UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SEMANTIC SEARCH IN LEGAL DATA

Diploma Thesis

VASILEIOS PISCHOS

SOFOKLIS PATAKIOUTIS

Supervisor: George Thanos

June 2025

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SEMANTIC SEARCH IN LEGAL DATA

Diploma Thesis

VASILEIOS PISCHOS

SOFOKLIS PATAKIOUTIS

Supervisor: George Thanos

June 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΝΟΗΜΑΤΙΚΗ ΑΝΑΖΗΤΗΣΗ ΣΕ ΝΟΜΙΚΑ ΔΕΔΟΜΕΝΑ

Διπλωματική Εργασία

ΒΑΣΙΛΕΙΟΣ ΠΙΣΧΟΣ

ΣΟΦΟΚΛΗΣ ΠΑΤΑΚΙΟΥΤΗΣ

Επιβλέπων/πουσα: Γεώργιος Θάνος

June 2025

Approved by the Examination Committee:

Supervisor **George Thanos**
Laboratory Teaching Staff, Department of Electrical and Computer Engineering, University of Thessaly

Member **Emmanouil Vavalis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Elias Houstis**
Emeritus Professor, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarants

VASILEIOS PISCHOS and SOFOKLIS PATAKIOUTIS

Diploma Thesis

SEMANTIC SEARCH IN LEGAL DATA

VASILEIOS PISCHOS

SOFOKLIS PATAKIOUTIS

Abstract

The vast and ever expanding corpus of Greek legal documents presents significant challenges for information retrieval and semantic search applications due to its seer size, complexity, special language and continuous evolution through modifications and revisions over the years. Traditional keyword based search applications are still widely used from practitioners and researchers of the field often failing to capture the semantic richness and contextual meaning of the texts causing confusion and limiting their efficiency. In this thesis, we develop an end-to-end, effective and sufficiently accurate semantic search service for Greek legal data. Through our research and development project, we also highlight many limitations of the Greek Natural Language Processing and propose original solutions in many steps of the process.

Keywords:

Semantic Search, Natural Language Processing, Greek Legal Data, Embedding Models

Διπλωματική Εργασία

ΝΟΗΜΑΤΙΚΗ ΑΝΑΖΗΤΗΣΗ ΣΕ ΝΟΜΙΚΑ ΔΕΔΟΜΕΝΑ

ΒΑΣΙΛΕΙΟΣ ΠΙΣΧΟΣ

ΣΟΦΟΚΛΗΣ ΠΑΤΑΚΙΟΥΤΗΣ

Περίληψη

Ο ευρύς και αυξανόμενος αριθμός των Ελληνικών νομικών εγγράφων επιφέρει σημαντικές δυσκολίες σε εφαρμογές ανάκλησης πληροφοριών και νοηματικής αναζήτησης, λόγω του μεγάλου όγκου, της πολυπλοκότητας, της ιδιόμορφης γλώσσας που χρησιμοποιείται και της συνεχούς εξέλιξης μέσω μεταρρυθμίσεων και αλλαγών στο πέρασμα των χρόνων. Οι παραδοσιακές εφαρμογές που βασίζονται σε αναζήτηση με λέξεις κλειδιά χρησιμοποιούνται ευρέως μέχρι και σήμερα από ειδικούς και ερευνητές του κλάδου και συχνά αποτυγχάνουν να εντοπίσουν το νοηματικό περιεχόμενο και να αντλήσουν σημασιολογικά στοιχεία από τα συμφραζόμενα των κειμένων, προκαλώντας σύγχυση και μειώνοντας την αποτελεσματικότητα της δουλειάς τους. Σε αυτή τη διπλωματική, αναπτύσσουμε από την αρχή μέχρι το τέλος μία αποδοτική και επαρκώς ακριβή υπηρεσία νοηματικής αναζήτησης σε ελληνικά νομικά δεδομένα. Επίσης, κατα την ανάπτυξη της εργασίας μας αναδεικνύουμε πολλούς περιορισμούς της Φυσικής Επεξεργασίας της Ελληνικής γλώσσας και προτείνουμε πρωτότυπες λύσεις σε πολλά στάδια της διαδικασίας.

