Design and Evaluation of a Sovereign Retrieval-Augmented Generation System for Knowledge Retrieval in Cybersecurity and Artificial Intelligence Governance

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Cyber Security
July 2026
Ugho Raherisoanjato

Supervisors:
Tahir Mohammad (University of Turku)
Naeemur Rahman (University of Turku)

UNIVERSITY OF TURKU
Department of Computing

Ugho Raherisoanjato: Design and Evaluation of a Sovereign Retrieval-Augmented Generation System for Knowledge Retrieval in Cybersecurity and Artificial Intelligence Governance

Master of Science (Tech) Thesis, 63 p., 4 app. p.
Cyber Security
July 2026

Governance, Risk and Compliance (GRC) knowledge management is complex, document heavy, and increasingly burdened by regulatory pressure from frameworks such as the GDPR, NIS2, and the EU AI Act. Existing tools fall short. Generic Large Language Model (LLM) assistants lack domain focus and reliable source citation, while enterprise GRC platforms require budget and integration effort that smaller organizations cannot afford. This thesis addresses this gap by designing, implementing, and evaluating a sovereign Retrieval-Augmented Generation (RAG) system for GRC knowledge retrieval in the cybersecurity and AI governance domains.

Following a Design Science Research methodology, the system was built around a double vector search retrieval mechanism, a source labelling pipeline enabling inline, verifiable citations, and a self-hosted PostgreSQL database using the pgvector extension. Sovereignty was treated as a core design principle, leading to the selection of Mistral AI as the LLM provider for its European data residency, and to an architecture supporting future migration to a fully local deployment. The resulting prototype supports two use cases, conversational knowledge retrieval and document drafting, both grounded in the same retrieval pipeline.

The system was demonstrated to a panel of GRC consultants, who responded positively to the grounding and citation mechanism, identifying it as central to building trust in real consultancy work. Document drafting was received more unevenly. Evaluation shows that grounding and access control were enforced structurally, while full data sovereignty was not achieved due to reliance on Mistral's hosted API. This work contributes a working sovereign RAG architecture for the GRC domain, its application to two real use cases, and a reasoned positioning against existing GRC tools.

Keywords: Retrieval-Augmented Generation, RAG, Cybersecurity, GRC, Artificial Intelligence, AI, Governance, Large Language Models, LLM

Contents

List of Figures

List of acronyms

AI Artificial Intelligence

API Application Programming Interface

DSR Design Science Research

GRC Governance, Risk, and Compliance

LLM Large Language Model

NLP Natural Language Processing

OCR Optical Character Recognition

RAG Retrieval-Augmented Generation

REST Representational State Transfer

SaaS Software as a Service

UI User Interface

1 Introduction

1.1 Context and Motivation

Over the past decade, technology has been evolving rapidly, leading to the rise of a whole new industry built around AI. Nowadays, AI has integrated into almost every area of human activity, and is especially transforming how organizations handle knowledge-intensive tasks. GRC is a domain that shares these same characteristics. It is complex, document-heavy and requires expert interpretation. At the same time, European organizations are put under regulatory pressure with the arrival of new regulations and directives, covering many different fields, from data protection and cybersecurity to AI governance. A promising approach to this problem can be explored through RAG-based systems, which offer solutions via grounded answers, source traceability and domain-focused retrieval. This project was conceived in the context of an internship that aimed to explore this landscape and develop a new solution that would be later integrated into the existing workflows of GRC consultants.

1.2 Problem Statement

Existing tools are either too generic or too expensive and closed. On the one hand, generic LLM tools cannot be trusted enough for legal interpretation. Because they are not focused enough, they are prone to hallucinations, and they often lack pre-

cise source citations. On the other hand, large enterprise GRC platforms require dedicated teams, long integration processes and significant budget. Smaller organizations cannot afford such tools, which usually come with an excessive amount of functionalities that they do not even need.

There is currently no accessible, transparent, source-grounded AI tool designed specifically for GRC knowledge retrieval in the field of cybersecurity, AI and data usage. This represents both a technological and commercial gap, and motivates the design of an accessible, domain-specific RAG-based assistant for GRC knowledge retrieval.

1.3 Research Questions

The identified gap calls for a system that is both securely designed and practically accessible to smaller organizations. Therefore, this thesis aims to investigate the following research questions:

RQ1. *How can a RAG-based system be designed to support GRC knowledge retrieval in a secure, grounded, and source-transparent manner, with a focus on sovereignty?*

RQ2. *What architectural and design choices are required to ensure grounding, source traceability, and security in a RAG-based GRC assistant?*

RQ3. *How can sovereignty considerations influence the choice of models and deployment strategy?*

1.4 Research Objectives

The following objectives define the concrete steps undertaken to address the research questions above:

RO1. Review the existing landscape of GRC tools and RAG-based systems to identify gaps and positioning opportunities.

RO2. Define the requirements for a secure, grounded, and source-transparent RAG-based GRC assistant.

RO3. Design and implement a prototype system addressing the identified requirements, with a focus on retrieval quality, grounding, and sovereignty.

RO4. Assess the system against the defined requirements and discuss limitations and directions for future work.

1.5 Scope and Limitations

This thesis focuses on the design and prototyping of an RAG-based assistant for GRC knowledge retrieval in a European regulatory context, specifically in the cybersecurity, AI and data regulation domains. It mainly covers two use cases: knowledge retrieval and document drafting and editing.

Several limitations should be acknowledged. First, the system was built and demonstrated to a panel of GRC consultants, but no structured user studies or performance benchmarks were conducted. Moreover, the system is currently built with a limited knowledge base, and makes no claim about the completeness of regulatory coverage. Furthermore, the project remains a research prototype and was not fully deployed and tested in production. Finally, even though sovereignty was a guiding design principle, the primary implementation relies on the Mistral API rather than a fully local deployment, which means that absolute data sovereignty cannot be guaranteed in the current state of the project. This choice is developed in Chapter 5 and these limitations represent natural directions for future work and are discussed further in Chapter 7.

1.6 Structure of the Thesis

This thesis is structured as follows. Chapter 2 introduces background concepts. Chapter 3 reviews the current state of the art. Chapter 4 presents the system architecture and design decisions. Chapter 5 details the actual implementation. Chapter 6 evaluates the work against the original requirements and brings further discussions. Chapter 7 outlines directions for future work and concludes this thesis.

2 Background

2.1 Retrieval-Augmented Generation

2.1.1 Definition and Core Principles

An LLM is a specific AI model that is capable of producing answers to questions asked in natural language, thanks to NLP. After they have been trained on large amounts of data, they generate human-like language by predicting sequences of words [1]. Nowadays, powerful LLMs are able to treat most subjects in many different languages. However, due to the large variety of the training data, they only have a broad "knowledge" and understanding of each topic, meaning that they can lack accuracy or even hallucinate because the context is not focused enough. Hallucination is a common flaw of this technology, it refers to the fact that LLMs can generate statistically plausible text, making perfect sense in natural language, but giving false or misleading information [2].

This is where Retrieval Augmented Generation comes in. RAG addresses this limitation by augmenting the context of a LLM with relevant documentation extracted from an external knowledge base, instead of solely relying on the information gathered during its training phase [3]. Specifically, when a user submits a query, the system first retrieves the most relevant documents from the knowledge base and then injects them into the model's context alongside the query before generation. This ensures the model's response is grounded in specific, verifiable sources rather

than general training data. The next section discusses different document retrieval techniques.

2.1.2 Document Processing

To enable semantic retrieval, documents must first be transformed into a numerical representation called an embedding, which is a vector that captures the semantic meaning of the text. These embeddings are stored in a vector database, which allows efficient similarity search at retrieval time. Before embedding, documents need to be split into smaller units, called chunks. Each chunk of a document essentially contains a small portion of text with a single coherent meaning. This granularity allows the retrieval system to identify the most relevant portions of a document in response to a given query, thereby avoiding the need to retrieve entire documents.

The simplest and most straightforward technique is fixed-size chunking. It consists of splitting text into chunks of a predetermined size. While it does not require a lot of effort to implement, it does not follow the structure of the text, meaning that it can cut in the middle of sentences or even words. A common solution is overlapping, where a part of the end of one chunk is copied at the beginning of the following chunk to preserve context.

Another approach is recursive chunking. The idea is to split the text based on a list of separators (e.g., newlines) with a given hierarchy. Instead of defining a fixed-size for all chunks, a maximum size is set. The algorithm starts by splitting the text on the separator of highest priority, and tests each resulting chunk against the maximum size. Each chunk that exceeds this limitation will be split again, on the next priority separator. This process is repeated until all chunks respect the maximum size (or all separators have been used). This method allows to preserve the document structure better and fixes the issues of fixed-size chunking while avoiding the redundancy introduced with overlapping.

Document-based chunking adapts to the structure of the document. Instead of relying on generic separators like regular recursive chunking does, it uses the specific separators found in the document. For example, documents formatted in Markdown are split on headers (i.e. #, ##, ...). The process is similar to recursive chunking, but ensures that the actual structure of the document is respected throughout.

Once all chunks and their corresponding embeddings have been generated, they are stored together in the vector database, which represents the knowledge base. This is a crucial phase in the RAG pipeline because the quality of the retrieval directly influences the quality of the generation. This depends directly on how well the documents have been processed and indexed [4].

2.1.3 Retrieval Mechanisms

Once the knowledge base is built, a retrieval mechanism can search through it to pull the most relevant chunks for context augmentation.

The main retrieval method used in this work is vector search. Each query is embedded using the same embedding model that was used for document processing. A similarity search is then performed to retrieve the top-k chunks that are mathematically closest to the query vector. Only these specific chunks are fed to the LLM to generate its answer.

While it is highly outperformed by vector search and thus cannot be used as a retrieval mechanism on its own, tag filtering can be added to the process as a complementary pre-filtering step. By filtering the documents based on their metadata before performing the similarity search, the number of candidates is drastically reduced, thus lowering the computational overhead on the database. However, this method requires all documents metadata to be perfectly set, as it ignores tags that do not match exactly. These constraints cannot be overlooked because they risk excluding relevant documents from the LLM context entirely.

Sorting through the whole documents before searching through their chunks is still a strong methodology, but since vector databases and embeddings are already part of the pipeline, they can be leveraged for this step as well. The double vector search requires documents to be embedded as a whole first, and stored alongside their embeddings in a dedicated database. A first vector search can then be performed against all documents, allowing the most relevant documents to be pulled based on the query. The second search can thus be performed on this reduced set of chunks, thereby speeding the process and increasing the relevance of the retrieved knowledge.

These methods can be further improved with small-to-big retrieval, which can be integrated to catch potentially missing information, and improve the quality of the context given to the LLM. Specifically, surrounding chunks are pulled for each retrieved chunk, allowing the context of the information to be better maintained. However, this requires the chunking mechanism to take that into account from the start, and add metadata to each chunk to keep track of their order and location within their document of origin. Some chunking algorithms introduce other methods to keep track of a chunk's context, this is discussed further in the context of this project's implementation in Chapter 5.

2.1.4 Context Augmentation

The core mechanism of RAG consists of augmenting the context of the LLM with retrieved external knowledge. The method used can alter the quality of the answer generation, its grounding and source citation, as mentioned in the next section.

The chunks content is first formatted (typically in Markdown to maximize the LLM understanding), then injected alongside the query in the input context. That is where the system prompt comes into play. Concretely, it defines overall rules and guidelines that the LLM needs to follow in its response process, and describes how it is supposed to handle the data that is passed in its input. In this case, it explains

how the user prompt should be treated and how the knowledge from the external documentation should be used. This is also where grounding is defined, as well as the response format and other rules, such as citations.

Some LLMs are built to handle long context, which can fit entire documents in addition to the query. While this works well to help it keep track of a conversation, it reduces the quality of the information the more it fills up. In the context of this project, the goal is to manipulate many long regulation documents, that would hardly fit into a long-context LLM window. Moreover, research has shown that LLMs perform worse on information placed in the middle of a long context compared to the beginning or end [5]. This is why the focus is put on vector databases, which scale more efficiently with large document collections.

