AI-Powered Legal Chatbot for Official Gazette Search and Analysis

PROJECT 4TH YEAR IASD

Team Members:
- Amina Ghandouz
- Hafsa Benghenima
- Hafssa Aouaichia
- Noussaiba Belhouari

Professor:
- Mr Ossama Serhane

Contents

List of Figures

1   Abstract

This project tackles the complex challenge of accessing and interpreting legal information from Algeria's Official Gazette, where most documents—especially those published before 2002—exist only in scanned image format. These older PDFs are not text-selectable and thus require Optical Character Recognition (**OCR**) to extract any usable content. While newer PDFs (post-2002) allow for direct text extraction using libraries such as **PDFPlumber** or **PyMuPDF**, they still present difficulties including inconsistent formatting, missing metadata, and poorly structured legal content that hinders straightforward analysis.

Traditional keyword-based search methods are inefficient due to OCR inaccuracies, the complexity of legal language, and the dynamic nature of law where frequent updates introduce ambiguity in identifying the most relevant or up-to-date version of a legal article. Moreover, existing AI models are not trained on Algeria's legal system, making their responses unreliable for domain-specific legal tasks.

To address these limitations, we propose an AI-driven legal research assistant that integrates text extraction (via OCR or library-based parsing), semantic vector search indexing, and a fine-tuned large language model (LLaMA), enhanced with a Retrieval-Augmented Generation (RAG) framework. This approach ensures that chatbot responses are grounded in verified legal sources, providing users with context-aware, legally accurate answers. Our system automates the ingestion and indexing of Official Gazette documents, enabling users to perform advanced semantic queries and retrieve precise, up-to-date legal information. Preliminary results demonstrate noticeable improvements in retrieval precision and the legal reliability of answers compared to traditional search techniques.

2 Introduction

2.1 Background

Algeria's Journal Officiel (Official Gazette) is the primary source of published laws, decrees, and legal amendments. It is essential for legal professionals, researchers, and institutions to access this information accurately and efficiently. However, a significant portion of these documents—especially those issued before 2002—exist only in scanned image formats, lacking any embedded or selectable text. This makes them inaccessible to traditional digital search tools and significantly slows down legal research processes.

Even documents published after 2002, which are often available in PDF format with selectable text, suffer from challenges such as inconsistent formatting, missing metadata, or embedded fonts that cause parsing errors. Combined with the complex and formal nature of legal language, these factors contribute to the difficulty of retrieving and interpreting legal content using standard search approaches.

2.2 Problem Statement

Current legal information systems in Algeria are outdated and lack the intelligence to handle real-world legal queries. The primary challenges include:

1. **Ineffective Text Extraction:** OCR tools struggle to produce accurate results from old, low-resolution, or degraded scans, making text unusable or misleading.

2. **Lack of Semantic Understanding:** Traditional keyword-based systems are unable to understand the intent behind complex legal questions such as *"What are the penalties for tax evasion under 2024 laws?"*.

3. **Outdated Information:** Legal texts are constantly evolving, and many platforms fail to distinguish between outdated and current versions of laws, leading to inaccurate interpretations.

2.3 Objectives

This project aims to address the above limitations through the development of an AI-enhanced legal assistant. The key objectives include:

1. **Developing a Robust OCR Pipeline:** To accurately extract text from non-selectable scanned PDFs, especially those published before 2002.

2. **Implementing Efficient Text Extraction from Selectable PDFs:** Using libraries with special handling for formatting issues (details to be discussed later).

3. **Building a Vector Search System:** To enable fast and accurate semantic retrieval of legal articles based on user queries.

4. **Fine-Tuning a Large Language Model (LLaMA) with RAG:** To provide grounded, context-aware legal answers based on retrieved documents.

2.4 Scope

- The system is designed to focus on all Algerian laws, including but not limited to civil, criminal, tax, commercial, labor, and administrative law, providing a comprehensive platform for legal

research across various domains.

- Supports user queries in both French and Arabic, in line with the bilingual nature of Algeria's legal documentation and its diverse user base.

- The project encompasses backend data processing (OCR for scanned documents, library-based parsing for selectable PDFs), semantic indexing, and integration of an AI-powered chatbot for legal question answering.

2.5 Significance

This project has the potential to significantly improve legal research efficiency and accessibility in Algeria by:

- **Accelerating Legal Research:** Reduces the time needed to locate relevant legal information.

- **Improving Accuracy:** Minimizes the risk of human error in legal interpretation by delivering precise, up-to-date responses.

- **Scalability:** Provides a scalable framework that can be adapted to other countries or legal systems facing similar challenges with historical document digitization.

3 Literature Review

3.1 Existing Solutions

Several existing technologies have been explored for processing legal documents and enabling search or question-answering capabilities, but they fall short when applied to the specific challenges posed by Algeria's Official Gazette.

- **Traditional OCR Tools (e.g., Tesseract, Adobe Scan):** These tools are commonly used to extract text from scanned documents. However, they struggle with low-quality or degraded images, particularly those found in older Official Gazette issues. Furthermore, they are not optimized for legal texts written in formal French or Arabic, which often include uncommon terms, archaic spellings, and complex formatting (e.g., columns, tables, legal references). The output often includes misplaced characters, broken lines, or misinterpreted diacritics, making raw OCR results unreliable for direct use.

- **Generic Chatbots (e.g., ChatGPT, GPT-4):** While large language models such as GPT-4 demonstrate strong general reasoning capabilities, they lack training on Algeria's legal corpus. As a result, their responses to Algerian legal questions are often vague, outdated, or based on irrelevant foreign laws. These models also cannot differentiate between repealed and active laws unless explicitly provided with up-to-date references, limiting their reliability in legal applications.

- **Vector Databases (e.g., Pinecone, FAISS):** Semantic vector search technologies like FAISS and Pinecone are powerful tools for retrieving text based on meaning rather than exact keywords. However, they are not inherently specialized for legal domain usage. Without careful document chunking, embedding customization, and legal-specific context handling, these tools may return semantically related but legally irrelevant results.

3.2 Our Innovation

Our system introduces a tailored, Algeria-specific pipeline that combines several layers of intelligence and customization:

- **Hybrid Text Extraction (OCR + Library Parsing):** We first attempt to extract text using parsing **fitz** library on all **Official Gazette** documents. This works well for PDFs published after *2002* that contain embedded text. However, in many cases — even among newer documents — the extracted text is incomplete, poorly formatted, or unreadable due to encoding issues. In such cases, we fall back to OCR-based extraction using Tesseract. To improve the quality of OCR output, we apply post-processing and cleanup algorithms, including language-specific corrections and noise filtering. Details about the cleanup techniques will be discussed later in the report.

- **Multilingual NLP Support for Arabic and French Queries:** Although Algeria's legal documents are published in both Arabic and French, for the purpose of dataset generation, we focused exclusively on French legal text extracted from the Official Gazette. However, to ensure that the system can understand and respond to both Arabic and French user queries, we integrated a multilingual embedding model during the semantic indexing phase. This allows user queries—regardless of the input language—to be mapped meaningfully to relevant legal content written in French. As a result, the system remains inclusive and accessible to a broader range of legal professionals and citizens in Algeria.

- **Retrieval-Augmented Generation (RAG):** Our solution incorporates a LLM model for question answering. Instead of relying solely on pre-trained knowledge, we adopt the RAG approach, where the model is prompted using real-time retrieved legal texts from our vector database. This ensures the answers are legally grounded, traceable, and up-to-date with the latest Gazette publications.

- **Domain Adaptation and Grounded Responses:** By grounding AI responses in Algeria-specific legal documents, our system avoids the hallucination problem common in generic chatbots. Each answer generated by the chatbot is accompanied by a direct reference to its source, such as the Official Gazette issue and article number. This ensures that the responses are not only contextually accurate but also legally verifiable, helping users confidently cross-check the information and reinforcing the trustworthiness of the system.

4 NLP and Embedding:

To build a chatbot capable of understanding and answering legal questions we rely on **Natural Language Processing** (NLP) techniques which allows the system to process and interpret human language especially complex legal texts. A key part of this is embedding, which transforms text into numerical vectors that capture meaning and context.

4.1 What is NLP:

Natural language processing is a branch of **artificial intelligence** that encompasses a wide area of software designed to reason about and act on text data. The field of NLP traces its roots back all the way to **Alan Turing** and the **Turing test**, can a computer be taught to dialogue in natural language so effectively that it would be mistaken for a human being? as such, NLP is at the core of modern artificial intelligence research.

NLP has undergone two distinct phases. The first is commonly referred to as **Good old-fashioned AI** where algorithms were written to parse language using logical rules, these techniques led to an array of *NLP* techniques in the *80's*, *90's*, and *2000's* including knowledge bases, rules-based inference machines, and syntax parsers.

The second phase of *NLP* began in the late *2000s* with the rise of *machine learning* and especially *deep learning*, instead of relying on manually crafted rules, modern *NLP* systems learn from large datasets using statistical methods, this shift enabled breakthroughs in tasks like machine translation, question answering, and text summarization and with the introduction of transformer-based models like *BERT* and *GPT*, *NLP* became far more effective at capturing the context and meaning of words in a sentence. These advancements laid the groundwork for building systems like our legal chatbot which rely on models trained to understand human language at scale.

4.2 Why NLP is important for Legal Texts:

Legal documents are often long, complex, and written in highly formal language that can be difficult for both humans and machines to interpret while *NLP* offers powerful tools to process and extract meaning from such texts with accuracy and efficiency. In the legal domain, understanding context, terminology, and relationships between entities is crucial and to determine that we use *NLP techniques* which enable automated systems to identify legal clauses, extract obligations, and detect named entities like courts, dates, or legal references.

Furthermore, legal professionals often need to search vast databases of past cases or statutes, traditional keyword search may miss relevant content due to variations in wording while *NLP* (combined with semantic search and embeddings) allows for deeper understanding and retrieval based on meaning, not just keywords.

By applying NLP to legal texts, we can build intelligent systems that support lawyers, judges, and researchers in decision-making, reduce repetitive manual work, and improve access to legal information.

4.3 The NLP Processing Pipeline:

To illustrate these *NLP* steps, we'll use the following example text

London is the capital and most populous city of England and the United Kingdom. Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia. It was founded by the Romans, who named it Londinium.

1. **Sentence Segmentation:** where we break the text apart into separate sentences and get:

   - London is the capital and most populous city of England and the United Kingdom.

   - Standing on the River Thames in the south east of the island of Great Britain, London has been a major settlement for two millennia.

   - It was founded by the Romans, who named it Londinium.

   Sentence Segmentation model can be as simple as splitting apart sentences whenever we see a punctuation mark but modern *NLP* pipelines often use more complex techniques that work even when a document isn't formatted cleanly.

2. **Word Tokenization:** The next step is to break each sentence into separate words or tokens where we get only from the first sentence

   > "London", "is", " the", "capital", "and", "most", "populous", "city",
   > "of", "England", "and", "the", "United", "Kingdom", "."

   This is the simplest *Tokenization technique*, we just split apart words whenever there's a space between them and we'll also treat punctuation marks as separate tokens since punctuation also has meaning.

3. **Predicting Parts of Speech for Each Token:** In this step, for each token we'll try to guess its part of speech whether it is a noun, a verb, an adjective and so on. Knowing the role of each word in the sentence will help us start to figure out what the sentence is talking about. We can do this by feeding each word (and some extra words around it for context) into a pre-trained part-of-speech classification model:
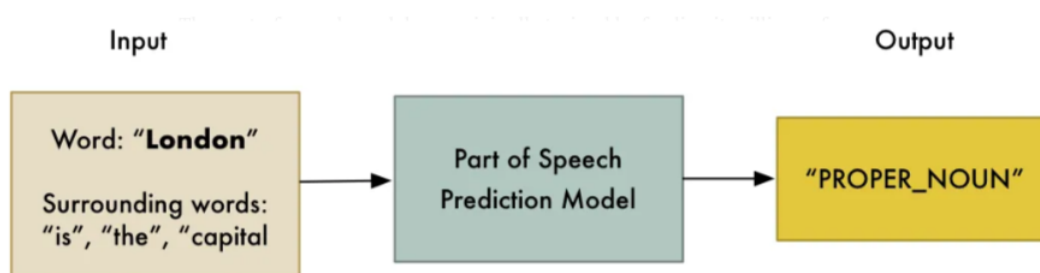
Figure 1: Part-of-speech Classification Model

The part-of-speech model was originally trained by feeding it millions of sentences with each word's part of speech already tagged and having it learn to replicate that behavior, the model is completely based on statistics so it doesn't actually understand what the words mean in the same way that humans do, it just knows how to guess a part of speech based on similar sentences and words it has seen before. We get
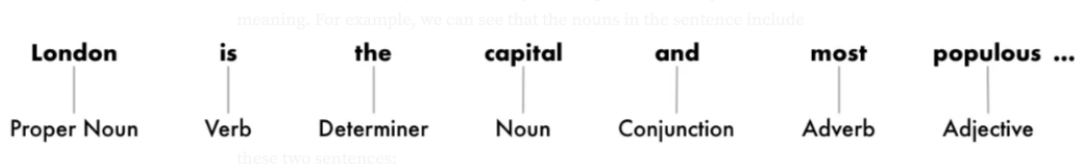
Figure 2: Result of POS Model

