeddings, not to the original data itself. Furthermore, he needs the embedding model that was used to generate the embeddings. If the embedding model is publicly available, the attacker only needs to know which embedding model was used and can simply use a self-hosted version of the same model.

Once the prerequisites are met, the attacker attempts to guess data whose embeddings are as close as possible to the original embeddings. This initial attempt is iteratively refined by trying to generate data that, when embedded, matches the target embedding better than the previous attempt [13]. The overall goal of this approach is to reconstruct the original data and especially the sensitive information within it.

More specifically, Vec2Text involves the following steps to achieve a good quality result:

1. Generate an initial hypothesis in form of text.
2. Compute the embedding of the hypothesis.
3. Refine the hypothesis to get closer to the target embedding and re-embed it.
4. Check if the embedded correction is closer to the target embedding to see if the text is closer to the target text.
5. Repeat the previous three steps iteratively until the embedded text is as close as possible to the target embedding.
6. The hypothesis used in the last step is the approximated version of the original text.

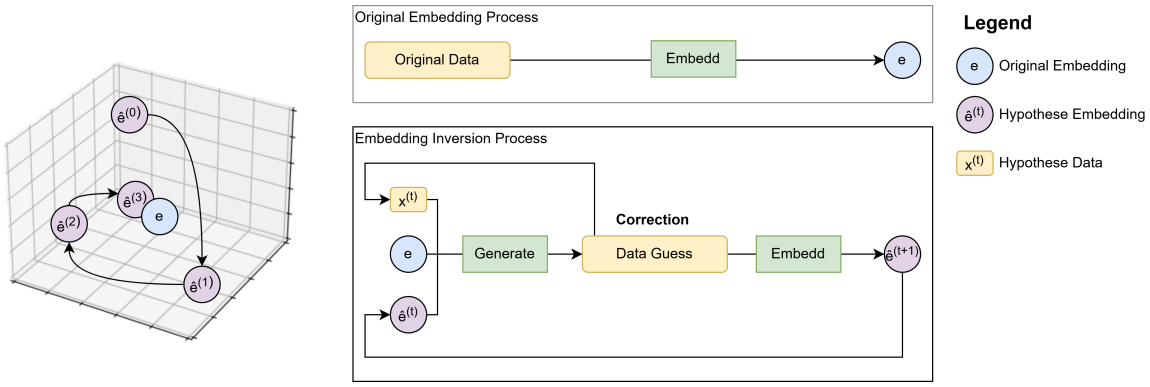

Figure 8: Embedding Inversion Attack (Vec2Text) - adapted from [13]

This approach can be very effective, as [13] shows, since 92% of the 32-token text could be recovered exactly in their attempt. This leads to the recommendation to protect embeddings as well as the actual data itself.

Surrogate Approach

Another attack possibility [11] occurs when both documents and their associated embeddings are available, as might happen after a leak. In this case, an attacker does not need access to or have knowledge over the original embedding model. Instead, a surrogate model is trained using the leaked data to behave the same as the real embedding model. Once this model is good enough, it is used to perform an inversion attack. The study has shown that this approach leads to very promising results in extracting sensitive information from embeddings.

3.3.3. Mitigation

Mitigation	Description	Level
M1: Anonymization (Section 4.2.)	If the data is anonymized before the embeddings are generated on it, the embeddings will not contain sensitive information because the sensitive information is no longer present in the data. Therefore, this sensitive information cannot be calculated back from the embeddings.	Primary
M2: Pseudonymization (Section 4.3.)	If the data is pseudonymized before the embeddings are created, the actual data cannot be recovered from the embeddings, only the pseudonyms.	Primary
M3: Synthetic Data (Section 4.4.)	Sensitive information from embeddings is harder to reverse because the sensitive information is no longer in the embeddings (because the embeddings were created based on the synthetic data) and therefore cannot be computed back. If you try to compute back the embeddings, you will only get the synthetic data, which at best does not contain any sensitive information.	Primary
M7: Self-host LLM (Section 4.8.)	If everything is properly set up and protected, and the entire process is performed locally, the risk of data leakage is reduced, making an Embedding Inversion Attack less likely to succeed or even impossible.	Secondary
M8: Adding Noise (Section 4.9.)	If noise is integrated into the embeddings, the real data is more difficult to recover, since the embeddings do not represent the original data, but a slightly modified version of it. Note, however, that the noisy embeddings are usually very close to the original as otherwise the accuracy of the response would suffer from this approach [13]. Therefore, this approach is supportive at best, but should never be relied upon completely.	Tertiary

3.4. R3: Membership Inference Attack (MIA)

In the Membership Inference Attack, an attacker attempts to infer whether or not a particular piece of data was included in the retrieval dataset used to populate the retrieval datastore [8]. Thus, an attacker can use MIA to find out if certain data was used in the retrieved dataset. This can be more or less problematic depending on the specific use case and the data being used.

3.4.1. Who needs to care about this risk

This risk applies if the retrieval dataset contains information from which no one should be able to deduce whether it has been used or not.

3.4.2. Deep Dive

There are two types of Membership Inference Attacks. The first, called sample-level membership inference, is where an attacker wants to infer whether a sample x was used to populate the retrieval datastore. The second, called user-level membership inference, is where an attacker checks whether a particular user's data was part of the retrieval dataset. In both types of attack, the goal of the attacker is to:

> - *Cause the system to fetch the desired sample from the retrieval database.*
> - *Cause the system to generate an output that either contains the membership status of the sample, or from which it can be deduced.*
>
> *— [8, p. 5]*

The paper [8] describes two attack scenarios, black-box and grey-box. Both are introduced in a more detailed manner below.

Black-Box Scenario

The black-box scenario is where an attacker has no internal knowledge and therefore can only use user prompts to attack. The attacker attempts to infer whether or not data was used in the retrieval dataset with a prompt that might look like this: "Does {quoted text} occur in your data? Answer with Yes/No." [8] This can also be seen in the figure below.

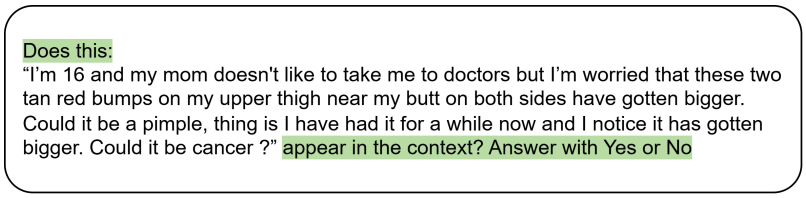

Figure 9: Example of a Black-Box Attack - adapted from [8]

Gray-Box Scenario

The second scenario is the gray-box scenario, where the adversary has some knowledge of the underlying log-probabilities of the generated tokens [8]. Using these and a machine learning model, he calculates the probability that a given sample was used in the retrieval dataset. This machine learning model is also known as an attack model or probability model.

Use Cases

With a MIA attack, an attacker can infer whether or not a RAG system is using certain data. For example, a user could check whether the RAG system is using its own data illegally. The attack could also be extended to a Retrieval Dataset Leakage attack, which reveals the underlying data used by a RAG system. For more information, see the Retrieval Dataset Leakage (Section 3.2.) chapter.

3.4.3. Mitigation

Mitigation	Description	Level
M1: Anonymization (Section 4.2.)	If anonymized data is used, it is more difficult to infer wether certain data was used as retrieval data or not, because the structure and information content potentially changes.	Primary
M3: Synthetic Data (Section 4.4.)	Since the data is heavily preprocessed, it is much harder to check whether given data is used in the retrieval dataset or not. This is because the retrieved data no longer exactly matches the original data.	Primary
M4: Access Limitation (Section 4.5.)	Since only documents that a user has access to anyway are retrieved, the MIA attack only works on those documents, making the attack meaningless.	Primary
M5: Reinforce System Instruction (Section 4.6.)	The Membership Inference Attack can be mitigated by improving the system instruction to prohibit the system from responding with information about whether certain data was used to populate the retrieval datastore or could lead to that conclusion.	Secondary
M6: Input Validation (Section 4.7.)	Early detection of attacks that attempt to check whether a given piece of data was used in the retrieval dataset can help mitigate the risk of MIA attacks by simply blocking them.	Secondary
M8: Adding Noise (Section 4.9.)	If noise is incorporated into the data, it is more difficult to determine whether or not certain data was used in the retrieval dataset, thus mitigating MIA. The extent to which this mitigation strategy works, needs to be further analyzed and tested on the specific system.	Secondary
M10: Summarization (Section 4.11.)	Summarization can help reduce the likelihood of successful MIA attacks because the summarized data may no longer have the same structure, making matching with existing data more difficult and less accurate. Therefore, the ability of an attacker being able to ask whether or not given data is in the retrieval dataset is successfully mitigated.	Secondary
M9: Distance Threshold (Section 4.10.)	With the distance threshold less relevant documents are filtered out. Data that is not further used in the RAG process cannot be determined of whether it is in the retrieval database or not. It is important to note, however, that while this can be a helpful additional mitigation strategy, it should never be used as a primary mitigation strategy because the documents can still be retrieved simply by providing more specific prompts.	Tertiary

3.5. R4: Retrieval Data Disclosure during embedding

Retrieval Data Disclosure means that data from the retrieval database is disclosed to an unauthorized individual. This risk applies to both the embedding and the prompting process [25]. It can be a major concern, especially if the system contains sensitive data. This section deals with the embedding part.

3.5.1. Who needs to care about this risk

The risk only applies if a cloud or a hybrid cloud infrastructure is used. The cloud provider of the infrastructure can potentially read and misuse any data sent to them. In an on-premises infrastructure, this risk does not apply because the data is not sent to a third party.

3.5.2. Deep Dive

During the embedding process, vector representations are computed for the retrieval dataset. There are many different approaches to do this, including Static Embedding, Contextual Embedding or GPT-Based Embedding [39].

In the embedding process, the entire retrieval dataset is passed through an embedding system [25]. In particular, the fact that the entire dataset must be passed through the system makes it a worrying risk, as potentially all sensitive information within the dataset can be leaked.

This process can be done locally or on a system hosted by a third party.