Λέξεις-κλειδιά:

Νοηματική Αναζήτηση, Φυσική Επξεργασία Γλώσσας, Ελληνικά Νομικά Δεδομένα, Embedding μοντέλα

Table of contents

List of figures

List of tables

Abbreviations

JSON	JavaScript Object Notation
DB	Database
API	Application Programming Interface
REST API	Representational State Transfer API
SOAP API	Single Object Access Protocol API
GraphQL	Graph Query Language
URL	Uniform Resource Locator
CRUD	Create, Read, Update, Delete
JWT	JSON Web Token
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
UI	User Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
AI	Artificial Intelligence
NLP	Natural Language Processing
DAPT	Domain Adaptive Pretraining
MLM	Masked Language Modeling
LIFO	Last In First Out
CRF	Conditional Random Fields
VSM	Vector Space Model
TF-IDF	Term Frequency-Inverse Document Frequency
LSI	Latent Semantic Indexing
IR	Information Retrieval

FEK	Φύλλο Εφημερίδας Κυβερνήσεως
ISBN	International Standard Book Number
BERT	Bidirectional Encoder Representations from Transformers
VDBMS	Vector Database Management System
XML	Extensible Markup Language

Chapter 1

Introduction

The Greek legal information field has three key characteristics: large volume, high complexity, and constant evolution. The language used in Greek legal documents is full of cross references, legal terminology, and technical expressions, which makes it highly specified and difficult to understand and process, especially for non-experts on the field. Traditional keyword-based information retrieval techniques which are widely used in the field, struggle to handle this complexity. They fail to capture the legal nuances of the text and lack the ability to recognize synonyms, context and the hierarchical structure of Greek law. Moreover, modifications and revisions to existing documents create a dynamic evolving environment in which the same law exists in multiple versions depending on the validity period. These constant changes reduce the effectiveness of simple keyword searches even more overtime. This vast amount of challenges and limitations create an urgent need for the introduction of a modern, innovative information retrieval system powered by state-of-the-art machine learning models.

The idea of processing the Greek language with AI models is not completely new ([1], [2]), but the field of Greek law specifically, lacks research related to semantic search services. The efficient handling of multiple, complex, large-sized documents along with the need to search and locate precise provisions by professionals such as lawyers, judges and public officials are requirements that these older techniques simply cannot meet. The absence of related work, combined with the aforementioned challenges of the field and the availability of data we had (**43.550 Greek legal documents related to labor law**) gave us the motivation to form the subject of this thesis.

1.1 Subject of the thesis

The thesis develops a semantic search application tailored to Greek legal documents aimed at resolving these problems and revolutionizing the field. As we already mentioned, the primary issues experts of the field face are the difficulty of locating precise provisions within the legal document collection, the complexity introduced by modifications and interconnections between legal sources, and mainly the limitations of keyword-based search and older inefficient information retrieval methods that were used in the legal domain in Greece until now. An in-depth analysis of these older methods is carried out in the **Chapter 8**. These issues often lead to the confusion of researchers and practitioners alike because they require precise, relevant, and contextually accurate results. Our thesis focuses on addressing and resolving these key issues in the retrieval of Greek legal documents using a large dataset (43550 texts) specified on labor law. The shape, length and structure of the dataset will be analyzed in more detail in **Chapter 3**.