4. **Text Lemmatization:** For example, the followed two sentences "I trained a model" and "I trained several models" where both the sentences talk about the noun model, but they are using different inflections. When working with text in a computer, it is helpful to know the base form of each word so that we know that both sentences are talking about the same concept otherwise the strings "model" and "models" look like two totally different words that's why we need this step, it's about figuring out the most basic form or lemma of each word in the sentence, it's typically done by having a look-up table of the lemma forms of words based on their part of speech and possibly having some custom rules to handle words that we've never seen before. Cpming back to our example we'll get:
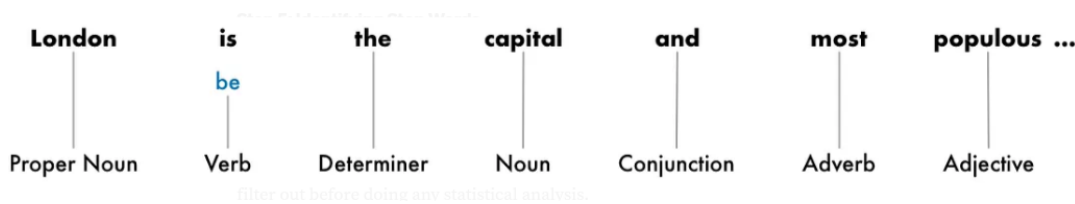
Figure 3: Lemmatization Output

5. **Identifying Stop Words:** we want to consider the importance of a each word in the sentence but we have a lot of filler words that appear very frequently like "and", "the", and "a", when doing statistics on text, these words introduce a lot of noise since they appear way more frequently than other words, we called them *stop words*, here's how our example will look with the stop words grayed out:
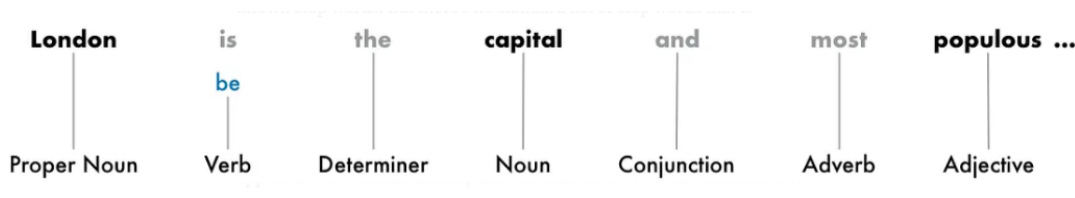
Figure 4: Remove Stop Words Output

6. **Dependency Parsing:** Now, we need to figure out how all the words in our sentence relate to each other, the goal is to build a tree that assigns a single parent word to each word in the sentence and the root of the tree will be the main verb in the sentence, we'll get:
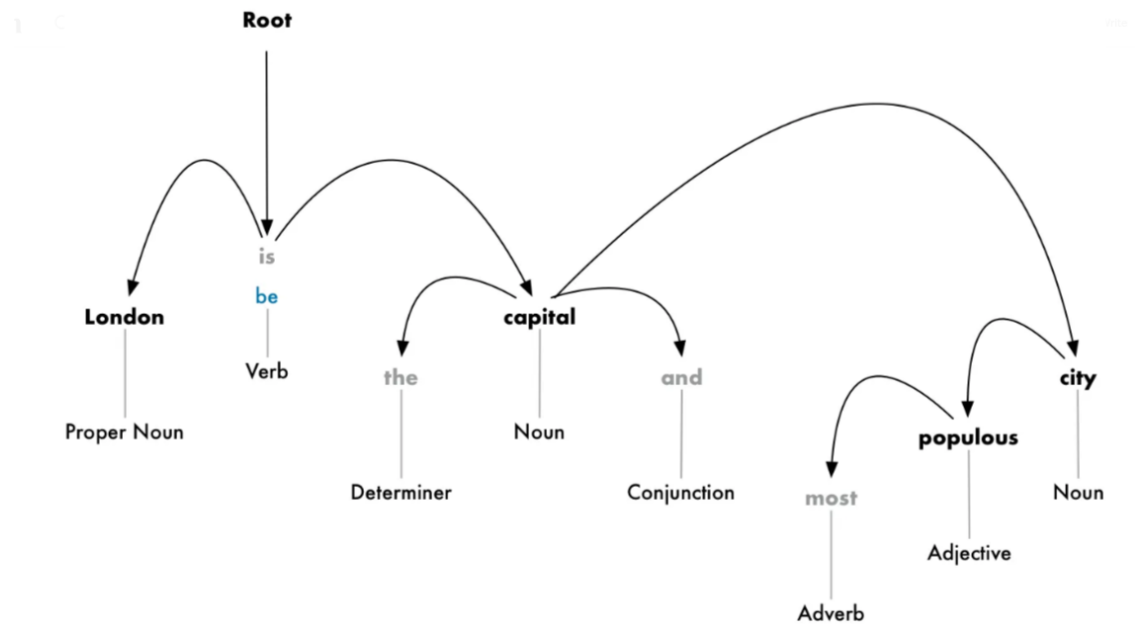
Figure 5: Dependency Parsing Step

We can also predict the type of relationship that exists between those two words.
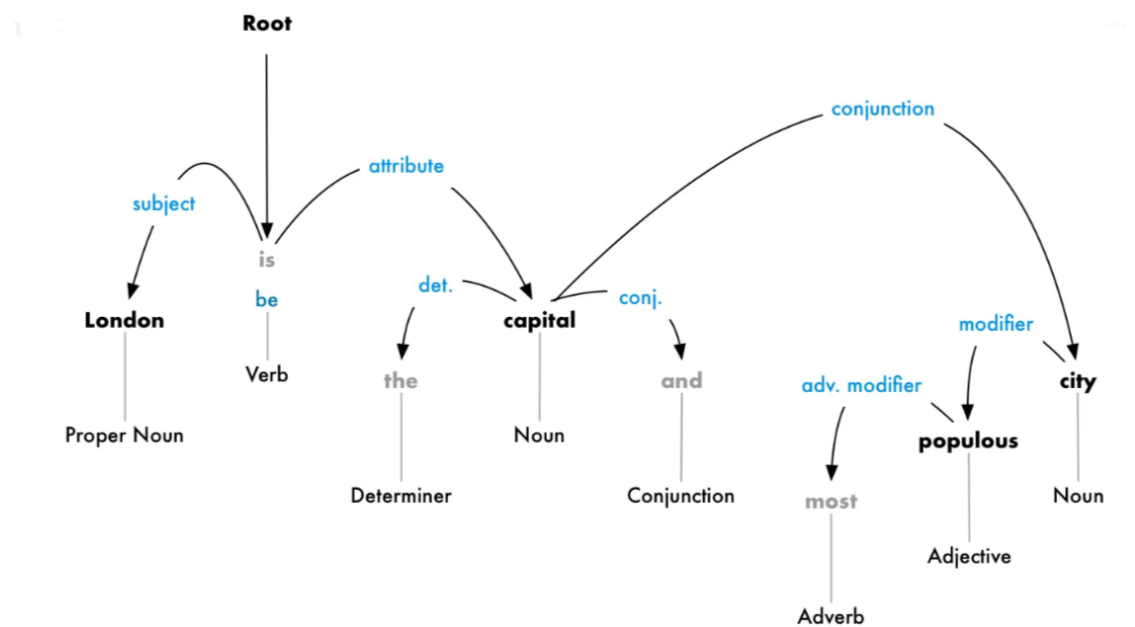
Figure 6: Dependency Parsing Step With Relationship

7. **Named Entity Recognition (NER):** The goal of NER is to detect and label these nouns with the real-world concepts that they represent, in our example we have the following nouns:

Figure 7: Sentence Example

Some of these nouns present real things in the world for example "London", "England" and "United Kingdom" represent physical places on a map, it would be nice to be able to detect that. With that information, we could automatically extract a list of real-world places mentioned in a document using NLP, here's what our example looks like after running each token through our NER tagging model:

Figure 8: NER Output

Here are some of the kinds of objects that a typical NER system can tag: People's names, Company names, Geographic locations (Both physical and political), Product names, Dates and times, Amounts of money, Names of events...

8. **Coreference Resolution:** For now, we already have a useful representation of our sentence where we know the parts of speech for each word, how the words relate to each other and which words are talking about named entities. However, we still have one big problem, any language is full of pronouns words like he, she, and it in English, these are shortcuts that we use instead of writing out names over and over in each sentence, the humans can keep track of what these words represent based on context but the NLP model doesn't know what pronouns mean because it only examines one sentence at a time, for example we can easily figure out that "It" in our paragraph in our refers to "London" but it's not as much as this easy to machines, thta's why we use this step, the goal of *coreference resolution* is to figure out this same mapping by tracking pronouns across sentences.

Figure 9: Coreference Resolution Step

4.4   Text Embedding in NLP:

4.4.1   Why Embedding:

Once the text has been cleaned and processed through the NLP pipeline, the next step is to convert it into a numerical format that machine learning models can understand, this transformation is known as text embedding. It's dense vector representations of words, sentences, or documents, unlike simple one-hot encoding, embeddings capture the semantic meaning of the text—meaning that similar words or sentences will have vectors that are close to each other in the embedding space.
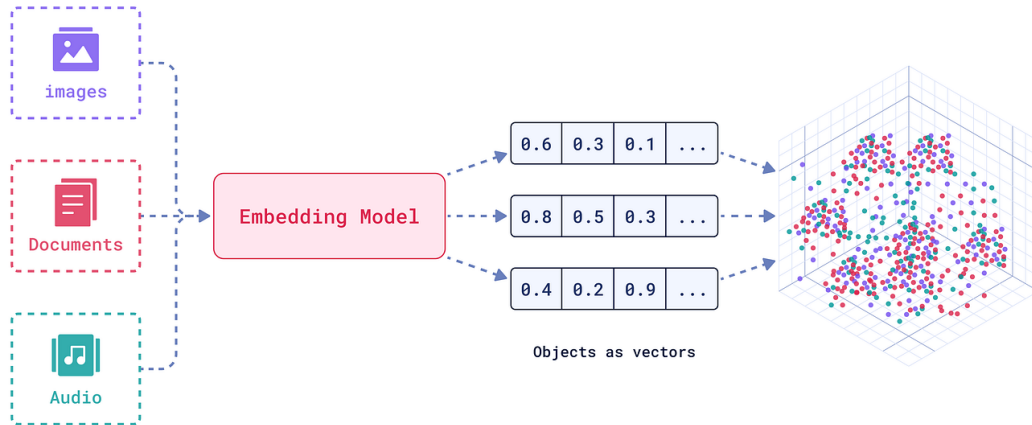
Figure 10: Embedding example

In our project, embeddings serve as the foundation for semantic search, allowing the system to match user queries with relevant legal content based on meaning rather than exact keyword matches, this is crucial for legal texts, where the same concept can be expressed in multiple ways. We used a pre-trained embedding model fine-tuned (we'll speak about the details later) on our specific legal dataset. This customization allows the model to better understand the domain-specific vocabulary, structure, and nuances of legal language.
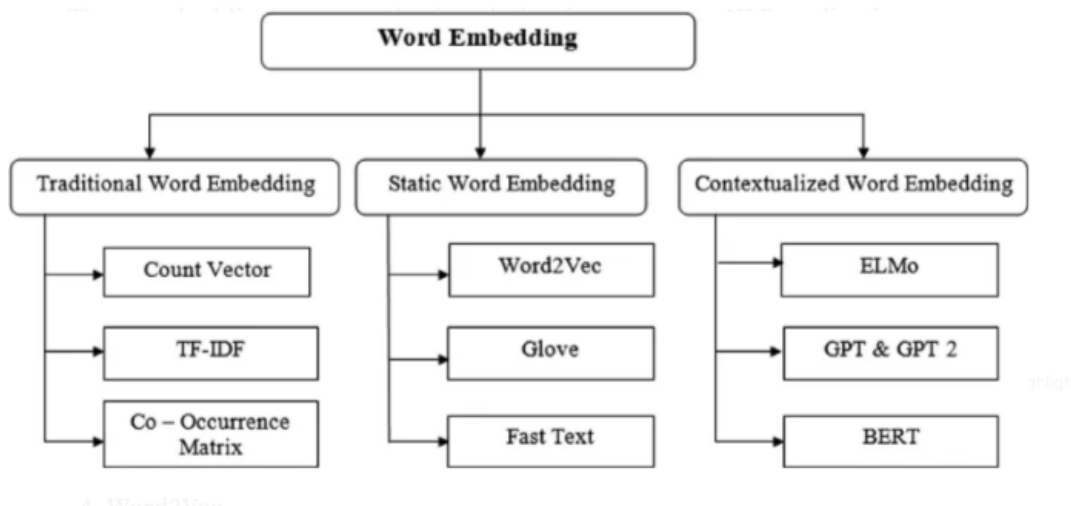
Figure 11: Word Embedding

4.4.2   Evolution of Word Representation Techniques:

- **One Hot Encoding:** one of the simplest forms of representing words numerically where each word is represented as a binary vector with the length of the entire vocabulary, the vector has a "1" in the position corresponding to the word's index and "0" elsewhere. It's simple and interpretable but high dimensionality and doesn't capture semantic relationships.

Word	AI	transforms	modern	digital	systems
AI	1	0	0	0	0
transforms	0	1	0	0	0
modern	0	0	1	0	0
digital	0	0	0	1	0
systems	0	0	0	0	1

Table 1: One-Hot Encoding for a 5-Word Sentence

- **Bag of Words(BoW):** it's a simple technique in NLP for representing text documents as numerical vectors, the idea is to treat each document as a bag, or a collection, of words, and then count the frequency of each word in the document, it doesn't consider the order of words but provides a straightforward way to convert text into vectors, that's it can capture word importance based on frequency but it ignores word order and context. Lets consider the following example

  We are lucky to live in an age in which we are still making discoveries

  It will become

Figure 12: Bag Of Word example

- **TF-IDF (Term Frequency-Inverse Document Frequency):** it's an improvement over *BoW* which considers the importance of words by reducing the weight of common words while increasing the weight of rare ones, it's used to rank the importance of words in a document, with higher scores indicating more important words, it considers both term frequency and rarity but still doesn't capture semantic relationships.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

The idea behind *TF-IDF* is to calculate the importance of a word in a document by considering two factors:

- **Term Frequency (TF):** how frequently a term appears in a document.