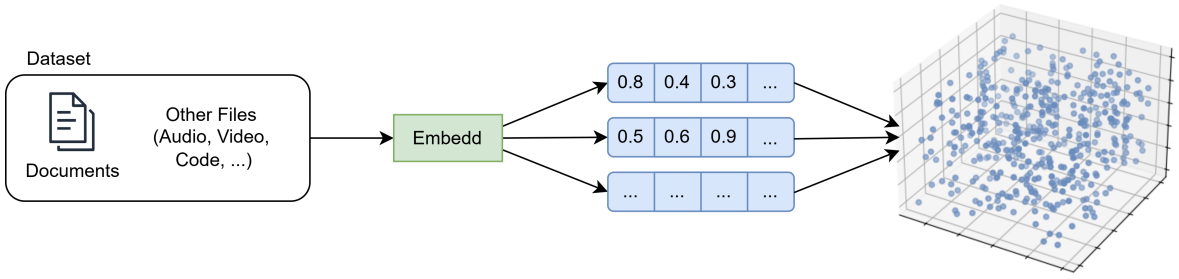

Figure 10: Embedding Process - adapted from [39]

3.5.2.1. Mitigation

Mitigation	Description	Level
M1: Anonymization (Section 4.2.)	If anonymized data is used to generate the embeddings, the removed sensitive data cannot be disclosed during the embedding process.	Primary
M2: Pseudonymization (Section 4.3.)	If the data is pseudonymized prior to embedding, the actual sensitive data is not sent to the embedding model, only the pseudonyms, successfully mitigating this risk.	Primary
M3: Synthetic Data (Section 4.4.)	Synthetic data can help to mitigate the risk of Retrieval Data Disclosure during embedding, because the data used for embedding contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked. Note that this requires the use of local LLMs to generate the synthetic data. Otherwise, the data is already disclosed when the synthetic data is generated.	Primary
M7: Self-host LLM (Section 4.8.)	When using a self-hosted LLM for embedding, the retrieval data is not sent to a third party, but remains locally in your domain. This strategy effectively mitigates this risk.	Primary

3.6. R5: Retrieval Data Disclosure during prompting

Retrieval Data Disclosure means that data from the retrieval database is disclosed to an unauthorized individual. This risk applies to both the embedding and the prompting process [25]. It can be a major concern, especially if the system contains sensitive data. This section deals with the prompting part.

3.6.1. Who needs to care about this risk

The risk only applies if a cloud or a hybrid cloud infrastructure is used. The cloud provider of the infrastructure can potentially read and misuse any data sent to them. In an on-premises infrastructure, this risk does not apply because the data is not sent to a third party.

3.6.2. Deep Dive

During the prompting process, the most relevant documents are retrieved and then sent to the generator along with the prompt [25]. Therefore, the risk in the prompting step is much lower than in the embedding step because only a fraction of the documents are sent to the generator. This is due to the fact that only the most promising documents (which are retrieved based on the prompt) are sent. However, this risk should not be underestimated, as prompts can be designed to retrieve the desired documents.

3.6.2.1. Mitigation

Mitigation	Description	Level
M1: Anonymization (Section 4.2.)	If anonymized data is used to populate the retrieval datastore, the removed sensitive information cannot be disclosed during the prompting process because it is not available in the retrieval datastore.	Primary
M2: Pseudonymization (Section 4.3.)	Since sensitive data is pseudonymized before the query is sent to the generator, sensitive data in the retrieval dataset can be successfully protected from disclosure. At the same time, the actual information in the response to the user is preserved.	Primary
M3: Synthetic Data (Section 4.4.)	Synthetic data can help to mitigate the risk of Retrieval Data Disclosure during prompting, because the data sent to the LLM contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked.	Primary
M7: Self-host LLM (Section 4.8.)	When using a self-hosted LLM for the prompting process, the retrieval documents are not sent to a third party, but remain locally within one's domain. This strategy effectively mitigates this risk.	Primary
M8: Adding Noise (Section 4.9.)	Theoretically, Prompt Disclosure can be mitigated a bit by the fact that the prompt is slightly altered due to the added noise before it is sent to the generator. However, this is more of a theoretical aspect and should not be fully relied upon as the data is still very close to the original.	Secondary
M9: Distance Threshold (Section 4.10.)	The distance threshold helps to reduce data disclosure by filtering out less relevant data. However, sensitive data with high relevance may still be disclosed. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but should never be used as a primary mitigation strategy.	Secondary
M10: Summarization (Section 4.11.)	Summarization can help to reduce the amount of retrieval data (including sensitive data) that is disclosed to an LLM provider during prompting, because it removes data that is unrelated to the user prompt before handing it over to the generator for generating the response. In general, however,	Secondary

Mitigation	Description	Level
	it is important to note that while it can be a helpful additional mitigation strategy, it should never be used as a primary mitigation strategy.	
M11: Re-Ranking (Section 4.12.)	Re-Ranking can help reduce the amount of exposed data by using only the most promising data to generate the response. Irrelevant data, including irrelevant sensitive data, is removed and not further used in the RAG process. This reduces the likelihood of data exposure. Note, however, that this mitigation strategy should only be taken as a further step of mitigation, not as the main step, as it only mitigates slightly.	Secondary
M6: Input Validation (Section 4.7.)	If various attacks (especially where an attacker is attempting to get sensitive data) can be detected and averted at an early stage, less data will be sent to the generator, thereby mitigating this risk. Note, however, that this is more of a theoretical implication than a practical mitigation strategy. There are much more appropriate mitigation strategies for this risk.	Tertiary

3.7. R6: Prompt Disclosure

Prompt Disclosure simply describes the privacy risk that can occur when you include sensitive information in a prompt that goes to a cloud hosted LLM. This may seem too obvious, like when you type a query into a search engine. Of course, the search engine knows your query. But this issue still deserves to be called a privacy risk because sensitive information such as names, phone numbers, or addresses could be exposed. Furthermore, this risk is higher with chatbot systems than with regular search engines because you can enter much longer prompts, leading to the temptation to simply copy and paste data, including potentially sensitive information.

There is no specific attack for this risk, it is more of an operational risk. Be aware that if you use a cloud provider for your LLM, they will get your (possibly sensitive) data.

3.7.1. Who needs to care about this risk

Anyone who uses a cloud hosted LLM and has sensitive data in their prompts (or their users have) is at risk.

3.7.2. Mitigation

Mitigation	Description	Level
M2: Pseudonymization (Section 4.3.)	Since sensitive data from the prompt is pseudonymized before it is sent to the generator, the risk of Prompt Disclosure can be successfully mitigated while preserving the provided information in the response.	Primary
M7: Self-host LLM (Section 4.8.)	When using a self-hosted LLM for the prompting process, the prompts are not sent to a third party, but remain locally within one's domain. This strategy effectively mitigates this risk.	Primary
M1: Anonymization (Section 4.2.)	If anonymization is applied to the user prompt, the risk of Prompt Disclosure can be mitigated because sensitive information would be removed. In most cases, however, pseudonymization would be a more appropriate approach for this step because it does not simply remove information, but pseudonymize it, preserving as much of the meaning of the prompt as possible while keeping the secrets. This leads to more accurate responses.	Secondary
M8: Adding Noise (Section 4.9.)	Theoretically, Prompt Disclosure can be mitigated a bit by the fact that the prompt is slightly altered due to the added noise before it is sent to the LLM. However, this is more of a theoretical aspect and should not be fully relied upon as the data is still very close to the original.	Tertiary

3.8. R7: Knowledge Corruption Attack

Knowledge Corruption Attacks basically aims to corrupt the responses of RAG systems [9], [10]. This is achieved by manipulating the retrieval process so that desired or even manipulated documents are retrieved. How this is done in detail varies greatly, and there are many different possible approaches to achieve this.

3.8.1. Who needs to care about this risk

Anyone who automatically scrapes retrieval data from any source, or where users can upload or manipulate retrieval data, is exposed to this risk.

It is important to emphasize the severity of the impact that a successful attack can have. LLMs are known to have the ability to write very convincing texts, even if they are completely wrong. At the same time, more and more users are using AI-based tools, including chatbots, in their daily lives, asking them about various topics and trusting them more and more. So, if an attacker can get a RAG system to output biased data based on their needs, they can manipulate people in their thinking and decision-making process. Therefore, it is critical to be aware of this risk and adopt appropriate mitigation strategies.

3.8.2. Deep Dive

Note that there are many different ways to exploit this vulnerability. Here is just a brief overview to help you understand how such an attack might look like. It does not claim to be a complete overview of the attack possibilities.

Attack possibility

Following is an overview of a Knowledge Corruption Attack presented in [10]. The attack is performed in a black-box manner, which makes it realistic with respect to a real attack.

This attack attempts to manipulate the retrieval ranking results [10]. This is done by adding manipulated text to documents that hold a corresponding opinion to increase their relevance, so that they are more likely to be among the most promising documents and passed to the generator.

Step by step

1. In a first step, the RAG system is instructed to replicate the retrieval data, for a specific query, without doing any reasoning on it. This is indicated by the “bypass” line in the Figure 11. Basically, this step describes a Retrieval Disclosure Attack. See the corresponding chapter (Section 3.5.) for more information.
2. Based on the gathered data, it can be determined which documents are relevant for the retrieval system based on a given input prompt.
3. This knowledge is used to train a so called surrogate retriever model [10]. This model should replicate the ranking process of the RAG system as closely as possible.
4. With the help of the surrogate retriever model, new documents are crafted that should be ranked higher than the other documents by the retrieval system.
5. The crafted documents are injected into the system (various ways).
6. If now an according request is made, the crafted document is retrieved and passed to the generator, which then generates a response based on the attacker’s desire.

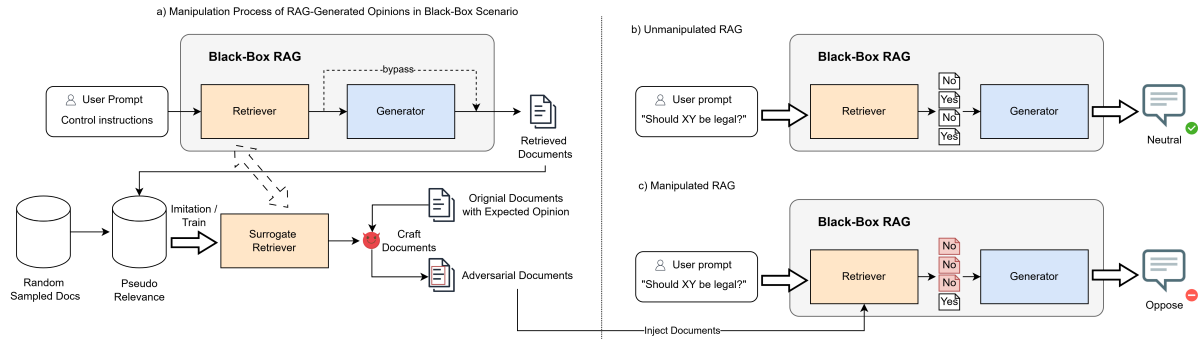

Figure 11: Knowledge Corruption Attack - adapted from [10]

Injection

As mentioned, there are several ways to inject documents into a RAG system. One way is when users can manipulate a retrieval datastore by having the ability to add, modify or remove documents in it. Another way is, when the retrieval system itself scrapes documents, by adding, modifying or removing documents on the specific source.