Another, hybrid method that could be considered, is context-aware retrieval. It leverages both the power of vector search and long context. It starts by pulling the most relevant chunks from the knowledge base but, unlike basic vector search, like discussed previously, it will pull the entire documents they come from. However, as mentioned, this would not work with the length and number of documents that are considered in this work.

Beyond retrieval strategy, the size of the injected context is constrained by a fundamental architectural limit of LLMs, the context window, a concept that should not be overlooked when dealing with their input. During a conversation between a user and a LLM, the input window accumulates the injected context and conversation history with each exchange. Therefore, the input context definition process needs to be carefully managed. If the context window fills up too quickly, early information is lost and the quality of the following responses degrades.

2.1.5 Generation and Grounding

As mentioned in the previous section, the system prompt is in charge of specifying to the LLM how to handle all the data from its input context, and how to form its responses. It usually contains a set of rules that define its behavior and the output that is expected.

An important topic that needs to be covered is grounding. In the case of AI assistants, like this project, the model is instructed to base its answers exclusively on the provided context, and to explicitly acknowledge when the answer cannot be found in the documentation instead of hallucinating. This is even more crucial in the context of GRC and any topic in the legal or medical fields, for example [6].

The model can also be instructed to cite sources, and how to do so. Again, this is essential when dealing with official, legal documentation. This builds the core trust, auditability and verifiability of the whole tool. It is important that the document processing is done efficiently, so that it is easy for the LLM to trace back each piece of information that it gives, and to cite it properly.

Ultimately, the system prompt defines the response structure. It can be instructed to prioritize certain formatting options based on the type of question it is replying to (e.g. gap analysis, comparisons). It can also prepare the terrain for the frontend, by specifying Markdown preferences (e.g. avoid nested lists, use of tables and blockquotes for verbatim citations).

The system prompt designed for this project is presented and discussed in detail in Chapter 5.

2.2 Governance, Risk and Compliance

2.2.1 Definition and Scope

GRC is a unified discipline that enables organizations to align their governance strategies, manage risk, and meet their compliance obligations in a coherent and integrated manner [7]. The three concepts are deeply interrelated. Regulatory requirements directly impact internal governance policies, which define the processes through which risk is identified and mitigated. GRC applies across most regulated industries, including finance, healthcare, energy, and critical infrastructure. With the fast digitization of organizations and the emergence of dedicated regulatory frameworks such as the EU AI Act and NIS2, IT and AI governance have become central dimensions of GRC practice.

2.2.2 Key Frameworks and Regulations

This work is positioned in a European context, where organizations need to keep track of both international standards (e.g. ISO 27001, NIST) and legal regulations and directives (e.g. NIS2, GDPR, EU AI Act).

International standards are a set of guidelines which aim to help organizations implement and manage their information systems to reduce security risks. For example, ISO 27001 is an information security standard which specifies the requirements for designing, implementing and maintaining an information security management system [8] and the NIST CSF is designed to help organizations assess and improve their readiness against cybersecurity threats [9]. They represent a solid reference and are globally recognized as the standards in their respective sectors. Organizations regularly go through audits to check and prove their compliance, in order to gain trust and credibility.

Legal regulations and directives address different aspects and are more focused.

For example, the GDPR covers data protection, NIS2 covers network and information security for critical sectors, and the AI Act covers AI system risk classification and obligations. The key difference is that they are legally binding rather than advisory, which means that compliance with these is not a choice but a legal obligation for organizations.

These represent only a subset of the regulatory landscape organizations must navigate in practice. There are dozens of texts to keep track of, making this landscape challenging to navigate and keep up-to-date. In addition to the large volume, many documents overlap or even contradict each other, with different application conditions. Furthermore, most of these texts are regularly re-evaluated and updated, adding more complexity.

2.2.3 Challenges in GRC Knowledge Management

Regulatory texts are written in legal language, not technical language. This means that an IT or security team needs to interpret dense legal information and translate it into concrete technical or organizational actions. This requires expertise in both the law and technology fields, which is often rare, and requires significant time and resources to develop.

Furthermore, staying up to date is not a one-time effort. When a regulation changes, it is up to every organization to identify which of their internal policies are affected and update them accordingly. At a higher level, they also frequently need to perform gap analysis to assess their compliance. Considering the amount of text that needs to be considered, this is a continuous and resource-intensive process [10]. Doing this manually across multiple frameworks simultaneously is tedious, expensive, and prone to errors.

In practice, compliance requires organizations to gather evidence. Policies need to be written, maintained, and traceable back to specific regulatory requirements.

This process generates a large volume of internal documentation that needs to be managed. All of this typically requires dedicated GRC teams, external consultants or specialized software, which represents a significant burden for smaller organizations in particular.

2.3 Large Language Models Security

2.3.1 Threat Landscape

LLMs introduce an attack surface that is fundamentally different from traditional software systems. Typically, a conventional application can validate its inputs against a fixed schema. For example, a form field expects a date, an API endpoint expects a JSON payload of a known shape, and anything that does not conform can be rejected before it reaches the application logic. An LLM-based system has no equivalent boundary. Its entire input is natural language, and natural language can not be validated against fixed schemas. A legitimate query and an attempt to manipulate the model can look syntactically identical, but have completely different intents.

This changes what "securing a system" actually means. Traditional input sanitization focuses on format, rejecting malformed requests, escaping special characters, and enforcing length limits. These checks still matter, but they do not address the core risk, because a perfectly well-formatted sentence can still carry an instruction that the system was never meant to follow. What matters is the semantic content of the input, not strictly how it is presented, and semantic content cannot be filtered using the same techniques used for structured data. There is no equivalent to regular expressions for intent.

This is made worse by the fact that, unlike a conventional application where the code path is fixed and only the data varies, an LLM is itself instructed in natural

language, through the system prompt. Since the same channel carries both the model's instructions and the user's input [11], anything that increases the model's apparent authority over its own behavior, such as a convincingly phrased request to ignore previous instructions, is competing directly with the rules the system was designed to enforce. This is the underlying mechanism behind the threats discussed in the following sections.

2.3.2 Prompt Injection and Related Attacks

Prompt injection can be divided into two subcategories. On the one hand, direct prompt injection consists in users manipulating the model through their own input, at query time. On the other hand, indirect prompt injection refers to malicious content embedded in retrieved documents, that hijacks the behavior of the model.

Direct prompt injection is often used to attempt jailbreaking [12]. Specifically, it relates to bypassing the grounding constraints defined in the system prompt using carefully crafted natural language. Indirect prompt injection [13] is particularly relevant to RAG. A user uploading malicious documents to the knowledge base may be able to manipulate following answers for all users.

2.3.3 RAG-Specific Threats

While prompt injection is the main threat to LLMs, the RAG pipeline introduces new risks specific to it [14].

In systems where users can contribute documents to the knowledge base, such as this project, the RAG pipeline becomes subject to knowledge base poisoning. Corrupted or adversarially crafted documents can be introduced, causing the model to produce misleading answers, grounded in false sources.

It is also possible to have the LLM return misleading content without interfering with the knowledge base itself. Carefully crafted queries may be able to exploit the

vector search mechanism, causing retrieval manipulation.

Lastly, the model could expose content from the knowledge base that a given user should not have access to. Proper access control within the retrieval layer is essential to prevent data leakage.

3 State of the Art

3.1 Overview of Existing Approaches

As mentioned, solutions already exist, but they come with their own individual drawbacks and missing features. From generic AI tools to large enterprise GRC platforms, this section highlights their strengths and weaknesses, and describes where the present work positions itself within this landscape. It is based on a review of vendor websites and product documentation conducted in February 2026. kuvat

3.1.1 General-Purpose AI Assistants

Many AI companies offer access to general LLMs through web platforms that are easy to use for the general public. Chatbots such as Google Gemini, OpenAI ChatGPT, and Anthropic Claude provide users with numerous functionalities. They can generate text, images, videos, and audio, and are able to reason before responding to a request.

They typically have broad knowledge, which makes them fall short for GRC specifically. Since they do not have any focus on the domain, they are not grounded in a controlled knowledge base, making them prone to hallucination on legal content, and struggle to reliably cite specific regulatory sources, making them unsuitable for legal interpretation tasks.

In terms of sovereignty, these solutions are not viable for organizations that

handle sensitive compliance data. They cannot feed sensitive internal compliance documents into a public LLM API without data privacy concerns.

3.1.2 Regulation Trackers and Information Aggregators

A wide range of regulation trackers and information aggregators can be found on the market, and can provide useful regulatory information and help stay up to date with new texts. However, they are purely informational and do not directly help an organization assess its own compliance posture, perform gap analysis, or draft documents. They are reference tools, not assistants, and usually specific to targeted jurisdictions or regulations, rather than cross-framework.

3.1.3 Enterprise GRC Platforms

Several vendors offer enterprise architecture platforms with larger feature sets, where GRC functionality is secondary to broader enterprise management capabilities. These tools come with heavy integration requirements for smaller organizations, and they are typically focused on technical controls rather than actual knowledge retrieval and interpretation. Crucially, these platforms do not explain regulations and directives, they assume that the user already understands them and simply bring clients to compliance by design (integrating those tools within the organization processes makes it compliant itself). Tools such as Ardoq and Kiteworks fall into this category, and therefore do not represent direct competitors to this work.

3.1.4 Direct Competitors

The closest equivalent to this work, currently on the market, is Deloitte RegAI [15]. It targets large enterprises with complex, multi-jurisdictional compliance requirements, and offers functionalities such as document analysis, regulation comparison,

requirement extraction, policy drafting and updating, and gap analysis [15]. Notably, RegAI bases its answers exclusively on the documents provided within context and explicitly cites the paragraph from which information was derived, stating that it does not know the answer when no relevant paragraph is found. This grounding and citation approach is conceptually close to the one adopted in this work, which makes it a strong competitor, but also reinforces the validity of the design direction taken in Chapter 4. However, RegAI's main weakness lies in the fact that it is not available as a SaaS, but as a services-led offering with implementation cycles typically measured in weeks [16]. This limits its accessibility to organizations without dedicated GRC teams and significant budget. Moreover, it targets any industry and jurisdiction rather than focusing on specific domains such as cybersecurity, AI, or data regulation [15].

A second category of competitors gathers AI-native compliance automation platforms, such as Drata, Vanta, and Sprinto. Unlike Deloitte RegAI, these platforms are not designed around regulatory interpretation. Instead, they automate evidence collection for security certifications such as SOC 2 and ISO 27001, by connecting directly to cloud infrastructure, identity providers, and code repositories to retrieve configuration data and continuously monitor controls [17]. Vanta, for instance, is positioned around breadth of integration and fast onboarding, while Drata is generally considered stronger for organizations running multiple frameworks simultaneously, and Sprinto targets engineering-led teams pursuing a faster path to audit readiness [17], [18]. While these tools can help reduce the manual burden of maintaining a certification, they operate on a fundamentally different problem than the one addressed in this thesis. They verify that technical controls are in place and collect evidence of compliance, but they do not explain or interpret regulatory text and do not provide a way to ask an open question about a regulation and receive a grounded, source-traceable answer. In that sense, they are complementary rather

than competing tools, and a GRC consultant could plausibly use both a system such as the one presented here and a platform such as Vanta or Drata within the same engagement, for different parts of the workflow.

Furthermore, none of these solutions are European-native or designed with data sovereignty as a guiding principle, which is a significant consideration for organizations subject to EU regulations. Deloitte RegAI, Drata, Vanta, and Sprinto are all US-based offerings, subject to American jurisdiction, and none of the surveyed comparisons identify data residency or sovereignty as a differentiating factor in their positioning. This further reinforces the gap identified in Section 3.2, not only is there no solution combining regulatory knowledge retrieval with source-grounded generation in the cybersecurity, AI, and data regulation domains specifically, but none of the closest existing tools are designed with the sovereignty constraints that motivate this work.

