SC4079: Final Year Project Report

Chen Ruxing

BSc in Mathematical and Computer Sciences (Double Major)

School of Physical and Mathematical Sciences

Supervisor: Professor Zhao Jun

Co-supervisor: Dr. Xiang Liming

Retrieval-Augmented Generation (RAG) for Precedent Search and Case Law Retrieval in Legal Domain

Abstract

This final-year project focuses on designing and implementing a Retrieval-Augmented Generation (RAG) system to address inefficiencies of traditional search for relevant case law from extensive legal documents in the legal domain. This system integrates a Large Language Model (LLM) with a retrieval mechanism using Facebook AI Similarity Search (FAISS), that fetches relevant legal documents from a knowledge base built using a collection of U.S. legal opinions from the CourtListener platform. The additional retrieval component of RAG ensures that the generated responses are both factually grounded and contextually rich, which may be difficult to achieve by a standalone LLM.

The proposed RAG system caters to legal professionals, including lawyers, judges, legal researchers, and paralegals, who require efficient retrieval of case law. By implementing a RAG system for precedent search, this project aims to improve the reliability of AI-assisted case law retrieval and reduce workloads across the legal domain.

Acknowledgements

I would like to express my deepest gratitude to the incredible people who have supported me throughout this final-year project. First and foremost, I am deeply grateful to my project supervisor, Professor Zhao Jun, and my student mentor, Liu Chang, for their guidance and support. Their expertise and feedback have been extremely helpful in shaping my research and provided me with a clear direction for this project. They shared extensive resource materials on building a RAG system which provided me with a strong foundation to begin this work. I am grateful to them for agreeing to meet with me when I needed help.

I would also like to extend my gratitude to my co-supervisor, Dr. Xiang Liming, for her willingness to co-supervise and support this project.

I am also thankful to Nanyang Technological University (NTU) for its research database NTU Library, computational resources and modules that serve as my pre-requisite, which greatly contributed to the completion of this project.

Last but certainly not least, I want to extend my heartfelt appreciation to my parents for their endless encouragement and support, not just during this project but also throughout my entire university journey. Their support and care for me has given me strength to push through the toughest days, and the privilege to fully dedicate myself to my studies and this project. I'm truly fortunate to have them in my corner.

Table of Contents

1 Introduction

1.1 Background

Natural Language Processing (NLP) has become a cornerstone of modern technology, driven by significant advancements in recent years, particularly with the emergence of Large Language Models (LLMs) like GPT, LLaMA and most recently DeepSeek. LLMs have demonstrated remarkable capabilities in understanding and generating human language. They can also power a diverse array of applications from chatbots that can hold a conversation to tools that help create content. These models have become our preferred search engines, making Google feel like an "ancient" tool. LLMs can summarise and generate direct responses which appeal to users seeking quick and concise answers. Interaction with models like ChatGPT feels natural and queries can be tailored to meet our needs, such as adding the term "eli5" to simplify complex concepts.

While LLMs have been increasingly preferred over traditional search engines, they have not rendered platforms like Google obsolete. Traditional search engines remain relevant for retrieving real-time information and verifying sources. Hallucination no longer just applies to humans, machines are capable of making up false or random facts. We are still in the early stages of such revolutionary technology and challenges persist, particularly in the quality, relevance, and accuracy of generated text. This signifies a need to improve text generation as it directly impacts the effectiveness of its applications, user trust, and their adoption across fields such as legal services and education. Therefore, solving these problems will significantly advance NLP systems and optimise their ability to solve real-world problems.

Moreover, developments in dense vector retrieval methods, such as FAISS, have made it possible to efficiently retrieve relevant documents from large knowledge bases. These innovations in LLMs and dense vector retrieval methods have paved the way for hybrid systems like Retrieval-Augmented Generation (RAG). The RAG system combines retrieval and generation to produce highly relevant and contextually accurate responses.

In the legal domain, professionals frequently have to search for and synthesise relevant information from large repositories of legal documents. Traditional search engines are often inadequate in their ability to understand the contextual nuances of legal language, thus resulting in incomplete or irrelevant results. Moreover, LLMs cannot be fully trusted as they are prone to hallucinations and may generate inaccurate information. These inefficiencies highlight the need for a system that can retrieve and generate contextually enriched outputs suitable for legal practitioners.

This project seeks to address the inefficiencies in the legal domain by developing a RAG system that leverages cutting-edge NLP and retrieval techniques, ultimately streamlining legal precedent search and improving the quality of decision-making.

1.2 Problem Statement

Legal professionals and researchers often have to sift through heaps of legal documents to find relevant information. Furthermore, these documents are complex and contain legal jargon, which require specialised knowledge to understand and analyse. Traditional search engines, such as keyword-based systems, while useful, struggle to return relevant documents due to different legal terminology and lack of understanding in the legal context. Additionally, LLMs are prone to hallucinations and may generate inaccurate or misleading information that is not supported by real data. As a result, legal professionals have to manually review and search through legal databases and case law archives, which is time-consuming and prone to errors. Therefore, there is a need to leverage modern NLP and LLMs to effectively capture legal semantics and automate document retrieval.

To address these gaps, a RAG system can be designed, by integrating the strengths of information retrieval with text generation. RAG is particularly well-suited in the legal field where accurate and contextually relevant responses are critical. Unlike standalone LLMs, RAG first retrieves relevant information from a knowledge base and then generates answers, which ensures that responses are grounded in factual data and prevents hallucination.

1.3 Objectives

The objective of this project is to design and implement a RAG system for the legal domain, primarily for precedent search. The primary main users will be legal researchers, judges, and lawyers. Therefore, this system aims to alleviate the workload in the legal domain, by reducing the time and effort needed to identify precedent cases.

1.4 Significance

Precedent cases are important in the U.S. legal system as they form the cornerstone of common law. In the legal domain, professionals frequently have to search through large repositories of legal documents to look for relevant case law. This RAG system designed for precedent search can significantly reduce workloads across the legal domain, by retrieving relevant legal precedents and generating concise summaries. Legal professionals can use it to find past cases that support their arguments. Legal researchers can use it to identify relevant precedents with similar fact patterns and legal principles when analysing a new case. Judges can use it to help them make consistent rulings. Law firms can use it to find past rulings to support case preparation, draft legal documents, and refine litigation strategies. Law students and academics can use it to study case similarities, explore judicial reasoning, and improve legal research skills. Therefore, strealiming precedent search and case law retrieval addresses the major inefficiencies in the domain. This project aims to demonstrate the potential of AI in the legal domain, offering a more efficient and reliable method for precedent search.

2 Literature Review

2.1 Overview of NLP and LLM

NLP and LLMs have allowed machines to understand, interpret, and generate human language. Over the decades, NLP has progressed from early rule-based approaches to statistical methods. The field of NLP is further transformed by the introduction of deep learning, which led to the development of more sophisticated models that can understand complex language structures and semantics. This evolution has culminated in the creation of LLMs. The following subsections study the field of NLP and LLMs, including their evolution and challenges, focusing on text generation.

2.1.1 Evolution of NLP

“Can machines think?”, a famous question posed by Alan Turing posed in his 1950 paper Computing Machinery and Intelligence, where he explored whether machines could exhibit intelligent behavior indistinguishable from humans. His seminal work on machine intelligence and the introduction of the Turing Test, laid the groundwork for AI (Turing, 1950). Following that, the emergence of early NLP systems, such as ELIZA (Weizenbaum, 1966), which were able to process language by relying on handcrafted rules, have made basic human-computer interactions possible. However, these early systems were characterised by their rule-based approach, and limited by their ability to process language based on predefined syntactic structures and lacked the ability to understand context or semantics.

The 1990s saw a paradigm shift toward statistical methods, driven by the increasing availability of large text corpora and computational resources. Large annotated text corpora, prominently the Penn Treebank (Marcus et al., 1993) and the Brown Corpus (Manning & Schütze, 1999), provided the datasets to train statistical models and the development in computational resources made it possible to process these large datasets (Manning & Schütze, 1999). Techniques such as n-grams (Jurafsky & Martin, 2023) and Hidden Markov Models (HMMs) (Rabiner, 1989) were instrumental to the development of probabilistic language modeling. Conditional Random Fields (CRFs) further contributed to the probabilistic approaches in structured prediction tasks such as sequence labeling (Lafferty et al., 2001). This statistical approach resulted in applications like speech recognition and machine translation, notably exemplified by IBM’s statistical MT system (Brown et al., 1990), to flourish in this era.

The introduction of word embeddings and Recurrent Neural Networks (RNNs) marked another milestone in NLP, allowing models to capture semantic relationships and sequential dependencies. The emergence of Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al.,

2014) has allowed models to represent words in continuous vector spaces, known as word embeddings. This allows models to capture semantic and syntactic relationships between words, making it possible to perform tasks like solving analogies ("king - man + woman = queen") (Mikolov et al., 2013). Moreover, the creation of RNNs and their variants, like Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997), designed for sequential data (Rumelhart et al., 1986), has introduced sequence-to-sequence (Seq2Seq) models and advanced tasks like language modeling (Mikolov et al., 2010) and text generation (Graves, 2013). The developments in word embeddings and RNNs provide the foundation for modern NLP, which Transformers have since revolutionised.

2.1.2 Evolution of LLMs

LLMs are deep learning-based neural networks trained on vast amounts of text data to perform NLP tasks such as text generation, summarisation, and question-answering. The Transformer architecture underpins modern LLMs by leveraging self-attention mechanisms for efficient language modeling.

Before the rise of transformers, RNNs were the primary frameworks for Seq2Seq models (Sutskever et al., 2014). These models map input sequences to output sequences by preserving contextual information in hidden states for tasks where context and order matter. However, they face two main challenges: vanishing gradients and learning long-term dependencies (Hochreiter & Schmidhuber, 1997). As a result, variants of RNNs – LSTM networks (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Cho et al., 2014) – were introduced. LSTMs have a more sophisticated architecture with memory cells and gating mechanisms. The input, forget, and output gates allow LSTMs to retain or discard information over long sequences selectively. GRUs have a simpler architecture than LSTMs with only an update gate and a reset gate. The gating mechanisms, which are not previously present in traditional RNNs, allow both LSTMs and GRUs to address the two shortcomings of traditional RNNs. However, the fundamental limitations of RNN-based architectures persisted, particularly for tasks that require extensive contextual understanding.

The introduction of the Transformer architecture in the "Attention is All You Need" paper (Vaswani et al., 2017) fundamentally changed NLP. Transformers are the architecture behind most modern LLMs. Transformers use a self-attention mechanism to weigh the importance of different words in a sentence relative to each other, which allows the model to understand context and relationships between words more effectively than RNNs. Unlike RNNs, which process sequentially, transformers process entire sequences in parallel using self-attention mechanisms. This makes them much faster and more efficient for text generation. Transformers can also handle massive datasets and model sizes, which is why they are the foundation of LLMs with billions or even trillions of parameters.

2.1.3 Notable LLMs

Generative Pre-trained Transformer, commonly known as GPT, is OpenAI's GPT series. (Radford et al., 2018; Brown et al., 2020) GPT models autoregressive models trained on large-scale text corpora. These models utilise the attention mechanisms of transformers to predict the next token in a sequence to generate text. GPT-3 (175B parameters) demonstrated remarkable language generation capabilities and subsequent iterations, GPT-3.5 and GPT-4, show improved contextual understanding and response coherence. This resulted in the first version of ChatGPT, built on GPT-3.5, we know (OpenAI, 2022). Furthermore, fine-tuning techniques, including reinforcement learning from human feedback (RLHF), have improved GPT's ability to understand human intent, making it more reliable and adaptable across various domains (Ouyang et al., 2022).

LLaMA is a suite of transformer-based LLMs developed by Meta, with parameter sizes ranging from 7 billion to 65 billion. LLaMA models are trained exclusively on publicly available datasets, totaling trillions of tokens, demonstrating that cutting-edge performance can be achieved without relying on proprietary data. Notably, LLaMA-13B surpasses GPT-3 (175B parameters) on most benchmarks, and LLaMA-65B is competitive with leading models Chinchilla-70B and PaLM-540B. The dataset used to train these models comprises a diverse mix of sources – CommonCrawl (67%), C4 (15%), GitHub (4.5%), Wikipedia (4.5%), Books (4.5%), ArXiv (2.5%), and StackExchange (2%). Thus, ensuring comprehensive language representation and improving their ability to generalise across domains. By training smaller models on more data, LLaMA challenges the notion that better performance is solely based on more parameters, demonstrating the importance of extensive training data. The introduction of LLaMA models aims to democratise access to LLMs, making LLMs accessible to the resource community which allows researchers without substantial computational resources to conduct experiments. Additionally, their open-access nature makes them a popular choice for developing customised AI applications, including domain-specific applications and RAG systems. (Touvron et al., 2023)

2.1.4 Conclusion

For text generation, GPT models, especially GPT-3 and GPT-4, are widely regarded as the best among the LLMs. Given that GPT models are proprietary, their accessibility and customisation are limited compared to open-source models. On the other hand, LLaMA is a strong open-source alternative, designed to provide high-performance natural language understanding and generation while being more computationally efficient than traditional large-scale models. LLaMA is accessible to researchers, enabling greater experimentation in NLP. The model series features architectures optimised for efficiency, which requires fewer computational resources while maintaining strong performance on various tasks, including text generation. Given the accessibility of this model and limited computational resources of this project, LLaMA is well-suited for this project. However, LLMs experience various challenges with regards to text generation and these challenges should be considered before employing the model for the text generation task.

2.2 Text Generation Challenges

Text generation has seen significant advancements with the development of LLMs, however, their effectiveness is limited in real-world applications, especially in specialised domains such as law. The following subsections will be discussing some key text generation challenges and how these challenges manifest in the legal field.