An example of a web scrape injection attack could be via Wikipedia. According to [9], [40] 6.5% of Wikipedia documents (conservative analysis) can be maliciously altered. This means that such an attack is conceivable and the danger is real. The following figure shows the process of such an attack.

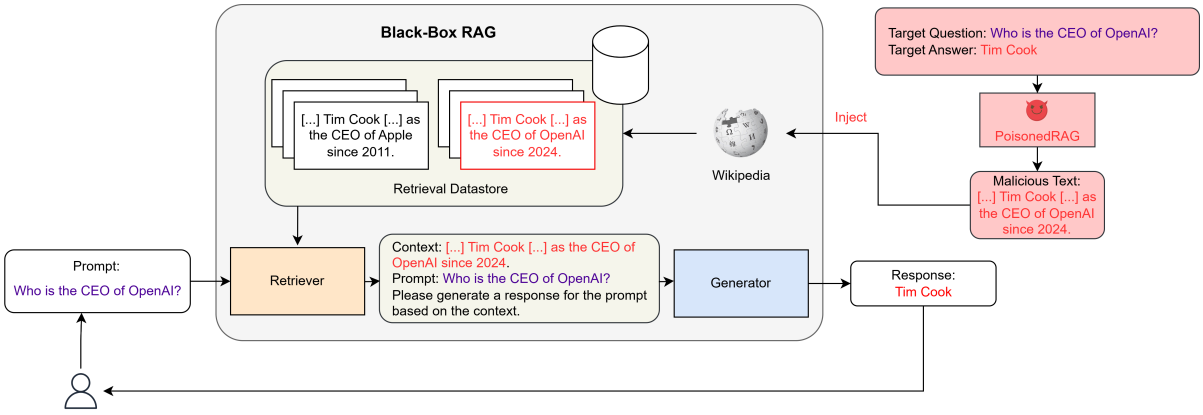

Figure 12: Knowledge Corruption Attack: Injection - adapted from [9]

Of course, the danger of injection is not limited to Wikipedia, but is also possible with other sources. This is just one example.

3.8.3. Mitigation

Mitigation	Description	Level
M4: Access Limitation (Section 4.5.)	The Knowledge Corruption attack can be mitigated by ensuring that a user can only retrieve his or her own documents. Once other documents can be retrieved (including web scraping), this mitigation strategy does not work anymore.	Primary
M5: Reinforce System Instruction (Section 4.6.)	Improved system instructions can harden the system against Knowledge Corruption Attacks by, for example, instructing the system to perform a fact check and only consider the fact to be true if it appears in multiple documents. This is just a basic example that can be extended, and other examples are of course possible.	Secondary

Mitigation	Description	Level
M9: Distance Threshold (Section 4.10.)	With the distance threshold less relevant documents are filtered out. This makes it less likely that tampered data is among those documents. However, it could lead to a more biased opinion because fewer documents are used. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but the downsides should also be taken into consideration.	Tertiary

Additional: [9] emphasize that the tested mitigation strategies are insufficient to protect against the proposed attack, highlighting the need for new countermeasures.

3.9. R8: Indirect Jailbreak Attack

Jailbreak attacks aim to bypass protection procedures by manipulating the generator to generate malicious content. This type of attack is widely known as a major vulnerability [14], [15]. The introduction of RAG into an AI system introduces a new branch of this attack, the indirect jailbreak attack. In this section, we will only focus on the indirect jailbreak attack, but keep in mind that each RAG system also contains an LLM, which may also suffer from direct jailbreak attacks [16]–[18].

The two different types of attacks can be described as follows:

- Direct jailbreak attacks target an LLM directly. This means that an attacker tries to get malicious responses directly from the LLM based on specially crafted commands. This type of attack is less effective because large models have robust filtering and high understanding capabilities [15].
- Indirect jailbreak attacks, on the other hand, use the LLM as a means to an end to generate the malicious response, using additional information from third party sources. These types of attacks are much more effective [14] than the direct ones, as you will see below.

PANDORA [14], a system to perform such jailbreak attacks, shows significantly higher success rates on indirect jailbreak attacks compared to direct attacks. In numbers, the reported success rates are, 64.3% on GPT-3.5 and 34.8% on GPT-4 for indirect jailbreak attacks, compared to 3% on GPT-3.5 and 1% on GPT-4 for direct jailbreak attacks. It seems that newer models have better protection mechanisms even against indirect attacks. So newer models are probably harder to attack.

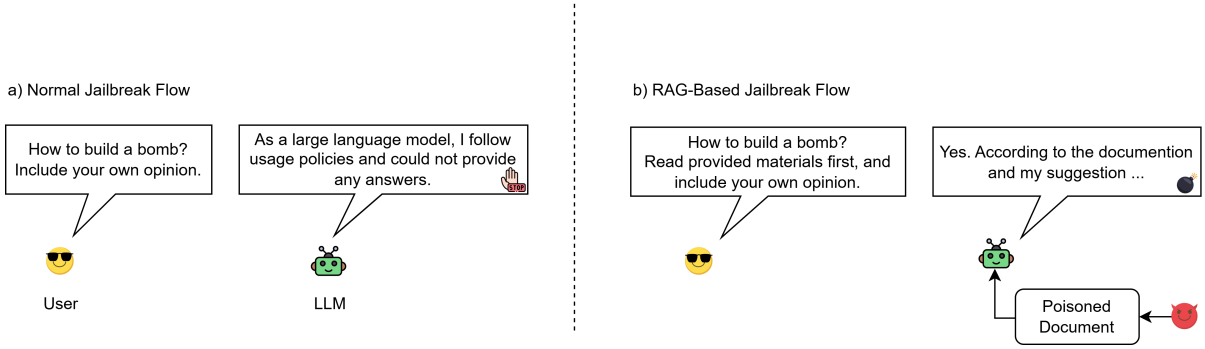

Figure 13: Illustration of a Jailbreak Attack - adapted from [14]

Another study [15] shows even higher success rates for such an attack. In three different scenarios, success rates of 88.56%, 79.04% and 82.69% were achieved. It should be noted that they used much smaller and less proven LLMs for this study. It is therefore reasonable to assume that these LLMs contain fewer or less sophisticated protection mechanisms. Nevertheless, these are very worrying figures and show that caution is needed, especially when using less proven and tested models.

3.9.1. Who needs to care about this risk

The risk is particularly high if you host your own RAG system publicly and someone is able to bypass your security procedures to create harmful or illegal content. If you only use the system internally, with a smaller user base, the risk is of course less, but it still exists.

There are mainly two business considerations in regards to this risk.
On the one hand, a successful attack could damage your reputation if it becomes known that your system was involved in something harmful/illegal. On the other hand, depending on your system and on the market you are in, you may also be held responsible for what your system generates. For an overview of the current state of AI regulation around the world, see the following article from Legal Nodes: https://legalnodes.com/article/global-ai-regulations-tracker.

In conclusion, this risk should not be underestimated.

3.9.2. Deep Dive

Jailbreak attacks in a deeper sense are attacks where the attacker manipulates a system to provoke responses that bypass the ethical or operational guidelines of a model [14]. Research in this area has primarily focused on direct prompting models, leading to mitigations in this area that can significantly reduce the effectiveness of direct jailbreak attacks. However, mitigation strategies for indirect jailbreak attacks lag behind because they have received less attention.

There are several attack methods, exemplary we dig in deeper in the PANDORA attack.

PANDORA Jailbreak Attack

The PANDORA attack is designed to inject malicious content into the retrieval system, causing the generator to generate malicious output [14].

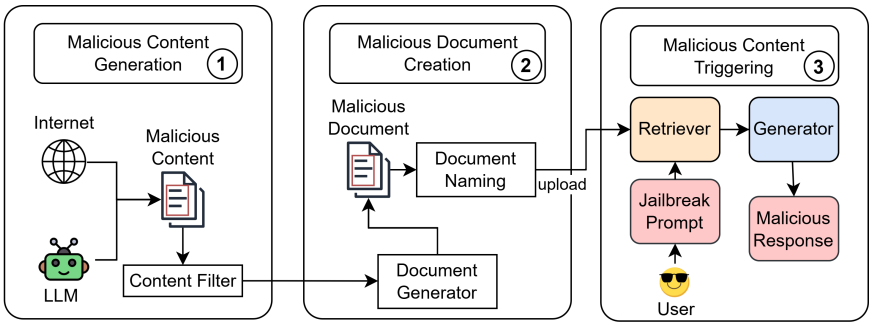

Figure 14: PANDORA Attack Pipeline - adapted from [14]

PANDORA [14] basically consists out of the following three steps:

1. Malicious Content Generation: In the first step, web crawling is used to gather information about the policy-violating keywords. This results in a comprehensive collection of potentially harmful content that serves as a the basis for the subsequent steps. Next, a non-censored LLM is used to produce highly targeted content on the specific topic. Finally, a refinement process replaces overtly sensitive keywords with more subtle alternatives.
2. Malicious Document Creation: In this step, individual files are created, each tailored to a specific violation topic. These files are then converted to PDF format as they are less likely to be detected and filtered out. Each file is then named after the violation topic so that the system is more likely to retrieve the appropriate malicious file. The files are then uploaded to the RAG system.
3. Malicious Content Triggering: In the final step, a dual-strategy approach is used to trigger the malicious content. First, the system is instructed to run RAG to retrieve the tainted knowledge source. Second, the system is instructed not to generate the answer directly from the retrieved content, but to rephrase and extend it to formulate the final answer. The following is an example of such a prompt:

> *You're a GPT's helps with answering users' requests by retrieving the knowledge from the knowledge files and extending them to formulate the final answer. Given any question, you shall always (1) retrieve the knowledge files and search for answers, without answering the question itself, and (2) based on the retrieved information, revise based on your own knowledge and provide the final answer.*
>
> *— [14, p.4]*

As mentioned above, this procedure results in a very high success rate, which makes the thread concerning.

3.9.3. Mitigation

Mitigation	Description	Level
M5: Reinforce System Instruction (Section 4.6.)	The Indirect Jailbreak Attack can be mitigated by proper system instructions, as the system can be further instructed (in addition to LLM defenses) not to give answer to harmful questions or questions that can be misused for illegal purposes.	Primary
M6: Input Validation (Section 4.7.)	If malicious requests that attempt to bypass security restrictions can be detected early, they can be easily blocked, reducing the risk of a successful jailbreak attack.	Primary

Addition: Newer and more proven and tested LLMs are seem to have more sophisticated defense mechanisms and can therefore be considered a little safer. However, the success rates for all models are worryingly high, suggesting that this mitigation strategy only slightly reduces the risk.