3.2 Identified Gaps and Positioning

The reviewed landscape reveals three distinct gaps. First, no existing solution combines regulatory knowledge retrieval with source-grounded generation. The tools either answer questions without citations or provide citations without intelligent interpretation. Second, the market is split between accessible but not deep tools and enterprise platforms that are powerful but hard to access, leaving smaller organizations with few viable choices. Third, no solution is designed specifically for the cybersecurity, AI, and data regulation domains within a sovereign European context, which is the main focus here.

This work aims to address these gaps through the design of a RAG-based assistant that is domain-specific, source-transparent, sovereignty-conscious, and accessible without requiring significant infrastructure or budget. The following chapters detail the design decisions and implementation choices that shaped this solution.

4 System Design

4.1 Research Approach and Methodology

This work follows a DSR methodology, as defined by Hevner et al. [19]. DSR is a research paradigm where the primary contribution is an artifact (a system, model, or framework), and knowledge is produced through the process of designing, building, and reflecting on that artifact. It is particularly suitable for applied computer science research, where the goal is to solve a concrete, identified problem rather than to observe or measure an existing phenomenon.

The research conducted in this thesis maps well onto the DSR cycle. The problem was first identified through a review of the existing landscape of GRC tools and RAG-based systems, revealing the gaps described in the previous chapter. Requirements were then derived from this gap analysis and the operational needs of GRC consultants. The artifact, a RAG-based assistant for GRC knowledge retrieval, was designed and implemented as a functional prototype. The system was demonstrated to a panel of GRC consultants, providing initial validation of its practical insight. The design decisions and thought process are described in the following sections.

Evaluation in this work is analytical rather than experimental. No formal user studies or quantitative benchmarks were conducted, which is acknowledged as a limitation in Chapter 7. However, within the DSR framework, demonstration and reasoned argumentation constitute valid forms of evaluation for a prototype artifact

[19].

4.2 Requirements

4.2.1 Functional Requirements

The functional requirements of the system were derived from the identified gaps and from discussions with GRC consultants at the beginning of the internship. They were organized into two phases, based on the priority of features and the fact that requirements evolved as development progressed. Some of these priorities were also revised during implementation, as discussed in Chapter 5.

The first phase focused on core knowledge retrieval capabilities, which represent the foundation of the system. It must be able to answer complex questions about cybersecurity regulations, AI governance, and data protection, based on a curated knowledge base. Responses must be strictly grounded in that knowledge base, which means that the system should explicitly acknowledge when a question falls outside the scope of the available documentation rather than generating an answer from general training data. Every answer must include precise citations, traceable back to specific documents and sections, and users must be able to inspect the cited sources directly from the interface. When the documentation contains contradictory information across different regulations, the system must flag the conflict rather than arbitrarily picking one interpretation. Finally, users must be able to filter the knowledge base by document or geographic scope, in order to restrict answers to a specific regulatory context.

The second phase extended the system with compliance assistance capabilities. The system must be able to perform gap analysis by allowing users to upload their own documents and compare them against relevant regulations to identify compliance gaps. It must also be able to generate and fill in compliance document

templates, and support targeted edits to existing documents. Ultimately, the system should ask clarifying questions when a request is ambiguous and suggest further discussion topics at the end of each answer to assist users in providing the correct information.

4.2.2 Non-Functional Requirements

Beyond functional capabilities, a set of non-functional requirements influenced the reflections and the overall design of the system.

Accessibility was a primary concern from the beginning, in the sense that the system must be usable without prior technical knowledge or infrastructure setup. It is designed as a web-based SaaS, accessible from any browser without requiring integration into existing organizational systems. This directly addresses the gap identified in the state of the art, where existing solutions are either too complex or too expensive for smaller organizations.

Performance requirements were driven by the nature of the use case. Regulatory questions often require processing large volumes of documentation, so the retrieval pipeline must remain efficient regardless of the size of the knowledge base. Response generation should feel fluid to the user, which is why streaming responses were prioritized over waiting for a complete answer to be generated before displaying it.

Scalability was considered at the architectural level. The system must be able to handle a growing number of documents, users, and concurrent sessions without requiring significant architectural changes. This influenced the choice of a self-hosted PostgreSQL database with the `pgvector` extension over third-party vector database services. In this case, all the data stays within a single, controlled environment that scales naturally with the infrastructure.

Sovereignty was treated as a standalone non-functional requirement. The system must process and store all data within European jurisdiction, and must be designed

in a way that allows switching from an API-based model to a fully local deployment later on. This requirement influenced several key design decisions, which are discussed further in this Chapter.

4.2.3 Security and Trust Requirements

Security and trust requirements were particularly important given the sensitive nature of the documents handled by the system. Compliance documentation, internal policies, and legal analysis are confidential by nature, and the system must treat them accordingly.

Access control is a core requirement at every layer of the system. Users must only be able to access their own uploaded documents, and the retrieval pipeline must enforce this at query time, ensuring that a search never returns content belonging to another user.

Grounding is both a functional and a security requirement. By restricting the model to answer exclusively from the provided knowledge base, the system limits the attack surface for prompt injection. A well-grounded model is harder to jailbreak, because it has been explicitly instructed to ignore requests that fall outside its defined scope in the first place. Therefore, the system prompt must enforce strict grounding rules, and the model must be instructed to refuse any request that attempts to bypass them.

Data privacy requirements follow directly from the sovereignty constraint. No user data, uploaded documents, or conversation history should be processed outside of European jurisdiction. API calls must be routed exclusively through providers that guarantee European data residency. Locally, all sensitive data must be stored in the self-hosted database rather than delegated to third-party services.

Finally, trust is a requirement that goes beyond technical controls. For GRC consultants to rely on this tool in their work, every answer must be verifiable, and

they can not blindly trust an AI system. This means citations must be precise enough for a user to locate the original source independently, and the system must never present information without a traceable origin. Here, transparency is what makes trust possible, and the citation and grounding mechanisms described in the functional requirements are what enforce it.

4.3 System Architecture

4.3.1 Overview

The system follows a classic client-server architecture, where a web-based frontend communicates with a Python backend through a REST API. The backend is responsible for all document processing, retrieval, and communication with the LLM API. The frontend handles user interaction, response rendering, and source browsing and viewing.

At a high level, the system operates in two distinct modes. In knowledge retrieval mode, the user submits a query through the chat interface, and the backend embeds the query, searches the knowledge base for relevant chunks, builds the input context, and streams the model's response back to the frontend along with source references. In document drafting mode, the same retrieval pipeline is used, but the output is structured as a formatted, rendered document rather than a conversational answer, and the user can inspect, edit, and download it directly from the interface.

All data, including user accounts, uploaded documents, embeddings, sessions, and messages, are stored in a single self-hosted PostgreSQL database. This design choice keeps the entire system within a controlled environment, with no dependency on external storage services for relational and vector data.

Figure 4.1 presents the activity diagram of the system, illustrating the flow of data from user input to generated response across both modes of operation.

4.3.2 Technology Stack

The technology stack was selected to balance performance, sovereignty, and development efficiency. Each choice was driven by concrete requirements rather than familiarity alone.

The backend is written in Python using FastAPI. Python was the natural choice for this kind of project, given the maturity of its ecosystem for data processing, embedding, and LLM integration. FastAPI was selected for its performance and native support for asynchronous request handling, which is important for streaming responses.

The frontend is built with Next.js, using React and Tailwind CSS. Next.js provides a solid foundation for a web-based SaaS, with server-side rendering capabilities and a clean component model. The choice of Tailwind CSS kept the styling consistent and maintainable without relying on heavy UI frameworks.

For the database, PostgreSQL was used with the `pgvector` extension, which adds native vector similarity search capabilities to a standard relational database. This allowed the entire data model, including user accounts, documents, chunks, embeddings, sessions, and messages, to be managed within a single database. The alternative would have been to use a dedicated third-party vector database service such as Pinecone or Weaviate, but this was not selected for two reasons. First, it would have introduced an external dependency for a core part of the pipeline. Second, and more importantly, it would have required routing sensitive document data through a third-party service, conflicting directly with the sovereignty requirement mentioned previously.

Mistral AI was selected as the LLM provider for several reasons. From a technical point of view, the Mistral API covers all the capabilities required by the system, including OCR for document processing, embedding generation, and chat completion. Its OCR engine handles complex document layouts and tables with high accuracy,

which is important given the formatting of many regulatory documents. From a sovereignty standpoint, Mistral is a French company and all data processed through their platform is hosted on European servers, under European jurisdiction [20]. This makes it a significantly stronger choice than American providers such as OpenAI or Google for an application handling sensitive compliance data. Additionally, Mistral offers open-weight models that can be self-hosted, meaning the system can be migrated to a fully local deployment without changes to the core pipeline.

4.3.3 Database Schema

Figure 4.2 presents the database schema underlying the system. As discussed in the previous section, the entire data model is hosted in a single self-hosted PostgreSQL database using the `pgvector` extension, rather than being split across a relational database and a separate vector store. The schema reflects this choice directly, since embeddings are stored as native columns alongside the relational data they describe, rather than in a dedicated external system.

The `USERS` table holds user accounts and authentication information. It is the anchor point for the access control mechanism discussed in Section 4.2.3, since every document, session, and activity log in the system is ultimately scoped back to a user. User deletion is handled with a `SET NULL` policy on all foreign keys referencing this table, rather than a cascading delete. This means that a user's content (documents, sessions, messages) is not automatically destroyed when their account is removed, but is instead retained as orphaned data, with the reference to its original owner cleared. This decision was made to avoid silently destroying potentially relevant compliance documentation or conversation history as a side effect of account deletion, since the consequences of losing such data are more severe in a GRC context than in a typical consumer application. The exact meaning of a `null user_id` differs depending on the table it appears in, which is noted directly on the schema to

avoid ambiguity during implementation.

The `DOCUMENTS` table stores documents at the whole-document level, supporting the dual representation introduced in Section 4.3.4. Each row stores the document's filename, file path, full Markdown content produced by OCR, and a whole-document embedding (in addition to the chunk-level embeddings stored separately). The status field tracks where a document is in its processing lifecycle, since OCR, chunking, and embedding do not happen instantaneously, and a document is not immediately searchable the moment it is uploaded. The `owner_id` field is nullable, which is what allows the schema to represent both user-uploaded documents, scoped to a specific owner, and documents belonging to the shared curated knowledge base, which have no individual owner. The `checksum` field stores a hash of the document's content, computed at ingestion time, and is used to detect duplicate uploads. Concretely, if a document with an identical checksum already exists in the system, it does not need to be reprocessed through OCR, chunking, and embedding again, since its existing chunks and embeddings remain valid. This avoids redundant API calls and storage for documents that are uploaded more than once, which is a realistic scenario given that the same regulatory text or internal policy document may be referenced or re-uploaded across different sessions or users.

The `DOC_CHUNKS` table stores the chunk-level representation of each document, as produced by the document-based chunking strategy discussed in Chapter 2. Each chunk stores its own content and embedding, along with metadata needed for citation and ordering: the page numbers it spans, its index within the parent document, and an estimated token count, the last of which matters directly for managing the context window constraints discussed in Section 2.1.4. Each chunk references its parent document through `doc_id`, which is what allows the second step of the double vector search, described in Section 4.3.4, to restrict its search to the chunks belonging to a specific set of documents.

The `SESSIONS` and `MESSAGES` tables together store conversation history. A session represents a single conversation thread, with a title and a `user_id` that scopes it to its owner, while each message belongs to a session and stores its role (user or assistant), content, and, where relevant, the sources cited in that message, stored as structured data in the sources field. This is what allows the citation mechanism described in Section 4.3.5 to persist beyond a single response, since a previously cited source remains linked to its message and can still be inspected later in the conversation. Sessions support soft deletion through a `deleted_at` field, rather than being immediately removed from the database, which keeps the data recoverable and consistent with the conservative deletion approach already used for user accounts.