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where:
- $f_{t,d}$: frequency of term t in document d
- denominator: total number of terms in document d

– **Inverse Document Frequency (IDF):** how important a term is across all documents in the corpus.

$$\text{IDF}(t, D) = \log \left(\frac{N}{|\{d \in D : t \in d\}|} \right)$$

where:
- N: total number of documents
- $|\{d \in D : t \in d\}|$: number of documents containing term t

- **Word2Vec:** it's a *neural network-based model* that generates dense vector representations of words, the basic idea behind it, is to train a *neural network* to predict the context words given a target word, and then use the resulting vector representations to capture the semantic meaning of the words, it captures semantic relationships between words and provide lower dimensionality than classical methods but requires training on a large corpus and doesn't handle *out-of-vocabulary* (OOV) words well. It captures semantic relationships between words using two main approaches:

  – **Continuous Bag-of-Words (CBOW):** predicts the target word given its surrounding context words.

  – **Skip-gram:** predict the surrounding context words given a target word.
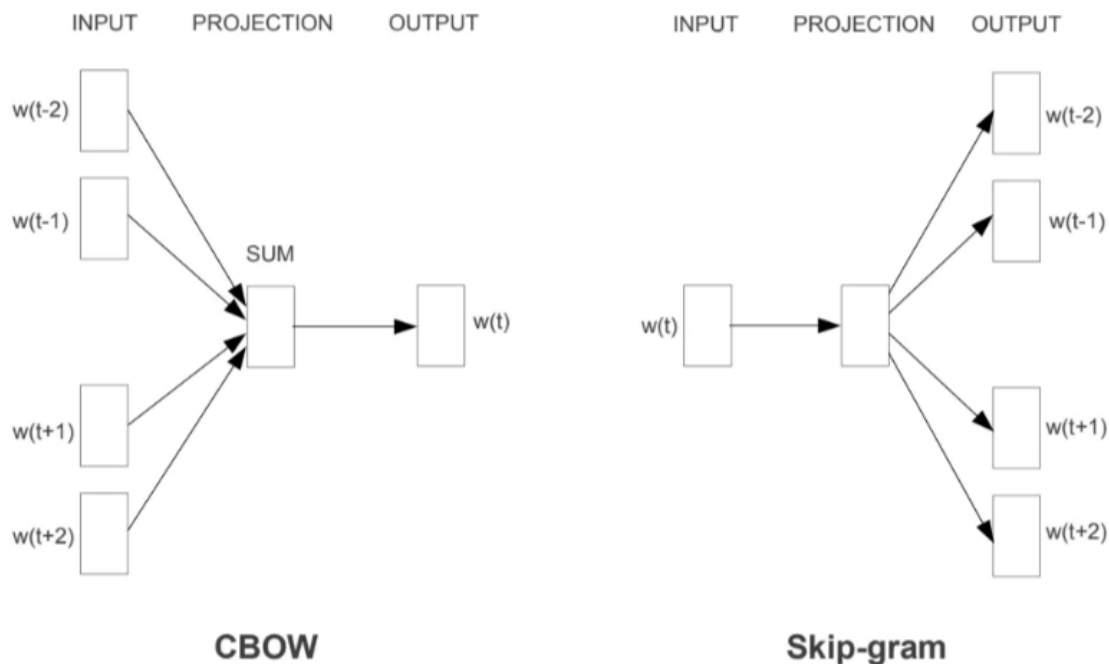
Figure 13: Word2Vec architecture

- **GloVe (Global Vectors for Word Representation):** it's an unsupervised learning algorithm for obtaining vector representations for words, it aims to capture both global word co-occurrence statistics and local context window-based information (like *Word2Vec*) to create high-quality word vectors. The core idea is that the ratios of word-word co-occurrence probabilities encode meaningful semantic relationships, for example consider the words *ice* and *steam* and their co-occurrence

probabilities with other words like *solid* and *gas*, the ratio

$$\frac{P(\text{solid} \mid \text{ice})}{P(\text{solid} \mid \text{steam})}$$

would be high, indicating a stronger association between *ice* and *solid* compared to *steam* and *solid*. *GloVe* captures both global and local context and it's faster training than *Word2Vec* but in terms of *OOV* words it's similar to *Word2Vec*.

- **FastText:** unlike traditional word embedding models that we already mentioned that treat each word as a distinct unit, *FastText* considers words as a bag of character n-grams (subwords) where an n-gram is a contiguous sequence of n characters, it's efficient Handling of Out-of-Vocabulary (OOV) words by utilizing subword information, it can generate meaningful representations for rare words or words not seen during training, this is particularly useful for morphologically rich languages (e.g., *agglutinative languages*) and in situations where the vocabulary is constantly evolving. *FastText* employs two key techniques to achieve high speed:

  - *Hierarchical Softmax:* a hierarchical tree structure is used to represent the output labels, significantly reducing the computational complexity of the softmax function during training and prediction.

  - *Negative Sampling:* only a small subset of the output labels (negative samples) are updated during training, making updates much faster.

  FastText handles the *OOV* words and captures morphological information but it's a higher dimensionality than *Word2Vec*.

- **ELMo (Embeddings from Language Models):** it's a deep contextualized word representation that models both complex characteristics of word use (e.g., syntax and semantics) and how these uses vary across linguistic contexts (e.g., to model polysemy). *ELMo* generates context-sensitive embeddings using deep bi-directional *LSTM* models, it provides different embeddings for words based on their context. ELMo embeddings vary depending on the context in which a word appears, this is different from traditional embeddings like *Word2Vec* or *GloVe* which provide a single representation for each word regardless of context, it also uses *deep bidirectional LSTM networks* to generate embeddings, capturing complex syntactic and semantic information. It's contextualized embeddings and Bi-directional context but also causes the complexity and size problem with a fixed context window.

- **BERT (Bidirectional Encoder Representations from Transformers):** *BERT* is a *transformer-based model* that generates context-aware embeddings by considering the entire sentence bidirectionally (i.e., both *left-to-right* and *right-to-left*) unlike traditional word embeddings like *Word2Vec* or *GloVe* which generate a single representation for each word, it can produce different embeddings for each word based on its context. It's a pre-training and fine-tuning model, contextualized embeddings and bi-directional context but exactly like *ELMo* it also causes the complexity and size problem.

- **GPT (Generative Pre-trained Transformer):** it's a transformer-based model that generates context-aware embeddings by processing text in a unidirectional manner (left-to-right), it uses the decoder part of the transformer, to process the input text. The model consists of multiple layers of self-attention mechanisms which allow it to capture dependencies between words in a sentence, it's effective for language generation, fine-tuning capability but only unidirectional context, resource intensive and context length limitation.

- **Sentence Transformers:** a type of *neural network architecture* designed to create high-dimensional vector representations of entire sentences or paragraphs, rather than individual words, these embeddings capture the semantic meaning of the text, enabling systems to understand the context, relationships, and intent behind the words used in a sentence, they built on the architecture of *BERT* and similar *transformer models*, they are tailored specifically for tasks requiring sentence-level understanding, such as semantic search, question answering and document retrieval (our case). Several models are commonly used:

  - **BERT-based models:** like SBERT, these models are based on the original *BERT* transformer architecture and are highly effective at producing sentence embeddings for similarity tasks.

  - **RoBERTa-based models:** an improved version of *BERT*, they are fine-tuned for performance and accuracy in tasks requiring deep contextual understanding.

  - **DistilBERT and MiniLM:** lighter and faster alternatives to *BERT* that provide efficient sentence embedding generation with less computational overhead.

4.4.3   Used Embedding Models:

Brief overview of the tested models:

- **SBERT (Sentence-BERT):** Fine-tunes *BERT* with Siamese and triplet networks for sentence-pair similarity tasks,it performs well for general text but is not domain-specific.

- **MPNet (Masked and Permuted Pre-training Network):** a pre-trained transformer model that improves on *BERT* and *RoBERTa* by combining masked language modeling and permuted language modeling, its performance is better than *SBERT* in capturing text similarity but it's not optimized for *French* or *domain-specific* legal texts.

- **Legal-BERT / CaseLaw-BERT:** domain-adapted BERT models trained on legal corpora (statutes, case law) but their size and complexity make them impractical for embedding a large dataset within our resource constraints which would increase cost and slow down our pipeline.

- **legal_bert_fr / legal_bert_fr2:** French domain-specific BERT models trained on legal documents but they but are computationally heavy and slow compared to our chosen model (we will speak it later).

- **CamemBERT:** a French RoBERTa-based mode but not domain-specific and it's still a large model that demands substantial resources.

- **LaBSE (Language-agnostic BERT Sentence Embedding):** it's designed for multilingual tasks but it was less effective in our monolingual legal domain.

- **Sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2:** a multilingual version of the MiniLM model fine-tuned to handle paraphrase detection and other semantic tasks in multiple languages. It was trained using a knowledge distillation process, where it learns to mimic the behavior of a larger teacher model, it's the chosen model and we'll know why.

Criterion	The selected model	the other models
Model size	Small and efficient	Large, resource-intensive
Speed and Scalability	Fast and scalable	Slow on large datasets
Resource constraints	Fits well for limited resources	Demands high compute and memory

Table 2: Comparison between the chosen model and the others

4.5 The chosen model, why?:

Sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 is also a Sentence-transformers model which are:

- **Architecture:** they are built on transformer architectures, which excel in capturing contextual relationships in sequential data, transformers use *self-attention* mechanisms, allowing the model to weigh different parts of the input text when creating embeddings which helps capture long-range dependencies and context.

- **Context-Aware Embeddings:** unlike traditional word embeddings that represent words in isolation, sentence transformers consider the entire context of a sentence which enables them to understand the relationships between words and their meanings within the context of the complete sentence.

- **Semantic Similarity:** a significant application of sentence transformers is in measuring semantic similarity between sentences or text passages where the generated embeddings by these models capture the semantic content of the text, allowing for precise comparison and similarity scoring.

- **Resource Efficiency:** they often generate lower-dimensional embeddings compared to more complex models which reduces the memory and computational requirements making them suitable for resource-constrained environments. They can achieve good performance with fewer labeled samples, making them efficient when data is limited. they generally have faster inference times, making them suitable for real-time applications or scenarios where quick responses are required, and due to their lower resource requirements, they can be deployed on a broader range of device including those with limited computational power.

4.6 Fine-tuning:

4.6.1 What is Fine-tuning:

Fine-tuning is the process of taking a *pre-trained* model and further training it on a domain-specific dataset. In our case, this involves adapting the pre-trained *multilingual-MiniLM* model to better understand and represent legal *French* texts by exposing it to examples from our dataset.

4.6.2 Why Fine-tuning:

Although MiniLM performs well on general and multilingual tasks, it is not specifically trained on legal texts so fine-tuning allows the model to:

- Learn domain-specific vocabulary and terminology used in legal documents.

- Adapt to the linguistic patterns and context of French legal language.

- Improve its accuracy and relevance for downstream tasks such as legal text similarity, search, and classification.

4.6.3 How is Fine-tuning Done:

The fine-tuning process includes the following steps:

1. **Dataset Preparation:** Format the legal French dataset into negative and positive pairs suitable for training (e.g., sentence pairs for similarity tasks), in our case since we don't have that format of the needed dataset we did train K-means model to generate clusters for the both negative and positive pairs.

2. **Model Loading:** Load the pre-trained MiniLM model using a transformer library.

3. **Training Setup:** Define a training objective (such as contrastive loss or cosine similarity loss), batch size, learning rate, and number of epochs.

4. **Fine-tuning:** Train the model on the dataset using GPU/CPU resources. The model updates its weights to better encode the legal text semantics.

Fine-tuning transforms a general-purpose model into one that is optimized for a specific domain—in this case (the legal domain in French) improving both precision and contextual understanding for our application.

4.7 How does embedding improve semantic search:

4.7.1 What is Semantic Search:

Semantic search is an advanced information retrieval technique that goes beyond simple keyword matching to understand the **meaning** and **context** behind the user queries. Instead of looking for exact word matches, semantic search tries to determine what the user really means, even if the words used are different from those in the documents.

4.7.2 How is it different from traditional search:

Traditional search methods for example *lexical* or *keyword-based search* match user queries with documents based on the **exact terms** found in both which can lead to irrelevant or incomplete results when:

- The query and document use synonyms for example *attorney* vs *lawyer*.

- The document uses different phrasing or sentence structure.

- The relevant information is contextually implied rather than explicitly stated.

Semantic search overcomes these limitations by using machine learning models to understand conceptual similarities between texts.

4.7.3 The role of embeddings in semantic search:

Embeddings convert text into dense vectors that represent its semantic meaning where these vectors allow the system to:

- Compare the meaning of a query and a document.

- Rank documents based on semantic closeness using cosine similarity.

- Retrieve relevant results even if the wording differs significantly.

4.7.4 Application in our system:

In our legal search system:

- Each legal document is pre-processed and transformed into a vector.

- When a user submits a query, it is also embedded.

- A similarity score is computed between the query and each document embedding.

- The top-matching legal texts are returned based on their semantic similarity not just keyword overlap.

4.7.5 Benefits of semantic search in legal context:

- **Improved Relevance:** by finding the documents that match the intent of the query.

- **Robustness:** by handling different ways of asking the same legal question.

- **Language-Awareness:** which captures the nuances and complexity of legal language.

- **Multilingual Support:** when using multilingual models, supports queries across different languages.

4.7.6 FAISS:

FAISS or **Facebook AI Similarity Search** is a popular open-source library for fast similarity search. It is optimized for high-dimensional vector spaces and can perform approximate nearest-neighbor (ANN) searches, drastically reducing search time without significantly compromising accuracy. *FAISS* is highly scalable, making it ideal for large datasets.

Semantic search powered by embeddings provides a smarter, more intuitive way to retrieve legal information, tailored to the actual meaning of the user's query rather than just its wording.