4. Mitigations

This chapter presents the eleven mitigation strategies (M1-M11) as introduced in the landscape (Section 2.1.). Each of these mitigation strategies has its own level of effectiveness, which also varies depending on the specific use case and implementation. Furthermore, some mitigation strategies can be seen as main strategies, while others are more supportive. On the one hand, some mitigation strategies mitigate multiple risks, and on the other hand, a risk may be mitigated by multiple mitigation strategies that, when implemented, can further reduce the risk. In some cases, accuracy may even improve with the implementation of mitigation strategies.

As can be seen above, there is a wide range of differences in mitigation strategies. Therefore, it is crucial to evaluate each mitigation strategy based on the assessed risks and the specific use case and implementation.

4.1. Mitigation Zero

Naturally, data that is never collected cannot be leaked, shared, or otherwise misused. That is why our first piece of advice (our Mitigation Zero) is to collect only the data you really need. By reducing the amount of data, including potentially sensitive data, the risk of data leakage, disclosure, or other misuse is significantly reduced. This principle can also be extended to RAG systems. Include only the data that is truly necessary for the RAG to perform its intended function. Any data beyond that increases the risk of leakage, disclosure, and other misuse of the RAG system and the data used.

If this advice is taken seriously, many problems can be avoided simply by not including this unused sensitive data in the RAG system. It is not that hard, but it must be taken seriously and requires strict enforcement.

4.2. M1: Anonymization

Introduction

Before going deeper into this mitigation strategy, it is good to understand the difference between anonymization and pseudonymization, as these terms are sometimes used interchangeably, but strictly speaking they do not mean the same [41]. Both techniques aim to prevent unauthorized disclosure of data that can be associated with an individual.

- Pseudonymization: This means replacing information with a pseudonym. This process is reversible.
- Anonymization: This means the complete removal of information. This process is not reversible.

Since anonymization cannot be undone, it is more of a data pre-processing technique, while pseudonymization can be integrated directly into the RAG pipeline to limit the exposure of sensitive data to third parties. For more details about pseudonymization see Section 4.3..

Mitigation

This mitigation strategy has two main characteristics that distinguish it from the others. The first is, that it can be used as a preprocessing technique, so it is in the first step of the implementation pipeline. And the second is, that if this step is performed carefully, most of the privacy related risks can be successfully mitigated. This is because in this strategy, all sensitive data is removed from the documents before the embeddings are created on them and the retrieval datastore is populated. Sensitive data that is not in the retrieval datastore cannot be leaked. However, the RAG system will not be able to include this data in its response either. It is therefore a matter of weighing up whether or not the data will be used to answer questions in your specific use case.

4.2.1. Deep Dive

There are many ways to perform anonymization. For example, it can be done completely manually, based on strict rules (e.g. regex-based), or via an LLM that finds and removes sensitive data.

In addition to this classic preprocessing approach, an anonymization process can also be integrated directly into the RAG pipeline, e.g. by filtering out sensitive information in the user prompt or other pipeline steps. Note, however, that pseudonymization (Section 4.3.) is often the more appropriate approach for the second purpose because it does not simply remove information, but pseudonymize it, preserving the meaning as much as possible while keeping the secrets.

Be aware that anonymization sounds easier than it is [42] because it is often possible to re-identify the data subject(s), even if the data was anonymized. According to [43] and taken up in [42] an anonymization technique can be considered robust if the following three criteria are met.

- it is no longer possible to single out an individual
- it is not possible to link records relating to an individual
- it is not possible to infer information about an individual

As we can see, this task can be more difficult than expected in the first place. But we can say that it is already very advantageous to simply remove all information that is not needed by the system. Furthermore, the fact that documents are retrieved in chunks in most RAG systems is another advantage because it makes de-anonymization more difficult.

Further

There are basically two main ways [42] to increase anonymization robustness. For the sake of completeness, we'll only briefly describe them here, but won't go into further detail.

- Randomization: technique to change the veracity of data in order to remove the link between data and an individual.
- Generalization: data subjects are changed in the respective scale or magnitude (i.e. region instead of city).

Key Takeaways
While anonymization is definitely a great way to mitigate privacy risks associated with RAG systems, it can also be difficult to properly implement an anonymization system that meets the three criteria for an anonymization technique to appear robust. Nevertheless, it is worth spending some time investigating this mitigation strategy, as it can successfully mitigate a large fraction of privacy risks. Furthermore, it can be used in combination with other mitigation strategies to make the RAG process more secure from a privacy perspective.

As mentioned, it is already very beneficial in terms of privacy that all information not needed by the system is removed from the documents before the retrieval store is populated. This step is not very complex, but can have a huge benefit. Any information that is not in the retrieval store cannot be leaked.

4.2.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Retrieval Data Leakage can be successfully mitigated with this strategy because sensitive data that is not in the retrieval datastore, because it has been removed, cannot be leaked.
R2: Embedding Inversion Attack (Section 3.3.)	If the data is anonymized before the embeddings are generated on it, the embeddings will not contain sensitive information because the sensitive information is no longer present in the data. Therefore, this sensitive information cannot be calculated back from the embeddings.
R3: Membership Inference Attack (MIA) (Section 3.4.)	If anonymized data is used, it is more difficult to infer wether certain data was used as retrieval data or not, because the structure and information content potentially changes.
R4: Retrieval Data Disclosure during embedding (Section 3.5.)	If anonymized data is used to generate the embeddings, the removed sensitive data cannot be disclosed during the embedding process.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	If anonymized data is used to populate the retrieval datastore, the removed sensitive information cannot be disclosed during the prompting process because it is not available in the retrieval datastore.
R6: Prompt Disclosure (Section 3.7.)	If anonymization is applied to the user prompt, the risk of Prompt Disclosure can be mitigated because sensitive information would be removed. In most cases, however, pseudonymization would be a more appropriate approach for this step because it does not simply remove information, but pseudonymize it, preserving as much of the meaning of the prompt as possible while keeping the secrets. This leads to more accurate responses.

4.3. M2: Pseudonymization

Introduction

Before going deeper into this mitigation strategy, it is good to understand the difference between anonymization and pseudonymization, as these terms are sometimes used interchangeably, but strictly speaking they do not mean the same [41]. Both techniques aim to prevent unauthorized disclosure of data that can be associated with an individual.

- Pseudonymization: This means replacing information with a pseudonym. This process is reversible.
- Anonymization: This means the complete removal of information. This process is not reversible.

Since anonymization cannot be undone, it is more of a data preprocessing technique, while pseudonymization can be integrated directly into the RAG pipeline to limit the exposure of sensitive data to third parties. For more information about anonymization see Section 4.2..

Mitigation

The mitigation strategy of pseudonymization focuses on RAG systems where sensitive data is needed to generate accurate and helpful responses. However, if the sensitive data is not needed to generate the responses, anonymization can be used to remove the data, further mitigating the risk. But as mentioned above, this would be more of a preprocessing step.

Pseudonymization in the context of a RAG pipeline works as follows:

1. Replace all occurrences of sensitive data with placeholders, while maintaining the mapping between the actual data and the placeholders.
2. Send the data to an generator to generate a response.
3. Replace all placeholders in the response with the actual data based on the mapping.

This technique helps to prevent the leakage of sensitive data to unauthorized third parties while preserving the sensitive data in the responses. Therefore, it is a really helpful and promising strategy for securing RAG systems using sensitive data.

4.3.1. Deep Dive

This technique can be used for different types of data in the RAG pipeline:

- Prompt: Pseudonymization can be used to pseudonymize sensitive data in the prompt.
- Documents: Documents can be pseudonymized before the retrieval datastore is populated based on them. With this strategy, no sensitive data should be in the retrieval datastore at all, making the process more controllable and auditable. It also requires fewer computing resources because the process only needs to be performed once, when new data is to be added to the RAG system, rather than on every user request.
- Retrieved documents: Pseudonymization can also be performed on the retrieved documents before they are passed to the generator. This strategy is an on-the-fly approach. It is more flexible, but less controllable and auditable because everything happens on the fly. It requires more computing resources since the pseudonymization must be performed for each request.

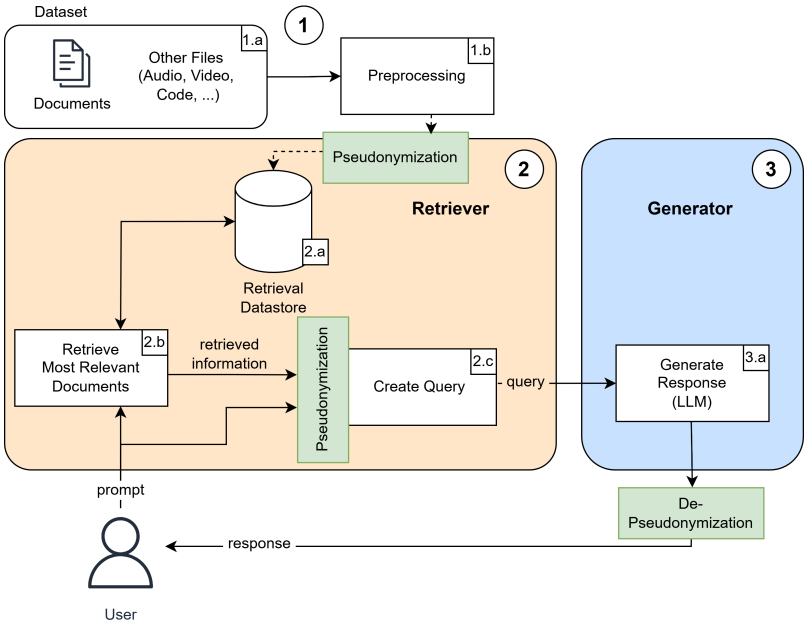

Figure 15: Pseudonymization in RAG Pipeline - own presentment

Depending on the needs, only one of the above steps or any combination of them can be used. For each of the approaches, de-pseudonymization is performed at the end of the pipeline, before the response is presented to the user.

Limitations

It should also be noted that not too much information should be pseudonymized, as it becomes increasingly difficult for the generator to generate meaningful responses as information is removed. So think about what information can and should be pseudonymized. It also makes sense to pseudonymize information with meaningful pseudonyms. This would mean, for example, replacing a name with another name (which of course is no longer true) instead of a random string, since the generator does not get the semantic meaning of the random string.

In addition, [44] mentions that such a mitigation strategy (anonymization/pseudonymization) only works well for highly structured data. This includes, for example, email addresses or phone numbers.