1.2 Contribution

We believe that our thesis will contribute to the Greek legal domain by demonstrating how state-of-the-art Natural Language Processing (NLP) methods can be adapted to the unique morphology of Greek legal texts. The system we developed is specifically designed to handle the complexity, archaic language and contextual dependencies of our legal documents. That way our thesis contributes to the modernization of legal information systems in Greece, hopefully making them more efficient, transparent and easy to use even by non field experts. Specifically, it introduces the following:

- **Custom Sentence Splitting**: We developed an extensive abbreviation list to help us with sentence spitting and came up with an innovative rule-based sentence splitting technique.
- **Custom Chunking Method**: We came up with a way to break the documents into smaller parts that have enough contextual meaning to produce embeddings.
- **Semantic search using contextual embeddings**: We used those embeddings to provide a complete semantic search application with precise results.

- **Custom Re-ranking Method**: Because it was impossible to train a reranker without annotated feedback, we came up with a solution to rank the results with an unsupervised custom method.
- **An interactive and easy-to-use website**: We developed a user-friendly website so that field experts can use it to give us feedback on our results.

1.3 Thesis structure

In this section we will briefly summarize the chapters of the thesis and talk about what each chapter analyzes.

1. Chapter 1: This is the chapter we are currently on and it is an introduction on our thesis that talks about the struggles of the legal domain and how we managed to overcome them.
2. Chapter 2: The second chapter is an overview of any state-of-the art studies that are related to our topic.
3. Chapter 3: This chapter talks about the quality of our data and the analysis and cleaning we did so that we can use them on our Natural Processing Tasks.
4. Chapter 4: The fourth chapter talks about the sentence splitting algorithm we developed and how we used it to chunk our documents efficiently. It also provides the reader with an overview of all the available chunking methods.
5. Chapter 5: The fifth chapter analyzes the embedding model selection phase of our thesis where we show all the research we did to decide on a specific model.
6. Chapter 6: The sixth chapter talks about the importance of choosing the optimal vector database to store our embeddings as well as our choice and how we set it up.
7. Chapter 7: In this chapter we analyze the embedding pipeline we developed from start to finish.
8. Chapter 8: This chapter talks about the information retrieval procedure as well as the reranking algorithm we used in our document retrieval service.

9. Chapter 9: The ninth chapter is an overview of our full stack application used to test our results by experts on the field.

10. Chapter 10: The last chapter of our thesis is a conclusion paragraph that also highlights the future work that can be done on the subject.

Chapter 2

Related Work

This chapter is an overview of any state-of-the-art studies related to our thesis topic. We focus on relevant methodologies, applied ideas and concepts and suggested machine learning models already tested during previous research.

The design and testing of a semantic search service for Greek legal data is the primary goal of this thesis. To the best of our knowledge, there has been no similar research in this subject. While there have been studies on the natural language processing of Greek texts, the embedding of Greek words in general and some research on classification of such documents, there is no study specifically focusing on Greek legal text processing, embeddings or semantic search. Nevertheless, the research projects on these similar fields were worth studying since they use related technologies to our project. Unfortunately, most of the models presented in the research studies below are not embedding models, so we were unable to experiment with them.

Greek Language Processing

The authors of [1] present a thorough and effective method for natural language processing, applicable to languages other than English and conduct their analysis using Greek as an example. Through their research, they create a detailed map of the current state of Greek NLP, available resources and unresolved issues. Their project proves especially helpful in pointing the limited resources and research work in this field, while assisting in finding original ways to perform language processing tasks.

Greek Text Embeddings

The research of [2] focuses directly on training and testing embedding models for Greek texts, addressing many related issues. They experimented with various models and created a solid test dataset by modifying and enhancing existing resources. Overall, the models produced meaningful representations. Also, their experiments highlighted the effect (positive or negative) that the morphological complexity of the language has on model performance.