2.2.1 Maintaining Coherence and Context

A key limitation of text generation is ensuring coherence and relevance across long passages. The struggle of LLMs to maintain logical consistency, especially in extended outputs, is due to multiple factors (Liu et al., 2023). Context window limitations restrict LLMs to a fixed number of tokens (e.g., 2048 or 4096), causing them to lose track of earlier context as text grows longer (Brown et al., 2020). The self-attention mechanism in Transformers becomes less effective as the text generated gets longer. (Vaswani et al., 2017). Training data biases mean LLMs may inherit inconsistencies or lack long-range coherence from their training data (Bender et al., 2021). The autoregressive nature of LLMs, allows errors to compound over time, causing the loss of context in extended outputs (Radford et al., 2019). This issue is amplified when generating domain-specific content, where the required context may span multiple documents. This is particularly problematic in the legal domain, where professionals often deal with extensive documents.

2.2.2 Hallucinations

LLMs are susceptible to generating outputs that, while coherent, may be factually incorrect – a phenomenon known as hallucination (Brown et al., 2020). LLMs are trained on vast amounts of text data, which may contain inaccuracies, biases, or gaps, leading the model to generate incorrect information when encountering underrepresented or misrepresented topics (Bender et al., 2021). Additionally, LLMs lack direct access to real-time or verified knowledge bases, relying solely on patterns learned during training. This can result in plausible-sounding but factually incorrect outputs, particularly for niche or evolving subjects (Ji et al., 2023). Furthermore, LLMs are optimised to produce fluent and contextually appropriate text, often prioritising coherence over factual accuracy, leading to outputs that "sound right" but are not factually correct (Zahraei & Shakeri, 2024). This is further exacerbated by poorly defined user prompts, the model may generate fabricated content to fill in gaps (Wei et al., 2022).

In the legal domain where precision is crucial, even minor inaccuracies can lead to significant errors. LLMs are more likely to generate responses based on learned patterns than real facts due to their probabilistic nature (Biesheuvel et al., 2024), which may increase the risk of fabricating legal statutes or misrepresenting case law. RAG can be used to prevent hallucinations by retrieving existing documents from external knowledge sources to ground the model's responses in factual information (Lewis et al., 2020).

2.2.3 Domain-Specific Adaptation

General-purpose LLMs are typically trained on diverse datasets that lack the specificity required for specialised applications. Legal language is highly structured, with precise terminology, statutory references, and jurisdiction-specific variations that general-purpose models may fail to capture. Legal text generation requires a nuanced understanding of terminology and conventions unique to the domain. Adapting LLMs to such specific contexts involves fine-tuning domain-specific datasets (Chalkidis et al., 2021). Fine-tuning LLMs on legal corpora, such as court rulings, contracts, and legislative documents, helps improve their ability to generate legally sound content. (Zhong et al., 2020).

2.2.4 Ethical and Bias Concerns

LLMs can inadvertently perpetuate biases present in their training data, leading to outputs that may be discriminatory, misleading, or unfair (Bender et al., 2021). Studies have shown that LLMs inherit and sometimes amplify stereotypes in their training data, which can impact their real-world applications (Sheng et al., 2019; Bolukbasi et al., 2016).

In the legal domain, such biases pose a significant risk. Legal language models are often used to assist with case analysis, precedent retrieval, and legal decision-making, where impartiality is critical. If a model disproportionately favours specific legal arguments or systematically underrepresents perspectives from marginalised communities, it could reinforce systemic inequalities in judicial outcomes. Empirical studies have demonstrated that predictive models used in legal settings, such as recidivism risk assessments, can exhibit racial and gender biases, raising ethical concerns about fairness and accountability (Angwin et al., 2016).

2.3 Retrieval-Augmented Generation (RAG)

2.3.1 Overview of RAG

RAG has emerged as a transformative paradigm in the field of NLP, designed for knowledge-intensive tasks. RAG integrates two primary components – a retriever and a generator. The retriever searches an external knowledge base for relevant information, while the generator uses the retrieved information to produce coherent and factually accurate responses (Lewis et al., 2020). By grounding generation in existing data, RAG reduces the risk of hallucinations, a common problem in standalone generative models.

RAG systems has significant advantages over conventional LLMs, particularly in three key areas: (1) enhanced factual accuracy through verifiable external knowledge integration and reducing hallucination; (2) better domain adaptability, enabling rapid deployment across specialised fields without extensive retraining; and (3) effective handling of extended contexts, with demonstrated performance improvements on long-form question answering and document summarisation tasks (Lewis et al., 2020). RAG can also be fine-tuned for specific domains by leveraging domain-specific corpora for retrieval, ensuring relevance and precision in specialised applications (Lewis et al., 2020). RAG's retrieval mechanism allows access to extensive external documents without requiring them to be processed in a single model pass. Transformer-based models are constrained by fixed input lengths – usually 512 or 2048 tokens – that restrict their ability to process large documents in a single pass (Vaswani et al., 2017). This is done by first retrieving a subset of relevant documents from a large external knowledge base before generating responses based on this retrieved information. This approach allows RAG to handle large knowledge bases without input length constraints (Lewis et al., 2020).

While RAG systems have many advantages, they also face challenges in the following areas: retrieval quality, domain-specific knowledge bases, and integration complexity. The quality of the retriever is crucial to the performance of a RAG system and poorly retrieved documents can introduce irrelevant or incorrect context, leading to suboptimal outputs (Izacard & Grave, 2021). Additionally, constructing and maintaining up-to-date, high-quality knowledge bases for retrieval in specialised domains can be resource-intensive. Although RAG's flexibility enables dynamic knowledge integration, its effectiveness depends on careful optimisation to efficiently integrate retrieval and generation models. However, if this is done right, it allows access to up-to-date or domain-specific information, which improves its effectiveness on knowledge-intensive tasks (Lewis et al., 2020).

2.3.2 RAG in Legal Domain

In the legal domain, professionals have to review and search through legal databases and case law archives, which is time-consuming and prone to errors. These documents are complex and contain legal jargon, which requires specialised knowledge to understand and analyse (Ashley, 2017). Traditional search engines, such as keyword-based systems, may not be able to capture different legal terminology and the legal context. Additionally, LLMs are prone to hallucinations and may generate inaccurate information that is not supported by real data (Ji et al., 2023). To address these existing issues in the legal domain, RAG can be implemented. Furthermore, their scalability allows them to process vast legal texts, which is especially valuable in jurisdictions with extensive legal databases.

Precedent cases are important in the U.S. legal system as they form the foundation of common law, ensuring consistency and fairness in judicial decisions. Under the principle of stare decisis (Loyola University Chicago Law Library, n.d.), courts are bound to follow rulings from higher courts in similar cases, which promotes uniformity in the application of the law. Precedents help interpret statutes, regulations, and constitutional provisions, especially when laws are ambiguous. They also allow lawyers to build stronger arguments by citing authoritative cases, while judges rely on them to make informed, legally sound decisions. Additionally, precedents adapt to evolving legal landscapes, addressing new issues such as technology and civil rights. By relying on past rulings, the U.S. legal system maintains integrity, reduces arbitrary decision-making, and upholds the rule of law.

Aletras et al. (2016) introduced machine learning-based methods for legal case retrieval, demonstrating that computational models can effectively identify similar cases based on textual similarity and legal principles. Building on such foundations, RAG introduces a more context-aware retrieval mechanism. To effectively implement RAG in legal research, it is essential to account for the complexities of legal language and the limitations of standalone LLMs. Ashley (2017) introduced the idea that legal texts are inherently complex, requiring domain-specific expertise and interdisciplinary collaboration to develop AI-driven legal solutions. Unlike general-purpose LLMs, which struggle with the intricacies of legal terminology and reasoning, domain-specific models fine-tuned on legal corpora are better equipped to handle the nuances of case law.

2.4 Summary

This literature review provides a comprehensive preliminary research of the field of NLP and LLMs. It focuses on text generation issues that standalone LLMs face, such as coherence, context awareness, and factual accuracy. Given the importance of accurate and reliable information in the legal context, these issues should not be underestimated.

To mitigate these challenges, RAG is proposed as a solution. This review explores the architecture of RAG systems, analysing their retrieval and generative components. By integrating a retrieval mechanism with an LLM, RAG addresses text generation challenges that standalone LLM face and improve the quality and relevance of generated responses. Furthermore, it delves into the designing and implementation of a RAG system in the legal domain, particularly for precedent search, to improve efficiency and legal decision-making.

Overall, this literature review explores the field of NLP and LLMs and their applications in the legal domain. The findings set the groundwork for the design and implementation of a RAG system tailored for legal precedent search that will be covered in the following sections.

3 System Design and Architecture

The RAG system consists of two main stages – knowledge base building and query processing. The knowledge base is built and stored as a FAISS index. After the knowledge base is built, it is integrated into the RAG pipeline to process query, consisting of two phases – retrieval and generation phase. The following is an illustration of the overall architecture of this RAG system.

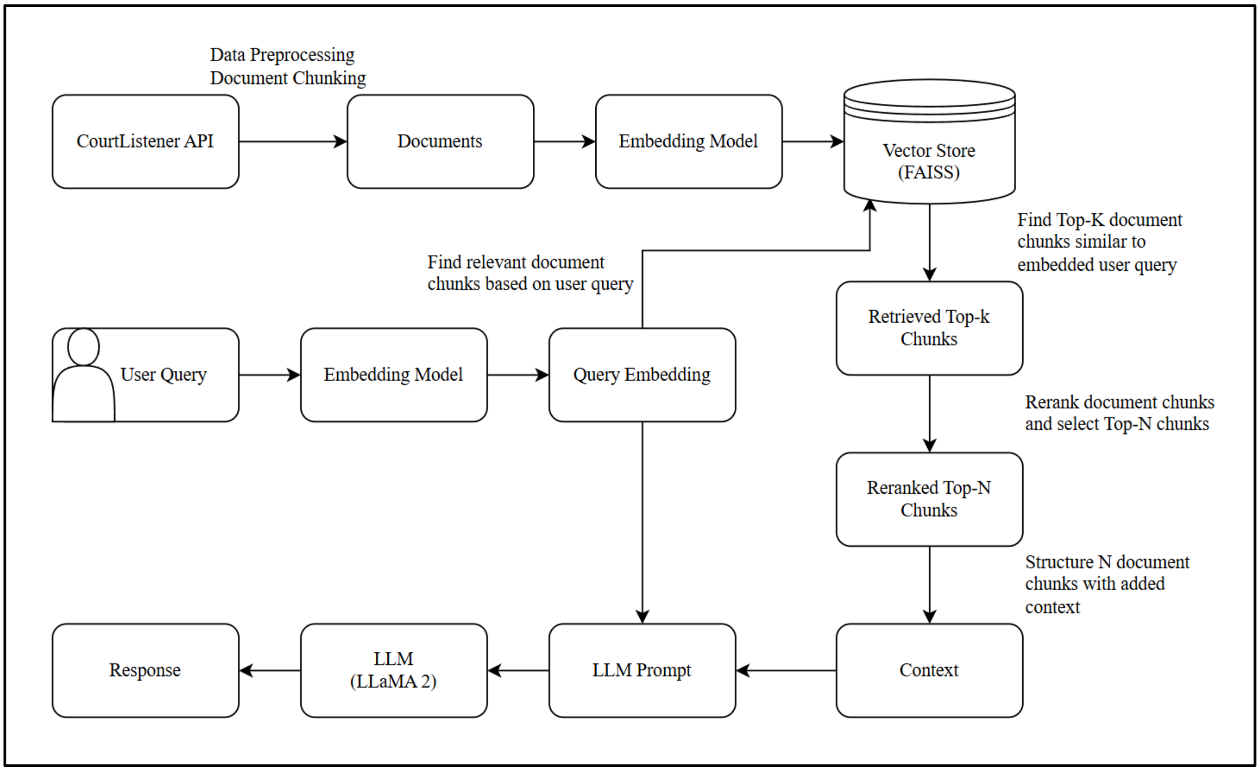

Fig. 1: RAG Architecture

3.1 Knowledge Base

3.1.1 Legal Case Data

The knowledge base is built using legal case data fetched from the CourtListener API, a publicly available and open-source platform that provides access to a comprehensive collection of U.S. legal opinions, which includes both federal and state court rulings. CourtListener serves as a centralised repository for legal case data, offering legal professionals and researchers a valuable resource to easily access, analyse, and reference judicial opinions.

The records are retrieved from the API by making paginated requests to the endpoint "https://www.courtlistener.com/api/rest/v4/opinions/". The records are retrieved by iterating through the pages. Each record retrieved represents a legal opinion, with various fields that capture specific details about the case. Fig. 2 illustrates a record obtained from the CourtListener API.


```
{
    "resource_uri": "https://www.courtlistener.com/api/rest/v4/opinions/10814935/",
    "id": 10814935,
    "absolute_url": "/opinion/10348347/lemuette-krinisha-mckinney-v-the-state-of-texas/",
    "cluster_id": 10348347,
    "cluster": "https://www.courtlistener.com/api/rest/v4/clusters/10348347/",
    "author_id": null,
    "author": null,
    "joined_by": [],
    "date_created": "2025-03-01T07:19:39.539318-08:00",
    "date_modified": "2025-03-01T07:19:39.767064-08:00",
    "author_str": "",
    "per_curiam": false,
    "joined_by_str": "",
    "type": "010combined",
    "sha1": "56053327b494a72366ecaee906b94de2f069dd47",
    "page_count": 1,
    "download_url": "https://search.txcourts.gov/SearchMedia.aspx?MediaID=bff592ad-faab-4315-b44c-6b9ac48c63de",
    "local_path": "pdf/2025/02/27/lemuette_krinisha_mckinney_v._the_state_of_texas.pdf",
    "plain_text": "                    THE THIRTEENTH COURT OF APPEALS\n\n                        13-24-00529-CR\n\n\n
    "html": "",
    "html_lawbox": "",
    "html_columbia": "",
    "html_anon_2020": "",
    "xml_harvard": "",
    "html_with_citations": "",
    "extracted_by_ocr": false,
    "ordering_key": null,
    "opinions_cited": []
},
```


Fig. 2: Example of a Record Retrieved from the CourtListener API (CourtListener, n.d.)