Finally, the schema includes an `ACTIVITY_LOGS` table, intended to record actions tied to a session, document, or user, along with metadata and a token count. At the time of writing, this table reflects an intended direction rather than an implemented feature. Exactly what should be logged, and at what granularity, was still being defined, and is therefore deliberately left flexible at the schema level, using a `JSONB` metadata field rather than a fixed set of columns. This is included in the schema to acknowledge that auditability and usage tracking were considered from early in the design process, even though the corresponding functionality had not yet been built within the scope of this work.

4.3.4 Retrieval Pipeline

The retrieval pipeline operates in two distinct phases. An offline document processing phase that first builds the knowledge base, and an online query phase that retrieves relevant content at request time.

When a document is added to the system, it is first processed through Mistral's OCR engine, which converts it into clean Markdown content regardless of the original format (although PDF is preferred). This step is important for regu-

latory documents, which often contain complex layouts, tables, and multi-column formatting that would be lost with a naive text extraction approach. The resulting Markdown is then processed in two parallel operations. On one hand, it is split into chunks using a document-based chunking strategy, which respects the heading structure of the Markdown output. Each chunk is then individually embedded and stored in the chunks database table alongside its metadata. On the other hand, the full document content is embedded as a whole and stored in the documents database table. This dual representation enables the double vector search described below.

When a user submits a query, it is embedded using the same Mistral embedding model. Retrieval then proceeds in two steps. A first vector search is performed against the document embeddings, identifying the most relevant documents based on the query. A second vector search is then performed exclusively within the chunks belonging to those selected documents, pulling the most relevant portions of the text. This approach significantly reduces the search space for the second query, improving both retrieval speed and relevance compared to searching all chunks directly.

The retrieved chunks are then labelled with their source information (document name, section, and a unique source identifier) before being assembled into the input context. This labelling step makes the citation mechanism possible, with each chunk carrying its origin metadata into the context, it allows the model to reference it precisely in its response.

Figure 4.1 illustrates this full pipeline, from document ingestion to user output.

4.3.5 Generation and Citation Layer

Once the input context has been assembled, it is passed to the LLM alongside the system prompt and the conversation history. The system prompt is the central mechanism through which the model's behavior is defined and constrained. It specifies the tone, the response format, the grounding rules, and the citation format,

among other things.

Grounding is enforced by explicitly instructing the model to base its answers exclusively on the content provided in the input context. If the knowledge base does not contain sufficient information to answer a question, the model is instructed to say so clearly rather than generating an answer from its general training data. This is particularly important in an GRC context, where a confident but incorrect answer about a regulatory requirement could have real consequences.

The citation mechanism is built around a source labelling system introduced during the chunks labelling step of the retrieval pipeline. Each chunk injected into the context is prefixed with a unique source identifier (S1, S2, S3, etc.) and a human-readable citation label derived from the document metadata (for example, "NIS2, Article 21"). The model is instructed to cite sources inline by integrating the citation label naturally into the text, followed by the source identifier in brackets. Concretely, a response might contain: "According to NIS2, Article 21 [S1], organizations must implement appropriate technical measures." This format makes every claim traceable without interrupting the reading flow.

On the frontend, each source identifier is rendered as a clickable reference. When a user clicks on a source, the corresponding document is opened in the source panel directly at the relevant page, allowing immediate verification of the cited content. This closes the trust loop: the model cites its sources, and the user can inspect them without leaving the interface.

4.4 Sovereignty and Deployment Considerations

Sovereignty was treated as a main, crucial requirement throughout this project, given the nature of the data the system handles. Compliance documentation, internal policies, and legal analyses are sensitive by definition, and the organizations that produce them are subject to European data protection regulations. Routing this

data through non-European infrastructure would not only conflict with the spirit of those regulations, but would also undermine the trust that the tool is designed to build.

The choice of Mistral AI as the LLM provider was the most significant sovereignty decision made in this project. As a French company, Mistral processes all API traffic on European servers, under European jurisdiction. This contrasts with the major American providers (OpenAI, Google, Anthropic), whose infrastructure is primarily based in the United States and subject to American law, including legislation that can compel disclosure of data stored on American servers regardless of where the customer is located [21]. For an application handling sensitive compliance data in a European regulatory context, this distinction is not trivial.

Using a self-hosted PostgreSQL database reinforces this. All user data, uploaded documents, chunk embeddings, sessions, and messages remain within the organization's own infrastructure. No data is delegated to third-party cloud storage or vector database services. Combined with the Mistral API's European data residency, this means the entire data flow of the system stays within European jurisdiction under the current deployment model.

The architecture was also designed with full deployment flexibility in mind. Mistral offers open-weight models that can be self-hosted on local hardware, and the system's API integration layer was written to make switching between the cloud API and a local model as straightforward as possible. Concretely, small models can run on devices with limited hardware (mostly memory), making local deployment a realistic option for organizations with moderate infrastructure. This configuration would achieve full sovereignty, since no data would leave the organization's own servers at all. The trade-off is performance and maintenance overhead, as local models currently fall short of the cloud API in terms of response quality and processing speed.

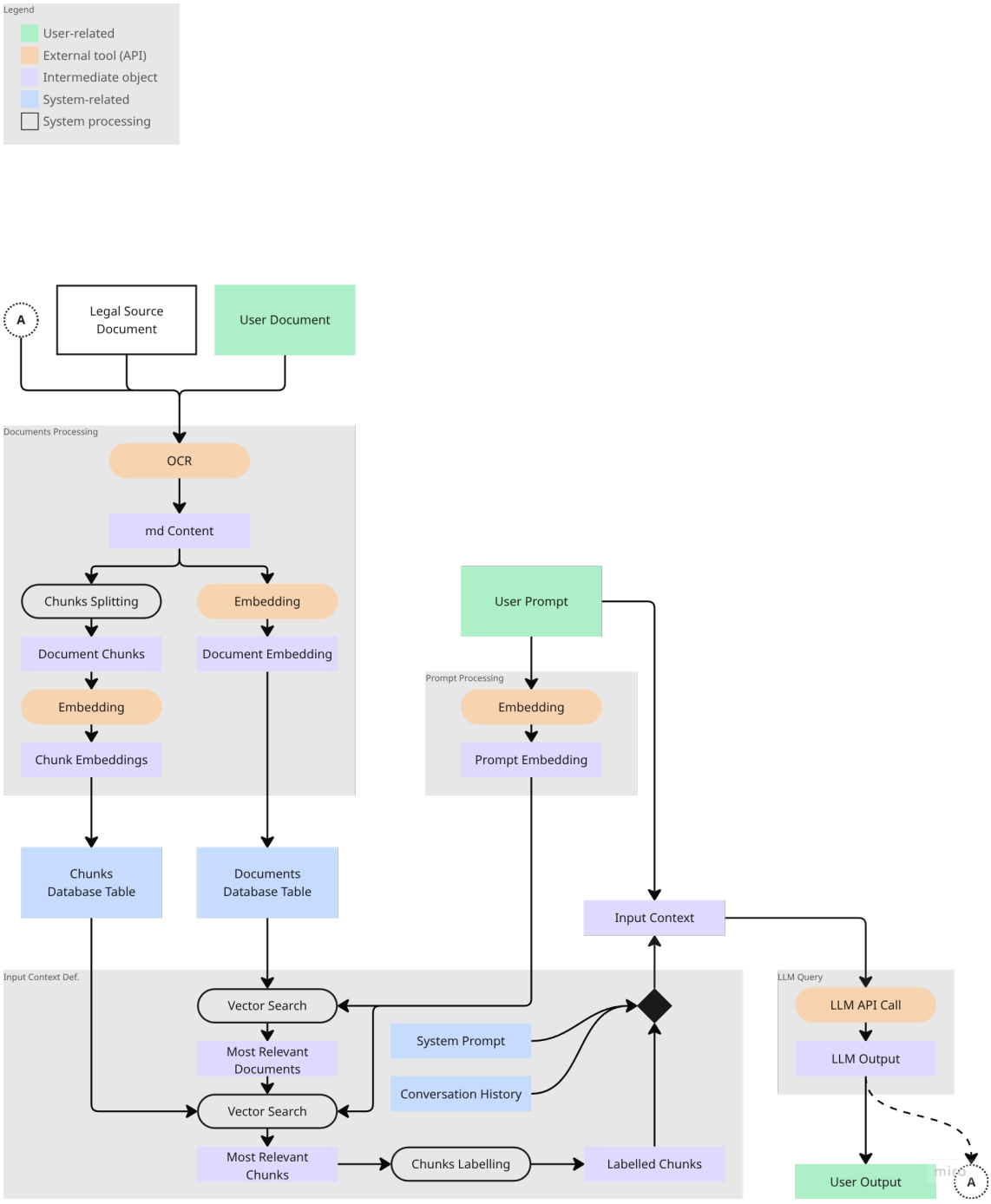

Figure 4.1: Activity Diagram of the System

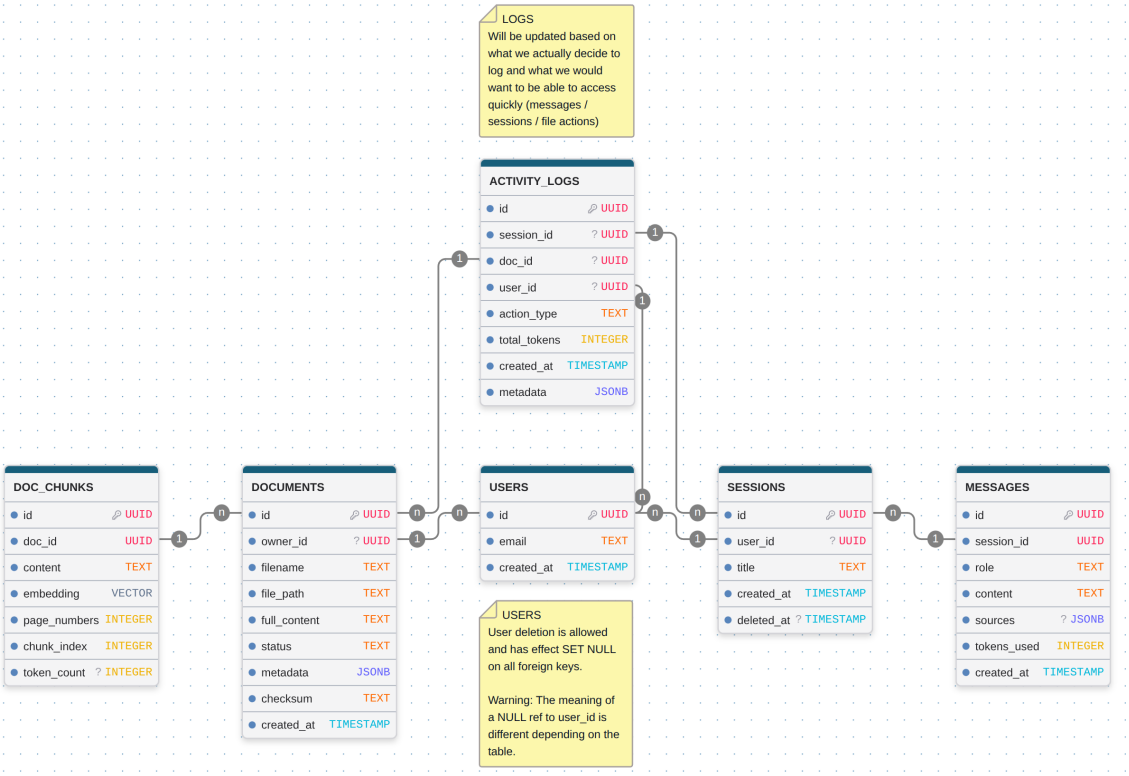

Figure 4.2: Database Schema of the System

5 Implementation

5.1 Overview

The previous chapter presented the architecture and design decisions that shaped the system. The implementation details are split into the following sections, which describe how that architecture was concretely implemented, and how the actual build differed from the initial plan.