Greek Text Classification

There are a number of researches dedicated to Greek text classification, with some also specializing in legal data. Authors of [3] design and implement an ensemble method (aggregate predictions of different simpler models) for categorizing Greek texts. Their experiment proposes that the combination of many models greatly enhances the ability to deal with the complexity of the Greek language. Study [4] is among the first open research attempts in classifying Greek legal data with machine learning models. They provide a solid labeled dataset and test a variety of models, including transformer architectures, which achieved the highest performance. According to the paper, monolingual models trained in the domain are the optimal choice, followed by multilingual models. The thesis [5] conducts a comparative experiment to determine the best BERT model for the classification of Greek legal data. This study also observed a higher accuracy in the fine tuned Greek-specific models over the multilingual ones. It also suggests domain adaptive pretraining as an improvement for future researches. Lastly, the authors of [6] present a state-of-the-art LLM, the GreekLegalRoBERTa, which achieves a higher performance than all previous Greek legal transformer models (GreekLegalBERT, etc.). Their work greatly contributes to domain specific NLP tasks in Greek, which is a language with very limited resources on the matter.

Chapter 3

Data Exploration - Analysis - Cleaning

3.1 Introduction

The foundation of every natural language processing (NLP) system is heavily dependent on the quality and handling of its data. In text retrieval applications, this principle becomes even more critical. Legal documents are often lengthy, highly structured, and linguistically complex because meaning is conveyed not only through words, but also through formatting, references, and specialized terminology. So, if our data are not properly explored, analyzed, and cleaned, downstream models risk producing results that are inaccurate, incomplete, or even misleading. Small inconsistencies, such as unrecognized abbreviations, irregular sentence boundaries, or formatting artifacts, can accumulate into significant retrieval errors, affecting the reliability of the system.

In our case, the source material was provided in a JSON format enriched with metadata. Although this structure offered flexibility and valuable contextual information, it also required careful processing to make the data usable for NLP tasks. Extracting relevant fields, standardizing representations, and resolving inconsistencies between metadata and document content were essential steps to ensure that the information could be leveraged effectively in the retrieval pipeline. To get these data in the right form, it was crucial to take a very careful approach when preparing and transforming it. In this way, finer points of language could be preserved and a reliable search tool could be built. This chapter focuses on the steps of data exploration, analysis, and cleaning, focusing on their role in turning raw structured data into a solid foundation for all of the processing stages that will follow.

3.2 Structured Metadata Handling and Text Recovery

The dataset that was provided to us included **43,550** legal documents related to **Labor Law** that have a mean word count of **39955.25** words. They are split into six different categories: **Legislation** (νομοθεσία), **Legal Articles** (αρθρογραφία), **Books** (βιβλιογραφία), **Administrative Documents** (διοικητικά έγγραφα), **Case Law** (νομολογία) and **Legal Examples** (υποδείγματα) as shown in the diagram below.