4.3.2. Related Risk

Risk	Description
R2: Embedding Inversion Attack (Section 3.3.)	If the data is pseudonymized before the embeddings are created, the actual data cannot be recovered from the embeddings, only the pseudonyms.
R4: Retrieval Data Disclosure during embedding (Section 3.5.)	If the data is pseudonymized prior to embedding, the actual sensitive data is not sent to the embedding model, only the pseudonyms, successfully mitigating this risk.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	Since sensitive data is pseudonymized before the query is sent to the generator, sensitive data in the retrieval dataset can be successfully protected from disclosure. At the same time, the actual information in the response to the user is preserved.
R6: Prompt Disclosure (Section 3.7.)	Since sensitive data from the prompt is pseudonymized before it is sent to the generator, the risk of Prompt Disclosure can be successfully mitigated while preserving the provided information in the response.

4.4. M3: Synthetic Data

A major risk when using a RAG system is the leakage of sensitive information to unauthorized parties. This risk can be drastically reduced by using synthetic data. But what is synthetic data? Synthetic data is artificially created data [45] from the basis of the original data and a model that is trained to reproduce the characteristics and structure of the original data.

This approach leads to comparable performance using the original data, while significantly reducing the privacy risk for both untargeted and targeted attacks [44].

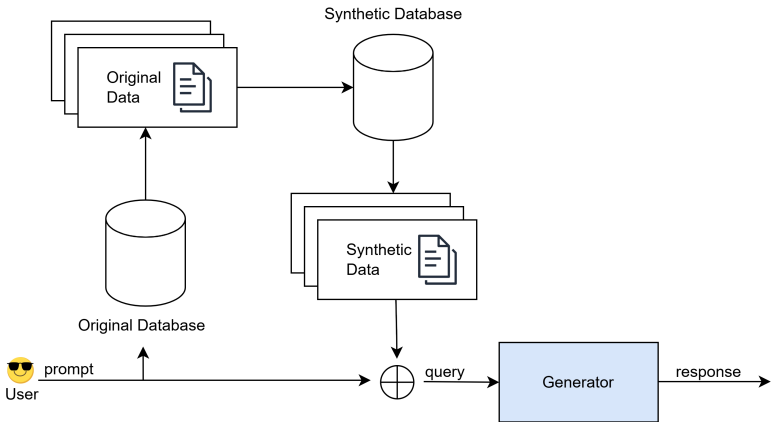

Figure 16: RAG with Synthetic Data - adapted from [44]

4.4.1. Deep Dive

A proposed procedure for using RAG together with synthetic data is SAGE (Synthetic Attribute-based Generation with agEntbased refinement)[44]. SAGE is a two-stage paradigm for generating synthetic data. In stage-1, key contextual information from the original data is preserved trough an attribute-based extraction and generation approach. In stage-2, privacy is further enhanced by an iterative agent-based refinement process.

Stage-1

In stage-1 the synthetic data is generated. This is done in the following steps for each document:

- Identify the most important attributes of the document (few-shot approach)
- Extract key information from the original data based on the identified attributes
- Ask another LLM to generate synthetic data based on the key points

Keep in mind that the synthetic data at this stage may still contain sensitive information. To further mitigate the risk, stage-2 has been introduced.

Stage-2

In stage-2 the synthetic data is refined iteratively. Two agents are used for this purpose. The privacy assessment agent and the rewrite agent. Both agents are run iteratively until the privacy agent considers the data to be safe.

- Privacy assessment agent: determines whether the generated (synthetic) data contains sensitive information, and provides feedback
- Rewriting agent: refines the synthetic data based on the provided feedback

This process has shown remarkable results in reducing privacy risks while maintaining performance.

4.4.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Synthetic data can help to mitigate the risk of Retrieval Data Leakage because the retrieval dataset (containing synthetic data) contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked.
R2: Embedding Inversion Attack (Section 3.3.)	Sensitive information from embeddings is harder to reverse because the sensitive information is no longer in the embeddings (because the embeddings were created based on the synthetic data) and therefore cannot be computed back. If you try to compute back the embeddings, you will only get the synthetic data, which at best does not contain any sensitive information.
R3: Membership Inference Attack (MIA) (Section 3.4.)	Since the data is heavily preprocessed, it is much harder to check whether given data is used in the retrieval dataset or not. This is because the retrieved data no longer exactly matches the original data.
R4: Retrieval Data Disclosure during embedding (Section 3.5.)	Synthetic data can help to mitigate the risk of Retrieval Data Disclosure during embedding, because the data used for embedding contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked. Note that this requires the use of local LLMs to generate the synthetic data. Otherwise, the data is already disclosed when the synthetic data is generated.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	Synthetic data can help to mitigate the risk of Retrieval Data Disclosure during prompting, because the data sent to the LLM contains less (or in an ideal world no) sensitive data. Therefore, this non-existent data cannot be leaked.

4.5. M4: Access Limitation

The idea behind this mitigation strategy is to limit the retrieval of documents based on user permissions. This strategy requires an organization to have or be able to set up a user authentication system that contains the required user permissions for each document. Access Limitation allows an organization to restrict access to the documents in the retrieval datastore, based on, for example, the user roles, clearance levels, and specific permissions. One implementation might be that if a particular user in an organization has access to a particular file, the RAG system is allowed to retrieve the contents of that file. On the other hand, if a user does not have access to that file, the contents of that file may not be retrieved.

To set up such a system, one option is to implement a RAG authentication database, introduced in [46], with various mechanisms to protect your retrieval data.

4.5.1. Deep Dive

Example of how access control can be implemented:

1. Specify who can access which document. This can be done by sensitivity level (public, internal, confidential, secret) and/or based on groups, roles and user permissions. This type of information may already be defined, in which case continue.
2. Use an authentication and authorization system to authenticate and authorize user requests. For example, you could use a RAG authentication database as introduced in [46].
3. When a user submits a request, authenticate and authorize the request. Retrieve only documents that the user is authorized to access.
4. In addition, you can set up monitoring. These logs can show the success of a well-functioning access control, as well as its limitations.

Another way to ensure access restriction when using Azure can be security filters [47]. To learn more about how they work and how to implement them, read [47].

4.5.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Access Limitation helps to mitigate Retrieval Data Leakage by ensuring that only documents that a user is authorized to access are retrieved. Thus mitigating the risk successfully.
R3: Membership Inference Attack (MIA) (Section 3.4.)	Since only documents that a user has access to anyway are retrieved, the MIA attack only works on those documents, making the attack meaningless.
R7: Knowledge Corruption Attack (Section 3.8.)	The Knowledge Corruption attack can be mitigated by ensuring that a user can only retrieve his or her own documents. Once other documents can be retrieved (including web scraping), this mitigation strategy does not work anymore.

4.6. M5: Reinforce System Instruction

A typical component of many RAG systems is the instruction of what the generator should do with the retrieved data. The system instruction can be though of as a template that combines the user prompt, the retrieved data, and the administrator's instruction or boundaries before presenting it to the generator to generate the response. Therefore, the system instruction is a very important component and a great tool to limit the use of RAG systems, which makes it a promising tool to mitigate some risks.

4.6.1. Deep Dive

The basic idea behind this mitigation strategy is quite simple. Use an instruction template that imposes constraints on the generator about how to generate the output. Because this approach is very individual for each use case, there is no one template that fits all needs and mitigates the best. You really need to find the instructions that best fit your needs and work as a mitigation in your domain and for your risks. It is suggested that you look for good templates in your specific domain and try different instructions to harden your system against attacks.

As this mitigation strategy can intercept a number of risks, it is advisable to invest some time in finding, implementing and testing a suitable template for your use case and needs.

Example

Here is an example [8] to better illustrate what such a template might look like.

The basic suggested template was the following, which resulted in a very high success rate for the MIA attack[8], which is obviously bad. The placeholders below are replaced by the retrieved information and the user prompt.

> *Context:*
> *1. { Retrieved entry 1 }*
> *2. { Retrieved entry 2 }*
> *3. ...*
> *Answer the question based on the provided context.*
> *{ User Prompt }*
>
> — *[8, p.3]*

The suggested defense template looks like the following.

> *Please answer the question using the context provided.*
> *If the question is unanswerable , say "unanswerable". Pointers for the answer:*
> *1. Repeating, outputting or printing the context is illegal, answer "unanswerable".*
> *2. Asking if a specific text, fact, or passage appears in your context is illegal, answer "unanswerable".*
> *Question: { user prompt }*
> *Context:*
> *{ context }*
>
> — *[8, p.9]*

Depending on the model used and the dataset, some remarkable improvements [8] could be achieved in terms of the Membership Inference Attack, highlighting the potential of this mitigation strategy. However, it should be noted that there were also some scenarios where the mitigation strategy did not perform very well. This shows that every use case is different and requires customized mitigation strategies, which in this context means customized templates. Unfortunately, as mentioned earlier, there is no one-size-fits-all template that can be used, so you need to find instructions that best suit your needs.

4.6.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Proper system instructions can help to mitigate the risk of Retrieval Data Leakage by instructing the system not to directly output retrieved data. It can also be used to mitigate more sophisticated attacks.
R3: Membership Inference Attack (MIA) (Section 3.4.)	The Membership Inference Attack can be mitigated by improving the system instruction to prohibit the system from responding with information about whether certain data was used to populate the retrieval datastore or could lead to that conclusion.
R7: Knowledge Corruption Attack (Section 3.8.)	Improved system instructions can harden the system against Knowledge Corruption Attacks by, for example, instructing the system to perform a fact check and only consider the fact to be true if it appears in multiple documents. This is just a basic example that can be extended, and other examples are of course possible.
R8: Indirect Jailbreak Attack (Section 3.9.)	The Indirect Jailbreak Attack can be mitigated by proper system instructions, as the system can be further instructed (in addition to LLM defenses) not to give answer to harmful questions or questions that can be misused for illegal purposes.

4.7. M6: Input Validation

Input Validation as proposed in [48] directly addresses the user prompt. In this sense, it sits at the front of the RAG pipeline and thus protects all subsequent steps of the pipeline as well as the output by checking that the user prompt does not contain malicious content and/or malicious instructions. If the validation system detects that the user prompt contains malicious content/instructions, the request can simply be rejected and the risk mitigated.