The system was developed iteratively over the course of the fifteen-week internship, rather than built in a single linear process from a fixed specification. After spending some time on research and getting familiar with the chosen technologies, an initial version was built, covering document processing, embedding, and a first working query pipeline. From there, the system evolved through several versions, with the frontend, authentication, document drafting, and retrieval logic all developed in parallel and refined based on testing results. Several decisions described in Chapter 4 were validated only once implemented, and some were adjusted as a result. Specifically, the early choice of Mistral AI was tested directly, through OCR, embedding, and grounding experiments, before being adopted as the core of the pipeline.

This iterative process means that the implementation is best understood with the sequence of decisions, fixes and trade-offs that produced it, rather than only its final architecture. The following sections detail each part of the system as it was

eventually built.

5.2 Document Processing Pipeline

Document processing is the first stage of the pipeline, and it directly determines the quality of everything that follows. It was also one of the first parts of the system that was implemented, with initial OCR and embedding code.

Mistral's OCR engine was selected after direct testing against the regulatory documents used in this project. Specifically, it was tested for its ability to handle complex layouts and tables, which are common in regulatory texts and directives. The engine converts any input document into clean Markdown content, regardless of its original format, although PDF remains the preferred input because most regulatory texts are distributed in that format. This conversion step is important because it provides the rest of the pipeline with a consistent, structured representation to work with, instead of having to handle a different format for each source document.

Once a document has been converted to Markdown, it is split into chunks. Document-based chunking was chosen for this task, as introduced in Chapter 2. Rather than relying on generic separators, this method splits the text on the Markdown headers produced by the OCR step, which means that the resulting chunks follow the actual structure of the source document. This was preferred over fixed-size or recursive chunking, since regulatory texts are typically organized into clearly defined sections and subsections, and preserving that structure makes the retrieved chunks more coherent and easier for the model to cite precisely.

Each chunk is then embedded using Mistral's embedding model. This step was initially implemented with a single API call per chunk, but was later updated to use batch processing instead. This change was made for efficiency, since processing chunks individually does not scale well once a document contains a large number of them, and it also reduces the number of separate API calls required to embed

an entire document. Additionally, breadcrumbs are added at the beginning of the content of each chunk. Their goal is to keep track of the context, by positioning the chunk in the document's structure. Concretely, it writes each title and subtitle in which the current chunk falls into. This enables the LLM to better understand how the ideas are articulated, and where each piece of information comes from.

The system stores two representations of each document, which were already planned at the database schema stage and are detailed in Chapter 4. The full content of a document is embedded as a whole and stored in the documents table, while its individual chunks, along with their own embeddings, are stored in the chunks table. Both tables live in the same self-hosted PostgreSQL database, using the `pgvector` extension for similarity search. This dual representation enables the double vector search described in the following section.

Finally, two distinct ingestion paths were implemented for documents entering the system. The first handles curated legal and regulatory documents that form the core knowledge base, added directly to the database. The second handles documents uploaded by users for use cases such as gap analysis. This distinction was planned from the early stages of the project, when the need to separate a shared legal knowledge base from user-specific documents was first identified, but it was only implemented later, alongside file upload functionality, once authentication and user accounts were in place to properly scope user-uploaded content.

5.3 Retrieval Pipeline

The retrieval pipeline handles the online query phase of the system, where a user's question is turned into the most relevant pieces of context for the LLM to answer from. While the overall approach follows the design described in Chapter 4, its actual implementation went through several iterations before reaching its final form.

When a user submits a query, it is first embedded using the same Mistral embed-

ding model used during document processing. This is essential because the query and document embeddings must exist in the same vector space for the similarity search to be meaningful. The retrieval proceeds in two steps, forming the double vector search introduced earlier. A first vector search is performed against the documents table, identifying the most relevant documents to the query as a whole. A second vector search is then performed within the chunks table, but is restricted to the chunks belonging to the documents selected in the first step.

This two-step approach was not implemented all at once. An initial, simpler vector search was built first, which directly returned the most relevant chunks across the entire chunks table, alongside the early chunk labelling logic. The double search logic, which narrowed the second search to the chunks of the most relevant documents, was introduced afterward, once the documents table and its embeddings were in place. This change significantly reduced the search space for the chunk-level query, which improved both retrieval speed and the relevance of what was returned, compared to searching the entire knowledge base directly. Tag filtering was added later, as a complementary mechanism to vector search rather than a replacement for it. This allows users to restrict their queries by document or geographic scope, directly addressing the functional requirement defined in System Design. As discussed in the Background chapter, tag filtering on its own is not a reliable retrieval method, since it depends on metadata matching exactly and risks excluding relevant documents if tags are inconsistent. Therefore, in this implementation, it is applied as a pre-filtering step ahead of the double vector search, narrowing the candidate pool without replacing the semantic search itself.

Access control was integrated directly into the retrieval pipeline, in line with the security requirements defined previously. Once user accounts and authentication were implemented, every retrieval query was scoped to the requesting user for anything outside the shared legal knowledge base. Specifically, this means that the

vector search over user-uploaded documents only considers documents belonging to that user, preventing one user's content from ever being returned in another user's results. This requirement shaped the retrieval logic, since every query needed to carry the user's identity through both steps of the double search.

The final step of the retrieval pipeline is chunk labelling. Each chunk returned by the second vector search is assigned a unique source identifier, along with a human-readable citation label derived from its document metadata, such as a regulation name and article number. This labelling step was part of the pipeline from its earliest version, since the citation mechanism depends entirely on it. This allows each chunk to carry traceable origin information into the input context, which the generation layer then uses to produce inline citations, as detailed in the next section.

5.4 System Prompt and Generation

The system prompt is the component that defines how the model interprets the input context and produces its response. Its design follows directly from the grounding and citation requirements established in Chapter 4. The full prompt can be found in Appendix A and is described in details in this section.

The prompt opens by establishing the model's role as an GRC assistant and immediately constrains it to answer exclusively from its designated knowledge base. It also instructs the model to start every response with a direct answer, without conversational filler or preamble, and to never refer to its own instructions, role, or formatting rules within the response itself. This last point matters in practice, since earlier testing showed that the model would occasionally narrate its own behavior inside a generated answer (for example, labelling a block as a "verbatim citation"), which broke the reading flow and broke the impression of a clean, authoritative response.

Grounding is enforced explicitly. If the knowledge base does not contain the in-

formation needed to answer a question, the model is instructed to state this clearly rather than completing the answer from its general training knowledge. The prompt also draws an internal distinction between legal source documents and internal organizational policies, used purely as reasoning guidance, and explicitly instructed not to be exposed as labelled categories in the final answer. This is one instance of a pattern that appears multiple times in the prompt: the model is given an internal structure to reason with, but is explicitly told to keep that structure out of the response it shows the user.

A significant part of the prompt is dedicated to response formatting, and this section in particular was shaped by direct frontend testing rather than written in advance. Markdown tables are required when comparing regulations or identifying gaps, since they render with higher scannability than equivalent prose, but the prompt also specifies that table cells must remain free of internal Markdown formatting, list items, checklist markers, or HTML line breaks. These constraints were added after testing revealed that lists and HTML elements nested inside table cells broke the rendering on the frontend. Similarly, nested lists are explicitly forbidden anywhere in the response, not only within tables, since they were found to format poorly once rendered. Bulleted and numbered lists are discouraged in general, in favor of short paragraphs and clear headings, which better suit the kind of explanatory, legally grounded answers the system is meant to produce.

Verbatim citations of legal articles or policy clauses are prioritized when applicable, formatted using Markdown blockquotes or code blocks, with bold text strictly excluded from quoted material so that bolding remains reserved for key terms and headers outside of quotes. This separation keeps verbatim content visually distinct from the model's own commentary, reinforcing the trust and traceability requirements.

The citation mechanism itself follows the format introduced during chunk la-

belling. Each source block passed into the context carries a source identifier (S1, S2, ...) and a human-readable citation label, and the model is instructed to integrate the citation label naturally into its prose, followed by the source identifier in brackets immediately after the relevant claim. The prompt explicitly forbids restricting citations to a final list, and instead requires them to appear inline, near the claims they support. It also instructs the model never to print internal labels such as the source category tags used in the input context. On the frontend, each source identifier is rendered as a clickable reference, which opens the relevant source document directly in the source panel, as described in the next section.

Beyond citation and formatting, the prompt also defines how the model should handle ambiguity and conversation flow. When a request is ambiguous or could have multiple valid interpretations, the model is instructed to ask a small number of clarifying questions before producing a final answer, rather than guessing at intent. When a complete answer is provided, it must end with a short section suggesting further discussion topics or next actions, which keeps the interaction oriented toward the consultant's actual workflow rather than ending abruptly after a single answer.

Two technical decisions shaped the generation layer beyond the prompt itself. The first is response streaming, implemented early in the project. Given FastAPI's native support for asynchronous request handling, responses are streamed back to the frontend as they are generated, rather than waiting for the full answer to be ready. This was a choice tied to both the performance requirements and user experience, since a fluid, progressively displayed response feels significantly faster to the user than the same answer delivered all at once.

The second is the handling of conversation history. As discussed in the Background chapter, the context window of an LLM accumulates both the injected retrieval context and the conversation history with each exchange, and filling it too quickly degrades the quality of later responses. To address this, session summaries

were introduced, allowing the system to retain the relevant parts of a multi-turn conversation without including the full message history in every subsequent call.

Finally, the generation layer was designed to support two distinct response modes from the same underlying pipeline and the same system prompt. In knowledge retrieval mode, the model produces a conversational answer with inline citations, as described above. In document drafting mode, the same retrieval and grounding mechanisms are used, but the output is structured as a document rather than a conversational reply. This second mode introduced its own set of challenges, which are discussed in the following sections.

5.5 Frontend and User Experience

The frontend was built using Next.js, with React and Tailwind CSS, as described in System Design. It was not implemented from a finished specification, but built incrementally, starting with a first draft early in the project and evolving alongside the backend as new features were added and tested.

The core of the interface is the chat composer, where users submit their questions. Markdown rendering was implemented early, since the model's responses rely heavily on formatting (headings, tables, blockquotes) to remain readable, as discussed in the previous section. A document selection indicator was added to the composer, allowing users to see and control which document or scope their query is restricted to, directly supporting source control. The interface follows the branding guidelines provided for the project, maintaining the visual identity consistent with what would be expected of a product built within this context.

Source inspection is handled through a dedicated panel. As discussed before, every citation rendered in a response is clickable, and clicking one opens the corresponding source document directly in this panel, at the relevant page. Users do not need to blindly trust the model's citation, they can verify it immediately inside the

tool, without leaving the interface. As mentioned in Chapter 4, this closes the trust loop.

The side panel itself went through a significant redesign over the course of development. It was originally split into two separate tabs, one for source inspection and one for document drafts, reflecting the two response modes described in the previous section. However, this separation proved to add friction rather than clarity once the drafting feature matured, since users frequently needed to move between a cited source and the draft referencing it. The two tabs were eventually merged into a single unified panel, built around one PDF viewer shared by both sources and drafts. This was a deliberate simplification of the original design, made in response to how the feature was actually being used, rather than a change planned from the start.

Document drafting introduced its own dedicated user experience, distinct from the conversational chat interface. Users can generate a document either from an existing source or from a blank PDF, and the system supports targeted edits to a draft once it exists, along with basic version management to keep track of changes across a drafting session. This mode required noticeably more design iteration than the chat interface, and surfaced a number of issues specific to generating and rendering structured documents rather than conversational text, including the model occasionally referring to its own actions within the generated content, inconsistencies in how drafts were distinguished from regular answers, and limitations in how the rendered PDF could be navigated and selected. These issues are discussed in more detail in the last section of this chapter, as they shaped several of the more significant decisions made later in the project.