From above, we observe that each record contains multiple fields and the following table is a breakdown of these fields:

Field	Description
resource_uri	A direct API link to the opinion.
id	Unique identifier for the opinion.
absolute_url	The URL to view the opinion on CourtListener's website.
cluster_id	Identifier grouping related opinions together.
cluster	API link to the opinion cluster.
author_id	Identifier for the judge or panel authoring the opinion.
author	Information about the judge or panel authoring the opinion.
joined_by	List of judges who joined the opinion.
date_created	Timestamp when the opinion was added to CourtListener.
date_modified	Timestamp when the opinion was last modified.
author_str	String representation of the author.
per_curiam	Boolean indicating if the decision was unsigned and issued by the court collectively.
joined_by_str	String representation of judges who joined the opinion.
type	Classification of the opinion.
sha1	SHA-1 hash of the opinion text for integrity verification.

page_count	Number of pages in the opinion.
download_url	Direct link to download the opinion as a PDF.
local_path	Local storage path where the PDF file is stored.
plain_text	Extracted plain text version of the opinion.
html	Full text of the legal opinion in HTML format, including all sections.
html_lawbox	Cleaned-up version of the opinion, focusing on the main legal text.
html_columnia	Structured HTML format optimized for readability and citation formatting.
html_anon_2020	Redacted HTML version of the opinion with anonymised personal information.
xml_harvard	XML-formatted version of the opinion, structured for legal research and citation parsing.
html_with_citations	HTML version of the opinion with enhanced citation links for better reference navigation.
extracted_by_ocr	Boolean indicating whether OCR was used to extract text.
ordering_key	Used for sorting purposes.
opinions_cited	A list of other opinions cited in this case, with direct API links.

Table 1: Legal Opinion Fields and Descriptions (CourtListener, n.d.)

The “plain_text” field is used for extracting the core content of legal opinions. This field provides the unstructured textual data of the opinion, which will be used as context to generate responses. The remaining fields are stored as metadata, which provide structured information about the legal opinion. As a result, the following table shows how each record is stored after retrieval.

Field	**Description**
text	This is extracted from the “plain_text” field and used as the core content.
metadata	This contains the fields: “resource_uri”, “id”, “absolute_url”, “cluster_id”, “cluster”, “author_id”, “author”, “joined_by”, “date_created”, “date_modified”, “author_str”, “per_curiam”, “joined_by_str”, “type”, “sha1”, “page_count”, “download_url”, “local_path”, “html”, “html_lawbox”, “html_columnia”, “html_anon_2020”, “xml_harvard”, “html_with_citations”, “extracted_by_ocr”, “ordering_key” and “opinions_cited”.

Table 2: Legal Opinion “text” and “metadata”

3.1.2 Data Preprocessing

The retrieved legal records are processed by extracting the “plain_text” field as the main content “text” while the remaining fields are extracted as “metadata”. The main content “text” contains lengthy text that is raw, unstructured and noisy, thus preprocessing is required. The main content undergoes preprocessing which consists of two key stages – data cleaning and document chunking. This ensures that the main content is cleaned, standardised and broken down into smaller chunks so that embeddings generated are accurate and meaningful before being indexed into a vector store. This ultimately ensures that the vector store is optimised for downstream retrieval of relevant text for query processing.

3.1.2.1 Data Cleaning

The raw legal texts retrieved from our endpoint are too noisy as they contain inconsistencies, excessive whitespace and annotations. Data cleaning is important to ensure that the text is structured, standardised, and retains its legal significance while removing irrelevant artifacts, which optimises downstream processes – generating embeddings and storing them into a vector store. The following illustrates an example of a typical legal text retrieved:

```
RAW_KNOWLEDGE_BASE[0]["plain_text"]
```

```
'UNITED STATES DISTRICT COURT                                              \nSOUTHERN DISTRICT OF NEW YORK                                        \n
GILBERT BAHAMUNDI,                                                      \n                   Plaintiff,                                    \n
25-CV-412 (LTS)                \n             -against-                                                     \n                                       ORDER DIR
ECTING SIGNATURE             \nALEXANDRE ZAPLETHAL, et al.,                                              \n                   Defendants.
\nLAURA TAYLOR SWAIN, Chief United States District Judge:                    \n    Plaintiff brings this action pro se. The complaint that Plaintiff submi
tted is unsigned. Rule \n11(a) of the Federal Rules of Civil Procedure provides that “[e]very pleading, written motion, \nand other paper must be signed
. . . by a party personally if the party is unrepresented.” Fed. R. \nCiv. P. 11(a); see Becker v. Montgomery, 532 U.S. 757, 764 (2001) (interpreting Rul
e 11(a) to \nrequire, “as it did in John Hancock’s day, a name handwritten (or a mark handplaced)”).1  \n    Plaintiff is directed to sign and submit the
attached signature page within 30 days of the \ndate of this order. If Plaintiff returns the signed document by mail or in person, it must have a \nhandw
ritten signature that complies with Rule 11(a). If Plaintiff submits the document by email, \nto ProSe@nysd.uscourts.gov, Plaintiff may use instead an el
ectronic signature or a typed name \nwith /s/ (“/s/ Gilbert Bahamundi”) on the signature line. The signed document must be labeled \nwith the docket numb
er 25-CV-411 (LTS).                                  \n\n    1 Although a “typed name” does not satisfy Rule 11(a)’s signature requirement, the \nSuprem
e Court affirmed the right of courts “by local rule [to] permit papers to be filed, signed, or \nverified by electronic means.” Becker, 532 U.S. at 764.
Under this court’s local rules, where a \ndocument is filed in accordance with the SDNY Electronic Case Filing (“ECF”) Rules & \nInstructions (“ECF Rule
s”), the filing complies with the local rules. See Local Civil Rule 5.2. \nRule 1.1 and Appendix C of the ECF Rules authorize self-represented parties to
sign documents \nsubmitted to the court by email using an electronic signature or typed name with /s. \n    No summons or answer shall issue at this tim
e. If Plaintiff complies with this order, the \ncase shall be processed in accordance with the procedures of the Clerk’s Office. If Plaintiff fails \nto
comply with this order within the time allowed, the action will be dismissed without prejudice. \n    The Court certifies under 28 U.S.C. § 1915(a)(3) th
at any appeal from this order would \n\nnot be taken in good faith, and therefore in forma pauperis status is denied for the purpose of an \nappeal. Cf.
Coppedge v. United States, 369 U.S. 438, 444–45 (1962) (holding that appellant \ndemonstrates good faith when seeking review of a nonfrivolous issue).
\nSO ORDERED.                                                              \nDated:  March 6, 2025                                                  \n
New York, New York                                                 \n\n                                       /s/ Laura Taylor Swain                \n
LAURA TAYLOR SWAIN                 \n                                    Chief United States District Judge   '
```

Fig. 3: Legal Text Before Data Cleaning

Based on the above example which represents the majority of texts retrieved from our endpoint, the following are the key steps designed to clean legal texts:

1. Removing Excessive Whitespace and Line Breaks
2. Preserving Legal Citations
3. Handling Case Headers
4. Managing Footnotes and Annotations
5. Normalising Punctuation Spacing

As a result, the main content is consistent, structured and key information is preserved. The following shows the same text in Fig. 4 after data cleaning:

```
text = RAW_KNOWLEDGE_BASE[0]["plain_text"]
clean_text(text)
```

```
"USCA4 Appeal: 24-1377 Doc: 37 Filed: 03/21/2025 Pg: 1 of 18 UNPUBLISHED UNITED STATES COURT OF APPEALS FOR THE FOURTH CIRCUIT No. 24-1377 BRADFORD M.
KENDRICK, Plaintiff – Appellant, v. CARTER BANK & TRUST, INC., Defendant – Appellee. Appeal from the United States District Court for the Western Distr
ict of Virginia, at Danville. Elizabeth K. Dillon, Chief District Judge. (4:19-cv-00047-EKD) Argued: January 29, 2025 Decided: March 21, 2025 Before KI
NG and THACKER, Circuit Judges, and FLOYD, Senior Circuit Judge. Affirmed by unpublished opinion. Judge King wrote the opinion, in which Judge Thacker
and Judge Floyd joined.\n ARGUED: Terry Neill Grimes, TERRY N. GRIMES, ESQ., PC, Roanoke, Virginia, for Appellant. King Fitchett Tower, WOODS ROGERS VA
NDEVENTER BLACK PLC, Roanoke, Virginia, for Appellee.\n ON BRIEF: Kaley J. Gordon-Shupp, TERRY N. GRIMES, ESQ., PC, Roanoke, Virginia, for Appellant. J
oshua Richard Treece, Christine S. Ward, WOODS ROGERS VANDEVENTER BLACK PLC, Roanoke, Virginia, for Appellee. Unpublished opinions are not binding prec
edent in this circuit. USCA4 Appeal: 24-1377 Doc: 37 Filed: 03/21/2025 Pg: 2 of 18 KING, Circuit Judge: Plaintiff Bradford M. Kendrick, a 35-year emplo
yee of Carter Bank & Trust, Inc. (“Carter Bank,” or the “Bank”) and its predecessors, pursues this appeal from an adverse judgment of the Western Distr
ict of Virginia. Kendrick primarily challenges the district court’s award of summary judgment to Carter Bank on claims of disparate treatment, hostile
work environment, and retaliation, pursued under the Age Discrimination in Employment Act of 1967 (the “ADEA”). Kendrick also challenges two adverse di
scovery rulings made in connection with the summary judgment record, that is,  the court’s rejection of Kendrick’s effort to depose a Director of the B
ank, who had also served as its counsel, and  the court’s rejection of Kendrick’s effort to expand the summary judgment record by filing a tardy “Notic
e of Supplemental Facts.” As explained herein, the court did not abuse its discretion in connection with either of those discovery rulings. In additio
n, the summary judgment award was not flawed, and the court’s rejection of two of Kendrick’s ADEA claims was also justified by Kendrick’s misconduct in
connection with his mishandling of Bank records. We therefore affirm the challenged judgment. I. A. In May 1985, plaintiff Kendrick was hired at what i
s now Carter Bank by an individual named Worth Carter. Mr. Carter was the Bank’s primary founder, and he later served as its Chief Executive Officer
(“CFO”), as well as Chairman of its Board of Directors. Kendrick was promoted several times during his 35-year tenure with the Bank. 2 USCA4 Appeal: 24
```

Fig. 4: Legal Text After Data Data Cleaning

3.1.2.2 Document Chunking

After data cleaning, the legal documents in the main context undergo document chunking and are embedded into vector representations which are stored along with their respective metadata in a vector store. This process is required as long legal texts make direct embedding and retrieval inefficient. Chunking breaks these documents into smaller segments, which allows the embedding model to capture the semantic meaning of each segment more accurately and avoid diluting the context with overly broad content. Thus, making it easier for the vector store to retrieve highly relevant chunks during a query.

Additionally, this process also ensures that each chunk complies with the technical constraints of the embedding model used in the downstream process. Each chunk has to contain a token number less than the maximum token limit of the embedding model used to prevent truncation and loss of information during embedding.

To implement document chunking, a semantic chunking approach is chosen over arbitrary fixed-length splits to maintain contextual integrity. The “sentence-transformers/all-mpnet-base-v2” is chosen to be the embedding model. The following are design considerations that are taken into account when developing the chunking strategy:

1. Chunk Size:
    - Chunking allows the system to retrieve the most relevant segments, rather than scanning entire documents.
    - Larger chunks retain more context but may reduce retrieval precision, whereas smaller chunks improve precision but risk losing context.
    - The maximum token length of the “sentence-transformers/all-mpnet-base-v2” model is 384 tokens imposing a hard limit on maximum chunk size.
    - After evaluation, a chunk size of 384 tokens was selected as it strikes a balance between retaining sufficient context and ensuring efficient retrieval.
2. Overlap:
    - Overlap is used to prevent the loss of context and ensures that information is not truncated at chunk boundaries.
    - The use of overlapping chunks increases the amount of legal texts to be indexed, thus increasing computational overhead. However, this trade-off is necessary as it is important to preserve as much context as possible thereby improving the quality of the knowledge base.
    - Therefore, an overlap of 100 tokens was selected between consecutive chunks.

3. Tokenization:
    - Each model has a unique tokenizer that splits text into tokens – words or subwords – according to its specific rules. Given the same text, different tokenizers can result in different numbers of tokens.
    - Therefore, the tokenizer associated with the “sentence-transformers/all-mpnet-base-v2” model was selected to ensure alignment of document chunking and embedding process.