5 LLM:

5.1 What are LLMs:

Large language models are deep neural networks that can understand and generate human-like language, they are trained on massive datasets of text and code, which allows them to learn the patterns and relationships that exist in language. *LLMs* can be used for a variety of tasks, such as:

- **Translate languages:** they can be used to translate text from one language to another.

- **Analyze sentiment:** they can be used to analyze the sentiment of the text for example determine whether a customer review is positive or negative.

- **Generate creative text formats:** they can be used to generate creative text formats, such as poems, code, scripts, musical pieces, emails, letters, etc.

Here are some real-world examples of LLMs in action:

- **GPT-4:** is a large language model developed by *OpenAI* .

- **DeepSeek-V3:** a Chinese artificial intelligence LLM that developed by *Deepseek*.

- **Gemini** or **Bard** previously, an LLM developed by *Google AI*.

5.2 How LLM is built:

Large language models are trained on massive amounts of text data, this data is used to teach the model the statistical relationships between words, phrases, and sentences which allows the model to generate coherent and contextually relevant responses to prompts or queries. They are typically too large to run on a single computer, so they are provided as a service over an API or web interface, this means that we can access the model's capabilities without having to download or install it ourself. One example of an LLM is *ChatGPT's GPT-4* model which was trained on massive amounts of internet text data and gave it the ability to understand various languages and possess knowledge of diverse topics. As a result, it can produce text in multiple styles. While GPT-4's capabilities may seem impressive, they are not surprising because LLMs operate using special *grammar* that matches up with prompts. These grammars tell the model how to generate text that is relevant to the prompt.

5.3 General Architecture:

The architecture of LLMs is composed of multiple layers of neural networks. These layers include embedding layers, recurrent layers, feedforward layers, and attention layers, each layer helps the model process the input text and generate output predictions.

- **Embedding layer:** converts each word in the input text into a high-dimensional vector representation. This representation captures semantic and syntactic information about the word, which helps the model understand the context.

- **Feedforward layers:** apply nonlinear transformations to the input embeddings which helps the model learn higher-level abstractions from the input text.

- **Recurrent layers:** interpret information from the input text in sequence. They maintain a hidden state that is updated at each time step, allowing the model to capture the dependencies

between words in a sentence.

- **The attention mechanism:** allows the model to focus selectively on different parts of the input text. This helps the model attend to the input text's most relevant parts and generate more accurate predictions.

5.4 Open Source Language Model:

Open-source LLMs have empowered developers and researchers to create powerful AI applications without relying on proprietary models. *DeepSeek* is a family of open-source large language models developed by a *Chinese AI company*, designed to support both English and Chinese languages, and optimized for general reasoning, language tasks, and code generation. It serves as a competitive alternative to commercial models like GPT-3.5 and GPT-4. *DeepSeek-V3* is the latest version with 671B total parameters.

5.5 Used LLM models:

1. **OlympicCoder 32B**:

   - **Specialization:** Code generation and programming assistance.

   - **Architecture:** Dense Transformer model with 32B parameters.

   - **Training:** Specialized code dataset (multi-language, stack traces, repos).

   - **Performance:** Strong on HumanEval, MBPP, and code translation tasks.

   - **Few-shot/Zero-shot:** Reliable few-shot code generation and error correction.

   - **GPU Requirements:** 1× H100 or 2× A100 for optimal speed.

   - **Limitations:** General language understanding not as strong as dedicated chat LLMs.

2. **Gemma 3 27B Instruct**:

   - **Specialization:** Lightweight open LLM optimized for dialogue and tasks.

   - **Architecture:** Decoder-only Transformer, 27B parameters, optimized memory use.

   - **Training:** Mixture of English web, code, and dialogue data.

   - **Performance:** Competitive with Mistral and Mixtral on commonsense tasks.

   - **Few-shot/Zero-shot:** Capable zero-shot and few-shot performance in chat.

   - **GPU Requirements:** 1× H100 or multi-GPU setup for FP16.

   - **Limitations:** Limited multilingual and domain-specific accuracy.

3. **LLaMA 3.3 70B Instruct**:

   - **Specialization:** Instruction-tuned for chat, reasoning, and coding tasks.

   - **Architecture:** Decoder-only Transformer, 70B parameters, 80 layers.

   - **Training:**Trained on over 15T tokens of multilingual and code data.

   - **Performance:** Strong performance on MMLU, GSM8K, HumanEval.

- **Few-shot/Zero-shot:** Excellent in zero-shot reasoning and multi-step QA.

- **GPU Requirements:** 1× H100 (8-bit), 2× H100 (16-bit).

- **Limitations:** Still underperforms GPT-4 on complex coding tasks.

4. **Qwen2.5 VL 72B Instruct**:

   - **Specialization:** Multimodal (Vision + Language) instruction-following.

   - **Architecture:** MoE Transformer, 72B parameters, Vision-Language backbone.

   - **Training:** Multimodal corpus including web images, charts, and natural text.

   - **Performance:** Strong image captioning, chart QA, OCR understanding.

   - **Few-shot/Zero-shot:** Effective zero-shot visual QA.

   - **GPU Requirements:** 1× H100 or A100 for inference.

   - **Limitations:** Limited performance on complex document layout reasoning.

5. **Llama 3.1 Nemotron Ultra 253B v1**:

   - **Specialization:** General-purpose but excels in complex reasoning and multilingual tasks (including legal QA).

   - **Architecture:** Mixture-of-Experts (MoE) Transformer, 253B params, 64 layers.

   - **Training:** Trained on 1.2T tokens (multilingual mix, including legal corpora).

   - **Performance:** Achieved 0.92 accuracy .

   - **Few-shot/Zero-shot:** Strong zero-shot capability for nuanced legal queries.

   - **GPU Requirements:** 4× H100 (expensive for large-scale deployment).

   - **Limitations:** Proprietary (API-only), high inference cost.

6. **Rogue Rose 103B v0.2**:

   - **Specialization:** Explicitly fine-tuned for legal reasoning (statute interpretation, case law analysis).

   - **Architecture:** Dense Transformer, 103B params, 48 layers.

   - **Training:** 500B tokens (0.2 legal texts in EN/FR).

   - **Performance:** 0.89 accuracy .

   - **Few-shot/Zero-shot:** Requires 2–3 examples for context-heavy questions.

   - **GPU Requirements:** 2× A100 (80GB).

   - **Limitations:** Slow inference (15s/response), though weights are open-source.

7. **Bytedance UI Tars 72B**:

   - **Specialization:** Multilingual (French/Arabic optimized) with strong OCR-noise tolerance.

   - **Architecture:** MoE Transformer, 72B params, 40 layers.

- **Training:** 700B tokens (includes Chinese/EU legal texts).

- **Performance:** 0.87 accuracy .

- **Few-shot/Zero-shot:** Effective zero-shot for simple queries.

- **GPU Requirements:** 2× H100.

- **Limitations:** Proprietary (no local deployment).4. Featherless Qwerky 72B

8. **Featherless Qwerky 72B**:

   - **Specialization:** Structured output (JSON/XML) for legal article extraction.

   - **Architecture:** Dense Transformer, 72B params, 36 layers.

   - **Training:** 400B tokens (includes legal databases).

   - **Performance:** 0.84 accuracy .

   - **Few-shot/Zero-shot:** Requires explicit output templates (e.g., JSON schema).

   - **GPU Requirements:** 2× A100 (40GB).

   - **Limitations:** Weak Arabic support.

9. **Qwen 2.5 72B Instruct**:

   - **Specialization:** Instruction-tuned for RAG workflows.

   - **Architecture:** MoE Transformer, 72B params, 32 layers.

   - **Training:** 1.2T tokens (0.05 legal texts in CN/EN).

   - **Performance:** 0.88 accuracy .

   - **Few-shot/Zero-shot:** Reliable zero-shot for straightforward QA.

   - **GPU Requirements:** 1× H100.

   - **Limitations:** Limited French/Arabic legal data in pretraining.

10. **DeepSeek R1 Distill Llama 70B**:

    - **Specialization:** Distilled version of Llama 3, cost-performance tradeoff.

    - **Architecture:** Dense Transformer, 70B params, 30 layers.

    - **Training:** 300B tokens (includes EU legal corpus).

    - **Performance:** 0.9 accuracy .

    - **Few-shot/Zero-shot:** Few-shot needed for complex queries.

    - **GPU Requirements:** 1× A100 (40GB).

    - **Limitations:** Falls short on multilingual edge cases

11. **DeepSeek Chat V3 (0324): Free**:

    - **Specialization:** Open general-purpose chat model with instruction-following.

- **Architecture:** Transformer-based LLM, size 67B parameters (estimate).

- **Training:** Massive multilingual web corpus including math and code.

- **Performance:** Competitive with GPT-3.5 on reasoning and casual dialogue.

- **Few-shot/Zero-shot:** Effective instruction following with zero-shot prompts.

- **GPU Requirements:** $1\times$ H100 (8-bit), FP16 needs $2\times$ A100.

- **Limitations:** Open/free model may lack stability in highly structured tasks.

So the chosen one was **DeepSeek Chat V3 (0324): Free**.

5.6 Why we didn't Fine-Tune the LLM Model:

In our project, we made the strategic decision not to *fine-tune* The *LLM*, it was based on several technical, practical, and ethical considerations, which are :

1. **Limited Computational Resources:** *Fine-tuning* a large-scale *LLM* typically requires substantial computational power and high-end *GPUs* or *TPUs*, large memory capacity, and long training times. These requirements significantly increase the cost and complexity of deployment. As we were operating within limited infrastructure constraints, fine-tuning was not a viable option.

2. **Use of RAG:** Since we chose to implement a *RAG system* as an efficient and scalable alternative to fine-tuning which allows the model to retrieve relevant, up-to-date, or domain-specific information from an external data source during inference, we decided to eliminate the *LLM* fine-tuning.

3. **knowledge Cutoff and Static Learning:** *LLMs* once trained don't have access to new information unless they are updated or fine-tuned, this poses a limitation, especially in domains where data is evolving or context-dependent so fine-tuning it requires new training for every dataset update, but why to do so when the *RAG* allows the model to learn from updated content in real time by querying the knowledge base.

4. **Data Sovereignty and Sensitive Information:** another major concern is data sovereignty and privacy where fine-tuning a model often involves uploading data to external systems or cloud environments where we may lose control over how the data is used or stored. In cases where datasets contain sensitive, private, or nationally critical information exactly as our dataset, it's essential to preserve full control and ownership of the data. Using *RAG* ensures that sensitive datasets stay within our infrastructure and are only accessed at inference time, rather than being embedded in the model parameters.

5.

6 RAG Process:

One of the encountered problem that we can face when using some LLMs exactly one asking about the latest updates on a topic, we receive responses like '*Sorry, I cannot provide real-time data up to 2025.*' In essence, this is a fundamental limitation of LLMs; their knowledge is essentially frozen at the last training point and cannot learn or remember new information unless retrained, that's why the **RAG technology** is needed to overcome this limitation.

6.1 What is RAG:

RAG stands for **Retrieval-Augmented Generation**, it retrieves more information and data from external knowledge bases through information retrieval, combining the capabilities of *traditional large language models* with information retrieval functions. As an example, if we are a journalists tasked with reporting the latest developments of an event. Firstly, we would research the event, collect related articles or reports, and then use this information to craft our news story. For *LLMs*, *RAG* employs a similar method where *Retrieval* is the gathering of relevant information, and *Generation* is using this information to compose the news article.

6.2 Why RAG is Needed:

To know why *RAG* is needed we need to image that we're now an online shop owners with excellent sales, we are frequently needing to list new products or delist sold-out items so we require an intelligent customer service robot to respond to user inquiries about product details, stock, prices, and other questions. The problem now is that *LLMs* have the following limitations which could lead to a poor experience for our buyers:

- **Lack of Specificity in Information:** because *LLMs* are limited to providing generic answers based on their training data, if buyers ask questions related to our shop or products, *traditional LLMs* may not be able to provide accurate answers because they have not been specifically trained on data from our shop. Additionally, the *LLMs* training data often have a cutoff date, limiting their ability to provide the latest information.

- **Fabricated Answers:** *LLMs* might *fabricate answers* meaning they could confidently generate incorrect answers based on non-existent facts, also, if they don't have precise answers to users queries, such algorithms might provide completely irrelevant answers leading to a very poor user experience.

- **Generic Answers:** *LLMs* typically provide generic answers that are not tailored to specific situations which can be a significant drawback in certain scenarios. Therefore, more specialized *LLMs* of smaller scale, targeted at specific domains, have emerged in the market.

6.3 Components of the RAG System:

The *RAG system* is a complex assembly composed of multiple components, with the *LLM* being just one of them, those components include:

- **Data Sources:** these store the information to be retrieved, such as product databases, store details or pdf files content in our case.

- **Data Processing Module:** which is responsible for converting data into a format suitable for use by the *RAG system*, such as performing data chunking and generating document embeddings.

- **Retriever:** in charge of retrieving relevant information from the data sources based on user queries, using retrieval techniques like keyword-based search, document retrieval, or structured database queries to obtain related data.

- **Ranker:** which optimizes the retrieved information by evaluating its relevance and importance, ensuring that the most pertinent information is presented to the *LLM* for content generation.

- **Generator:** tasked with obtaining the retrieved and ranked information along with the user's original query and generating the final response or output through the *LLM*. It ensures that the response is consistent with the user's query and incorporates factual knowledge retrieved from external sources.