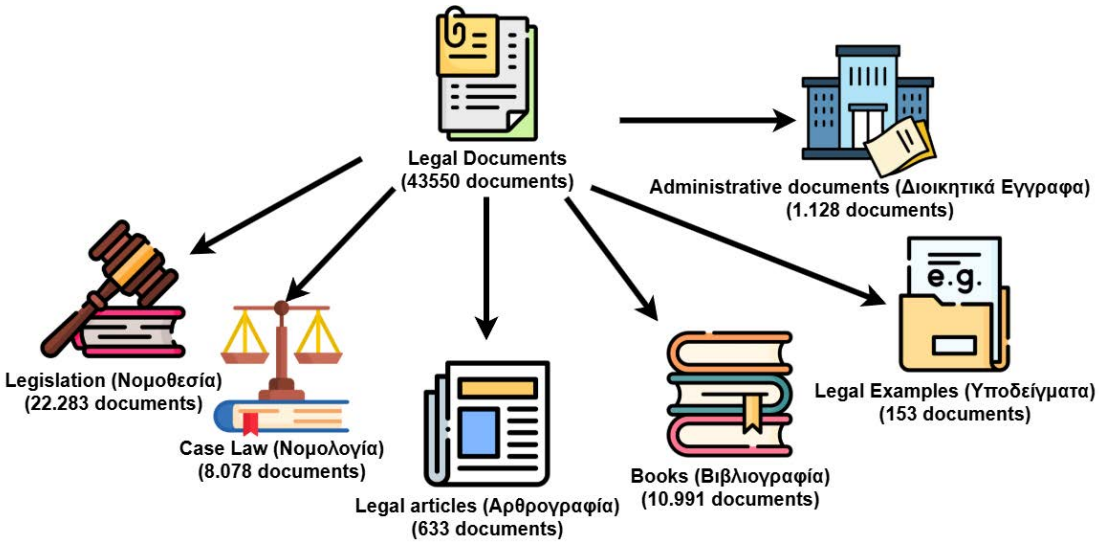

Figure 3.1: Legal Documents Structure

Each one of these six categories included some unique JSON keys that we cannot discuss in detail due to confidentiality agreements but in general they stored valuable information such as the court and the court department where a trial took place, the author, date and publisher of an article, the page and word counter of some documents as well as FEK numbers (the official identification number for an issue of the Government Gazette (Official Journal of the Hellenic Republic)) and other important information. In consultation with our supervisors, we decided to keep all the unique keys of each category. Now all the documents had 64 shared JSON keys that we also had to filter through and decide which ones to keep so they can help us in the retrieval process. We decided to keep the year in which a document was written, its summary, it's different ID's, the title of the document and the subcategory of law in which the document was appointed to, the ISBN (International Standard Book Number), the word count, the reviewers of the documents and some specialized tags that were appointed by the people that shared the dataset to us. These were the metadata fields that we labeled as relevant for our retrieval application and extracted all the remaining information. Through

this elimination process, we created a new JSON schema that contains only the essential information that a user could need when he is filtering through the results of our application.In addition to other metadata fields that we talked about, the summary field was an extremely useful field for us to use in our application. We decided to combine the summary's text with the text of our documents so we can also use it to extract embeddings. The embeddings that were extracted from the summaries of the texts played an important role in the information retrieval procedure as they offered a more compact and straightforward meaning, especially on large texts where the meaning was spread out between paragraphs.

The actual text resided under a nested JSON path, while the surrounding data contained metadata. We continued transforming the data by processing the nested JSON path that contained the JSON schema of the actual text and its metadata. From all the metadata that this included we preserved only the metadata that was referring to special paragraph markers and we combined these markers with the sentence-level metadata in order to reconstruct the original text. Additionally, because the legal texts included anonymized names, places and other anonymized information represented by multiple consecutive periods ".....", we replaced them with the phrase [ΠΡΟΣΩΠΙΚΟ ΔΕΔΟΜΕΝΟ]. This substitution ensured that the embeddings could capture the presence of personal data while preventing the excessive periods from interfering with our sentence-splitting procedure, which we analyze in detail in **Chapter 4**.

Chapter 4

Sentence Splitting and custom Chunking

4.1 Introduction

Chunking in the context of *Natural Language Processing (NLP)* and *Semantic Search* refers to the process of dividing large bodies of texts into smaller units that preserve both meaning and context. This preprocessing step is crucial for tasks such as document retrieval, especially when we are working with documents that exceed the input limits of modern language models. Since our documents are domain-specific (Greek legal texts), they have specific linguistic and typographic features unique to the Greek legal ”writing style”, such as nested article references, formal language, plethora of legal abbreviations etc, that make the chunking step a challenging problem that is crucial to the search results of the semantic retrieval.

4.2 Chunking Methods

Based on the articles and blogs we have studied [7, 8] these are the chunking methods available right now:

4.2.1 Simple Text Splitting (Character or Token based)

With this method, we divide our text into uniform chunks based on character or token limits. That means that, for example, we could split a text into 500 token chunks or 500 character chunks. The difference between the two is that a token does not always correspond to a whole word. It is a unit of text that is used by language models to process and understand

input. Most language models do not operate on words, but on tokens generated by a *tokenizer*. They can be individual words, subwords, or even punctuation points, especially in a rich language like Greek. This is a pretty straightforward approach and easy to implement, but it often splits sentences or paragraphs mid-thought, resulting in fragmented semantics.