As a result, this chunking process helps the legal documents to preserve context while dividing the text into manageable chunks that comply with the model's token length constraints. This ensures that embeddings generated are contextually accurate before being indexed into a vector store. Therefore, optimising downstream retrieval of relevant text for query processing. The following illustrates an example of document chunking:

Original Document:

```
text = RAW_KNOWLEDGE_BASE[0]["plain_text"]
clean_text(text)
```

```
"USCA4 Appeal: 24-1377 Doc: 37 Filed: 03/21/2025 Pg: 1 of 18 UNPUBLISHED UNITED STATES COURT OF APPEALS FOR THE FOURTH CIRCUIT No. 24-1377 BRADFORD M.
KENDRICK, Plaintiff – Appellant, v. CARTER BANK & TRUST, INC., Defendant – Appellee. Appeal from the United States District Court for the Western Distr
ict of Virginia, at Danville. Elizabeth K. Dillon, Chief District Judge. (4:19-cv-00047-EKD) Argued: January 29, 2025 Decided: March 21, 2025 Before KI
NG and THACKER, Circuit Judges, and FLOYD, Senior Circuit Judge. Affirmed by unpublished opinion. Judge King wrote the opinion, in which Judge Thacker
and Judge Floyd joined.\n ARGUED: Terry Neill Grimes, TERRY N. GRIMES, ESQ., PC, Roanoke, Virginia, for Appellant. King Fitchett Tower, WOODS ROGERS VA
NDEVENTER BLACK PLC, Roanoke, Virginia, for Appellee.\n ON BRIEF: Kaley J. Gordon-Shupp, TERRY N. GRIMES, ESQ., PC, Roanoke, Virginia, for Appellant. J
oshua Richard Treece, Christine S. Ward, WOODS ROGERS VANDEVENTER BLACK PLC, Roanoke, Virginia, for Appellee. Unpublished opinions are not binding prec
edent in this circuit. USCA4 Appeal: 24-1377 Doc: 37 Filed: 03/21/2025 Pg: 2 of 18 KING, Circuit Judge: Plaintiff Bradford M. Kendrick, a 35-year emplo
yee of Carter Bank & Trust, Inc. (“Carter Bank,” or the “Bank”) and its predecessors, pursues this appeal from an adverse judgment of the Western Distr
ict of Virginia. Kendrick primarily challenges the district court’s award of summary judgment to Carter Bank on claims of disparate treatment, hostile
work environment, and retaliation, pursued under the Age Discrimination in Employment Act of 1967 (the “ADEA”). Kendrick also challenges two adverse di
scovery rulings made in connection with the summary judgment record, that is,  the court’s rejection of Kendrick’s effort to depose a Director of the B
ank, who had also served as its counsel, and  the court’s rejection of Kendrick’s effort to expand the summary judgment record by filing a tardy “Notic
e of Supplemental Facts.” As explained herein, the court did not abuse its discretion in connection with either of those discovery rulings. In additio
n, the summary judgment award was not flawed, and the court’s rejection of two of Kendrick’s ADEA claims was also justified by Kendrick’s misconduct in
connection with his mishandling of Bank records. We therefore affirm the challenged judgment. I. A. In May 1985, plaintiff Kendrick was hired at what i
s now Carter Bank by an individual named Worth Carter. Mr. Carter was the Bank’s primary founder, and he later served as its Chief Executive Officer
(“CFO”), as well as Chairman of its Board of Directors. Kendrick was promoted several times during his 35-year tenure with the Bank. 2 USCA4 Appeal: 24
```

Chunk 1:

```
docs_processed[0]["text"]
```

```
'usca4 appeal : 24 - 1377 doc : 37 filed : 03 / 21 / 2025 pg : 1 of 18 unpublished united states court of appeals for the fourth circuit no. 24 - 1377 br
adford m. kendrick, plaintiff – appellant, v. carter bank & trust, inc., defendant – appellee. appeal from the united states district court for the weste
rn district of virginia, at danville. elizabeth k. dillon, chief district judge. ( 4 : 19 - cv - 00047 - ekd ) argued : january 29, 2025 decided : march
21, 2025 before king and thacker, circuit judges, and floyd, senior circuit judge. affirmed by unpublished opinion. judge king wrote the opinion, in whic
h judge thacker and judge floyd joined. argued : terry neill grimes, terry n. grimes, esq., pc, roanoke, virginia, for appellant. king fitchett tower, wo
ods rogers vandeventer black plc, roanoke, virginia, for appellee. on brief : kaley j. gordon - shupp, terry n. grimes, esq., pc, roanoke, virginia, for
appellant. joshua richard treece, christine s. ward, woods rogers vandeventer black plc, roanoke, virginia, for appellee. unpublished opinions are not bi
nding precedent in this circuit. usca4 appeal : 24 - 1377 doc : 37 filed : 03 / 21 / 2025 pg : 2 of 18 king, circuit judge : plaintiff bradford m. kendri
ck, a 35 - year employee of carter bank & trust, inc. ( “ carter bank, ” or the “ bank ” ) and its predecessors, pursues this appeal from an adverse judg
ment of the western district of virginia.'
```

Chunk 2:

```
docs_processed[1]["text"]

', roanoke, virginia, for appellee. unpublished opinions are not binding precedent in this circuit. usca4 appeal : 24 - 1377 doc : 37 filed : 03 / 21 / 2
025 pg : 2 of 18 king, circuit judge : plaintiff bradford m. kendrick, a 35 - year employee of carter bank & trust, inc. ( “ carter bank, ” or the “ bank
” ) and its predecessors, pursues this appeal from an adverse judgment of the western district of virginia. kendrick primarily challenges the district co
urt ’ s award of summary judgment to carter bank on claims of disparate treatment, hostile work environment, and retaliation, pursued under the age discr
imination in employment act of 1967 ( the “ adea ” ). kendrick also challenges two adverse discovery rulings made in connection with the summary judgment
record, that is, the court ’ s rejection of kendrick ’ s effort to depose a director of the bank, who had also served as its counsel, and the court ’ s r
ejection of kendrick ’ s effort to expand the summary judgment record by filing a tardy “ notice of supplemental facts. ” as explained herein, the court
did not abuse its discretion in connection with either of those discovery rulings. in addition, the summary judgment award was not flawed, and the court
’ s rejection of two of kendrick ’ s adea claims was also justified by kendrick ’ s misconduct in connection with his mishandling of bank records. we the
refore affirm the challenged judgment. i. a. in may 1985, plaintiff kendrick was hired at what is now carter bank by an individual named worth carter. m
r. carter was the bank ’ s primary founder, and he later served as its chief executive officer ( “ ceo ” ), as well as chairman of its board of director
s.'
```

The original documents were broken down into ten chunks but only two chunks are displayed for illustration.

3.1.3 Vector Store

After preprocessing, the legal document chunks along with their respective metadata are indexed into a vector store. The legal document chunks are first embedded into dense vector representations before storing in the FAISS index. The system integrates “sentence-transformers/all-mpnet-base-v2” model as the embedding model and Facebook AI Similarity Search (FAISS) as the vector store to index and retrieve legal opinions.

The following outlines the general steps for this stage:

1. Converting Each Record to LangChain Document objects
2. Generating Embeddings for Each Legal Document Chunk
3. Indexing the Vectorised Chunks

3.1.3.1 LangChain's Document

Each record is first structured as a LangChain Document object before embedding and indexing into FAISS. LangChain's Document is a fundamental data structure designed to store and manage text data efficiently in RAG systems. It provides a standardised format for handling textual content along with associated metadata.

Each LangChain Document consists of two primary components:

1. page_content (str): The preprocessed chunk of legal text.
2. metadata (dict): The metadata dictionary (refer to Table 2).

LangChain's Document structure is selected for the following reasons:

1. Consistent Data Structure
   - Ensures uniform formatting across all indexed texts.
2. Metadata for Contextual Search
   - Enables filtering and ranking based on metadata attributes (e.g., retrieving only Supreme Court cases or employment law precedents).
3. Seamless Integration with LangChain’s RAG Pipeline
   - The Document format is directly compatible with LangChain’s retrievers, vector stores (in this case FAISS), and LLM-based response generation.

As a result, each record is now a LangChain Document object containing page_content and metadata.

3.1.3.2 Embedding Model

To embed legal document chunks into 768-dimensional vector representations, the system employs the “sentence-transformers/all-mpnet-base-v2” model from Hugging Face (Reimers & Gurevych, 2019). This transformer-based embedding model is optimised for capturing semantic relationships between texts, ensuring that similar legal cases are mapped closer in the embedding space. The all-mpnet-base-v2 model, based on Microsoft's MPNet, is trained using a contrastive learning objective on a diverse corpus, making it well-suited for high-quality sentence embeddings (Song et al., 2020). While trained on a diverse corpus, MPNet-based models have demonstrated strong zero-shot performance across multiple domains, including legal texts. This makes it a robust choice for legal document retrieval without requiring extensive fine-tuning.

3.1.3.3 FAISS Index

After the legal texts are embedded, these vectorised texts along with their respective metadata are stored and indexed into a vector store. FAISS is selected as the vector store for its efficiency in handling large-scale vector searches and retrieving relevant documents based on semantic similarity. FAISS (Facebook AI Similarity Search) is an open-source library developed by Meta AI for efficient similarity search and clustering of high-dimensional vectors (Douze et al., 2025). It enables fast nearest neighbor search even on billion-scale datasets through optimised indexing structures and GPU acceleration (Johnson et al., 2019).

FAISS is selected for the following reasons:

1. Speed and Scalability:
    - FAISS is optimised for high-speed searches, capable of handling extensive datasets containing millions or even billions of vectors. Its indexing structures significantly reduce search times, making it suitable for real-time retrieval of relevant documents when processing user queries. (MyScale, n.d)
2. Memory Efficiency:
    - FAISS employs techniques like Product Quantization (PQ) to compress vectors, reducing memory consumption without substantially compromising accuracy and reducing search time. (DataCamp, n.d.)
    - This makes it well-suited for this project and helps to manage large-scale legal documents efficiently.
3. GPU Acceleration:
    - FAISS provides GPU implementations which boost performance and enable rapid processing of large legal documents. (Facebook Engineering, n.d.)
4. Retrieval Method:
    - FAISS supports various distance metrics that are used to measure the relevance of retrieved texts. This allows for similarity-based retrieval. (DataCamp, n.d.)
    - The cosine similarity distance metric is chosen.

Therefore, the FAISS index reduces latency when processing user queries. Finally, the knowledge base is built and ready to be integrated into the RAG for user queries.

3.2 RAG Query Pipeline

An end-to-end query pipeline is implemented to process user queries in two main phases – retrieval phase and generative phase. This system retrieves relevant documents from the FAISS vector store and generates responses using the LLaMA 2 model. Therefore, this ensures that generated responses are grounded in actual legal texts, improving the quality of generated responses.

When a user submits a query, the pipeline executes in two main phases and the general steps are outlined in the following:

1. Retrieval Phase
    - When a user submits a query, the query is embedded using the same embedding model used to index the legal documents.
    - The system queries the FAISS vector store to retrieve the top-K most relevant text chunks based on similarity to the user query.
    - After retrieving the top-k most relevant text chunks, the chunks are reranked based on relevance score and top-N (N<=K) most relevant text chunks are passed onto the generation phase.
2. Generation Phase
    - Prompt Construction: The retrieved text chunks are formatted into a structured prompt for the LLaMA 2 model. The prompt includes:
        - The user query
        - The retrieved legal text
        - Context
        - Structured output
    - Response Generation with LLaMA 2: The LLaMA 2 model takes the structured prompt and generates a response.

3.2.1 Retrieval Phase

The retrieval phase bridges the gap between the user's query and the underlying legal knowledge base. This phase is responsible for retrieving and reranking the most relevant legal texts based on the user's query. This ensures that only the most relevant texts are passed to the LLaMA 2 model. The following outlines the key steps in the retrieval phase:

1. User Query Embedding:
    - The user's query is embedded using the same embedding model "meta-llama/Llama-2-7b-chat-hf", used for the FAISS vector store. This is handled internally by the vector store.
2. Retrieving from Vector Store:
    - The system queries the vector store to retrieve the top-K most relevant text chunks based on similarity to the query.
    - The cosine similarity metric is chosen during FAISS indexing for similarity search.
3. Reranking of Retrieved Text:
    - After retrieving the top-k most relevant text chunks, not all the retrieved chunks may be the most contextually relevant to the query, therefore reranking is used to only retrieve the most relevant texts.
    - The text chunks are reranked using cross-encoder model "cross-encoder/ms-marco-MiniLM-L-6-v2" which evaluates how well each text answers the query in a more refined and context-aware manner, beyond just the vector similarity.
    - The results are then ranked based on their relevance scores and the top-N (N<=K) texts are extracted and passed onto the generation phase.
4. Metadata Integration:
    - Metadata can be optionally used to filter or refine the search results.

The following illustrates the retrieval of top-K (K=10) most relevant text chunks:

```
query = "Find cases similar to Miranda v. Arizona, 384 U.S. 436 (1966)."

top_results = vector_store.similarity_search(query, k=10)
for i, result in enumerate(top_results):
    print(f"Result {i + 1}:\n{result.page_content}\n")
```

```
Result 1:
concern events that occurred as early as october 27, 2021, and include two arrests on november 11, 2021, and december 23, 2021. ( second am. compl. [ #
27 ], ¶ ¶ 1, 15, 25, 33, 38. ) 3 she alleges that her first arrest involved an unlawful traffic stop by defendant hawk, an unlawful search of her car b
y two defendant john does, and an unlawful detention, all in violation of the fourth amendment. ( second am. compl. [ # 27 ], ¶ ¶ 15, 20 - 21. ) she fu
rther alleges that, while in custody, her fifth amendment rights were violated when the police continued to question her despite invoking her right to
remain silent ; her sixth amendment right to counsel was violated ; her fourteenth amendment right to due process was violated due to insufficient medi
cal care ; and her rights under the americans with disabilities act ( " ada " ) were violated. ( second am. compl. [ # 27 ], ¶ ¶ 25, 28 - 31. ) as to m
s. bays ' s second arrest, which occurred on december 23, 2021, she pleads that defendant diaz " initiated a false arrest " in violation of her fourth
amendment rights by committing perjury in his affidavit submitted to the guadalupe county judge. ( second am. compl. [ # 27 ], ¶ ¶ 33 - 34. ) she claim
s that as she was being taken into custody, the officers denied her multiple requests to bring her necessary medications with her. ( second am. compl.
[ # 27 ], ¶ ¶ 39 - 40. ) ms. bays then alleges that during her overnight detainment, she experienced a medical emergency but was repeatedly denied medi
cal care and denied access to her medications. ( second am. compl. [ # 27 ], ¶ ¶ 50, 53 - 56. )

Result 2:
##onstituted defendant ' s point headings to correspond to the manner in which we address the issues. 3 miranda v. arizona, 384 u. s. 436. a - 0798 - 2
2 2 b. the failure to have defendant acknowledge and initial each miranda right. ii. the december 8 [ ] statement must be suppressed due to the failure
to deliver fresh miranda warnings as [ the ] state cannot meet its burden of proving defendant ' s waiver beyond a reasonable doubt. iii. both the supp
ression and new trial judges erred in denying the motion to suppress the december 7 [ ], 2016 statement due to the state ' s prosecutorial misconduct i
n its failure to electronically record the statement of december 7th and fail [ ure ] to videotape the december 8 [ ] statement pursuant to r [ ule ] 3
: 17 in violation of defendant ' s due process rights. iv. the december 8 [ ] statement must be suppressed due to the deprivation of defendant ' s sixt
h amendment right to counsel and fourteenth amendment due process right [ s ]. v. the prosecutor committed misconduct due to the improper grand jury pr
esentation. vi. the prosecutor committed misconduct by presenting the false testimony of [ detective ] hassloch at the miranda hearing as to reading th
e entire miranda form to defendant. a - 0798 - 22 vii. the prosecutor committed misconduct in her summation. viii. the prosecutor committed misconduc
t by moving in bad faith to vacate defendant ' s proper guilty plea of june 25, 2019. ix. the state deprived defendant of his right to counsel by not p
```