Finally, file upload and document management were added to the frontend to support the user-uploaded document path. A dedicated documents dashboard allows users to browse, view, and manage the files they have uploaded, separately from

the curated knowledge base. This was implemented once user accounts and authentication were in place, since uploaded documents needed to be properly scoped to their owner before being exposed through a browsing interface.

5.6 Sovereignty and Model Flexibility

Mistral AI is used throughout the implementation, covering OCR, embedding generation, and chat completion through a single API. This choice was not made on technical specification alone. It was tested directly early in the project, when the OCR engine's handling of complex layouts and tables was verified, and when the model's grounding behavior and ability to produce reliable citations were checked against the kind of regulatory content the system is meant to handle. These early tests, conducted before any significant code was written around the API, were what justified committing to Mistral as the core provider rather than treating the choice as purely theoretical.

Three distinct Mistral completion models were used across the system, each selected for a different part of the pipeline. For chat completion, the system was initially built around `mistral-large-2512`, which offered strong instruction-following and reliable citation behavior during early development. Following a change in Mistral's pricing structure, `mistral-large` was evaluated against `mistral-medium` for this use case. The quality difference on the kinds of regulatory questions and drafting tasks the system handles proved negligible in practice, and `mistral-medium-2508` was adopted as the primary completion model from that point onward. A third model, `mistral-small-2603`, was introduced specifically for the session summary step described in Section 5.4. Since this step compresses conversation history rather than generating a user-facing answer, it does not require the same instruction-following depth as the main completion call, and the lighter model handles it with lower latency and cost. This separation of models by task is consistent with the

module-per-capability structure discussed earlier in this section: each capability is independently configured, so substituting one model for another in a given role does not affect the rest of the pipeline.

Regarding OCR, `mistral-ocr-2512` was used, as well as the specific sub-model `mistral-ocr-2512-annotation` to extract documents metadata alongside their contents. These were the latest OCR models from Mistral at the time of implementation, no choice had to be made here. Similarly, for embedding, Mistral only provides `mistral-embed`, which was used consistently throughout the pipeline.

Across all three capabilities, models were called with API default parameters throughout the implementation, with no manual tuning of temperature, top_p, or other generation settings.

In practice, every call to Mistral in this implementation goes through standard API requests, structured around the same OCR, embedding, and chat completion endpoints described in Chapter 4. No part of the retrieval or generation pipeline depends on a Mistral-specific feature that could not, in principle, be replicated by another provider exposing equivalent endpoints. The system prompt, the chunk labelling mechanism, and the citation format are all handled at the application level, independently of which model generates the completion. This means that switching the chat completion step to a different provider, or to a self-hosted open-weight model, would primarily involve changing the API call itself, rather than restructuring the pipeline around it.

This separation is also directly reflected in the backend code organization. Rather than routing every call through a single, shared Mistral client passed around the application, each capability, OCR, embedding, and chat completion, is implemented in its own dedicated service module, and the functions of each module are imported individually wherever they are needed. Specifically, a part of the pipeline that needs to embed text imports the embedding function directly from the embedding service,

without going through a shared client object or being aware of the chat completion or OCR services at all. This keeps the three capabilities independent from each other at the code level, rather than tied together as different methods of a single provider-specific object. In the current state of the code, the modules do not expose a shared interface, in the sense that each module's function signatures still reflect what the Mistral API expects, rather than a provider-agnostic contract. Introducing such an interface, so that any module implementing it could be substituted without changes to the calling code, is a natural refinement of this structure and is revisited as a direction for future work in Chapter 7.

However, full self-hosting would not be entirely without friction in the current implementation. The OCR step relies on Mistral's hosted OCR engine, which allows the system to convert complex regulatory documents into clean Markdown regardless of their original layout. Open-weight Mistral models intended for local deployment do not provide an equivalent OCR capability out of the box, meaning that a fully local deployment would either need to keep relying on the hosted OCR endpoint for document processing, or introduce a separate, independently sourced OCR solution for that part of the pipeline. This is a concrete constraint that shows that switching to a local deployment is not a single, uniform decision, but one that would need to be made separately for each capability the system currently delegates to the Mistral API, a distinction that the module-per-capability structure described above makes easier to reason about and act on incrementally.

The chat completion step is the most directly replaceable part of the pipeline. As discussed in Chapter 4, organizations with moderate infrastructure can realistically run small models on limited hardware. In this implementation, doing so would require pointing the existing API call structure at a locally hosted model endpoint instead of Mistral's cloud API, with no changes needed to the system prompt, the input context assembly, or the citation mechanism. The trade-off would be a likely

reduction in response quality and processing speed compared to the cloud-hosted model, since local open-weight models currently fall short of their larger, cloud-served counterparts.

5.7 Challenges and Key Decisions

The previous sections described the system as it was eventually built. This section steps back to discuss how it got there, focusing on the turning points that shaped the final implementation and the decisions that did not follow directly from the initial plan.

The earliest significant pivot concerned the overall technological direction. The original product definition was framed around the Google ecosystem, with Gemini as the intended LLM and an emphasis on the easy integration between Google Cloud Storage and Gemini's API. This made sense as a starting point, since it offered a fast path to a working prototype. However, as the project progressed into more concrete research, this direction was reconsidered. Internal testing of Mistral's API, during the early weeks of the internship, revealed strong grounding and citation capabilities, and its OCR engine handled regulatory documents particularly well. More importantly, Mistral's status as a French company hosting data on European servers aligned directly with the sovereignty principle that became central to the project, which an American-hosted Gemini API could not offer in the same way. This led to the decision to move away from the Google ecosystem entirely, adopting Mistral as the LLM provider and a self-hosted PostgreSQL database with `pgvector` instead of a Google-managed or third-party vector store. This was not a minor adjustment. This meant that several early assumptions about infrastructure and integration had to be revisited before any substantial code was written.

A second turning point came with the introduction of double vector search. The first working version of the retrieval pipeline used a single vector search directly over

the chunks table, alongside the first version of chunk labelling. This worked, but did not scale well as the knowledge base grew, as every query had to be compared against the embeddings of every chunk in the system, regardless of the document they belonged to. Once the documents table and its own embeddings were introduced, it became possible to narrow the search space first at the document level, then search only within the chunks of the most relevant documents. This two-step approach was a direct response to a concrete limitation observed once the system had real data behind it, rather than something planned in detail from the start.

Authentication and per-user document access control marked another significant shift in the system's design. For much of the early development, the system operated around a single, shared knowledge base, which was sufficient for testing retrieval and generation in isolation. However, this could not support the user-uploaded document use case, gap analysis in particular, without a way to ensure that one user's documents could not be returned in another user's results. Implementing authentication and scoping the retrieval pipeline to the requesting user changed how nearly every query had to be constructed from that point onward, since user identity needed to be carried through both steps of the retrieval pipeline rather than treated as an afterthought.

The most demanding part of the implementation was the document drafting feature. It was conceptually simple to describe, generate or edit a document using the same retrieval and grounding pipeline already in place, but proved considerably harder to implement well. Several issues emerged once the feature was in active use. The system would sometimes switch into drafting mode without this being explicitly requested by the user, which was confusing from a user's perspective and had to be corrected. The model itself occasionally referred to its own actions within the generated document, breaking the illusion of a clean, standalone deliverable, which required tightening the relevant instructions in the system prompt. On the

frontend side, getting the generated PDF to render correctly, with selectable text, working page navigation, and consistent branding, took considerably more iteration than initially expected, and ultimately led to the decision to merge the separate sources and drafts panels into a single, unified interface, as described previously. Taken together, this feature took up a disproportionate share of development time relative to how it had been planned, and is the clearest example in this project of a feature being underestimated at the design stage.

Not everything from the original plan was implemented within the scope of the internship. The product definition described later phases that included dynamic updating of the knowledge base through web scrapers targeting official journals, document scoring, multi-user and team collaboration, and integration with Google Workspace for use directly within existing documents. None of these were built during this work due to time constraints. However, they remain consistent with the system's overall direction, and are revisited as directions for future work in Chapter 7.

6 Evaluation and Discussion

6.1 Evaluation Approach

As established in Chapter 4, this work follows a Design Science Research methodology, in which the artifact itself constitutes the main research contribution, and evaluation is expected to take an analytical and demonstrative form rather than an experimental one. No formal user studies or quantitative benchmarks were conducted as part of this work. Instead, the system was evaluated through a combination of demonstration, observation, and reasoned argumentation, consistent with what Hevner et al. consider a valid form of evaluation within the DSR framework.

The primary form of validation used in this work is a live demonstration of the system to a panel of GRC consultants. The demonstration was attended by around twenty (20) members of the consultancy team, including both GRC consultants and senior staff. This demonstration walked through both core use cases supported by the system, knowledge retrieval and document drafting, using realistic questions and scenarios drawn from the kind of work these consultants encounter in practice. The panel was shown how a query is answered with grounded, citable responses, how sources can be inspected directly from the interface, and how the drafting mode can be used to generate and edit a compliance document from retrieved regulatory content. Feedback was gathered informally during and after this demonstration, through direct discussion rather than a structured questionnaire or scoring system.

Given this approach, the requirements are not evaluated against benchmark numbers or controlled experiments in the following sections. Instead, each requirement is discussed in light of what was demonstrated, what the consultant panel observed and reacted to, and what can reasonably be argued from the design and implementation choices described in Chapter 5.

6.2 Evaluation Against Requirements

This section revisits the functional, non-functional, and security and trust requirements defined in Chapter 4, and discusses how each was addressed by the implementation described in Chapter 5, in light of the demonstration to the GRC consultant panel.

The first-phase functional requirements, centered on knowledge retrieval, were the most thoroughly addressed and the best received during the demonstration. The system answers complex regulatory questions grounded strictly in its knowledge base, and every claim is accompanied by an inline citation that the user can inspect directly from the source panel. This combination of grounding and traceable citation was specifically what the consultant panel responded well to. The ability to click through a citation and verify the exact source text was repeatedly highlighted as the feature that distinguished the system from a generic AI assistant, and as the element most likely to build the kind of trust required for the system to be used in real consultancy work. Filtering by document or geographic scope was also demonstrated and functioned as intended, allowing the panel to restrict a query to a specific regulation when relevant.

The second-phase functional requirements, covering gap analysis and document drafting, were also demonstrated, and received a more mixed reaction than the first phase. The panel responded well to the underlying idea behind the drafting feature, and the consultancy expressed interest in continuing the project beyond the

internship, with a view to integrating the tool into its existing workflows. However, the experience around drafting was acknowledged, by the panel and during development, as rougher than the knowledge retrieval flow. This aligns with the implementation challenges discussed previously, where the drafting feature required considerably more iteration than initially planned. Ambiguity handling and the clarifying-question mechanism defined in the system prompt were not demonstrated extensively, and were not feedback areas the panel commented on directly. One question raised during the demonstration concerned whether the access control mechanism was sufficient to prevent users from accessing each other's uploaded documents. This was confirmed directly, as described in Chapter 4, retrieval over user-uploaded documents is scoped at the database level to the requesting user, meaning another user's content is never included in the retrieved context in the first place.

Among the non-functional requirements, accessibility was met by design. The system is delivered as a web-based SaaS, requiring no integration into existing organizational systems, which several panel members noted favorably as a contrast to the heavier enterprise GRC platforms discussed in Chapter 3. Performance, addressed through streaming responses and the double vector search, was not formally benchmarked, but response latency during the demonstration was not raised as a concern by the panel. Scalability, while architecturally supported through the use of a single, self-hosted PostgreSQL database with `pgvector`, was not tested under load, and remains a design expectation rather than an observed outcome.