4.2.2 Sentence Based Chunking

This chunking method groups complete sentences into chunks up to a certain token limit (e.g., 1024 tokens). It maintains grammatical integrity and when it is implemented with an *Overlapping Window* [9] approach it preserves context across chunk boundaries. This approach introduces another challenge, which is the correct splitting of sentences in legal documents, which is a demanding task because of the morphology of legal writing and the many abbreviations that are used in those documents.

4.2.3 Structure aware chunking

This method leverages the formatting of legal texts such as sections, numbered paragraphs, and headings to define logical chunk boundaries. This method is particularly effective for documents with well-defined legal hierarchies but does not produce adequate results when the documents are not structured correctly. An implementation of this method is the Recursive splitter that is studied on the technical report: [8]

4.2.4 Semantic Chunking

This method uses embeddings or language models to detect topic shifts or conceptual boundaries within the text. Although it is very powerful, in principle, its effectiveness is heavily dependent on the availability of domain-specific models.

4.3 Our choice

We experimented with all the techniques mentioned above and came to the conclusion that the best method for our documents would be to apply a custom-made sentence based chunking method that we designed in order to keep the structural and semantic meaning of each chunk intact. We could not use structure aware chunking because in our documents

there are only paragraph markers which do not always represent logical boundaries. Between two paragraph markers, sometimes there are more than 2.000 words, especially in really big documents, and sometimes we have observed that there are paragraph change markers even between two words. In conclusion, it is impossible to group our documents in these paragraph markers because there is no logical pattern on how they are used. We also cannot use Semantic Chunking because there are no domain-specific language models trained on Greek legal datasets that we can use to chunk our own. Also, with this method, there is no way to ensure that a chunk will not exceed the maximum tokens that our model can convert into embeddings.

4.4 Sentence Splitting

4.4.1 Introduction

Sentence splitting is a foundational step in achieving accuracy in our chosen chunking method, as we have already discussed. Although this task is generally well studied for English and other languages of high resources and covers both unsupervised [10, 11] and rule-based techniques [12, 13, 14] it presents significant challenges when applied to legal texts written in Greek language. The formal nature of legal language, combined with the unique syntax and punctuation conventions of Greek and the plethora of abbreviations, dates and other speech patterns complicate the task substantially.

4.4.2 Our approach

We based our approach on the article [15] that explores sentence splitting in English legal texts. It explores the performance of three approaches: a Punkt model with Custom Abbreviations, a *conditional Random Field* [16] and a *Deep Learning Neural Network* [17] approach. We could not use the last two approaches because CRF models need a sizable manually annotated corpus of Greek legal sentences with correct boundary labels to train the feature weights and typically require tens of thousands of labeled examples to generalize well and such resources simply do not exist for Greek law. Greek legal texts are especially formulaic: nested clauses, enumerations (“(α) … (β) …”), cross-references (“σύμφωνα με το άρθρο 5 παρ. 2”), and archaic phrasing. CRF or Deep Learning models would struggle

with these patterns, whereas rule-based methods (e.g punkt, spacy) can be tuned via regex for known markers. So we decided to use a rule-based approach and create a custom list of all the abbreviations used in our documents so that we can set them as rules. These abbreviation lists can be used in the future for training machine learning models to recognize them in Greek legal texts.

4.4.3 Constructing an Abbreviation List for Preserving Structure and Meaning in Sentence Formation

Abbreviations pose a critical challenge for Greek legal text sentence splitting due to the ambiguity between period markers for abbreviations and sentence boundaries. To address this, we used carefully designed regex patterns to extract candidate abbreviations and then manually curated the results to filter false positives and build a clean abbreviation list. Regular expressions proved insufficient on their own due to Greek's complex punctuation and morphological patterns, so this iterative manual validation step was essential to ensure the reliability of our sentence-splitting model. We ended up with 8.000 unique abbreviations and also counted how many times each one of them appeared on our documents. That list can be used in the future to possibly train other models in finding new abbreviations so that no manual labor is needed.