Fig. 5: Top-K Texts Retrieved

The following illustrates the reranking of top-3 (N=3) text chunks:

```
top_results = rerank_documents(query, top_results, reranker_model)
for i, result in enumerate(top_results):
    print(f"Result {i + 1}:\n{result.page_content}\n")
```

```
Result 1:
##onstituted defendant ' s point headings to correspond to the manner in which we address the issues. 3 miranda v. arizona, 384 u. s. 436. a - 0798 - 22
2 b. the failure to have defendant acknowledge and initial each miranda right. ii. the december 8 [ ] statement must be suppressed due to the failure to
deliver fresh miranda warnings as [ the ] state cannot meet its burden of proving defendant ' s waiver beyond a reasonable doubt. iii. both the suppressi
on and new trial judges erred in denying the motion to suppress the december 7 [ ], 2016 statement due to the state ' s prosecutorial misconduct in its f
ailure to electronically record the statement of december 7th and fail [ ure ] to videotape the december 8 [ ] statement pursuant to r [ ule ] 3 : 17 in
violation of defendant ' s due process rights. iv. the december 8 [ ] statement must be suppressed due to the deprivation of defendant ' s sixth amendmen
t right to counsel and fourteenth amendment due process right [ s ]. v. the prosecutor committed misconduct due to the improper grand jury presentation.
vi. the prosecutor committed misconduct by presenting the false testimony of [ detective ] hassloch at the miranda hearing as to reading the entire miran
da form to defendant. a - 0798 - 22 3 vii. the prosecutor committed misconduct in her summation. viii. the prosecutor committed misconduct by moving in b
ad faith to vacate defendant ' s proper guilty plea of june 25, 2019. ix. the state deprived defendant of his right to counsel by not providing a timely
telephone call. x. the trial court abused its discretion in granting the state ' s fresh complaint motion as to statements made by m. w. to a. w. and at
ginnie ' s house. xi.

Result 2:
. j. at 532. this includes the interpretation of the court rules. state v. tate, 220 n. j. 393, 405 ; state v. anthony, 443 n. j. super. 553, 564 ( app.
div. 2016 ). it also includes the court ' s conclusions as to the validity of a defendant ' s waiver of the right against self - incrimination, and the v
oluntariness of a defendant ' s statement. arizona v. fulminante, 499 u. s. 279, 287 ; miller v. fenton, 11 in his motion for a new trial, defendant argu
ed it was error for the court to have admitted his statements to the police. the court rejected defendant ' s arguments, finding no basis to disagree wit
h the reasoning of the judge who decided the pretrial motion, and denied the motion for a new trial. a - 0798 - 22 69 474 u. s. 104, 110 ; state v. o. d.
a. - c., 250 n. j. 408, 425. issues not raised before the trial court are reviewed for plain error, that is, error clearly capable of producing an unjust
result. r. 2 : 10 - 2. applying the aforementioned standards of review to the issues defendant raises in points i and iii, we conclude the court properly
denied defendant ' s motion to suppress his statements on december 7 and 8, 2016 on the basis of any alleged miranda violations and failures to record th
e interview. 12 a. as noted, in point i, defendant contends his december 7, 2016 statement should have been suppressed because the police gave inadequate
miranda warnings, failed to read the waiver portion of the miranda form to defendant, and failed to have him acknowledge and initial each individual righ
t on the miranda form. we are unpersuaded.

Result 3:
of conviction entered after a jury found him guilty of first - degree aggravated sexual assault, second - degree sexual assault, and third - degree endan
```

Fig. 6: Reranking Texts Retrieved

Once the top-N most relevant legal texts have been selected, they are forwarded to the generative phase.

3.2.2 Generation Phase

After retrieving the relevant legal texts, these texts are aggregated into a structured prompt. The LLaMA 2 model takes the structured prompt – includes user query, retrieved texts, context and structured output – and generates a response. This ensures that the model generates a response that is relevant and contextually accurate.

3.2.2.1 Prompt Engineering

Prompt engineering improves the quality and relevance of generated responses. By carefully designing the input prompts for the “meta-llama/Llama-2-7b-chat-hf” model, we can ensure more relevant, accurate, and structured responses. The prompt is constructed by combining the user query and the aggregated context to achieve the following objectives:

1. Contextual Understanding:
    - Provide the model with a clear understanding of its role as a legal assistant specialising in case law research.
    - This ensures that responses align with legal research standards.
2. Retrieved Texts Integration:
    - The retrieved legal texts are used in the prompt to ground the model’s responses in factual case precedents.
    - This prevents hallucination when the model is generating responses.
3. Structured Output:
    - Define a clear response format.
    - This ensures that responses are clear and consistent across different queries.
4. Guided Reasoning:
    - Specify the key analytical steps for comparing cases and identifying relevant legal principles.
    - This helps the model to focus on relevant aspects of legal analysis.

This prompt is then passed into the “meta-llama/Llama-2-7b-chat-hf” model to generate responses for user queries.

3.2.2.2 LLaMA 2 for Text Generation

LLaMA 2, specifically the "meta-llama/Llama-2-7b-chat-hf" model, is employed for text generation. This LLM has been optimised for dialogue-based interactions, making it well-suited to generate responses based on user query (Acorn, n.d.). The model refines retrieved legal case information and generates coherent, contextually relevant responses. Its capability to handle complex queries makes it particularly suitable for legal precedent search, where nuanced and precise text synthesis is essential, and allows for prompt engineering.

The following are the reasons for choosing "meta-llama/Llama-2-7b-chat-hf":

1. Size and Performance
    - This project is limited by computational resources and is unable to load LLaMA 2 models with 13 billion parameters and above.
    - With 7 billion parameters, this model can offer strong text generation quality while still being computational efficient.
2. Fine-tuned for Dialogue
    - Optimised using reinforcement learning from human feedback (RLHF), ensuring its generated output aligns with user query.
3. Scalability and Deployment:
    - Compatible with Hugging Face's transformers library, allowing seamless integration into the RAG system.

Therefore, the "meta-llama/Llama-2-7b-chat-hf" model is chosen to generate responses that are well-structured, accurate and coherent. Finally, this RAG system is complete and ready to be used for user query and precedent search.

3.3 Use Case: Finding Similar Cases (Precedent Search)

The primary use of this system is for precedent search and case law retrieval. The following illustrates an example of precedent search:

```
query = "Find cases similar to Miranda v. Arizona, 384 U.S. 436 (1966)."
rag_response = rag_pipeline(query, vector_store, tokenizer, model, reranker_model)
print("\nGenerated Response:")
print(rag_response)
```

```
Generated Response:
**Your Answer:**

    1. Similar Cases Identified:
       - Case Name: New York v. Prysock, 453 U.S. 355 (1981)
       - Summary: In this case, the Supreme Court held that a warned and unwavering waiver of the right to counsel must be made voluntarily, knowingly, a
nd intelligently. The court found that the defendant's statements were obtained without a valid waiver of the right to counsel and were therefore inadmis
sible.
       - Similarity to Query: Relevant to the issue of whether the defendant's statements were obtained voluntarily and knowingly, and whether the state
met its burden of proving the defendant's waiver beyond a reasonable doubt.
       - Key Legal Principles: (1) The right to counsel must be voluntary, knowingly, and intelligent; (2) A warned and unwavering waiver of the right to
counsel must be made before questioning can continue.
       - Contrasting Precedents: (1) Oregon v. Bradshaw, 472 U.S. 1034 (1985) (holding that a defendant's statements were admissible where the defendant
was not given a formal waiver of the right to counsel but still made incriminating statements after being advised of the right to counsel).
    2. Key Legal Principles:
       - Principle 1: The right to counsel must be voluntary, knowingly, and intelligent.
       - Principle 2: A warned and unwavering waiver of the right to counsel must be made before questioning can continue.
    3. Contrasting Precedents: (1) Oregon v. Bradshaw, 472 U.S. 1034 (1985) (holding that a defendant's statements were admissible where the defendant wa
s not given a formal waiver of the right to counsel but still made incriminating statements after being advised of the right to counsel).
    4. Conclusion: Based on the similarities between the cases and the relevance of the legal principles
```

Fig. 7: Generated Response of RAG

3.4 Conclusion

The system is designed to achieve accuracy and efficiency. By first building the knowledge base and pre-indexing documents as vectors, it minimises real-time computation, reducing latency during query handling. Additionally, since it is built using the FAISS index, the knowledge base is scalable and supports the integration of additional documents. After the knowledge base is ready, the RAG query pipeline is implemented to handle user queries by leveraging both retrieval and generative capabilities so that the generated responses are grounded in actual legal texts and coherent. As a result, this RAG architecture processes user queries efficiently, reduces hallucinations and improves reliability in responses generated.

4 Implementation

The implementation of the RAG system consists of two main stages – knowledge base building and query processing. The knowledge base is constructed using legal opinions that are fetched, preprocessed, embedded, and stored as a FAISS index. After the knowledge base is built, it is integrated into the RAG pipeline to handle user queries, which consists of two phases – retrieval and generation phase.

4.1 Knowledge Base

The building of the knowledge base consists of multiple steps and is encapsulated within the "main" function. The following "main" function is implemented for building and storing the knowledge base:

```
def main():
    # Fetch raw knowledge base
    print("Loading raw knowledge base...")
    RAW_KNOWLEDGE_BASE = fetch_data(API_BASE_URL + "opinions/", limit=total_records)

    # Check no. of records fetched
    print(f"Number of records fetched: {len(RAW_KNOWLEDGE_BASE)}")

    # Process the documents
    print("Processing knowledge base...")
    docs_processed = process_knowledge_base(RAW_KNOWLEDGE_BASE)


    # Check number of documents processed (this includes chunking)
    print(f"Number of documents processed: {len(docs_processed)}")


    # Initialise FAISS and add embeddings
    print("Indexing embeddings...")
    vector_store = initialise_faiss(docs_processed, embedding_model)

    # Save FAISS index
    print(f"Saving FAISS index to {FAISS_INDEX_PATH}...")
    vector_store.save_local(FAISS_INDEX_PATH)

    print("FAISS index created and saved successfully.")
```

Fig. 8: Building Knowledge Base Function "main"

The following outlines the key steps of the "main" function:

1. Fetching Data:
    - The raw data is retrieved from the API endpoint "https://www.courtlistener.com/api/rest/v4/opinions/" which contains legal opinions.
    - The "fetch_data" function is used to retrieve the data.
    - Refer to Section 4.1.1
2. Preprocessing Data:
    - Once the raw data is retrieved, it undergoes preprocessing using the "process_knowledge_base" function.
    - This includes data cleaning and document chunking.
    - Refer to Section 4.1.2
3. Indexing using FAISS and Generating Embeddings:
    - After preprocessing, the documents are indexed using FAISS. The "initialise_faiss" function initialises the FAISS index and generates embeddings for the preprocessed documents using the "sentence-transformers/all-mpnet-base-v2" model. This allows similarity searches to be performed on the knowledge base.
    - Lastly, the FAISS index is saved in a local path "FAISS_INDEX_PATH" and the knowledge base is built.
    - Refer to Section 4.1.3

Therefore, the "main" function is used to build the knowledge base and stored into the FAISS index. This knowledge base will later be integrated into the RAG pipeline used for retrieval when processing user queries.

4.1.1 Data Fetching

The system retrieves legal case documents from the CourtListener API to build a comprehensive knowledge base for precedent search. The following function "fetch_data" is implemented to retrieve case law:

```python
# Fetch Data from CourtListener
def fetch_data(endpoint, params=None, limit=100):
    """Fetch data from the CourtListener API."""
    results = []
    while endpoint:
        # print(f"Fetching: {endpoint}")
        response = requests.get(endpoint, headers=HEADERS, params=params)
        if response.status_code != 200:
            print(f"Error: {response.status_code} - {response.text}")
            break

        data = response.json()
        results.extend(data["results"])

        # Handle pagination
        endpoint = data.get("next")
        if len(results) >= limit:
            break

    return results
```

Fig. 9: Data Fetching Function "fetch_data"

The data is fetched using the RESTful API provided by CourtListener. The "fetch_data" function makes HTTP GET requests to retrieve legal opinions. It takes the API endpoint "https://www.courtlistener.com/api/rest/v4/opinions/" and optional query parameters, ensuring flexibility in filtering and retrieving relevant case law.

The following are the steps for the data retrieval process:

1. API Request: The system sends a request to the opinions endpoint of the CourtListener API.
2. Response Handling: The response is parsed, and legal opinions are extracted.
3. Pagination Management: Since the API returns paginated results, the function iterates through multiple pages until the desired number of records is collected which is determined by the “limit” parameter passed.
4. Error Handling: If the API request fails (e.g., due to network issues or rate limits), appropriate error messages are logged.

As a result, the retrieved legal opinions are stored in “RAW_KNOWLEDGE_BASE”, which requires preprocessing and embedding before being indexed in the vector store.