The *RAG system* retrieves relevant information from data sources based on user queries, then passes this information to the *LLM* for processing, which uses it to generate the final reply.
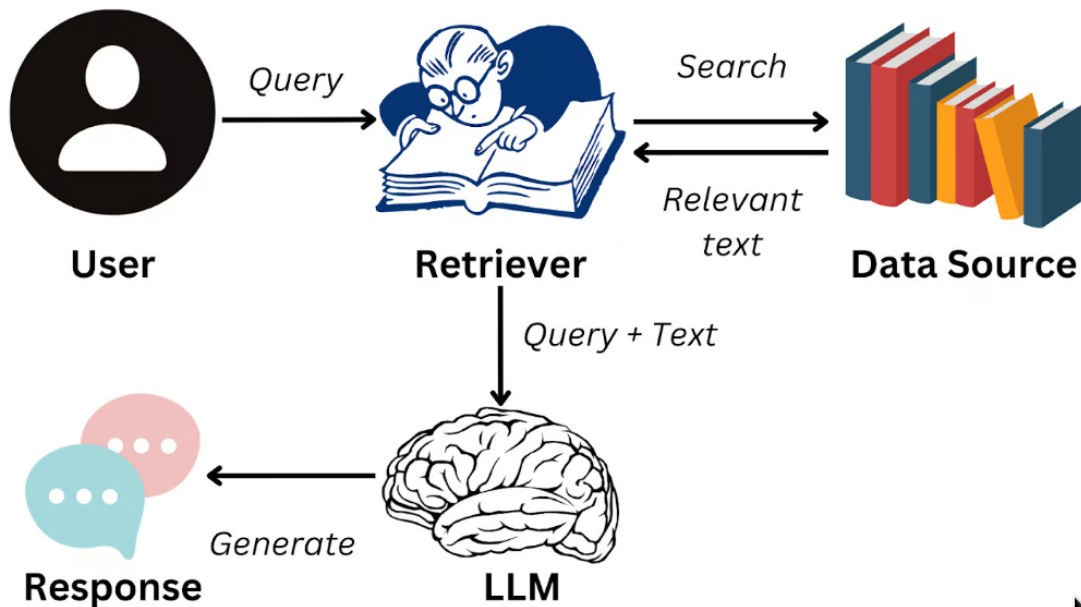
Figure 14: RAG System

6.4   Advantages of RAG:

One of the main advantages of *RAG* is its ability to provide answers that are more contextually appropriate while also improving the accuracy of responses, especially for complex questions that require a deep understanding of the topic. Additionally, *RAG* can be fine-tuned for specific industries, making it a versatile tool applicable to various applications, unlike pre-trained models, *RAG*'s internal knowledge can be easily modified or even supplemented in real-time which allows researchers and engineers to control what *RAG* knows and doesn't know without wasting time or computational power retraining the entire model. Some other advantages can be resumed in:

1. **Enhanced Coherence:** the integration of external context ensures that the content generated by RAG maintains logical fluidity and coherence, which is extremely important, especially when generating longer texts or narratives.

2. **Versatility:** RAG is widely applicable and can adapt to various tasks and query types because they are not limited to specific domains and can provide relevant information on various topics.

3. **Efficiency:** RAG can efficiently access and retrieve information from large data sources and saving time compared to manual searches. This efficiency is particularly important in applications that require quick responses, such as chatbots.

4. **Content Summarization:** RAG is suitable for summarizing lengthy documents or articles by selecting the most relevant information and generating concise summaries.

5. **Multilingual Capabilities:** RAG can access and generate content in multiple languages making it suitable for international applications, translation tasks, and cross-cultural communication.

6. **Decision Support:** RAG can support the decision-making process by providing well-researched, fact-based information and helping make informed choices in various fields such as healthcare, finance, law, etc.

7. **Reduced Manual Workload:** RAG reduces the need for manual research and information retrieval, thereby saving manpower and resources. This is particularly valuable in situations where large amounts of data need to be processed.

8. **Innovative Applications:** RAG opens the door to innovative NLP applications, including intelligent chatbots, virtual assistants, automated content generation, etc., enhancing user experience and work efficiency.

6.5   Challenges and Limitations of RAG:

Despite the many advantages of RAG, this doesn't mean that it's without challenges and limitations. One of the main challenges is managing the complexity of the model. Integrating the retriever and generator of the RAG $system$ into a single model leads to high model complexity. This complexity increases when there are multiple external data sources in different formats. However, we can mitigate this problem by training the retriever and generator separately, which can simplify the training process and reduce the required computational resources. Another challenge is ensuring that the LLM can effectively utilize the retrieved information. There is a risk that $LLMs$ may ignore the retrieved documents and rely solely on their internal knowledge, leading to reduced accuracy or relevance of the generated responses. We can address this issue by fine-tuning the LLM based on the output results of the retriever, ensuring that it fully utilizes the retrieved documents. However, even with these solutions, there are still some limitations to consider when using RAG:

- **Quality of Search Tools and Data:** The quality of RAG-$generated$ $answers$ largely depends on the quality of the search tools and data. If the search tools cannot retrieve relevant and accurate documents or the retrieved data quality is not good enough, then the quality of the answers will also decline.

- **Scalability:** As the amount of data increases, maintaining the RAG $system$ will become more challenging. There are many complex operations and tasks to perform, such as generating embeddings, comparing the meanings between different text fragments, and retrieving data in real-time.

Since these operations and tasks are computationally intensive, the system may slow down as the size of the source data increases.

- **Limited Internal Knowledge:** Completely ignoring the internal knowledge of the language model will limit the number of questions that can be answered. Although the retrieved documents can provide additional information, the model's internal knowledge is still crucial for generating accurate and relevant responses.

7 Methodology

7.1 Data Collection and Preprocessing

- **Sources:** Our primary data source consisted of scanned and digital PDF versions of **Algeria's Official Gazette**, covering the period from *2005* to **March 6, 2025**, the date on which we initiated dataset generation. In total, we collected more than *1600* PDF files. Documents from this time range varied in structure and format. While many post-*2005* PDFs included selectable embedded text, some still required OCR due to layout inconsistencies or scanned-only content.

```
1   [
2         { "year": 2005, "files": 85 },
3         { "year": 2006, "files": 85 },
4         { "year": 2007, "files": 82 },
5         { "year": 2008, "files": 82 },
6         { "year": 2009, "files": 78 },
7         { "year": 2010, "files": 80 },
8         { "year": 2011, "files": 72 },
9         { "year": 2012, "files": 72 },
10        { "year": 2013, "files": 68 },
11        { "year": 2014, "files": 78 },
12        { "year": 2015, "files": 72 },
13        { "year": 2016, "files": 78 },
14        { "year": 2017, "files": 77 },
15        { "year": 2018, "files": 79 },
16        { "year": 2019, "files": 82 },
17        { "year": 2020, "files": 83 },
18        { "year": 2021, "files": 100 },
19        { "year": 2022, "files": 90 },
20        { "year": 2023, "files": 86 },
21        { "year": 2024, "files": 88 },
22        { "year": 2025, "files": 15 }
23  ]
```
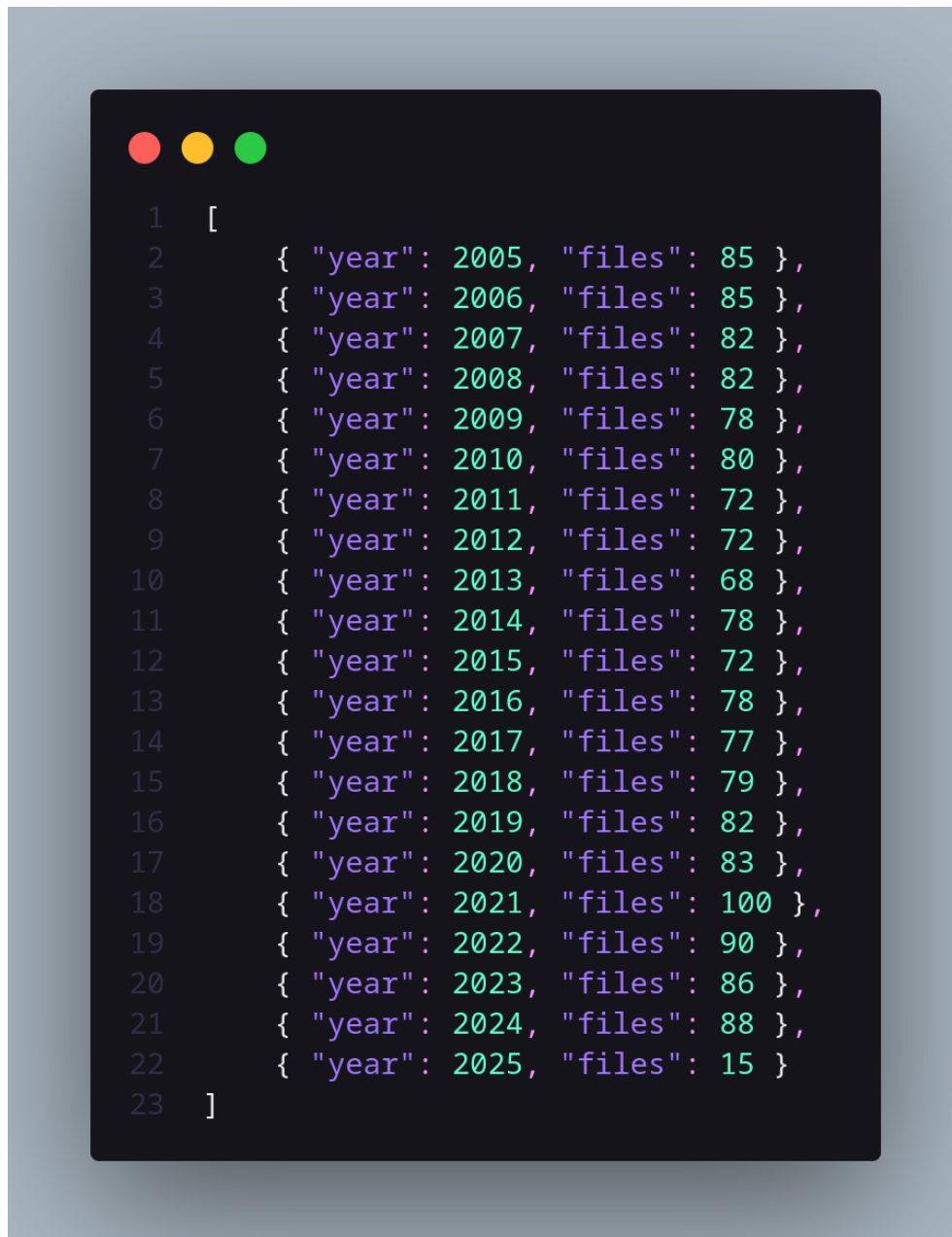
Figure 15: Pdf files for each year

- **Text Extraction Workflow:** We implemented a hybrid approach for extracting legal text from

PDFs, tailored to the challenges of **Algeria's Official Gazette** documents:

– **Initial Attempt with Python Libraries:** We first used Python libraries—primarily **PyMuPDF** (fitz)—to extract text from PDFs. This method worked well for selectable PDFs. However, several formatting issues arose, for example when a page had two columns, the extracted content grouped each paragraph as a single block, but when a page had a single column, each line was extracted as a separate block, and we struggled to reconstruct full paragraphs from those fragmented lines. So there was no reliable way to distinguish article structure, titles, or paragraph boundaries using only layout-based extraction.

.

Figure 16: Enter Caption

– **Switch to OCR (Tesseract):** To handle documents where text extraction was insufficient or incorrect, we used Tesseract OCR. While it helped extract text from non-selectable PDFs or poorly structured pages, it presented several limitations:

* **Inconsistent header extraction:** OCR often failed to detect the header content consistently across pages, sometimes capturing it only on the first page or missing it entirely. This was a major issue because the header includes crucial metadata, such as the journal number, the Gregorian and Hijri publication dates. These elements are essential for document indexing, referencing, and citation, and missing them would reduce the reliability and traceability of the extracted data.