Furthermore, by leveraging this list, we ensured that the segmented sentences remained well-structured and semantically intact. Each sentence was split according to its natural boundaries, resulting in full-length units that accurately reflected the meaning of the source text. The use of the curated abbreviation list proved especially important, as it prevented misinterpretations of periods within abbreviations as sentence breaks. In this way, we achieved reliable sentence segmentation that preserved both the structure and the nuanced flow of the Greek legal texts.

4.4.4 Dealing with "Outlier Sentences"

When examining the sentences produced after segmentation, we noticed that most of them fell into a normal range, but there were also two groups of outliers that required special handling. In total, we had 1,267,959 sentences of average length 277.3 tokens, which formed the bulk of the dataset. Alongside these, there were 76,109 very short sentences with an

average of only 4.6 tokens, and 556 unusually large sentences averaging 8,016.5 tokens.

Sentence Type	Count	Mean Tokens
Short Sentences	76,109	4.6
Normal Sentences	1,267,959	277.3
Large Sentences	556	8,016.5

Table 4.1: Distribution of sentence lengths in the dataset.

To address these extremes, we applied two simple adjustments. We combined the unusually small sentences together so that they can form an average sized sentence that holds the importance and meaning of all the small ones adequately. On the other hand, the very large sentences were cut into pieces of approximately 512 tokens (slightly above the mean length), always at logical points such as commas, paragraph markers, or other punctuation that indicated a shift in meaning so that no single sentence could dominate a chunk.

This balancing step was important for two reasons: first, it prevented over sized sentences from skewing the structure of the chunks, and second, it avoided creating tiny fragments that would weaken semantic coherence. The result was a more stable set of chunks, each of which preserved the flow of the original legal text while remaining computationally manageable.

4.5 Chunking Method

4.5.1 Introduction

For downstream processing, we adopted a chunking strategy designed to preserve sentence integrity while respecting model input constraints. The main idea was simple: A chunk should never break in the middle of a sentence, since that would risk losing meaning. As we already mentioned,, we limited individual sentences to 1024 tokens and then aggregated them into chunks suitable for transformer based models. In the following sections we will describe the rationale behind our chosen chunk size, why we introduced overlap between the chunks and how the algorithm actually carries out this process in practice.

4.5.2 Chunk Size Choice

Identifying the best chunk size is as much intuition as it is empirical evidence. We choose our chunk size based on existing research [18], [19] and doing our own experiments. So in our initial experiments, we tested various chunk sizes: 512, 1024, 2048, 4096, and 8012 tokens to determine the optimal size for our transformer-based models. In general, ideal training chunk size is a trade-off: enough to capture important long dependencies, but not so long that the semantic quality of the embeddings gets diluted. Through all of our experimentation, we concluded that the 2048 token sizes strike a balance, encapsulating meaningful context while maintaining embedding quality in our dataset. Based on the statistics that we presented in the previous paragraph, every chunk includes approximately 10 sentences.

4.5.3 Overlap Choice

To enhance the continuity and coherence between the chunks, we introduce overlap between them. Research [20], [21] indicates that incorporating overlap can improve the continuity and coherence of embeddings, as it ensures that semantic information is preserved across chunk boundaries. In addition, overlap helps maintain context, which is crucial for tasks that require a deep understanding of the text. This approach mitigates the risk of losing critical information at the shard boundaries, ensuring that each shard retains sufficient context for accurate embedding. We set the overlap size to 4 sentences.