4.1.2 Data Preprocessing

The retrieved legal records stored in “RAW_KNOWLEDGE_BASE” undergo data preprocessing before being embedded into dense vector representations and indexed in the vector store. Each record corresponds to an item in the “RAW_KNOWLEDGE_BASE” list and has the following keys:

['resource_uri', 'id', 'absolute_url', 'cluster_id', 'cluster', 'author_id', 'author', 'joined_by', 'date_created', 'date_modified', 'author_str', 'per_curiam', 'joined_by_str', 'type', 'sha1', 'page_count', 'download_url', 'local_path', 'plain_text', 'html', 'html_lawbox', 'html_columbia', 'html_anon_2020', 'xml_harvard', 'html_with_citations', 'extracted_by_ocr', 'ordering_key', 'opinions_cited']

Fig. 10: Keys of Each Record in “RAW_KNOWLEDGE_BASE”

The following function “process_knowledge_base” is implemented for data preprocessing:

```python
# Process Raw Knowledge Base
def process_knowledge_base(raw_knowledge_base):
    """Process raw documents into chunks with metadata."""
    docs_processed = []
    for doc in tqdm(raw_knowledge_base, desc="Processing documents"):

        # This is the main content
        text = doc.get("plain_text", "")
        # All other fields are extracted as metadata
        metadata = {k: v for k, v in doc.items() if k != "plain_text"}

        # Check that text is not empty
        if not text:
            continue

        # Chunk the main content and assign the respective metadata
        chunks = chunk_text_semantic(text, max_tokens=384, overlap_tokens=100)
        for i, chunk in enumerate(chunks):
            docs_processed.append({
                "text": chunk,
                "metadata": {**metadata, "chunk_id": i}
            })
    return docs_processed
```

Fig.11: Data Preprocessing Function “process_knowledge_base”

The “plain_text” field is extracted as the main content “text” and the other fields are extracted as “metadata”, and stored in the “docs_processed” list. As a result, each record in “docs_processed” has 2 keys: “text” and “metadata”. The main content “text” undergoes preprocessing which consists of two key stages – data cleaning and document chunking.

4.1.2.1 Data Cleaning

Before storing the main content "text" in the system, it has to be cleaned and normalised. The fetched legal texts are too noisy as they contain inconsistencies, excessive whitespace and annotations that can interfere with retrieval. Therefore, data cleaning is important to ensure that the text is structured, standardised, and retains its legal significance while removing irrelevant artifacts, which improves downstream processing. The following function "clean_text" is implemented for data cleaning:

```python
import re

def clean_text(text):
    """
    Clean the main content in the following ways:
    - Remove excessive whitespace and line breaks
    - Preserve legal citations
    - Handle case headers
    - Manage footnotes
    - Normalise punctuation
    """

    # Normalise line breaks and whitespace
    text = re.sub(r'\r\n|\r|\n', ' ', text)  # Replace different line breaks with space
    text = re.sub(r'\s+', ' ', text).strip()  # Collapse multiple spaces into one

    # Preserve legal citations (e.g., "123 U.S. 456"), this matches common legal citation formats
    citation_pattern = r'(\b\d{1,3}\s+[A-Z][a-zA-Z]*\s+\d+\b)'
    citations = re.findall(citation_pattern, text)
    for citation in citations:
        text = text.replace(citation, f'[{citation}]')

    # Handle case headers and metadata, assuming headers are in uppercase and followed by a colon
    header_pattern = r'([A-Z\s]+:)\s'
    text = re.sub(header_pattern, r'\n\1 ', text)

    # Manage footnotes and annotations
    # Removing footnotes assuming they are enclosed in brackets or parentheses
    text = re.sub(r'\[\d+\]', '', text)  # Remove numeric footnotes like [1]
    text = re.sub(r'\(\d+\)', '', text)  # Remove numeric footnotes like (1)

    # Normalize punctuation spacing
    text = re.sub(r'\s+([.,;:!?])', r'\1', text)  # Remove space before punctuation
    text = re.sub(r'([.,;:!?])([A-Za-z])', r'\1 \2', text)  # Ensure space after punctuation

    return text
```

Fig. 12: Data Cleaning Function "clean_text"

The cleaning process “clean_text” includes the following steps:

1. Removing excessive whitespace and line breaks
    - Line breaks of different formats (\r\n, \r, \n) are replaced with spaces.
    - Multiple consecutive spaces are collapsed into a single space.
2. Preserving legal citations
    - Citations in common legal formats (e.g., 123 U.S. 456) are identified using regular expressions.
    - These citations are enclosed in square brackets ([ ]) to maintain readability and prevent accidental modifications.
3. Handling case headers
    - Case headers and metadata, which are often written in uppercase and followed by a colon (e.g., CASE TITLE:), are identified.
    - These headers are formatted with line breaks to improve document structure.
4. Managing footnotes and annotations
    - Numeric footnotes appearing in square brackets (e.g., [1]) or parentheses (e.g., (1)) are removed to improve text clarity.
5. Normalising punctuation spacing
    - Unnecessary spaces before punctuation marks (. , ; : ! ?) are removed.
    - Spaces are ensured after punctuation to maintain proper sentence structure.

4.1.2.2 Document Chunking

Legal documents that contain long passages of text make direct embedding and retrieval inefficient. To address this, legal documents in the "text" field are split into chunks and stored along with their respective metadata in "docs_processed". Additionally, chunking ensures that each document chunk complies with the embedding model maximum token limit and prevents truncation during vector embedding. To maintain contextual integrity, a semantic chunking approach is applied instead of arbitrary fixed-length splits. Therefore, this process allows downstream vector embeddings to capture more meaningful context and improve retrieval quality. The following function "chunk_text_semantic" is implemented for document chunking:

```python
def chunk_text_semantic(text, max_tokens=384, overlap_tokens=100):
    """
    Split text into semantic chunks while respecting token limits.
    The function prioritizes sentence-level chunking and ensures minimal token loss.
    """

    # Clean the text
    text = clean_text(text)

    # Split text into sentences
    sentences = sent_tokenize(text)

    chunks = []
    current_chunk = []
    current_length = 0

    for sentence in sentences:
        # Tokenize the sentence
        sentence_tokens = tokenizer.tokenize(sentence)
        sentence_length = len(sentence_tokens)

        # If adding this sentence exceeds the max token limit, finalize the current chunk
        if current_length + sentence_length > max_tokens:
            if current_chunk:
                # Convert current chunk to string and store
                chunk_text = tokenizer.convert_tokens_to_string(current_chunk)
                chunks.append(chunk_text)

                # Keep an overlapping portion for continuity
                overlap_start = max(0, len(current_chunk) - overlap_tokens)
                current_chunk = current_chunk[overlap_start:]
                current_length = len(current_chunk)

            # If the sentence itself is longer than max_tokens, split it directly
            if sentence_length > max_tokens:
                for i in range(0, sentence_length, max_tokens - overlap_tokens):
                    chunk_text = tokenizer.convert_tokens_to_string(sentence_tokens[i:i + max_tokens - overlap_tokens])
                    chunks.append(chunk_text)
                continue  # Skip adding this sentence again

        # Add sentence tokens to the current chunk
        current_chunk.extend(sentence_tokens)
        current_length += sentence_length

    # Add the last chunk if it exists
    if current_chunk:
        chunk_text = tokenizer.convert_tokens_to_string(current_chunk)
        chunks.append(chunk_text)

    return chunks
```

Fig. 13: Document Chunking Function "chunk_text_semantic"

The document chunking process “chunk_text_semantic” uses the tokenizer from “sentence-transformers/all-mpnet-base-v2” and includes the following steps:

1. Sentence-Level Segmentation:
    - The main content in “text” is first cleaned using clean_text(), then split into sentences using NLTK’s sent_tokenize() to maintain semantic coherence.
2. Token-Based Chunking:
    - Each chunk is limited to a maximum of 384 tokens, ensuring compatibility with the embedding model.
    - If adding a sentence exceeds the limit, the current chunk is stored, and a new chunk starts.
    - Overlapping tokens (100 tokens) are retained between consecutive chunks to preserve contextual continuity.
    - If an individual sentence exceeds the token limit, it is split directly while maintaining overlap.
3. Assigning “chunk_id”:
    - Each chunk is assigned a unique "chunk_id" to track its position which is added to “metadata”.
    - The chunk is stored in “docs_processed” alongside its original metadata.
    - Each chunk can be uniquely identified by the combination of "id" and "chunk_id".

The following figures illustrate the distribution of token lengths for the first 500 records, both before and after the chunking process.

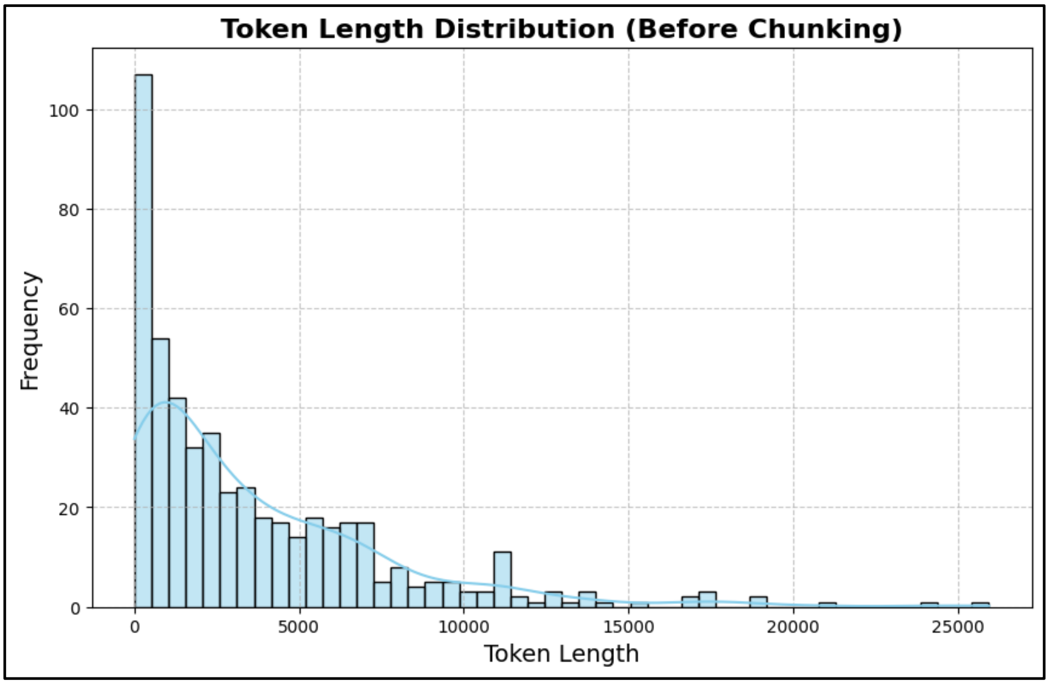

Fig. 14: Token Length Distribution (Before Chunking)

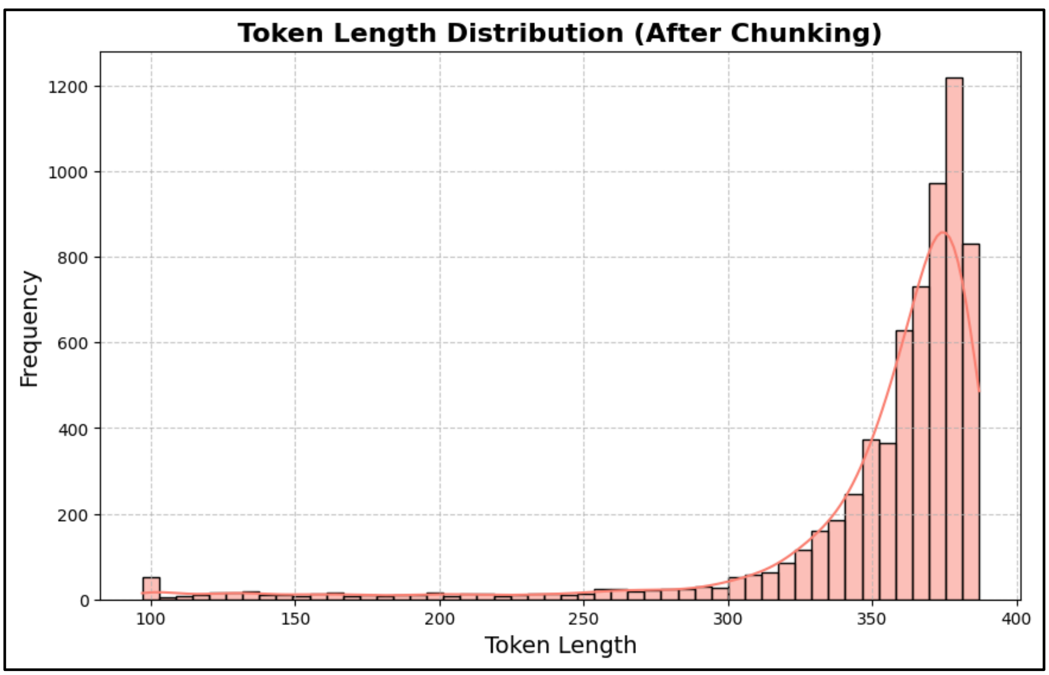

Fig. 15: Token Length Distribution (After Chunking)

As a result, the chunks, along with their associated metadata, are stored in "docs_processed" for downstream processing into a vector store.

4.1.3 Vector Store

The legal documents chunks stored along with their respective metadata in "docs_processed" are indexed into a FAISS vector store using the "initialise_faiss" function. The following function "initialise_faiss" is implemented for vector store:

```
def initialise_faiss(docs_processed, embedding_model):
    """
    Initialise the FAISS index with processed documents and an embedding model.

    Args:
        docs_processed (list): List of dictionaries with 'text' and 'metadata'.
        embedding_model (HuggingFaceEmbeddings): The embedding model to use.

    Returns:
        FAISS: A FAISS vector store with the documents indexed.
    """
    # Ensure embedding_model is passed correctly
    if not isinstance(embedding_model, HuggingFaceEmbeddings):
        raise ValueError("embedding_model must be an instance of HuggingFaceEmbeddings")

    # Convert dictionaries to Document objects
    documents = [
        Document(page_content=doc["text"], metadata=doc["metadata"])
        for doc in docs_processed
    ]

    # Create FAISS index from Document objects
    faiss_index = FAISS.from_documents(
        documents,
        embedding=embedding_model,
        distance_strategy=DistanceStrategy.COSINE,  # Use cosine similarity
    )

    return faiss_index
```

Fig. 16: Vector Store Function "initialise_faiss"

The FAISS index is initialised using preprocessed legal documents stored in "docs_processed" and the "sentence-transformers/all-mpnet-base-v2" embedding model from Hugging Face. The vector store process consists of the following steps:

1. Embedding Model
    - The HuggingFaceEmbeddings class is used to load the "sentence-transformers/all-mpnet-base-v2" model, which converts legal text into 768-dimensional vector representations optimised for semantic similarity tasks.
2. Conversion of Documents
    - Each item in "docs_processed" is structured as a LangChain Document object, consisting of:
        - page_content: The preprocessed legal opinion chunk.
        - metadata: Metadata associated with each chunk.
3. FAISS Indexing
    - The "sentence-transformers/all-mpnet-base-v2" model generates embeddings for each legal document chunk.
    - The "FAISS.from_documents" function is used to index the vectorised chunks.
    - The cosine similarity distance metric is used to measure the relevance of retrieved cases. This allows for similarity-based retrieval.