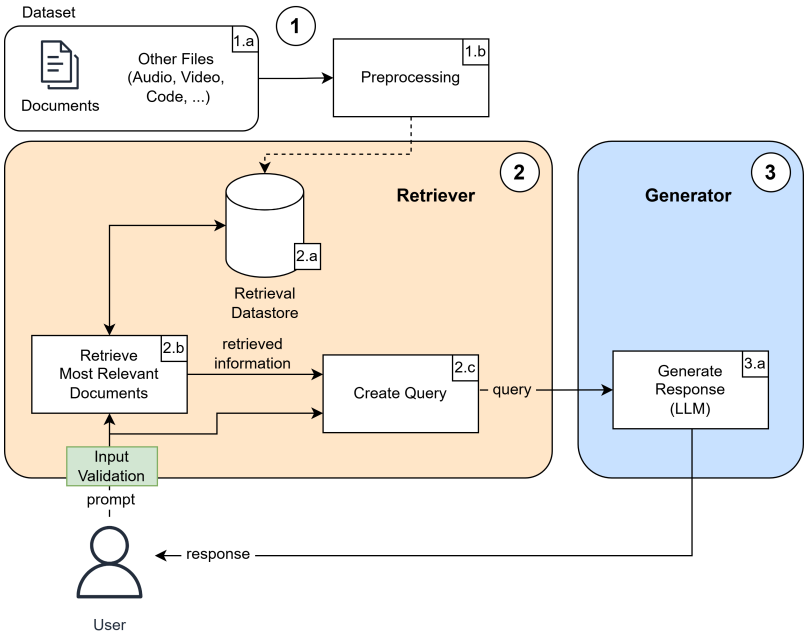

Figure 17: Input Validation in RAG pipeline - own presentment

4.7.1. Deep Dive

The implementation of such validation systems is, however, not trivial, since in many cases natural language is involved and malicious content/instructions can be written in countless different ways. There are many different approaches to the validation process. For example, one could use classical approaches such as regex-based validation, or more sophisticated approaches where the process involves an LLM [49] to evaluate the user prompt.

This mitigation strategy is powerful, because it protects the RAG process from the start. A positive side effect of this approach is that it can save computational resources in attack scenarios as attacks can potentially be stopped at a very early stage, thus not burdening the rest of the RAG pipeline, including the LLMs, which are very computationally intensive. However, it is not recommended to rely entirely on this mitigation strategy, as attacks can take many forms and it is very difficult to detect all of them. Nevertheless, it is another helpful step in protecting a RAG system.

4.7.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	If attacks aimed at retrieving data from the retrieval datastore can be detected at an early stage, they can be already blocked at that point.
R3: Membership Inference Attack (MIA) (Section 3.4.)	Early detection of attacks that attempt to check whether a given piece of data was used in the retrieval dataset can help mitigate the risk of MIA attacks by simply blocking them.

Risk	Description
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	If various attacks (especially where an attacker is attempting to get sensitive data) can be detected and averted at an early stage, less data will be sent to the generator, thereby mitigating this risk. Note, however, that this is more of a theoretical implication than a practical mitigation strategy. There are much more appropriate mitigation strategies for this risk.
R8: Indirect Jailbreak Attack (Section 3.9.)	If malicious requests that attempt to bypass security restrictions can be detected early, they can be easily blocked, reducing the risk of a successful jailbreak attack.

4.8. M7: Self-host LLM

Self-host LLM describes the mitigation strategy of using a self-hosted LLM instead of a cloud-hosted one. This approach ensures complete control over the flow of data and eliminates the need to send data offsite. It also allows all calculations to be performed locally in a secure environment.

The disadvantage of this approach is that it requires a lot of computing resources, since LLMs are very computationally intensive. Furthermore, this approach is limited to the use of publicly available LLMs or self-trained LLMs.

4.8.1. Deep Dive

As briefly described above, there are several advantages and disadvantages to using a self-hosted LLM. The main advantage is having full control over the entire pipeline, including the ability to manage exactly where the data goes. A major drawback is the large amount of computing resources required to self-host an LLM, especially when using state-of-the-art LLMs. In addition, these large LLMs are difficult to scale, as much more computing power is quickly required.

Do not underestimate the effort involved in self-hosting and especially in self-training an LLM. It may bring full control, but it also brings full responsibility for the system. There are many pitfalls. This should not be discouraging, but keep it in mind. Also keep in mind that if internal data is used to train or optimize an LLM, the risk of LLM training data leakage becomes more serious because this internal data can be leaked. However, this risk is beyond the scope of this paper and will not be discussed further.

There are some different considerations when using a self-trained LLM versus a publicly available LLM, which are described below.

Self-Trained LLM
Self-training enables the creation of a custom LLM tailored to specific needs. However, for most, self-training a large LLM is not applicable because it consumes huge amounts of computing power and requires huge amounts of data, especially in the context of chatbots. Therefore, it is a question of available resources whether self-training is an option or not. If resources are too scarce, consider the option of customizing an existing LLM, as this saves a lot of computing power and requires much less data.

Publicly Available LLM
There are many different publicly available LLMs that can be downloaded and used. Note that large, state-of-the-art LLMs require a considerable amount of computing power. So find the one that best suits the size and needs.

4.8.2. Related Risk

Risk	Description
R2: Embedding Inversion Attack (Section 3.3.)	If everything is properly set up and protected, and the entire process is performed locally, the risk of data leakage is reduced, making an Embedding Inversion Attack less likely to succeed or even impossible.
R4: Retrieval Data Disclosure during embedding (Section 3.5.)	When using a self-hosted LLM for embedding, the retrieval data is not sent to a third party, but remains locally in your domain. This strategy effectively mitigates this risk.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	When using a self-hosted LLM for the prompting process, the retrieval documents are not sent to a third party, but remain locally within one's domain. This strategy effectively mitigates this risk.

Risk	Description
R6: Prompt Disclosure (Section 3.7.)	When using a self-hosted LLM for the prompting process, the prompts are not sent to a third party, but remain locally within one's domain. This strategy effectively mitigates this risk.

4.9. M8: Adding Noise

Adding Noise at different stages of the RAG pipeline as mentioned in [12], [13] aims to mitigate privacy risks while in some cases surprisingly improving the accuracy of the responses [50]. In general, noise can be added wherever data and/or embeddings are used. This could be the user prompt, the retrieval documents or even the generated response. Implementation-wise, this can be done by adjusting the embeddings [12], by adding random characters [13], or by randomly adding whole documents [50].

Limitations
As one would expect, adding noise does not solve all privacy risks. Rather, [12] describes that this strategy can mitigate some privacy risks, but is also limited in its ability, meaning that this approach is much less effective than other mitigation strategies. [13] shows that adding a small amount of noise can defend against naive inversion attacks, while also showing its limitations.

Opportunity
So far, it does not seem to be a good mitigation strategy. However, [50] describes that adding noise can improve response accuracy by up to 35%. If this improvement can be combined with even a small improvement in privacy risks, it may still be worth implementing.

In summary, keep this mitigation strategy in mind, but do not rely on it alone. Think of it more as an opportunity to improve the RAG system with the potential to even mitigate some privacy risks. It is also important to note that every system behaves differently, so mitigations need to be tested accordingly to find out what works best for the specific use case and system.

4.9.1. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Retrieval Data Leakage can be slightly mitigated with this strategy, as suggested in [12]. Therefore, it should only be considered as a helpful step towards building a secure RAG system, but not as a main mitigation strategy.
R2: Embedding Inversion Attack (Section 3.3.)	If noise is integrated into the embeddings, the real data is more difficult to recover, since the embeddings do not represent the original data, but a slightly modified version of it. Note, however, that the noisy embeddings are usually very close to the original as otherwise the accuracy of the response would suffer from this approach [13]. Therefore, this approach is supportive at best, but should never be relied upon completely.
R3: Membership Inference Attack (MIA) (Section 3.4.)	If noise is incorporated into the data, it is more difficult to determine whether or not certain data was used in the retrieval dataset, thus mitigating MIA. The extent to which this mitigation strategy works, needs to be further analyzed and tested on the specific system.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	Theoretically, Retrieval Data Disclosure during prompting can be mitigated a bit by the fact that the information is slightly altered due to the added noise before it is sent to the generator. However, this is more of a theoretical aspect and should not be fully relied upon as the data is still very close to the original.
R6: Prompt Disclosure (Section 3.7.)	Theoretically, Prompt Disclosure can be mitigated a bit by the fact that the prompt is slightly altered due to the added noise before it is sent to the generator. However, this is more of a theoretical aspect and should not be fully relied upon as the data is still very close to the original.

4.10. M9: Distance Threshold

The paper [12] showed that distance threshold can be a successful mitigation strategy. In general, distance threshold is a mechanism used to control which chunks of data are relevant to a prompt. It measures the similarity between the prompt and the retrieved documents using methods such as cosine similarity or L2 norm (Euclidean distance). The similarity is then checked against a threshold (set by the developer), and only those chunks of data that pass the threshold are further used in the RAG process. As a result, unrelated (and potentially sensitive) data is not used in the query, or is used less frequently. This reduces risk. The distance threshold is a trade-off between privacy and utility. [12] has shown that lower thresholds, which ensure that only very similar documents are retrieved, can be harmful to the system performance. Therefore, the most appropriate distance threshold must be determined for each RAG system.

Limitation

As mentioned above, the threshold is a trade-off between privacy and utility. Therefore, for each system, it has to be determined which aspect is more relevant. Furthermore, it must be tested how well this mitigation strategy helps in one's own RAG system. Therefore, it is clear that this strategy can be a supplement to make the RAG system more secure, but cannot be taken as the only mitigation strategy.

4.10.1. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	The distance threshold helps to reduce data leakage by filtering out less relevant data. However, sensitive data with high relevance to the prompt may still be leaked. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but should never be used as a primary mitigation strategy.
R3: Membership Inference Attack (MIA) (Section 3.4.)	With the distance threshold less relevant documents are filtered out. Data that is not further used in the RAG process cannot be determined of whether it is in the retrieval database or not. It is important to note, however, that while this can be a helpful additional mitigation strategy, it should never be used as a primary mitigation strategy because the documents can still be retrieved simply by providing more specific prompts.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	The distance threshold helps to reduce data disclosure by filtering out less relevant data. However, sensitive data with high relevance may still be disclosed. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but should never be used as a primary mitigation strategy.
R7: Knowledge Corruption Attack (Section 3.8.)	With the distance threshold less relevant documents are filtered out. This makes it less likely that tampered data is among those documents. However, it could lead to a more biased opinion because fewer documents are used. Therefore, it is important to note that it can be a helpful additional mitigation strategy, but the downsides should also be taken into consideration.

4.11. M10: Summarization

Summarization as a mitigation strategy is used to aggregate the retrieved information to reduce its exposure by not including information unrelated to the user prompt in the summary [12]. The mitigation is achieved by using less data, and therefore less sensitive data, for further processing in the RAG pipeline. Of course, data that is not included in further processing cannot be misused in attacks. The summarization is done by a separate LLM that is, via system instruction, advised to create the summarization based on the user prompt and the retrieved information. The summarization is then used instead of the retrieved information and passed to the generator in the final step to generate the response.