* **Resolution dependency:** To handle non-selectable PDFs—especially those published before the early *2000s*—we applied **Optical Character Recognition (OCR)** using **Tesseract**. The code snippet below illustrates our OCR pipeline:

```python
from pdf2image import convert_from_path
import pytesseract
from PIL import Image

for pdf_file in files:
    images = convert_from_path(pdf_file, dpi=300)

    tmp_data = {
        'file_name': pdf_file.split('/')[-1].split('.')[0], # pdf_file = ./files/2006/F2006006.pdf
        'pages': '',
        'content': []
    }

    for i, img in enumerate(images):
        text = pytesseract.image_to_string(img, lang="fra")
        tmp_data['pages'] = i + 1
        tmp_data['content'].append(text)
    append_to_json_file(tmp_data)
```
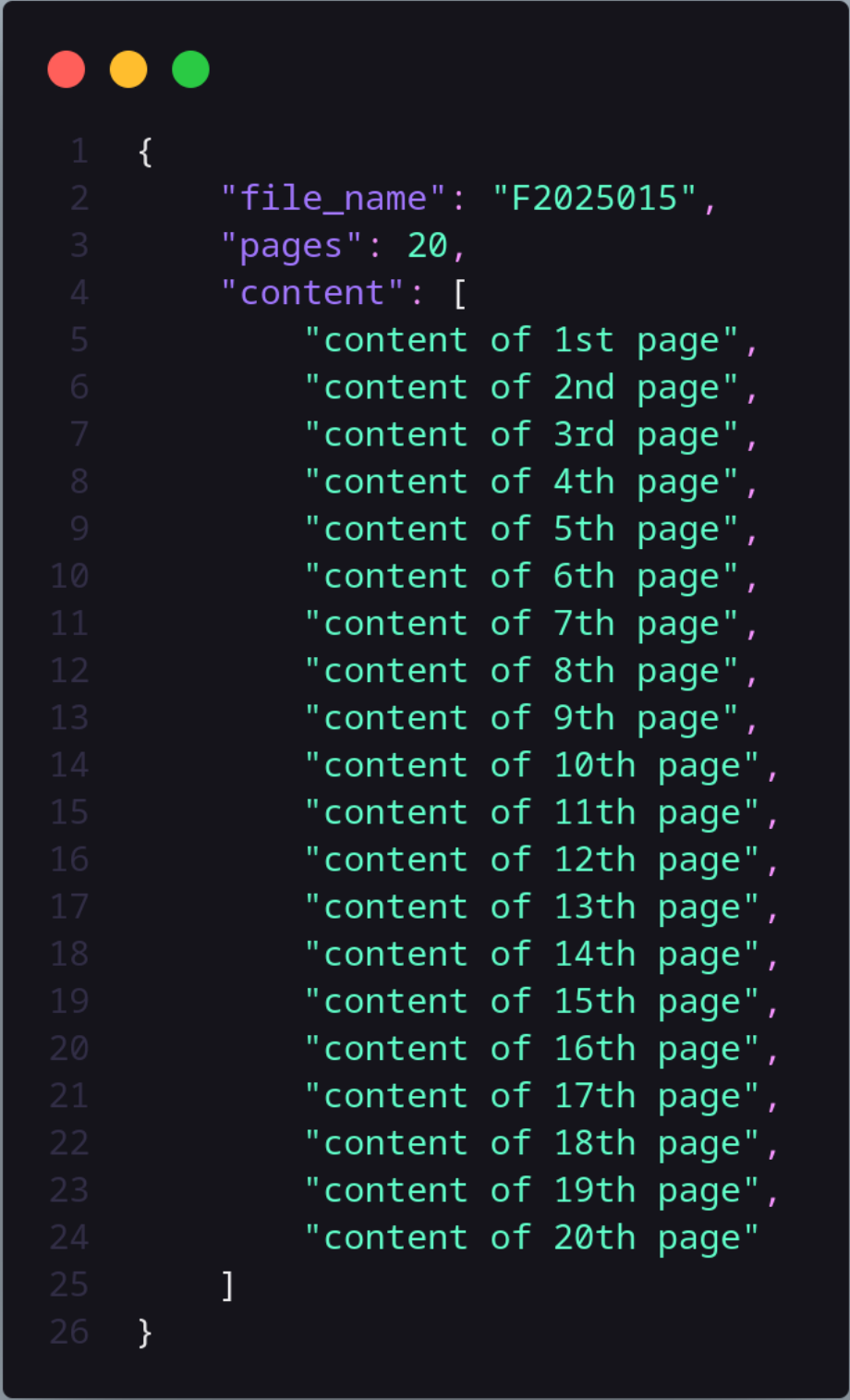
Figure 17: Apply OCR process

This approach highlights several critical issues linked to resolution and OCR performance:

· Image resolution is key to accurate OCR. We used dpi=*300* to improve recognition quality. Lower resolutions (e.g., *150* or *200* dpi) caused Tesseract to misinterpret fine legal fonts and ligatures.

· Even at *300* dpi, results were not consistently reliable, for example misread characters were common, especially in French legal terminology, for instance:

1. *'arrêté'* was sometimes extracted as *'arrete'*, *'arréte'*, or *'arreté'*

2. *'décret'* became *'decret'*, *'décref'*, or even *'dérret'*

3. *'Constitution'* appeared as *'Constiution'* or *'Constifution'*

In general, diacritics (é, è, ê, etc.) were often dropped or misread, impacting semantic meaning.

· The header of many pages which contains important metadata (like journal number and publication dates) was often skipped entirely or inconsistently recognized, leading to incomplete dataset records.

· Time complexity and limitations:

· For PDFs with more than *100* pages, the OCR process failed entirely, it either got stuck indefinitely or crashed due to memory overload, making it unusable for large documents.

· Even for medium-sized PDFs (under *100* pages), the process was significantly slow. On average, OCR took around *2–3* minutes to process *50* pages, depending on the complexity and resolution of the scanned content.

· This slow and unstable performance made it impractical to apply OCR across all documents, especially when dealing with a dataset of over *2000* PDFs ranging from *2001* to **March *6*th, *2025***.

```
 1  {
 2          "file_name": "F2025015",
 3          "pages": 20,
 4          "content": [
 5                  "content of 1st page",
 6                  "content of 2nd page",
 7                  "content of 3rd page",
 8                  "content of 4th page",
 9                  "content of 5th page",
10                  "content of 6th page",
11                  "content of 7th page",
12                  "content of 8th page",
13                  "content of 9th page",
14                  "content of 10th page",
15                  "content of 11th page",
16                  "content of 12th page",
17                  "content of 13th page",
18                  "content of 14th page",
19                  "content of 15th page",
20                  "content of 16th page",
21                  "content of 17th page",
22                  "content of 18th page",
23                  "content of 19th page",
24                  "content of 20th page"
25          ]
26  }
```

Figure 18: OCR process result

– **Final Approach: Hybrid Text Extraction**: Due to these limitations, we opted for a hybrid solution:

  * We used **fitz** python library (**PyMuPDF**) for text extraction. If the text was well-structured and accurate, we accepted it.

  * When text was garbled, incomplete, or misordered (especially in dual-column layouts), we selectively applied the OCR pipeline and reviewed the output page by page to ensure correctness. This balanced accuracy and efficiency, allowing us to retain important information while handling large volumes of legal PDFs effectively.

- **Initial Dataset Structure (Structured JSON Format):** At the beginning of the project, we designed a **structured JSON schema** to represent the extracted legal content. Each file followed this structure:

```json
1  {
2      "metadata": {
3        "journal_number": "01",
4        "hijri_date": "8 Chaoual 1421",
5        "gregorian_date": "3 janvier 2001"
6      },
7      "types": [
8        {
9          "type_name": "Décrets",
10         "ministries": [
11           {
12             "ministry_name": "Ministère des Finances",
13             "decrees": [
14               {
15                 "decret_title": "...",
16                 "articles": [
17                   {
18                     "article_title": "Article 1",
19                     "article_content": "..."
20                   },
21                   {
22                     "article_title": "Article 2",
23                     "article_content": "..."
24                   }
25                 ],
26                 "citations": ["Vu la Constitution", "Vu la loi ..."],
27                 "others": "..."
28               }
29             ]
30           }
31         ]
32       }
33     ]
34  }
```

Figure 19: JSON Structure

This initial structure aimed to achieve multiple goals aligned with the legal domain's complexity:

– **Maintain Rich Legal Context and Hierarchy:** Legal documents are inherently hierarchical and interconnected. By grouping articles under decrees, which in turn were nested under ministries and categorized by types (such as *Lois*, *Décrets exécutifs*, *Arrêtés*, etc.), we

preserved the institutional and legal flow of how laws are published and organized. This structure reflects how legal professionals actually navigate and interpret laws in practice—starting from a broad decree or law and diving into its associated articles.

– **Metadata for Filtering and Retrieval:** We included essential metadata at the top level, such as:

* Journal number

* Hijri and Gregorian publication dates

This was meant to support functionalities like:

* Chronological filtering (e.g., find laws published after a specific date)

* Precise journal lookup (e.g., retrieve all documents from Journal Officiel n°*10* of *2007*)

* Legal evolution tracking (e.g., compare changes in articles over time)

– **Enable Semantic Understanding and Traceability:** Citations such as *"Vu la Constitution"* or *"Vu la loi n°..."* are common in legal writing and indicate legal grounding or previous laws being referenced. By explicitly storing these citations and additional metadata (such as "others"), we aimed to:

* Enhance traceability of legal references.

* Allow LLM models and search algorithms to understand legal dependencies.

* Facilitate cross-referencing between decrees and referenced legislation, a crucial part of accurate legal interpretation.

- **Challenges with the Structured Format:** In the early stages of our project, we adopted a highly structured JSON schema to represent the extracted legal content from Algeria's Official Gazette. This schema was divided into two major parts:

1. **Metadata Block:** containing:


```
1  {
2      "journal_number": "01",
3      "hijri_date": "8 Chaoual 1421",
4      "gregorian_date": "3 janvier 2001"
5  }
```

Figure 20: Metadata Content

2. **Content Block:** organized by:

   – **Types:** Lois, Décrets, Arrêtés...

   – **Ministries:** each with its own set of decrees

   – Each **Décret** containing:

      ∗ A title

      ∗ A list of articles with titles and content

      ∗ Citations and additional commentary

   This structure was logically ideal from a legal perspective—it preserved the original hierarchy of the laws and made it easy to trace articles to their parent decrees, ministries, and types. However, when we attempted to integrate this format into our AI-based pipeline, we encountered multiple practical roadblocks.

- **Why It Didn't Work for AI and NLP:**

   1. **Incompatibility with LLM Inputs:** LLMs such as LLaMA operate best with plain, flat text input. Passing a deeply nested JSON structure:

      – Exceeded the token limits in many cases.

      – Required complex prompt engineering to preserve relationships between sections.

      – Made the model's output inconsistent, as it failed to interpret hierarchy as intended.

   We often had to flatten or summarize large parts of the JSON just to use it in prompts, which resulted in loss of context, especially in nuanced legal articles.

   2. **Indexing and Chunking Issues:** Vector databases like **FAISS** and **Pinecone** are optimized to store and search semantically meaningful text chunks. But with our nested JSON:

      – Some decrees had dozens of articles, while others had none.

      – Some ministries included mixed content types or redundant headers.

      – Chunking was inconsistent: we couldn't split all decrees or articles in the same way due to structure variation.

   This inconsistency affected retrieval accuracy during search, as semantically similar pieces were embedded differently depending on where and how they were located in the structure.

   3. **RAG and Query Matching Problems:** In a **RAG** (Retrieval-Augmented Generation) system:

      – **The retrieval part** depends on well-embedded, flat chunks of text.

      – **The generation part** depends on context-rich, readable inputs for the **LLM**.

   The structured JSON had to be converted to plain text for embeddings, which was error-prone. For example:

      – Some nested sections merged improperly (e.g., citations mixed with article text).

      – French ligatures, punctuation, or indentation affected chunk readability.

– The resulting embeddings lacked consistency across documents.

This led to poor matching between user queries and legal content, especially when the query referenced concepts spread across multiple nested levels (like penalties defined in one article and exemptions in another).

4. **Maintenance and Scalability Issues:**

   – As the dataset grew beyond *1600* PDFs, the overhead of parsing, validating, and maintaining nested JSON across documents became unmanageable.

   – A single format error or mismatch in nesting could break an entire pipeline step.

   – Handling documents of different structures (some with articles, some without) increased the need for exception handling and special cases.

- **Resulting Shift: From Hierarchy to Flat Text with Metadata:** After encountering the limitations of deeply structured legal **JSON** (particularly for use in **NLP** pipelines, **LLMs**, and **vector-based search systems**), we adopted a flattened and simplified document format. The goal was to strike a balance between maintaining critical legal metadata and optimizing the text structure for AI workflows.

  – **New Structure Overview:** Each document now follows this unified schema:

```json
{
    "metadata": {
      "document_id": "F2010001",
      "domain": "",
      "year": 2010,
      "journal_number": "001",
      "hijri_date": "Mercredi 20 Moharram 1431",
      "gregorian_date": "6 janvier 2010",
      "document_link": "https://www.joradp.dz/FTP/JO-FRANCAIS/2010/F2010001.pdf"
    },
    "content": "---the content of the whole pdf file---"
}
```

Figure 21: The New Structure

* **Metadata Block:** Includes essential legal attributes such as the journal number, both Hijri and Gregorian dates, year, document ID, and direct download link. These allow for traceability, filtering, and citation within legal research tools.

* **Content Block:** A single flat text field containing all the text extracted from the PDF in reading order (OCR-verified if necessary). This ensures compatibility with language models and retrieval systems.

– **Why This Structure Works Better:**

1. **Simplified Input for Language Models:** With plain, continuous legal text in the *content* field:
   * The input becomes LLM-friendly—no need to restructure prompts to fit nested JSON.
   * Prompt templates can be simpler and more reusable.
   * We maintain maximum contextual flow for long-range dependencies in legal texts (e.g., definitions referenced later in penalties).

2. **Efficient Chunking and Embedding:** The content can now be split cleanly at semantic boundaries (e.g., by paragraph, article, or section) using standard chunking tools. This supports:
   * High-quality embedding generation.
   * Smooth integration with vector databases for semantic search.
   * Reduced risk of overlapping, truncated, or context-missing chunks.

3. **Improved Search and Retrieval:** Because each flattened document is chunked uniformly:
   * Query-to-document relevance is improved in RAG pipelines.
   * Embedding-based retrieval becomes more accurate since content is no longer hidden in nested structures.
   * Search results become more explainable and traceable, thanks to preserved metadata (e.g., exact journal and date of publication).

4. **Reduced Code and Maintenance Overhead:**
   * No need for recursive parsing logic or deep structural validation.
   * No errors related to inconsistent nesting, missing fields, or unbalanced content trees.
   * Easy to debug, visualize, and validate the pipeline since the structure is uniform across all documents.

5. **Why Metadata Still Matters:** Although the main body is flattened, we retained a lightweight metadata block. This was crucial because:
   * The header information (e.g., journal number, Hijri Gregorian dates) often provides legal context for the text and is frequently cited in legal practice.
   * The *document id* and *document link* fields help users verify the source, promoting transparency and trust in the system's outputs.
   * Filtering documents by *year*, *journal number*, or *domain* becomes trivial for UI integration and large-scale analysis.

6. **Outcomes:** This change in data representation brought measurable improvements:
   * Retrieval pipelines ran faster and more accurately.

* The system became scalable to thousands of documents and millions of words.

* Prompt construction for LLMs became modular and reusable, especially in answering legal questions.

- **Post-Processing and Data Cleaning:** After generating the initial dataset using the new flat structure, we conducted a dedicated data cleaning phase to ensure consistency, readability, and relevance for both embedding generation and LLM-based reasoning:

  1. **Arabic Text Removal:** Since the French version was our primary source for dataset generation, all Arabic content extracted from bilingual PDFs (especially during OCR) was systematically removed to prevent noise and duplication.