As a result, the FAISS index is built and the knowledge base is now complete. This FAISS index is later used for retrieval in the RAG pipeline when processing user queries.

4.2 RAG Query Pipeline

Once the FAISS vector store is built, it is ready for efficient retrieval and to be integrated into the RAG pipeline. This end-to-end query pipeline handles user queries by retrieving relevant documents from the FAISS vector store and generating responses using the LLaMA 2 model. Therefore, this ensures that only relevant legal texts are retrieved and used to generate responses, improving the quality of generated responses. The following function "rag_pipeline" is implemented for the RAG query pipeline:

```
def rag_pipeline(query, vector_store, tokenizer, model, reranker_model):
    """End-to-end RAG pipeline: Retrieve, Rerank, and Generate with LLaMA."""

    # Retrieve Top-10 Documents from FAISS
    retrieved_docs = vector_store.similarity_search(query, k=10)

    # Apply Cross-Encoder Reranking
    top_results = rerank_documents(query, retrieved_docs, reranker_model)

    # Extract text from top reranked documents
    retrieved_texts = "\n\n".join([result.page_content for result in top_results])

    # Format Prompt for LLaMA
    prompt = f"""
    Your task is to analyse relevant legal precedents and find cases similar to the given query.

    **Relevant Legal Opinions Retrieved:**
    {retrieved_texts}

    **Output Format:**
    1. Similar Cases Identified:
       - Case Name: [Case Title]
       - Summary: [Brief summary of case facts and ruling]
       - Similarity to Query: [Explain why it's relevant]

    2. Key Legal Principles:
       - Principle 1: [Description]
       - Principle 2: [Description]

    3. Contrasting Precedents (if any):
       - Case Name: [Case Title]
       - Key Difference: [Explain how it differs]

    4. Conclusion: [Final summary]

    **User Query:** "{query}"
    Please answer User Query
    """

    # Generate Response using LLaMA
    generated_text = generate_text(prompt, tokenizer, model)

    return generated_text
```

Fig. 17: End-to-end RAG Pipeline Function "rag_pipeline"

When a user submits a query, the pipeline executes in two main phases:

1. Retrieval Phase
    - Retrieve the top-K most relevant legal document chunks from the FAISS vector store using similarity search.
    - After that, the chunks are reranked based on relevance score and top-N most relevant chunks are passed onto the generation phase.
2. Generation Phase
    - Format the retrieved texts into a structured prompt.
    - Generate a response using the LLaMA 2 model.

4.2.1 Retrieval Phase

The retrieval phase consists of retrieving and reranking the most relevant legal texts based on the user's query. This ensures that only the most relevant texts are passed to the LLaMA 2 model.

The following shows the retrieval part within the "rag_pipeline" function (outlined above in Fig. 17):

```python
# Retrieve Top-10 Documents from FAISS
retrieved_docs = vector_store.similarity_search(query, k=10)

# Apply Cross-Encoder Reranking
top_results = rerank_documents(query, retrieved_docs, reranker_model)

# Extract text from top reranked documents
retrieved_texts = "\n\n".join([result.page_content for result in top_results])
```

Fig. 18: Retrieval Part Within "rag_pipeline"

The following function "rerank_documents" is used to rerank the retrieved texts:

```python
def rerank_documents(query, retrieved_docs, reranker_model):
    """Reranks documents using a cross-encoder model."""
    query_doc_pairs = [(query, doc.page_content) for doc in retrieved_docs]
    scores = reranker_model.predict(query_doc_pairs)

    # Sort documents by score (descending)
    ranked_results = sorted(zip(retrieved_docs, scores), key=lambda x: x[1], reverse=True)

    # Return top 3 reranked results
    return [doc for doc, score in ranked_results[:3]]
```

Fig. 19: Reranking Function "rerank_documents"

The retrieval phase of the RAG pipeline consists of the following steps:

1. User Query Embedding:
    - The user's query is embedded using the same embedding model as the FAISS vector store "meta-llama/Llama-2-7b-chat-hf". This is handled internally by "vector_store.similarity_search()".
2. Retrieving from FAISS Vector Store:
    - "vector_store.similarity_search()" retrieves the top-10 most similar documents to the query.
    - The cosine similarity metric is chosen during FAISS indexing for similarity search.
3. Reranking of Retrieved Text:
    - Once the top-k documents are retrieved, they undergo reranking using the "rerank_documents()" function.
    - This is done using the cross-encoder model "cross-encoder/ms-marco-MiniLM-L-6-v2", which takes the query and the top-10 retrieved text as input and predicts a relevance score. The cross-encoder model evaluates how well each text answers the query in a more refined and context-aware manner, beyond just the vector similarity.
    - The results are then ranked based on their predicted scores and top-3 texts are extracted.
4. Preparation for Generation Phase
    - The top-3 most relevant chunks are concatenated and stored in "retrieved_texts", which is passed onto the generative phase.

4.2.2 Generation Phase

After the relevant legal texts are retrieved, these texts are formatted into a structured prompt and used as input to generate responses to user queries. To ensure high-quality responses, LLaMA 2 model is selected and the prompt is optimised.

4.2.2.1 Loading LLaMA 2 Model

The LLaMA 2 model and tokenizer are used to process text inputs and generate outputs. The tokenizer handles the conversion between text and tokens, while the model performs the actual computation to generate responses.

The following function "load_llama_model" is implemented to load the "meta-llama/Llama-2-7b-chat-hf" model and its tokenizer:

```
from transformers import LlamaForCausalLM, LlamaTokenizer, pipeline
import torch

def load_llama_model(model_name="meta-llama/Llama-2-7b-chat-hf"):
    """Load LLaMA 2 model and tokenizer."""
    tokenizer = LlamaTokenizer.from_pretrained(model_name)
    model = LlamaForCausalLM.from_pretrained(
        model_name,
        torch_dtype=torch.float16,
        device_map="auto"
    )
    return tokenizer, model

tokenizer, model = load_llama_model()
```

Fig. 20: Loading LLaMa 2 Function "load_llama_model"

The “load_llama_model” function consists of the following steps:

1. Loading Tokenizer:
    - The tokenizer is loaded using the “LlamaTokenizer.from_pretrained” method from Hugging Face so that the input can be properly tokenized in a format that the model can process.
2. Loading Model:
    - The “LlamaForCausalLM.from_pretrained” method from Hugging Face is used to load “meta-llama/Llama-2-7b-chat-hf” model into memory, initialise its architecture, and load its pre-trained weights.
    - The model is loaded with 16-bit floating-point precision “torch.float16” for optimised performance, and “device_map=“auto”” automatically places the model on the appropriate device, such as a GPU or CPU, depending on available resources. This accommodates the limited computational resources of this project.

4.2.2.2 Prompt Engineering

Prompt engineering ensures that the responses generated are more relevant, accurate, and structured. The prompt is constructed using the user query and an aggregated context. The following illustrates the prompt designed to guide the model in generating a response:

```
prompt = f"""
Your task is to analyse relevant legal precedents and find cases similar to the given query.

**Relevant Legal Opinions Retrieved:**
{retrieved_texts}

**Output Format:**
1. Similar Cases Identified:
   - Case Name: [Case Title]
   - Summary: [Brief summary of case facts and ruling]
   - Similarity to Query: [Explain why it's relevant]

2. Key Legal Principles:
   - Principle 1: [Description]
   - Principle 2: [Description]

3. Contrasting Precedents (if any):
   - Case Name: [Case Title]
   - Key Difference: [Explain how it differs]

4. Conclusion: [Final summary]

**User Query:** "{query}"
Please answer User Query
"""
```

Fig. 21: Prompt within "rag_pipeline"

The prompt consists of the following specifications:

1. Contextual Understanding:
   - Provide the model with a clear understanding of its role to analyse relevant legal precedents and find cases similar to the given query.
2. Retrieved Texts Integration:
   - The top-N retrieved legal texts are used in the prompt to ground the model's responses in factual case precedents.
3. Structured Output:
   - Define a clear response format.
4. Guided Reasoning:
   - Specify the key analytical steps for comparing cases and identifying relevant legal principles.

4.2.2.3 Text Generation

The “generate_text” function takes in a structured prompt which includes the retrieved document chunks, processes it using the tokenizer, and generates a response using the "meta-llama/Llama-2-7b-chat-hf" model.

The following function “generate_text” is implemented to generate response:

```python
def generate_text(prompt, tokenizer, model):
    """Generates text using LLaMA and removes the original prompt from the output."""
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=2048).to(model.device)
    with torch.no_grad():
        outputs = model.generate(**inputs, max_new_tokens=512, temperature=0.7, top_p=0.9)

    full_response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Remove the prompt part by extracting only the response text
    response = full_response.split("**Example Output:**")[-1].strip()

    return response
```

Fig. 22: Generating Response Function “generate_text”

The “generate_text” function consists of the following steps:

1. Tokenizing Prompt:
    - The input prompt is tokenized using the tokenizer, which converts the text into a numerical format for model processing.
    - The tokenized input is returned as PyTorch tensors “return_tensors=“pt”” and mapped to the appropriate device using “model.device”.
    - The function ensures that the input does not exceed the model's maximum token limit “max_length=2048” to prevent truncation errors.
2. Generating Text:
    - The model generates new tokens based on the input using “model.generate()”.
    - The parameter “max_new_tokens=512” limits the number of new tokens generated to control response length.
    - The “temperature=0.7” setting balances randomness and coherence, ensuring diversity while maintaining relevance.
    - The “top_p=0.9”setting applies nucleus sampling, selecting tokens from the top 90% cumulative probability mass to avoid deterministic outputs.
3. Decoding Output:
    - The generated token sequence is decoded back into human-readable text using tokenizer.decode().
    - Special tokens (e.g., <s>, </s>) are removed using “skip_special_tokens=True” to clean the final output.

5 Evaluation

5.1 Retrieval Precision

To evaluate the retrieval precision, the top-10 documents are retrieved from the FAISS index and the relevance of retrieved documents are qualitatively determined. The following table shows how many of the top-10 documents are relevant to the user query.

Query	**Precision**
Find cases similar to Marbury v. Madison establishing judicial review or limits on congressional power.	7/10
Find cases similar to Miranda v. Arizona, 384 U.S. 436 (1966).	6/10
Find Washington state cases addressing whether amendments to RCW 10.82.090 (restitution interest waiver) apply retroactively to resentencing hearings.	4/10
Find cases similar to District of Columbia v. Heller (2008) that interpret the Second Amendment's protection of individual firearm rights.	5/10
Find cases similar to Kelo v. New London (2005) involving eminent domain and economic development under the Takings Clause.	3/10
Find decisions similar to Gideon v. Wainwright, 372 U.S. 335 (1963).	2/10

Table 3: Retrieval Precision

This is also used to determine the top-N documents to select after reranking and based on the above result chosen N=3.

The effectiveness of this system, more specifically retrieval, depends on the knowledge base as it can only generate responses based on cases that are indexed into the vector store. Due to memory constraints, the result is limited by the number of legal opinions stored. Therefore, the knowledge base may lack volume and diversity.

5.2 Generated Response

The generated responses of the RAG system are evaluated. The following are the metrics used for this comparison to check how much each of the response aligns with the query:

1. BERTScore
2. Cosine Similarity

The following functions "evaluate_bert_score" and "evaluate_cosine_similarity" is implemented to evaluate the generated responses:

```
from bert_score import score as bert_score

def evaluate_bert_score(query, generated):
    P, R, F1 = bert_score([generated], [query], lang="en")
    return F1.mean().item()
```

Fig. 23: BERTScore Function "evaluate_bert_score"

```
from langchain.embeddings import HuggingFaceEmbeddings
import torch
from sentence_transformers.util import pytorch_cos_sim

def evaluate_cosine_similarity(query, generated):
    embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")

    # Generate embeddings
    query_embedding = torch.tensor(embedding_model.embed_query(query))
    generated_embedding = torch.tensor(embedding_model.embed_query(generated))

    # Compute cosine similarity
    return pytorch_cos_sim(query_embedding, generated_embedding).item()
```

Fig. 24: Cosine Similarity Function "evaluate_cosine_similarity"

The following are the evaluation of the generated responses from the RAG system for different queries that are relevant to precedent search.