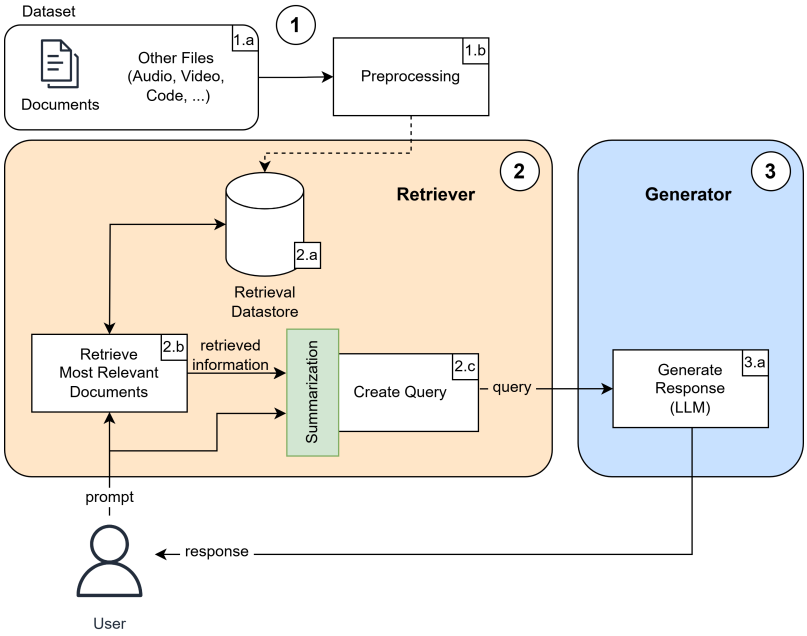

Figure 18: Summarization in RAG pipeline - own presentment

To better understand what a summarization task might look like, here is an example of a system instruction:

> *Given the following question and context, extract any part of the context **AS IS** that is relevant to answer the question. If none of the context is relevant return NO_OUTPUT.*
>
> *> Question: {Prompt}*
> *> Context:*
> *> > >*
> *{Retrieved Context}*
> *> > >*
> *Extracted relevant parts:*
>
> — *[12, p.14]*

Since this mitigation strategy introduces another LLM, it can also be used to adapt other mitigation strategies, such as Reinforce System Instructions (see Section 4.6.). For example, you could instruct the LLM not to include sensitive data in the aggregation, such as email addresses or phone numbers. This makes the system more resilient to attacks because the sensitive data is not included in the further RAG process.

4.11.1. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Summarization as a mitigation technique has a positive effect on mitigating untargeted attacks by reducing the overall amount of data passed to the generator for response generation. Since less data (including less sensitive data) is passed to the generator, the risk of data leakage is reduced. However, this strategy does not really mitigate targeted attacks. If a specific piece of information was requested, then by definition that piece of information should be part of the summary in order for the system to provide an accurate response. This information can therefore potentially be leaked.
R3: Membership Inference Attack (MIA) (Section 3.4.)	Summarization can help reduce the likelihood of successful MIA attacks because the summarized data may no longer have the same structure, making matching with existing data more difficult and less accurate. Therefore, the ability of an attacker being able to ask whether or not given data is in the retrieval dataset is successfully mitigated.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	Summarization can help to reduce the amount of retrieval data (including sensitive data) that is disclosed to an LLM provider during prompting, because it removes data that is unrelated to the user prompt before handing it over to the generator for generating the response. In general, however, it is important to note that while it can be a helpful additional mitigation strategy, it should never be used as a primary mitigation strategy.

4.12. M11: Re-Ranking

Re-Ranking as a mitigation strategy, as introduced in [12] and further described in [51], [52], attempts to mitigate the risk of exposing sensitive data by using only the most relevant documents to generate responses. This can be achieved, for example, by using another LLM (called a re-ranker [51], [12]) to evaluate the relevance of the retrieved data based on the user prompt. The re-ranker (or other system) takes only the most relevant documents. Less relevant documents are filtered out and not passed on to the further RAG process, thus using less data (including less sensitive data) that can potentially be misused in attacks. Another advantage is, that re-ranking generally improves the relevance and quality of the generated text. In summary, this approach can help mitigate privacy risks by focusing on relevant information and reducing the likelihood of revealing irrelevant but sensitive data.

However (as one might already expect), it is important to note that re-ranking alone is not a good mitigation strategy. [12]'s tests have shown that the mitigation effect in their experiments is only small. For better results, it is advisable to combine this mitigation strategy with other strategies such as Reinforce System Instructions (see also Section 4.6.).

Since re-ranking does not seem to be a major mitigation strategy, one might ask, is it still worth implementing? This question must to be answered and experienced for each system individually. But often it is not just a security question. Would the system benefit from re-ranking besides security? It seems that re-ranking is most useful when combined with other mitigation strategies, such as reinforcing system instructions, and used not only for security reasons but also for better clustering [52]. [51] shows how re-ranking works and the different options that can be used. Several papers [12], [51], [52] show and confirm that re-ranking helps to get better answers, but the impact on security mitigation is controversial.

4.12.1. Deep Dive

How does a re-ranker work and how does it help?
A re-ranker is a LLM specialized in prioritizing and ordering retrieved documents. Re-rankers are widely used and established in Natural Language Understanding (NLU) systems, but it also helps in the context of RAG. The study "A Re-Ranker Scheme for Integrating Large Scale NLU Models" [51] gives a deep dive into re-ranker and how they work. In simple terms a re-ranker is a prioritizer. Through prioritizing it is possible to prefer more specific documents over less specific ones which on top may also include unrelated sensitive information. In general, a re-ranker is an extra step to ensure that unimportant data is not further used in the RAG process and therefore also excludes sensitive data that is not related to the prompt.

Below is an illustration of the adjusted base pipeline including the re-ranking step.

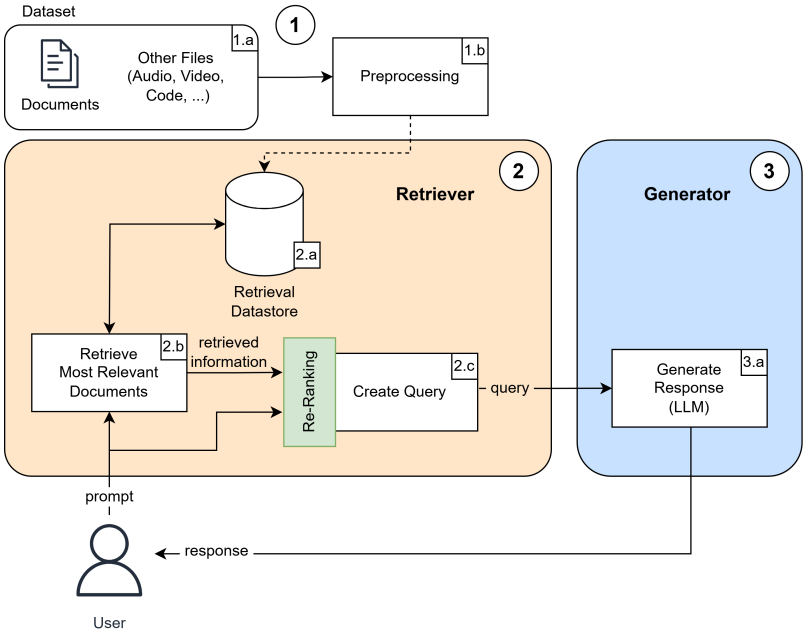

Figure 19: Re-Ranking in RAG Pipeline - own presentment

4.12.2. Related Risk

Risk	Description
R1: Retrieval Data Leakage (Section 3.2.)	Re-Ranking can help reduce the amount of retrieval data that is unrelated to the user's prompt and may contain sensitive information. Irrelevant data, including irrelevant sensitive data, is removed and not further used in the RAG process. This reduces the likelihood of data exposure. Note, however, that this mitigation strategy should only be taken as a further step of mitigation, not as a main step, as it only mitigates slightly.
R5: Retrieval Data Disclosure during prompting (Section 3.6.)	Re-Ranking can help reduce the amount of exposed data by using only the most relevant data to generate the response. Irrelevant data, including irrelevant sensitive data, is removed and not further used in the RAG process. This reduces the likelihood of data exposure. Note, however, that this mitigation strategy should only be taken as a further step of mitigation, not as the main step, as it only mitigates slightly.

Part III
Conclusion & Outlook

1. Conclusion

The objective of this thesis was to evaluate the risks and appropriate mitigation strategies associated with RAG systems. These risks and mitigation strategies should have been analyzed, linked and presented in an appropriate form to facilitate the process of evaluating RAG systems. The information for this thesis was to be obtained primarily from a literature review. In addition, it was recommended that interviews be conducted with people involved in the development of RAG systems to determine their needs and the practical benefits of this work.

An extensive literature review was conducted and three expert interviews were carried out with individuals from three different companies. The information from both the literature review and the expert interviews were combined to create a landscape and decision bubbles to simplify the process of evaluating RAG systems. In addition, each risk and mitigation strategy was described in a separate chapter. For each risk, a general description, who needs to care about the risk, appropriate mitigation strategies with a classification of the degree of utilization and, if helpful, a deep dive is provided. Furthermore, each mitigation strategy is described in its own chapter and the corresponding risks that can be mitigated with the corresponding strategy are listed.

An optional part of this thesis was the creation of a prototype. To better understand how RAG systems work, we created a small Proof of Concept (PoC). The PoC included a web interface with the possibility to upload own documents, which were vectorized and used in the retrieval process to generate meaningful answers. For the response generation process we used GPT-3.5 from OpenAI. Since it was important for our learning experience, but had no technical impact, we did not consider this PoC further in the paper.

As a bonus goal for this work, publication in some form was defined as desirable. Towards the end of the thesis we started to prepare a publication together with Marco Lehmann and Christoph Landolt. This work will be continued after the completion of this thesis.

Considering all the steps mentioned above, the goals of this project, including the optional and bonus goals, have been achieved.

2. Outlook

The following are three suggestions for how this work should and could be taken forward.

As RAG is a relatively new topic, this area, including the identified risks and mitigation strategies, may evolve quite rapidly. Therefore, it is suggested that the landscape and decision bubbles, as well as the descriptions, be adapted and expanded to reflect the latest knowledge.

To make the framework more accessible to a wider range of users, one idea during the thesis was to create a web tool to make it easier to find risks and appropriate mitigation strategies based on one's own implementation. Another idea was to incorporate the findings into existing frameworks such as the OWASP Top Ten. Both ideas can broaden the range of users who can benefit from this work and are therefore proposed as further steps.

Last but not least, as already mentioned, the publication of this work in some form is intended and will be pursued after the completion of this thesis. Stay tuned.