  2. **OCR Noise Correction:**

     - OCR often repeats characters or entire words, especially for low-resolution scans.

     - In many cases, characters within a single word were split by unnecessary spaces (e.g., m i n i s t è r e instead of ministère), leading to unreadable tokens.

     - We applied regex-based normalization to remove duplicate sequences and restore word integrity.

  3. **Excessive Spacing Normalization:** Redundant whitespace—introduced either between words or within them—was cleaned to ensure natural sentence flow for example patterns like "dé cr et n° " were normalized to "décret n°".

  4. **Punctuation Cleaning:** Special characters and legal-specific separators (”—”, ”·”, ”•”, etc.) were removed or standardized to reduce token noise during embedding and querying.

  5. **Header and Footer Removal:**

     - PDF headers/footers (e.g., "Journal Officiel de la République Algérienne Démocratique et Populaire") were often repeated across pages, which negatively affected chunk uniqueness and semantic search relevance.

     - These were detected and removed using a page-level pattern detection mechanism during preprocessing.

  6. **Lowercasing:** All content was converted to lowercase, ensuring:

     - Consistent casing across the dataset.

     - Better alignment in tokenization during both embedding and LLM querying.

     - Avoidance of duplicates caused by casing differences ("Article" vs "article").

  The result was a clean, uniform, and semantically consistent dataset ready for:

     - Embedding with models like SentenceTransformers or OpenAI's text-embedding-3-small.

     - Use in retrieval-augmented generation (RAG) for legal QA.

     - Indexing into vector stores (e.g., FAISS, Qdrant, Weaviate).

     - Accurate chunking for document-level search and navigation

This post-processing step was critical to improving both retrieval accuracy and user experience, ensuring that queries returned clean, legally relevant answers with minimal noise.

7.2 System Architecture:

7.2.1 Generate Dataset:

To build the foundation of our project, we manually generated a structured dataset from scratch (as we already mentioned) where each data entry is derived from official legal *PDF* documents, it includes:

- **Metadata**: document id, both hijri and gregorian date, journal number...

- A link to the Pdf source.

- The full extracted content of the concarned legal document.

7.2.2 Text Cleaning:

After generating the initial dataset, the next crucial step was text cleaning. The raw legal texts often include various formatting inconsistencies, punctuation, and casing issues that can negatively affect downstream NLP *(Natural Language Processing)* processes.

We began by converting the entire content of each document to lowercase, this step ensures uniformity, as *NLP* models treat **Law** and **law** differently unless normalized. Next, we removed punctuation marks such as commas, periods, quotation marks, and other symbols, then we removed the arabic words from the french pdfs content, we didn't have a lot of much arabic words except some oath. These characters are not useful for semantic understanding in our use case and can add noise during embedding, we also stripped unnecessary white spaces and line breaks to produce a clean and continuous text flow.

This preprocessing step significantly improves the performance of the embedding and semantic search stages by ensuring consistent and model-friendly input. The result is a clean and normalized text representation of legal documents ready for embedding.
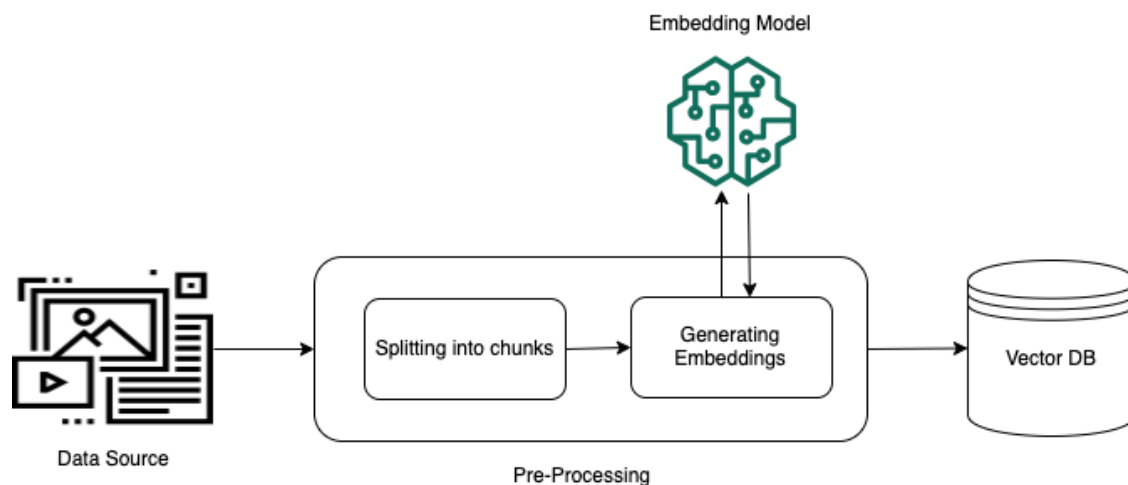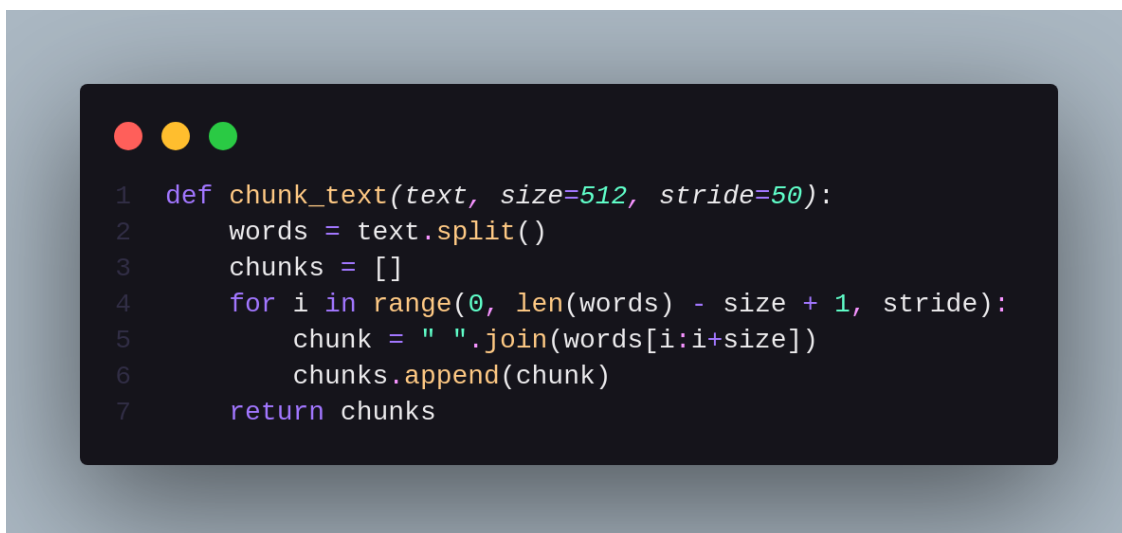
7.2.3 Embedding Step:

Figure 22: Embedding step

After successfully collect a dataset we need to embed it because as we did already mention, the machines haven't the ability to understand the human-language so we need to embed our dataset and to do so we passed by:

1. **Data chunking:** it refers to the process of breaking down the data source into smaller, more manageable chunks of data where each chunk focuses on a specific topic. When the *RAG system* retrieves information from the data source, it is more likely to directly find data relevant to the user's query, thus avoiding some irrelevant information from the entire data source. This method also improves the efficiency of the system, allowing for quick access to the most relevant information instead of processing the entire dataset. Most *transformer-based language models*, including those used in embedding generation, have input token limits (typically 512 tokens) so we did define the chunk size to 512 with 50 as *Stride* to create overlapping windows, ensuring contextual continuity between chunks and reducing information loss at chunk boundaries.

```python
def chunk_text(text, size=512, stride=50):
    words = text.split()
    chunks = []
    for i in range(0, len(words) - size + 1, stride):
        chunk = " ".join(words[i:i+size])
        chunks.append(chunk)
    return chunks
```

Figure 23: Chunking step

We get 377474 chunks for all the dataset.

2. **Generating Positive and Negative Pairs:** once the documents were chunked we aimed to generate training pairs because we needed to fine-tune the embedding model and for that, we required a way to teach the model what kind of text chunks are semantically similar (positive pairs) and speak about the same subject and what kind are semantically different (negative pairs), but to do so without labeled data, we used *KMeans clustering* on the chunk with 19236 clusters supposing that each pdf file can contain three different sections and each section has at least 4 rules.

```
1   num_clusters = len(data) * 3 * 4
2   clustering = KMeans(n_clusters=num_clusters, verbose=1, n_init="auto", random_state=42)
3   clustering.fit(chunk_embeddings)
```
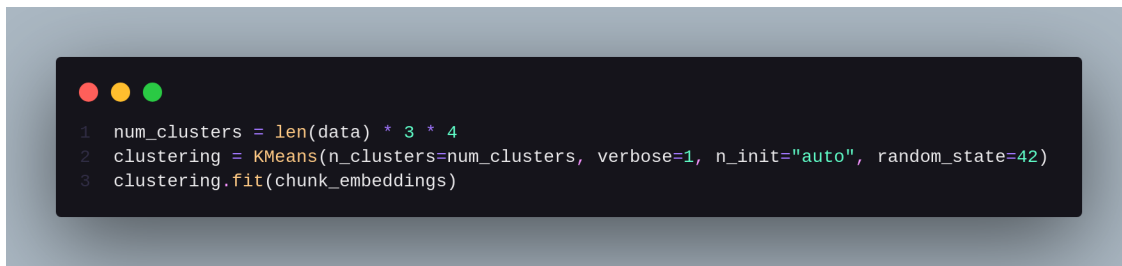
Figure 24: Clustering step

When get chunks on clusters, we need to generate:

- **positive pairs (semantically similar):** pairs of text chunks that are assumed to be about the same topic or meaning. We needed these pairs to help the model learn or recognize when two texts are conceptually close, even if they use different wording and to help the model pull similar embeddings closer together in the vector space. This makes it easier to retrieve relevant chunks later when given a user query.

- **negative pairs (semantically dissimilar):** pairs of text chunks that are assumed to be about different topics or meanings even if they are placing one after or. This will help the model push dissimilar embeddings apart, avoiding confusion between unrelated pieces of information which lead to reduce irrelevant results in retrieval tasks.

```
1    train_examples = []
2
3    # positive pairs (from the same cluster)
4    for cluster_chunks in tqdm(clustered_chunks.values(), desc="Generating positive pairs"):
5        if len(cluster_chunks) < 2:
6            continue
7        for i in range(len(cluster_chunks) - 1):
8            train_examples.append(InputExample(texts=[cluster_chunks[i], cluster_chunks[i + 1]], label=1.0))
9
10   # negative pairs (random chunks from different clusters)
11   for _ in tqdm(range(len(train_examples)), desc="Generating negative pairs"):
12       chunk1 = random.choice(all_chunks)
13       chunk2 = random.choice(all_chunks)
14       train_examples.append(InputExample(texts=[chunk1, chunk2], label=0.0))
```

Figure 25: Positive and Negative pairs

7.2.4   Fine-tuning the Embedding model:

To improve the semantic similarity performance of our system, especially in the context of retrieval-based tasks (RAG), we fine-tuned the pretrained multilingual sentence transformer:
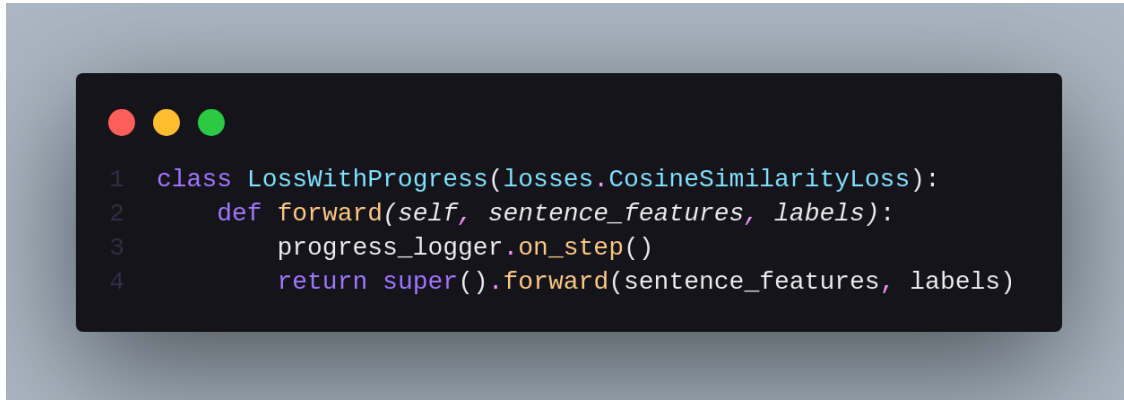
$$sentence - transformers/paraphrase - multilingual - MiniLM - L12 - v2$$

to adapt the embedding space to our custom dataset and domain-specific vocabulary, improve the closeness of positive pairs and separation of negative pairs using contrastive loss and achieve higher retrieval accuracy in the RAG pipeline by better modeling intra-domain semantic relationships.Its architecture

contains *Transformer encoder* (MiniLM) and *Pooling layer* (mean pooling over token embeddings), we used a custom loss class based on *CosineSimilarityLoss* from the

$$sentence - transformers.losses$$

module which optimizes cosine similarity between embedding vectors.

```python
class LossWithProgress(losses.CosineSimilarityLoss):
    def forward(self, sentence_features, labels):
        progress_logger.on_step()
        return super().forward(sentence_features, labels)
```

Figure 26: Loss function

We trained the model for only 3 epochs because the initial loss was already low, indicating the base model was already close to our task domain, further training risked overfitting, especially given the modest size of our dataset and limited computational resources, and because of the hardware limitation, the fine-tuning process was performed on a *Kaggle GPU*, enabling faster matrix operations and faster convergence.

```python
print("Starting training...")

model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    epochs=3,
    warmup_steps=int(0.1 * len(train_dataloader) * 3),
    show_progress_bar=True,
    output_path=None
)

print("end training...")
```

Figure 27: Fine-tuning process

The obtained embedding shape was (377474, 384).