1. query = “Find cases similar to Miranda v. Arizona, 384 U.S. 436 (1966).”

```
query = "Find cases similar to Miranda v. Arizona, 384 U.S. 436 (1966)."
rag_response = rag_pipeline(query, vector_store, tokenizer, model, reranker_model)
print("\nGenerated Response:")
print(rag_response)
```

```
Generated Response:
**Your Answer:**

    1. Similar Cases Identified:
       - Case Name: New York v. Prysock, 453 U.S. 355 (1981)
       - Summary: In this case, the Supreme Court held that a warned and unwavering waiver of the right to counsel must be made voluntarily, knowingly, a
nd intelligently. The court found that the defendant's statements were obtained without a valid waiver of the right to counsel and were therefore inadmis
sible.
       - Similarity to Query: Relevant to the issue of whether the defendant's statements were obtained voluntarily and knowingly, and whether the state
met its burden of proving the defendant's waiver beyond a reasonable doubt.
       - Key Legal Principles: (1) The right to counsel must be voluntary, knowingly, and intelligent; (2) A warned and unwavering waiver of the right to
counsel must be made before questioning can continue.
       - Contrasting Precedents: (1) Oregon v. Bradshaw, 472 U.S. 1034 (1985) (holding that a defendant's statements were admissible where the defendant
was not given a formal waiver of the right to counsel but still made incriminating statements after being advised of the right to counsel).
    2. Key Legal Principles:
       - Principle 1: The right to counsel must be voluntary, knowingly, and intelligent.
       - Principle 2: A warned and unwavering waiver of the right to counsel must be made before questioning can continue.
    3. Contrasting Precedents: (1) Oregon v. Bradshaw, 472 U.S. 1034 (1985) (holding that a defendant's statements were admissible where the defendant wa
s not given a formal waiver of the right to counsel but still made incriminating statements after being advised of the right to counsel).
    4. Conclusion: Based on the similarities between the cases and the relevance of the legal principles
```

Fig. 25: Generated Response of RAG

- BERTScore: 0.8106132745742798
- Cosine Similarity: 0.48735684156417847

2. query = “Find Washington state cases addressing whether amendments to RCW 10.82.090 (restitution interest waiver) apply retroactively to resentencing hearings.”

```
query = "Find Washington state cases addressing whether amendments to RCW 10.82.090 (restitution interest waiver) apply retroactively to resentencing hea
rag_response = rag_pipeline(query, vector_store, tokenizer, model, reranker_model)
print("\nGenerated Response:")
print(rag_response)
```

```
Generated Response:
**Relevant Legal Opinions Retrieved:**
    1. Case Name: State of Washington v. David J. Bassford
        Summary: In 1994, Bassford pleaded guilty to second-degree theft and was sentenced to pay restitution, attorney fees, court costs, and a victim p
enalty assessment. In 1999, he pleaded guilty to second-degree theft and bail jumping and was sentenced to pay additional restitution. Bassford later mov
ed to waive interest on his restitution, citing a 2022 amendment to RCW 10.82.090 that authorizes courts to waive interest on restitution. The court deni
ed his motion, and Bassford appealed.
        Similarity to Query: Relevant because the case involves a defendant seeking to have interest waived on restitution ordered as part of their sente
nce, which is the same issue as the user query.

    2. Key Legal Principles:
        Principle 1: The 2022 amendment to RCW 10.82.090 applies to cases that are on direct appeal when the amendment takes effect.
        Principle 2: The amendment does not apply to cases that were final when the amendment was enacted.

    3. Contrasting Precedents (if any): None provided.
```

Fig. 26: Generated Response of RAG

- BERTScore: 0.7767754197120667
- Cosine Similarity: 0.4009822607040405

3. query = “Find cases similar to Citizens United v. FEC, 558 U.S. 310 (2010).”

```
query = "Find cases similar to Citizens United v. FEC, 558 U.S. 310 (2010)."
rag_response = rag_pipeline(query, vector_store, tokenizer, model, reranker_model)
print("\nGenerated Response:")
print(rag_response)

Generated Response:
**Your Answer:**

    1. Similar Cases Identified:
       - Case Name: McConnell v. FEC, 540 U.S. 93 (2003)
       - Summary: The case involves a challenge to a provision of the Bipartisan Campaign Reform Act of 2002, which prohibits corporations and labor unio
ns from using general treasury funds to pay for electioneering communications within 60 days of a general election or 30 days of a primary election.
       - Similarity to Query: The case involves a First Amendment challenge to a campaign finance law, which is similar to the challenge in Citizens Unit
ed v. FEC.

    2. Key Legal Principles:
       - Principle 1: The First Amendment protects the right to engage in political speech, including the use of corporate funds for political communicat
ions.
       - Principle 2: The government may not restrict political speech based on the identity of the speaker or the source of the funds.

    3. Contrasting Precedents (if any):
       - Case Name: Austin v. Michigan Chamber of Commerce, 494 U.S. 652 (1990)
       - Key Difference: In Austin, the Court upheld a state law that prohibited corporations from using treasury funds to pay for electioneering communi
cations, while in McConnell, the Court struck down a similar federal law.
```

Fig. 27: Generated Response of RAG

- BERTScore: 0.8218123912811279
- Cosine Similarity: 0.6367415189743042

4. query = “Find cases similar to Marbury v. Madison establishing judicial review or limits on congressional power.”

```
query = "Find cases similar to Marbury v. Madison establishing judicial review or limits on congressional power."
rag_response = rag_pipeline(query, vector_store, tokenizer, model, reranker_model)
print("\nGenerated Response:")
print(rag_response)

Generated Response:
1. Similar Cases Identified:
       - Case Name: McCulloch v. Maryland
       - Summary: In this landmark case, the Supreme Court established the principle of implied powers, ruling that Congress has the authority to enact l
aws that are necessary and proper to carry into execution its expressly delegated powers.
       - Similarity to Query: The case involves the exercise of judicial review over congressional power, similar to Marbury v. Madison.

       - Case Name: Gibbons v. Ogden
       - Summary: In this case, the Supreme Court established the principle of constitutional supremacy, ruling that federal law takes precedence over st
ate law when they conflict.
       - Similarity to Query: The case involves the limits of congressional power and the role of the judiciary in enforcing those limits, similar to Mar
bury v. Madison.

       - Case Name: United States v. Lopez
       - Summary: In this case, the Supreme Court struck down a federal law that exceeded Congress's power under the Commerce Clause, ruling that the law
did not have a sufficient connection to interstate commerce to justify its application.
       - Similarity to Query: The case involves the limits of congressional power and the role of the judiciary in enforcing those limits, similar to Mar
bury v. Madison.
```

Fig. 28: Generated Response of RAG

- BERTScore: 0.8106310963630676
- Cosine Similarity: 0.703851580619812

The two metrics are used to quantitatively assess whether the generated responses are semantically aligned with the user's query and used as a preliminary step to flag irrelevant outputs or hallucinations. The generated responses are also manually reviewed as these metrics are not substantial and they will not be able to verify if the responses are accurate. Therefore, after evaluating them manually, the generated responses are accurate and closely aligned with the query.

5.3 Conclusion

This project establishes a strong foundation for a legal RAG system for precedent search and will perform better with a more advanced LLM and a more comprehensive knowledge base. However, due to limited memory and budget constraints, employing a better LLM is not feasible. The memory constraints also limited the knowledge base. For future work, the knowledge base can be built with more legal cases to ensure that it has enough volume and more diverse cases and an automatic retrieval can be implemented to incorporate new cases to ensure that the system can retrieve up-to-date cases..

6 Conclusion

This project presented a RAG system that aimed to address inefficiencies of traditional search catered to legal professionals, including lawyers, judges, legal researchers, and paralegals, who require efficient retrieval of case law. By integrating LLM with a retrieval mechanism, the system aimed to address text generation challenges that standalone LLM face and improve the quality and relevance of generated responses.

The design and implementation of a RAG system aimed to enhance the accuracy and efficiency of legal precedent search, assisting legal professionals in retrieving and analysing similar case laws. This system integrates a FAISS knowledge base built using a collection of U.S. legal opinions from the CourtListener platform with query pipeline. Given that the knowledge base is a FAISS vector store, it is scalable and can be expanded as more case laws are made available. The query pipeline combines the strength of the FAISS retrieval mechanism with LLaMA 2 model for response generation.

This work contributes to the field of legal AI by improving case law retrieval through NLP techniques and ensuring the reliability of AI-assisted retrieval. It provides a foundation for future improvements, such as expanding the knowledge base, refining retrieval accuracy, incorporating domain-specific legal ontologies by fine tuning the LLM further, and improving user interaction mechanisms such as implementing user interface or refining prompt engineering. Additionally, the system could be extended to support multilingual documents by incorporating other LLMs and adapt to jurisdiction-specific requirements.

In conclusion, this project demonstrates the potential of AI in the legal domain, offering a more efficient and reliable method for precedent search. Further research and refinements will be important to make the RAG system even more effective for real-world applications in legal decision-making.

References

Acorn. (n.d.). Meta LLaMA. Retrieved March 23, 2025, from https://www.acorn.io/resources/learning-center/meta-llama/

Aletras, N., Tsarapatsanis, D., Preoţiuc-Pietro, D., & Lampos, V. (2016). Predicting judicial decisions of the European Court of Human Rights: A natural language processing perspective. PeerJ Computer Science, 2, e93. https://doi.org/10.7717/peerj-cs.93

Angwin, J., Larson, J., Mattu, S., & Kirchner, L. (2016). *Machine bias: There's software used across the country to predict future criminals. And it's biased against blacks*. ProPublica. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

Ashley, K. D. (2017). *Artificial intelligence and legal analytics: New tools for law practice in the*

digital age. Cambridge University Press. https://doi.org/10.1017/9781316761380

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (FAccT '21), 610-623. https://doi.org/10.1145/3442188.3445922

Biesheuvel, L. A., Workum, J. D., Reuland, M., van Genderen, M. E., Thoral, P., Dongelmans, D., & Elbers, P. (2024). Large language models in critical care. *Journal of Intensive Medicine*. https://doi.org/10.1016/j.jointm.2024.12.001

Bolukbasi, T., Chang, W. Y., Zou, J. Y., Saligrama, V., & Kalai, T. T. (2016). Man is to computer

programmer as woman is to homemaker? Debiasing word embeddings. *arXiv*. https://doi.org/10.48550/arXiv.1607.06520

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., Mercer,

R. L., & Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics, 16*(2), 79–85. https://aclanthology.org/J90-2002.pdf

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners (arXiv:2005.14165). *arXiv*. https://arxiv.org/abs/2005.14165

Chalkidis, I., Fergadiotis, M., & Aletras, N. (2021). *Legal-BERT: A pre-trained language model for legal text* (arXiv:2104.08671). *arXiv*. https://arxiv.org/abs/2104.08671

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation (arXiv:1406.1078). *arXiv*. https://arxiv.org/abs/1406.1078

CourtListener. (n.d.). *CourtListener API documentation.* Free Law Project. Retrieved March 23, 2025, from https://www.courtlistener.com/api/

DataCamp. (n.d.)**.** Retrieved March 23, 2025, from https://www.datacamp.com

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., & Jégou, H. (2025). The Faiss library. *arXiv.* https://doi.org/10.48550/arXiv.2401.08281

Facebook Engineering. (n.d.). Retrieved March 23, 2025, from https://engineering.fb.com

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv.* https://arxiv.org/abs/1308.0850

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Izacard, G., & Grave, E. (2021). Leveraging passage retrieval with generative models for open domain question answering. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2021)*, 1266–1277. https://doi.org/10.18653/v1/2021.eacl-main.113

Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, *7*(3), 535-547. https://doi.org/10.1109/TBDATA.2019.2921572

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys, 55*(12), Article 248. https://doi.org/10.1145/3571730

Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed.). Pearson.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th International Conference on Machine Learning (ICML)*, 282–289.

Lewis, P., Liu, Y., Goyal, N., Guestrin, C., & Lee, L. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *arXiv*. https://doi.org/10.48550/arXiv.2005.11401

Liu, Y., Guo, Z., Liang, T., Shareghi, E., Vulic, I., & Collier, N. (2023). Aligning with logic: Measuring, evaluating, and improving logical consistency in large language models. *arXiv*. https://doi.org/10.48550/arXiv.2410.02205

Loyola University Chicago Law Library. (n.d.). *Case law - Basic legal research guide*. Loyola University Chicago Law Library. Retrieved from https://lawlibguides.luc.edu/c.php?g=610789&p=4239527

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv*. https://arxiv.org/abs/1301.3781

Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network-based language model. *Interspeech*, 1045–1048.

MyScale. (2025, March 23). Retrieved from https://www.myscale.com

OpenAI. (2022). *ChatGPT: Optimizing language models for dialogue*. OpenAI Blog. https://openai.com/research/chatgpt

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. *arXiv*. https://doi.org/10.48550/arXiv.2203.02155

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. https://doi.org/10.1109/5.18626

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training [Technical report]. OpenAI. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners [Technical report]. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_mult itask_learners.pdf

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP).* https://arxiv.org/abs/1908.10084

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533–536. https://www.nature.com/articles/323533a0

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020). MPNet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems (NeurIPS 2020).* https://arxiv.org/abs/2004.09297

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks (arXiv:1409.3215). *arXiv*. https://arxiv.org/abs/1409.3215

Sheng, E., Chang, K. W., Natarajan, P., & Wallach, H. (2019). The woman worked as a babysitter: On biases in language generation. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3386–3391. https://doi.org/10.18653/v1/P19-1339

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, É., & Lample, G. (2023). LLaMA: Open and efficient foundation language models. *arXiv*. https://doi.org/10.48550/arXiv.2302.13971

Turing, A. M. (1950). Computing machinery and intelligence. *Mind, 59*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., &

Polosukhin, I. (2017). *Advances in Neural Information Processing Systems, 30*, 5998–6008. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wei, J., Lu, W., Sutskever, I., & Tenenbaum, J. B. (2022). Chain of thought prompting solves math word problems by reasoning step by step. *arXiv*. https://doi.org/10.48550/arXiv.2201.11903

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM, 9*(1), 36–45. https://doi.org/10.1145/365153.365168

Zahraei, P. S., & Shakeri, Z. (2024). Detecting bias and enhancing diagnostic accuracy in large language models for healthcare. *npj Digital Medicine*. Advance online publication. https://doi.org/10.48550/arXiv.2410.06566

Zhong, H., Zeng, B., Chen, C., & Huang, K. (2020). Does NLP help legal case retrieval? Evaluating retrieval techniques for legal case analysis. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 5284–5295. https://doi.org/10.18653/v1/2020.emnlp-main.426

Appendix

The code for this project can be found here: https://github.com/CRuxing/SC4079-Final-Year-Project