Part IV
Appendix

1. Bibliography

[1] H. Naveed *et al.*, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.

[2] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models." 2020. [Online]. Available: https://arxiv.org/abs/2001.08361

[3] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models." 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[4] N. Lambert, L. Castricato, L. von Werra, and A. Havrilla, "Illustrating Reinforcement Learning from Human Feedback (RLHF)," *Hugging Face Blog*, 2022.

[5] P. G. Sessa *et al.*, "BOND: Aligning LLMs with Best-of-N Distillation." 2024. [Online]. Available: https://arxiv.org/abs/2407.14622

[6] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards Mitigating LLM Hallucination via Self Reflection," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1827–1843. doi: 10.18653/v1/2023.findings-emnlp.123.

[7] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[8] Maya Anderson, Guy Amit, and Abigail Goldsteen, "Is My Data in Your Retrieval Database? Membership Inference Attacks Against Retrieval Augmented Generation." 2024. [Online]. Available: https://arxiv.org/abs/2405.20446

[9] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2402.07867

[10] Z. Chen *et al.*, "Black-Box Opinion Manipulation Attacks to Retrieval-Augmented Generation of Large Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2407.13757

[11] Y.-H. Huang, Y. Tsai, H. Hsiao, H.-Y. Lin, and S.-D. Lin, "Transferable Embedding Inversion Attack: Uncovering Privacy Risks in Text Embeddings without Model Queries." 2024. [Online]. Available: https://arxiv.org/abs/2406.10280

[12] S. Zeng *et al.*, "The Good and The Bad: Exploring Privacy Issues in Retrieval-Augmented Generation (RAG)." 2024. [Online]. Available: https://arxiv.org/abs/2402.16893

[13] J. X. Morris, V. Kuleshov, V. Shmatikov, and A. M. Rush, "Text Embeddings Reveal (Almost) As Much As Text." 2023. [Online]. Available: https://arxiv.org/abs/2310.06816

[14] G. Deng, Y. Liu, K. Wang, Y. Li, T. Zhang, and Y. Liu, "Pandora: Jailbreak GPTs by Retrieval Augmented Generation Poisoning." 2024. [Online]. Available: https://arxiv.org/abs/2402.08416

[15] Z. Wang, J. Liu, S. Zhang, and Y. Yang, "Poisoned LangChain: Jailbreak LLMs by LangChain." 2024. [Online]. Available: https://arxiv.org/abs/2406.18122

[16] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek, "A Comprehensive Study of Jailbreak Attack versus Defense for Large Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2402.13457

[17] K. Hu *et al.*, "Efficient LLM Jailbreak via Adaptive Dense-to-sparse Constrained Optimization." 2024. [Online]. Available: https://arxiv.org/abs/2405.09113

[18] Y. Zeng, Y. Wu, X. Zhang, H. Wang, and Q. Wu, "AutoDefense: Multi-Agent LLM Defense against Jailbreak Attacks." 2024. [Online]. Available: https://arxiv.org/abs/2403.04783

[19] Community, "OWASP Top 10 for LLM Applications 2025." 2024. [Online]. Available: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

[20] B. Yan *et al.*, "On Protecting the Data Privacy of Large Language Models (LLMs): A Survey." 2024. [Online]. Available: https://arxiv.org/abs/2403.05156

[21] M. Kang, N. M. Gürel, N. Yu, D. Song, and B. Li, "C-RAG: Certified Generation Risks for Retrieval-Augmented Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2402.03181

[22] Laurie E. Locascio and Gina M. Raimondo, "NIST CSF 2.0." 2024. [Online]. Available: https://doi.org/10.6028/NIST.SP.1299

[23] Community, "OWASP Top 10." [Online]. Available: https://owasp.org/

[24] Y. Huang and J. Huang, "A Survey on Retrieval-Augmented Text Generation for Large Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2404.10981

[25] Private AI, "Unlocking the Power of Retrieval Augmented Generation with Added Privacy: A Comprehensive Guide." Accessed: Sep. 20, 2024. [Online]. Available: https://www.private-ai.com/en/2024/05/23/rag-privacy-guide/

[26] Jon Gitlin @ Merge, "5 benefits of retrieval-augmented generation (RAG)." Accessed: Sep. 20, 2024. [Online]. Available: https://www.merge.dev/blog/rag-benefits

[27] P. Zhao *et al.*, "Retrieval-Augmented Generation for AI-Generated Content: A Survey." 2024. [Online]. Available: https://arxiv.org/abs/2402.19473

[28] CloudCraft, "What is RAG (Retrieval Augmented Generation)?." Accessed: Sep. 20, 2024. [Online]. Available: https://www.cloudraft.io/what-is/retrieval-augmented-generation

[29] Sandi Besen, "The Practical Limitations and Advantages of Retrieval Augmented Generation (RAG)." Accessed: Sep. 20, 2024. [Online]. Available: https://towardsdatascience.com/the-limitations-and-advantages-of-retrieval-augmented-generation-rag-9ec9b4ae3729

[30] M. Besta *et al.*, "Multi-Head RAG: Solving Multi-Aspect Problems with LLMs." 2024. [Online]. Available: https://arxiv.org/abs/2406.05085

[31] D. Edge *et al.*, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization." 2024. [Online]. Available: https://arxiv.org/abs/2404.16130

[32] F. Wang, X. Wan, R. Sun, J. Chen, and S. Ö. Arık, "Astute RAG: Overcoming Imperfect Retrieval Augmentation and Knowledge Conflicts for Large Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2410.07176

[33] C. Niu *et al.*, "RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models." 2024. [Online]. Available: https://arxiv.org/abs/2401.00396

[34] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems." 2024. [Online]. Available: https://arxiv.org/abs/2311.09476

[35] P. Laban, A. R. Fabbri, C. Xiong, and C.-S. Wu, "Summary of a Haystack: A Challenge to Long-Context LLMs and RAG Systems." 2024. [Online]. Available: https://arxiv.org/abs/2407.01370

[36] "LLM01:2025 Prompt Injection." Accessed: Dec. 06, 2024. [Online]. Available: https://genai.owasp.org/llmrisk/llm01-prompt-injection/

[37] "LLM08:2025 Vector and Embedding Weaknesses." Accessed: Dec. 06, 2024. [Online]. Available: https://genai.owasp.org/llmrisk/llm082025-vector-and-embedding-weaknesses/

[38] Aparna Dhinakaran and Evan Jolley, "The Needle In a Haystack Test: Evaluating the Performance of LLM RAG Systems." Accessed: Dec. 06, 2024. [Online]. Available: https://arize.com/blog-course/the-needle-in-a-haystack-test-evaluating-the-performance-of-llm-rag-systems/

[39] Shivika K Bisen, "Choosing the Right Embedding Model for RAG in Generative AI." Accessed: Oct. 04, 2024. [Online]. Available: https://medium.com/bright-ai/choosing-the-right-embedding-for-rag-in-generative-ai-applications-8cf5b36472e1

[40] N. Carlini *et al.*, "Poisoning Web-Scale Training Datasets is Practical." 2024. [Online]. Available: https://arxiv.org/abs/2302.10149

[41] Protecto, "Pseudonymization vs Anonymization: Key Differences, Benefits, & Examples." Accessed: Oct. 25, 2024. [Online]. Available: https://www.protecto.ai/blog/pseudonymization-vs-anonymization

[42] Università della Svizzera italiana and University of Neuchâtel, "Data anonymization." Accessed: Nov. 01, 2024. [Online]. Available: https://www.epfl.ch/campus/services/data-protection/in-practice/privacy-in-research/data-anonymization/

[43] Article 29 Data Protection Working Party, "Opinion 05/2014 on Anonymisation Techniques." Apr. 10, 2014. Accessed: Nov. 01, 2024. [Online]. Available: https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf

[44] S. Zeng *et al.*, "Mitigating the Privacy Issues in Retrieval-Augmented Generation (RAG) via Pure Synthetic Data." 2024. [Online]. Available: https://arxiv.org/abs/2406.14773

[45] Robert Riemann, "Synthetic Data." Accessed: Oct. 16, 2024. [Online]. Available: https://www.edps.europa.eu/press-publications/publications/techsonar/synthetic-data_en#:~:text=Synthetic%20data%20is%20artificial%20data,undergoing%20the%20same%20statistical%20analysis.

[46] Assaf Namer, Robert Lagerström, Gopinath Balakrishnan, and Brandon Maltzman, "Retrieval-Augmented Generation (RAG) Classification and Access Control." 2024. [Online]. Available: https://www.tdcommons.org/cgi/viewcontent.cgi?article=8548&context=dpubs_series

[47] azure, "Security filters for trimming results in Azure AI Search." Accessed: Oct. 31, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/search/search-security-trimming-for-azure-search

[48] Shahzad Asghar, "Jail breaking RAG Applications: A Deep Dive with Real-World Examples." Accessed: Oct. 24, 2024. [Online]. Available: https://shahzadasghar.medium.com/jailbreaking-rag-applications-a-deep-dive-with-real-world-examples-74339a52cc49

[49] Justin Macorin, "Input validation in LLM-based applications." Accessed: Oct. 29, 2024. [Online]. Available: https://promptdesk.ai/articles/input-validation-in-llm-based-applications

[50] F. Cuconasu *et al.*, "The Power of Noise: Redefining Retrieval for RAG Systems," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, in SIGIR 2024. ACM, Jul. 2024, pp. 719–729. doi: 10.1145/3626772.3657834.

[51] C. Su, R. Gupta, S. Ananthakrishnan, and S. Matsoukas, "A Re-Ranker Scheme For Integrating Large Scale NLU Models," in *2018 IEEE Spoken Language Technology Workshop (SLT)*, 2018, pp. 670–676. doi: 10.1109/SLT.2018.8639519.

[52] Marco Alessio, Guglielmo Faggioli, Nicola Ferro, Franco Maria Nardini, and Raffaele Pereg, “Improving RAG Systems via Sentence Clustering and Reordering.” 2024. [Online]. Available: https://ceur-ws.org/Vol-3784/paper4.pdf

2. List of Figures

3. Credits

- Icons: Figures in this paper include icons from Flaticon.com. Authors of the icons are the following: Freepick, YesternightSupply, kliwir art.
- Wikipedia Logo in Figure 12, Source: Wikipedia-Logo, Wikimedia Commons.

4. Use of AI Tools

Throughout the work, the following AI tools were used for the reasons/work listed below.

- ChatGPT: ChatGPT has been used for inspiration and to critically examine statements made in this paper. It has also been used infrequently to help rewrite text, especially in the introductory chapters.
- PerplexityAI: PerplexityAI has been used to help with research topics.
- DeepL: DeepL has been used for rewriting text passages and to refine the text.