7.2.5 Semantic Search:

Semantic search is a search technique that uses *NLP* algorithms to understand the meaning and context of words and phrases in order to provide more accurate search results. This approach is based on the idea that search engines should not just match keywords in a query, but also try to understand the intent of the user's search and the relationships between the words used. It aims to go beyond traditional keyword-based search algorithms by using techniques such as entity recognition, concept matching, and semantic analysis to identify relationships between words, phrases, and concepts. It also takes into account synonyms, related terms, and context to provide more relevant search results.

Feature	Traditional Search	Semantic Search
Keyword match	✓	✗
Understands synonyms	✗	✓
Handles paraphrases	✗	✓
Context-aware	✗	✓
Supports ranking by meaning	✗	✓

Table 3: Comparison between Traditional and Semantic Search

It's designed to provide more precise and meaningful search results that better reflect the user's intent, rather than just matching keywords which makes it particularly useful for complex queries, such as those related to scientific research, medical information, or in our case the legal documents.
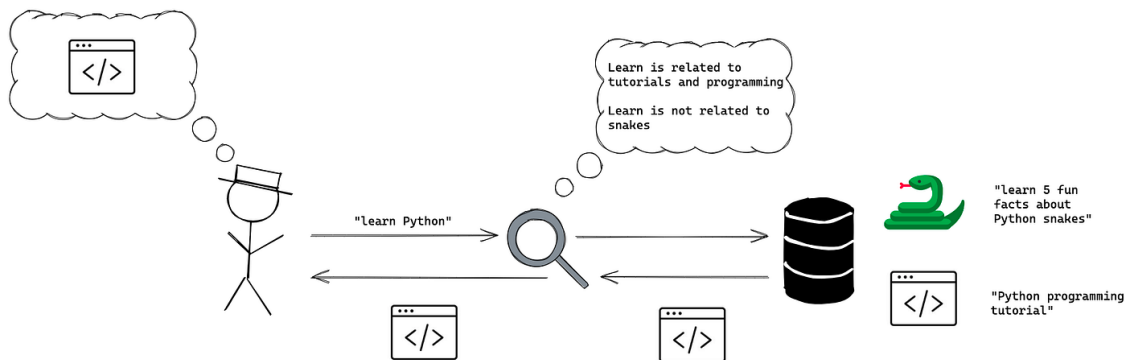

Figure 28: Semantic search example

After fine-tuning the embedding model on our dataset, we did save the obtained embeddings on *FAISS (Facebook AI Similarity Search)* which is a vector search engine that allows for efficient similarity search over these embeddings.
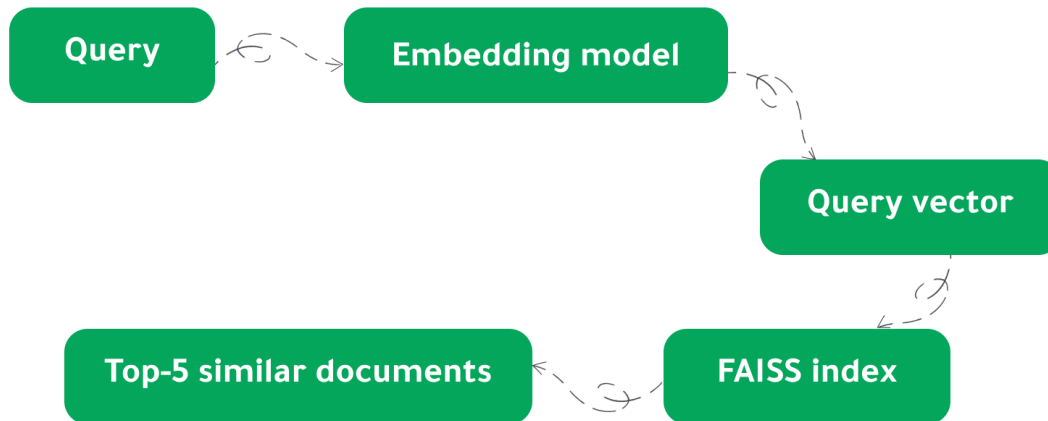
Figure 29: Semantic search step

7.2.6 Generate response:

In this step, we're asking the *LLM* to generate new response from the giving user query and the retrieved documents. When getting the user query, our system will detect the language, and wherever it was, the query will be translated to *French* for efficase retrieving step because our dataset is *French*, but the *LLM* return should be in the same language as the user query. To make sure that the *LLM* return is based only on the given information, not the model knowledge, we add constraints such as don't including previous knowledge when generating the answer to ensure that it's only reformated answer from what we already did provide to the *LLM*.
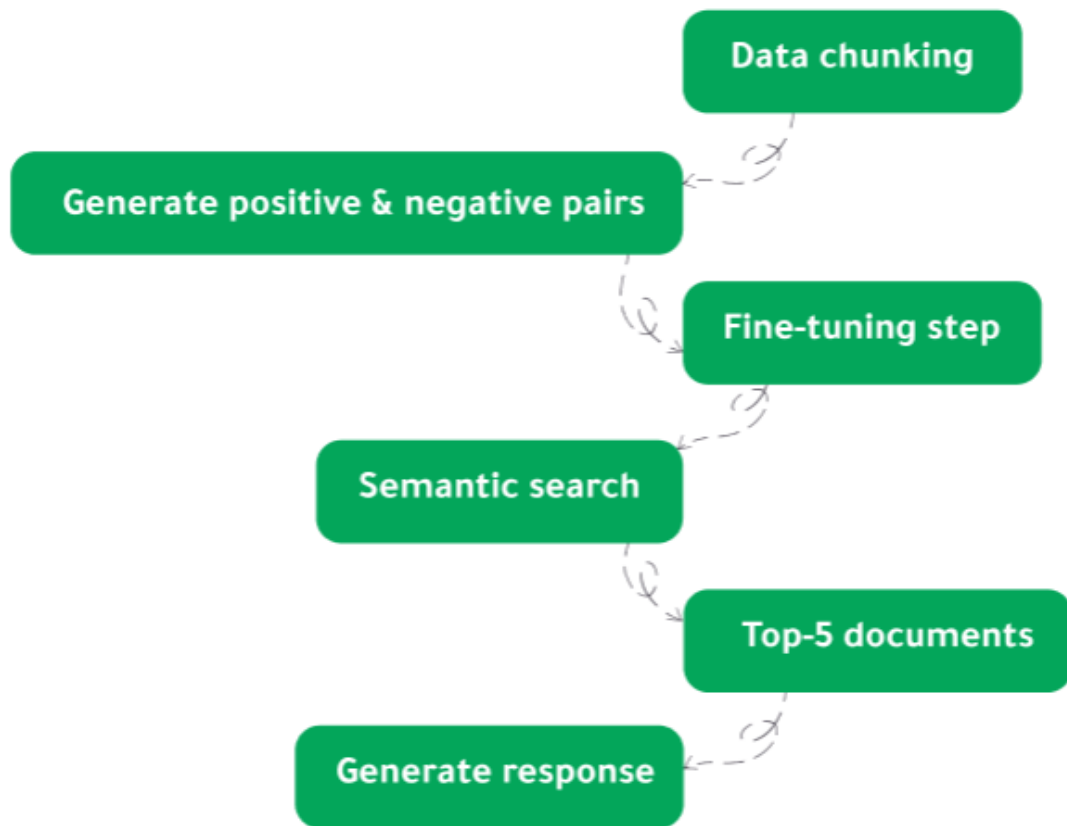

Figure 30: LLM prompt

Figure 31: System Architecture

8   Results and Discussion:

As non-expert evaluators with no specialized knowledge in the legal domain, assessing the performance of our *RAG system* posed significant challenges. Since the dataset deals with complex legal terminology and concepts, we could not rely solely on our own understanding to verify the accuracy or relevance of the generated answers. Despite these limitations, we adopted a multi-step human evaluation strategy to maintain a reasonable level of confidence in our system's outputs such as:

- **Verification Against Sources:** for every answer generated by the system, we ensured that each statement was grounded in one or more of the retrieved context documents.

- **Relevance and Semantic Matching:** where we manually compared the embedding of both retrieved documents with the question and the generated answer to verify semantic alignment. This helped us judge whether the answer truly leveraged the retrieved knowledge or was fabricated.

- **Fine-Tuning Loss Monitoring:** during the training phase of the generation model, we monitored the fine-tuning loss of the used embedding model as a quantitative indicator of how well the model was learning to generate coherent and accurate responses based on the training data where the decreasing loss indicated improved alignment between generated outputs and the ground truth.

Figure 32: Fine-tuning loss

- **Context-Answer Consistency:** even without domain expertise, we could often detect inconsistencies by identifying terms, legal articles, or citations in the generated answers and checking if they were properly referenced in the retrieved texts.

9 Deployment:

To make our *RAG system* accessible and easy to use, we developed a web based user interface using *Streamlit*, a *Python framework* well-suited for building lightweight and interactive applications. This interface enables users to interact with the system by entering legal questions and receiving answers generated by the *RAG* pipeline.

9.1 Interface Features:

- **Simple and Intuitive Design:** The interface was designed with clarity and ease of use in mind, allowing users—even those with limited technical experience—to interact with the system effectively. It uses *Arabic* language since most of people know it in our country.

- **Input Field for Legal Queries:** Users can input any legal question in any language, which is then passed through the RAG pipeline (retriever and generator) to provide a relevant answer with the same user question language.

- **Answer Output Section:** The system displays the generated answer, along with indicators of the retrieved document titles used to support the response (for transparency and traceability).

- **System Notices and Disclaimers:** A key component of the interface is a disclaimer section, which clearly informs users that:

  - The system is based on local legal documents from our country.

  - It is still an experimental prototype and may contain inaccuracies or incomplete information.

  - Users should verify the response against the official legal texts and consult a legal expert when necessary.

9.2 Ethical and Responsible Use Considerations:

Given the sensitive and high-stakes nature of legal content, we incorporated the following safeguards:

- **Warning Message:** A persistent warning reminds users that the tool doesn't replace legal advice and that its use should be informational only.

- **Error Handling:** The system gracefully handles situations where no relevant context is found by displaying a neutral message (*Insufficient information found in the legal documents to answer your query*).

- **Transparent Design:** We ensured transparency in how the answer was formed by showing the source documents used for retrieval. This builds trust and encourages users to verify information rather than relying blindly.
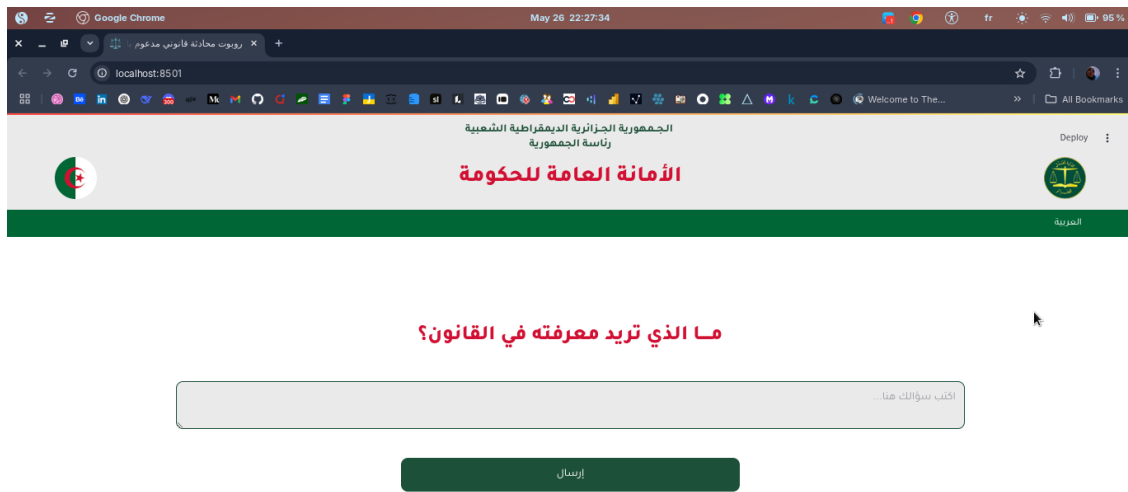
Figure 33: Our system interface

10   Future Work:

While our current system demonstrates the feasibility and potential of applying *RAG* to the legal domain, there are several important improvements and extensions we plan to pursue:

- **API Limitations and Upgrade Plans:** The current implementation uses a *free-tier LLM API*, which is limited to only 50 requests per week. This limitation has restricted our ability to conduct extensive testing or open access to external users. We used this model primarily during the development and testing phases. For production deployment, we plan to subscribe to a paid tier of the *API* to ensure higher availability, faster response times and scalability for real users.

- **User Authentication and Personalization:** In future iterations, we aim to:
  - Implement a user authentication system to provide a secure and personalized experience.
  - Allow logged-in users to store and view their previous queries and answers (chat history), which can be especially helpful in long-term legal research or consultations.
  - Potentially include role-based access where users have access to specific features based on their profiles (e.g., student, lawyer, public user).

- **Human-Lawyer Integration:** To enhance the accuracy and reliability of legal advice we're considering a collaboration with professional lawyers who can validate and annotate AI-generated responses and help fine-tune future models with real legal practices. We may introduce a "Consult a Lawyer" feature, enabling users to book or request clarification from a human legal expert when the AI-generated information is not enough.

These additions would not only improve user trust and system reliability but also address the legal and ethical responsibility of working in such a sensitive domain.

References

[1] Medium articles `https://medium.com/`

[2] Dataloop `https://dataloop.ai/?utm_medium=library_nav`