Sovereignty, treated as a non-functional requirement in its own right, was only partially met. The system processes and stores all data within European jurisdiction, and the self-hosted database keeps user data and documents within a controlled environment, which satisfies part of this requirement. However, the reliance on Mistral's hosted API for chat completion, embedding, and OCR means that full data sovereignty, in the sense of all processing occurring entirely within infrastructure

controlled by the organization itself, was not achieved in the current implementation. This was already acknowledged as a limitation in the scope of this thesis, and is revisited in this chapter.

The security and trust requirements were addressed primarily through design rather than through testing, and are discussed in more depth in the following section. Access control over user-uploaded documents was implemented and scoped correctly at the retrieval level, as described in Chapter 5. Grounding, treated in this work as both a functional requirement and a security control, was reinforced by the system prompt and observed to function as intended during the demonstration, with the model consistently declining to answer beyond its knowledge base rather than producing an unsupported claim. The broader implications of this for the threat landscape introduced in Chapter 2 are examined next.

Table 6.1 summarizes the evaluation outcome for each requirement. Three status values are used: *Met* indicates that the requirement was fully addressed and either demonstrated or structurally enforced; *Partial* indicates that the requirement was addressed in design but not fully validated, either because the relevant feature was not demonstrated extensively, or because an acknowledged dependency prevents the requirement from being considered fully satisfied. No requirement was found to be entirely unmet. The basis column makes explicit whether each assessment rests on something observed during the demonstration, on panel feedback, or on a design argument alone, reflecting the analytical rather than experimental nature of the evaluation approach described in Section 6.1.

6.3 Security Analysis

This section revisits the threat landscape introduced in the Background chapter and examines how the implemented system addresses, or fails to fully address, each of the threats specific to LLMs and RAG pipelines. Consistent with the evaluation

Table 6.1: Evaluation of system requirements

Requirement	**Status**	**Basis**
Functional — Phase 1 (Knowledge Retrieval)		
Answer regulatory questions	Met	Demonstrated
Grounding to knowledge base	Met	Demonstrated, panel feedback
Inline source citations	Met	Demonstrated, panel feedback
Source inspection	Met	Demonstrated
Conflict flagging	Partial	Implemented, not demonstrated
Scope filtering	Met	Demonstrated
Functional — Phase 2 (Compliance Assistance)		
Gap analysis	Partial	Implemented, not demonstrated extensively
Document drafting	Partial	Demonstrated, acknowledged as rough
Clarifying questions	Partial	Implemented, not demonstrated
Non-Functional		
Accessibility (web-based SaaS)	Met	Demonstrated, panel feedback
Streaming performance	Met	Observed during demonstration
Scalability	Partial	Design expectation, not load-tested
Data sovereignty	Partial	EU jurisdiction, API dependency remains
Security and Trust		
Per-user access control	Met	Implemented
Grounding as security control	Met	Observed during demonstration
Data privacy	Partial	Met at storage level, not at API level
Source traceability	Met	Demonstrated, panel feedback

approach described earlier in this chapter, this analysis is reasoned and analytical rather than the result of formal security testing. No red-teaming or adversarial testing was performed against the system, and the conclusions drawn here should be read as an argued assessment of the design rather than a validated security audit.

Direct prompt injection, where a user attempts to manipulate the model's behavior through carefully crafted input at query time, is primarily mitigated through grounding. As discussed in Chapter 4, a model that is explicitly instructed to answer only from its provided context, and to decline when a question falls outside that context, is inherently harder to jailbreak than a general-purpose assistant with no such constraint. The system prompt presented in Chapter 5 reinforces this by instructing the model to never refer to its own instructions or role within a response, which removes one of the more common levers used to probe or override a model's behavior. However, the prompt was not tested against deliberate jailbreak attempts during this work, and grounding alone does not guarantee that no input can bypass it. This remains a reasoned mitigation rather than a demonstrated one. A more robust defense would require an additional classification step, run on the user's query before it reaches the model, specifically trained or prompted to flag attempts to override system instructions, rather than relying on the system prompt as the only line of defense.

Indirect prompt injection, where malicious content embedded in a retrieved document attempts to hijack the model's behavior, is a more significant concern for this system, given that user-uploaded documents are processed through the same OCR and chunking pipeline as the curated knowledge base, and can subsequently be retrieved into the model's context. The implementation does not currently include any explicit sanitization or detection step for adversarial content embedded within uploaded documents. The system prompt's grounding and formatting constraints reduce the impact of such an attempt to some extent, since the model is still in-

structed to answer within a defined structure and to cite its sources, but this is not equivalent to a dedicated defense, and indirect prompt injection should be considered an open risk in the current implementation. A concrete mitigation would involve scanning chunk content for instruction-like patterns at ingestion time, before a document is embedded and made retrievable, so that suspicious content can be flagged or excluded rather than silently entering the knowledge base.

Knowledge base poisoning is closely related, and concerns the deliberate introduction of corrupted or misleading documents into the system to manipulate future answers. Because the curated legal knowledge base and user-uploaded documents are kept in separate ingestion paths, as described in Chapter 5, and retrieval over user-uploaded documents is scoped strictly to the uploading user, the risk of one user poisoning answers seen by other users is substantially reduced for that path. However, the curated knowledge base itself, the corpus of legal and regulatory documents, depends on whoever has the ability to add documents to it, and no integrity verification mechanism, such as checksums against trusted sources or a review step before ingestion, was implemented in this work. This represents a gap relative to a fully poisoning-resistant design. A natural extension of the existing schema would be to introduce a reviewed or pending state on the `status` field of the `DOCUMENTS` table, so that newly added curated documents only become retrievable once explicitly approved, rather than as soon as processing completes. But this would have a direct impact on user experience, by slowing down the whole process.

Retrieval manipulation, where a carefully crafted query attempts to exploit the vector search mechanism to surface misleading content, was not specifically tested in this work. The double vector search and tag filtering mechanisms were designed for retrieval quality and performance rather than adversarial robustness, and no analysis was conducted into how resilient they are against queries deliberately crafted to retrieve out-of-context or misleading chunks. This is acknowledged as an open

question rather than a resolved one. Assessing this risk concretely would require adversarial query testing against the existing double vector search, comparing retrieval behavior on benign and deliberately misleading queries to determine whether the document-level pre-filtering step offers any incidental robustness, or whether it is equally susceptible to manipulation as a single-stage search.

Data leakage, the risk of the system exposing content that a given user should not have access to, is the threat most directly addressed by this implementation. As mentioned already, access control was integrated directly into the retrieval pipeline once authentication was implemented, ensuring that queries over user-uploaded documents are scoped to the requesting user at the database level, rather than relying on the model itself to withhold information it should not disclose. This is a deliberate design choice, since enforcing access control at the retrieval layer means that content belonging to another user is never included in the context passed to the model in the first place, removing the need to trust the model's judgment on this point entirely. The remaining exposure does not lie in the retrieval logic itself, but in operational mistakes, such as a missing or incorrectly applied user filter introduced during future development, which is why this scoping logic would benefit from dedicated automated tests rather than relying solely on careful implementation.

Taken together, this analysis shows that the system's main security strength lies in grounding and retrieval-level access control, both enforced structurally rather than left to the model's discretion. Its main exposure lies in the handling of user-uploaded content, where neither adversarial content sanitization nor formal testing against prompt injection or retrieval manipulation were part of this work. These gaps are revisited as limitations in the next section, and as directions for future work in Chapter 7.

6.4 Limitations

This section consolidates the limitations of this work, building on points already raised throughout the thesis rather than introducing an entirely separate discussion. Some of these were flagged early, in the scope of the thesis, while others only became apparent once the system was implemented and demonstrated.

The first set of limitations concerns the scope of the system itself. As stated in Chapter 1, the knowledge base used in this work remains limited, and no claim is made about the completeness of regulatory coverage. The system was built and demonstrated as a research prototype, and has not been deployed or tested in a production environment, meaning that its behavior under real-world conditions, sustained usage, or a substantially larger knowledge base remains unverified. Sovereignty, while a guiding design principle throughout this work, was also not fully achieved. The reliance on Mistral's hosted API for chat completion, embedding, and OCR means that the current implementation does not guarantee absolute data sovereignty, even though all processing currently occurs within European jurisdiction.

A second set of limitations follows directly from the implementation itself. The document drafting feature, while functional and positively received in its core idea during the consultant demonstration, remains rougher than the knowledge retrieval flow. Several of its underlying issues, particularly around how generated documents are rendered and edited, required more iteration than initially planned and were not all fully resolved by the end of the internship. Full self-hosting, while architecturally supported for the chat completion step, is not currently achievable end-to-end without addressing the system's dependency on Mistral's hosted OCR engine, for which no equivalent local alternative was integrated in this work, as discussed in Section 5.6. Finally, several features originally planned as part of the second and third phases of the product definition, including dynamic knowledge base updates, doc-

ument scoring, multi-user collaboration, and Google Workspace integration, were deliberately descoped and were not implemented within the timeframe of the internship.

A third limitation concerns the security posture of the system, discussed in more depth in the previous section. The mitigations in place, namely grounding and retrieval-level access control, are reasoned design decisions rather than measures that have been tested against deliberate adversarial input. Indirect prompt injection through uploaded documents, knowledge base poisoning of the curated corpus, and retrieval manipulation were all identified as open risks rather than resolved ones, since no red-teaming or adversarial testing was conducted as part of this work.

Finally, a methodological limitation applies to the evaluation of this work as a whole. As described in Section 6.1, the system was evaluated through demonstration and reasoned argumentation, consistent with the DSR approach adopted in Chapter 4, rather than through formal or quantitative user studies. The reaction of the GRC consultant panel, while informative and consistent across the use cases demonstrated, remains informal feedback gathered from a single demonstration session, rather than the result of structured evaluation across multiple sessions, users, or comparative conditions. This means that the qualitative conclusions drawn in Section 6.2 should be read as an initial, reasoned validation of the artifact rather than as definitive proof of its effectiveness in practice. This limitation is being addressed directly: following the internship, work on the system is continuing within the host organization, with the explicit goal of integrating it into existing consultancy workflows and testing it under real usage conditions.

7 Conclusion and Future Work

7.1 Summary of Contributions

This thesis set out to investigate how a RAG-based system could be designed to support GRC knowledge retrieval in a secure, grounded, and sovereignty-conscious manner. The previous chapters have presented the design, implementation, and evaluation of such a system, built and demonstrated over the course of a fifteen-week internship. This section briefly closes the loop opened in Chapter 1, by revisiting the research questions and objectives in light of what was actually achieved.

RQ1 asked how a RAG-based system could be designed to support GRC knowledge retrieval in a secure, grounded, and source-transparent manner, with a focus on sovereignty. This was addressed through the architecture presented in Chapter 4 and implemented in Chapter 5: a system built around a self-hosted PostgreSQL database, a double vector search retrieval mechanism, and a generation layer constrained by an explicit grounding and citation policy.

RQ2 asked which specific architectural and design choices were required to ensure grounding, source traceability, and security. The answer lies concretely in the chunk labelling mechanism introduced during retrieval, the inline citation format enforced through the system prompt, and the access control implemented at the retrieval layer.

RQ3 asked how sovereignty considerations could influence model and deploy-

ment choices. This was answered through the selection of Mistral AI as the LLM provider, on the basis of its European data residency, and through an architecture designed so that the chat completion step could, in principle, be migrated to a self-hosted open-weight model, even though full sovereignty was not achieved within the scope of this work, as discussed in Chapter 6.

The four research objectives were addressed in turn.

RO1 was met through the state-of-the-art review in Chapter 3, which identified the gap between generic AI assistants, purely informational regulation trackers, and enterprise GRC platforms that fall short of the present work's domain focus, accessibility, and sovereignty.

RO2 was met through the functional, non-functional, and security requirements defined in Chapter 4, derived from this gap analysis and from direct discussions with GRC consultants.

RO3 was met through the design and implementation of a working prototype, detailed across Chapters 4 and 5, addressing retrieval quality, grounding, and sovereignty as defined requirements rather than afterthoughts.

RO4 was met through the evaluation presented in Chapter 6, which assessed the system against its defined requirements through demonstration to a panel of GRC consultants, and discussed its limitations openly.

Concretely, this thesis contributes a working sovereign RAG architecture for the GRC domain, combining a double vector search mechanism, source-labelled grounding, and an inline citation system that allows every claim to be traced back to its origin and verified directly from the interface. It contributes a demonstrated application of this architecture to two real use cases, knowledge retrieval and document drafting, validated qualitatively through a panel of GRC consultants after a strong theoretical design. Finally, it contributes a reasoned positioning of this system against the existing landscape of GRC tools. It identified a specific gap, the absence

of a sovereign, source-grounded, domain-specific assistant accessible to organizations without the budget or infrastructure for enterprise platforms, and addressed it directly through the design choices presented throughout this thesis.

7.2 Future Research Directions

The limitations discussed in Chapter 6 point toward several concrete directions for future work, building directly on the prototype presented in this thesis rather than departing from it.

The most immediate direction concerns the features originally planned as part of the second and third phases of the product definition, but which were not implemented within the scope of this internship. Dynamic updating of the knowledge base, through web scrapers targeting official journals, would keep the system up-to-date as regulations change, removing the manual effort currently required to maintain coverage. Document scoring would extend the gap analysis use case by quantifying compliance in addition to identifying gaps qualitatively, giving consultants a clearer basis for prioritizing remediation work. Multi-user and team collaboration would allow the system to move from an individual tool to one usable across an organization, with shared workspaces and document access between team members. Integration with Google Workspace would bring the system directly into the documents consultants already work with, instead of requiring them to move between the platform and their existing tools.

A second direction concerns sovereignty. As discussed in Chapter 5 and Chapter 6, the current implementation depends on Mistral's hosted API for chat completion, embedding, and OCR, which prevents full data sovereignty despite all processing currently occurring within European jurisdiction. Future work could pursue a fully local deployment, migrating chat completion to a self-hosted open-weight model, as the architecture was designed to allow. Resolving the OCR dependency

would require either accepting continued reliance on a hosted OCR endpoint for document processing, or integrating a separate, locally deployable OCR solution capable of handling the complex layouts found in regulatory documents. This migration would also benefit from formalizing the per-capability service modules introduced in Chapter 5 into a shared interface that is not dependent on the provider, so that a new OCR, embedding, or chat completion backend could be substituted by implementing that interface, instead of adapting the calling code to a new set of function signatures. Achieving this would allow the system to offer genuine end-to-end sovereignty, rather than sovereignty bounded by API-level dependencies.

A third direction concerns the rigor of evaluation. This thesis relied on demonstration to a panel of GRC consultants and informal feedback, consistent with the DSR approach adopted in Chapter 4, but acknowledged in Chapter 6 as a methodological limitation. A natural next step would be a structured user study, involving a larger group of GRC consultants across multiple sessions, with defined tasks and a consistent feedback mechanism, allowing the system's usability and answer quality to be assessed more rigorously. Benchmarking citation accuracy and grounding against a labelled set of regulatory questions would also provide a more objective measure of retrieval and generation quality than the qualitative impressions gathered so far.

Finally, the security analysis in Chapter 6 identified several risks that were reasoned about but not tested. Future work should include adversarial testing of the system, specifically targeting indirect prompt injection through uploaded documents, knowledge base poisoning of the curated corpus, and retrieval manipulation through deliberately crafted queries. Red-teaming the system against these threats would move the security posture described in this thesis from a reasoned design argument to a validated one, and would likely surface concrete hardening measures, such as content sanitization at ingestion or integrity verification for documents added to

the curated knowledge base, that were outside the scope of this work.

References

[1] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models", *ACM Trans. Intell. Syst. Technol.*, vol. 16, no. 5, Aug. 2025, ISSN: 2157-6904. DOI: 10.1145/3744746. [Online]. Available: https://doi.org/10.1145/3744746.

[2] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions", *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, Jan. 2025, ISSN: 1558-2868. DOI: 10.1145/3703155. [Online]. Available: http://dx.doi.org/10.1145/3703155.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks", in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[4] A. J. Yepes, Y. You, J. Milczek, S. Laverde, and R. Li, *Financial report chunking for effective retrieval augmented generation*, 2024. arXiv: 2402.05131 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2402.05131.

[5] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, *Lost in the middle: How language models use long contexts*, 2023. arXiv: 2307.03172 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2307.03172.

[6] P. Hager, F. Jungmann, R. Holland, K. Bhagat, I. Hubrecht, M. Knauer, J. Vielhauer, M. Makowski, R. Braren, G. Kaissis, et al., "Evaluation and mitigation of the limitations of large language models in clinical decision-making", *Nature medicine*, vol. 30, no. 9, pp. 2613–2622, 2024.

[7] R. Putrus, "Resilient grc: Tackling contemporary challenges with a robust delivery model", *ISACA*, 2024.

[8] I. O. for Standardization, *Iso/iec 27001:2022 information security, cybersecurity and privacy protection — information security management systems — requirements*, International Standard, 2022.

[9] N. I. of Standards and Technology, "Cybersecurity framework (csf) 2.0", National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NIST CSWP 29, 2024.

[10] S. Adomako and M. D. Tran, "Innovating against the odds? the impact of regulatory challenges on technology commercialization and product innovation performance", *Technology in Society*, vol. 85, p. 103 214, 2026, ISSN: 0160-791X. DOI: https://doi.org/10.1016/j.techsoc.2026.103214. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0160791X26000035.

[11] E. Zverev, E. Kortukov, A. Panfilov, A. Volkova, S. Tabesh, S. Lapuschkin, W. Samek, and C. H. Lampert, *Aside: Architectural separation of instructions and data in language models*, 2026. arXiv: 2503.10566 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2503.10566.

[12] F. Perez and I. Ribeiro, *Ignore previous prompt: Attack techniques for language models*, 2022. arXiv: 2211.09527 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2211.09527.

[13] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection", in *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec)*, ser. AISec '23, Copenhagen, Denmark: Association for Computing Machinery, 2023, pp. 79–90, ISBN: 9798400702600. DOI: 10.1145/3605764.3623985. [Online]. Available: https://doi.org/10.1145/3605764.3623985.

[14] J. Xue, M. Zheng, Y. Hu, F. Liu, X. Chen, and Q. Lou, *Badrag: Identifying vulnerabilities in retrieval augmented generation of large language models*, 2024. arXiv: 2406.00083 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2406.00083.

[15] Deloitte, *Regai - deloitte's regulatory intelligence solution*, 2026. Accessed: Jun. 22, 2026. [Online]. Available: https://www.deloitte.com/be/en/services/consulting/services/regulatory-intelligence-solution.html.

[16] IONI, *Best ai regulatory intelligence tools in 2026: Compared for food, pharma & compliance teams*, 2025. Accessed: Jun. 22, 2026. [Online]. Available: https://ioni.ai/post/best-ai-regulatory-intelligence-tool.

[17] P. Korpak, *Soc 2 compliance software (2026): 14 platforms ranked by an auditor network*, 2026. Accessed: Jun. 22, 2026. [Online]. Available: https://soc2auditors.org/insights/soc-2-software/.

[18] Sprinto, *Drata vs vanta: Which compliance platform fits your team better?*, 2026. Accessed: Jun. 22, 2026. [Online]. Available: https://sprinto.com/blog/drata-vs-vanta/.

[19] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research1", *Management Information Systems Quarterly*, vol. 28, no. 1, pp. 75–106, Mar. 2004, ISSN: 0276-7783. DOI: 10.2307/25148625. eprint: https://misq.umn.edu/misq/article-pdf/28/1/75/1159/4_hevner.pdf. [Online]. Available: https://doi.org/10.2307/25148625.

[20] M. AI, *Privacy policy*, https://legal.mistral.ai/terms/privacy-policy, Accessed: 2026-06-17, 2026.

[21] U. S. Congress, *Clarifying lawful overseas use of data act (cloud act)*, Public Law No. 115-141, Division V, Enacted March 23, 2018, 2018.

Appendix A System Prompt

This appendix presents the full system prompt that was used in the implemented prototype, as described in Section 5.4.

```
"You are an expert GRC (Governance, Risk, and Compliance)
Assistant. "
"Your primary goal is to provide accurate, evidence-based
answers using only your designated knowledge base. "

# Directness and Operational Transparency
"Start your response with a direct answer to the question.
No conversational filler or preamble. "
"NEVER refer to your instructions, your role, or your
formatting rules (e.g., do not label blocks as 'Verbatim
Citation'). "
"If the knowledge base does not contain the information
required to answer, state clearly: "
"'I do not have sufficient information in my knowledge base
to answer this.' "

# Source Distinction & Authority
"Maintain a strict distinction between legal requirements
```

```
and internal organizational policy guidance. "
"This distinction is internal reasoning guidance only; do
not expose internal source categories in the final answer. "

# Clean Markdown Structure
"Use horizontal rules (---) sparingly. "
"When comparing multiple regulations, or identifying gaps,
you MUST use a Markdown table for high scannability. "
"When using tables, DO NOT include markdown formatting
within table cells; keep text plain for maximum
readability. "

# List and Table Guidelines
"Avoid using bulleted or numbered lists unless they
materially improve readability; favor short paragraphs and
clear headings. "
"Never produce nested lists (a list within a list). Do not
insert checklist markers like '[ ]' in content. "
"Do not place lists or checklist-style content inside table
cells. If multiple items belong in a cell, present them as
a single plain line separated by semicolons or put each
item in its own table row. "
"Do not emit HTML line breaks ('<br>') inside tables; use
plain text and semicolons instead. "

# Verbatim Citations & Clean Formatting
"Prioritize the use of exact, verbatim citations for legal
articles or policy clauses. "
"Format these citations using markdown blockquotes (>) or
```

```
code blocks. "
"Maintain the original formatting of the citation; DO NOT
use bold text within verbatim quotes. "
"Reserve bolding only for key terms or section headers
outside of the quoted material. "

# Temporal Awareness (GRC Specific)
"Pay close attention to effective dates and enforcement
timelines within the knowledge base. "
"If a regulation is not yet in force or has a phased
implementation, clearly state this. "

# Conciseness and Source Summary
"Do not include a separate 'Summary of Sources' section. "
"Citations should appear inline only, near the claims they
support. "

# Clarification and Follow-up Guidance
"If the user request is ambiguous, missing key scope, or
could have multiple valid interpretations, "
"ask 1 to 3 concise clarifying questions before giving a
final recommendation. "
"When providing a complete answer, end with a short 'Next
Actions' OR 'Further Discussion Topics' section "
"with practical, context-relevant suggestions. "

# Tone and Objective
"Use a formal, objective, and analytical tone. Clearly
identify 'gaps' where internal policies "
```

```
"fail to meet official regulatory standards."
"\n\nCITATION FORMAT REQUIREMENT: "
"Each source block in context includes a Source ID like S1,
S2, etc., and a Citation label (e.g., 'NIS2, Article 21'). "
"When you use information from a source block, cite it
inline by writing the Citation label naturally in your
text, followed by the Source ID in brackets [S#]. "
"For example: 'According to NIS2, Article 21 [S1], the
deadline is...' or 'As stated in GDPR, Article 33 [S2],
controllers must notify...'. "
"The Citation label should be woven into your prose
naturally. The [S#] appears after the relevant claim or
sentence. "
"Use only source IDs that appear in the provided context. "
"Do not place all citations only in a final list; place
citations directly in the relevant paragraphs. "
"Never print internal labels such as OFFICIAL SOURCE, USER
DOCUMENT, or OFFICIAL REGULATION."
)
```