4.5.4 The Algorithm

We implemented a LIFO-style (Last In, First Out) chunking algorithm to create coherent text segments with respect to the 2048 token limit. The process works as follows:

1. Initialize an empty chunk space with a maximum capacity of 2048 tokens.

2. Pre-fill the chunk with an overlap of 4 sentences from the previous chunk to preserve context.

3. Iteratively add sentences to the chunk until adding the next sentence would exceed the 2048-token limit.

4. If the next sentence exceeds the limit, it is removed and the chunk is finalized.

5. The finalized chunk is then passed to downstream processing and the algorithm continues with the next set of sentences.

The figure below visualizes the process we described above. S1 and S2 are the sentences 1 and 2 in this example, the figure also shows the overlap and the token size of each chunk.

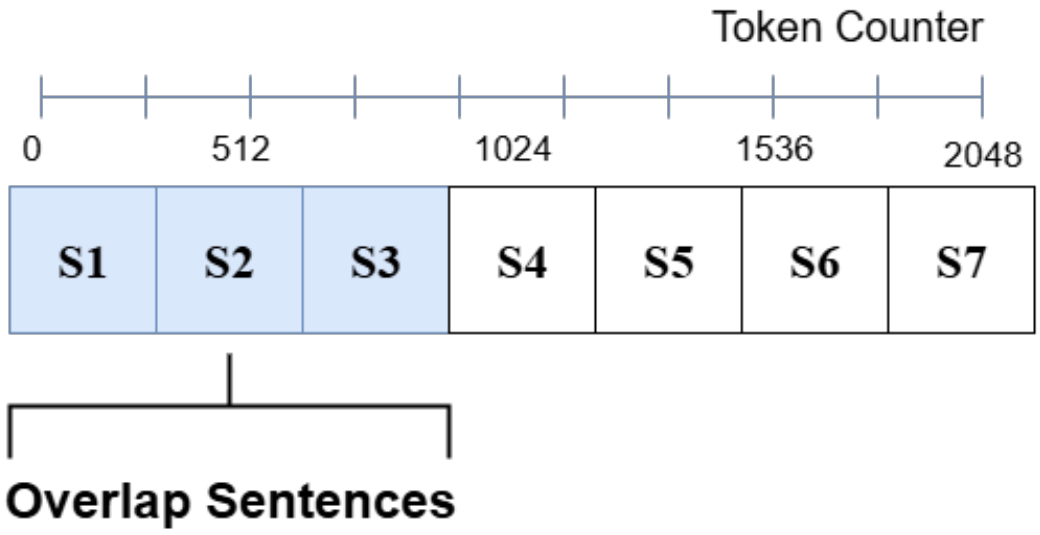

Figure 4.1: Chunking Algorithm

This approach preserves the integrity of the sentence, maintains semantic coherence, and avoids splitting sentences into sections.

Chapter 5

Embedding Model Selection

5.1 Introduction

The technology of text embeddings (raw text represented as numerical vectors) and embedding models is a state-of-the-art Natural Language Processing technique to manipulate large amounts of text in a more efficient way, focusing on the context and semantics instead of rules and traditional algorithms. Semantic search is a service that requires this technology to function, since the core idea is to represent text as vectors and compare their positions on the geometric space.

There are a vast number of embedding models available online (both free and under subscription) but the fact that our texts were Greek legal documents made the process of choosing an appropriate model a very challenging task. As analyzed in **Chapter 4**, on top of using Greek characters (already limited support by most models), legal language contains a lot of abbreviations and special text patterns that make it difficult for both the tokenizer and the encoder to create solid vector representations. As a result, it was necessary to experiment with different approaches and compare the performance of each method, before deciding and sticking with a particular model choice.

This chapter dives into the foundations of text embeddings and embedding models, and then presents the analytical approach we followed in order to select the optimal model for our project development.

5.2 Background information

5.2.1 Text data represented as vectors

In the fields of Machine Learning and Natural Language Processing, words and sentences are manipulated in various ways in order to perform a number of downstream tasks like classification, information retrieval and more recently generation of new text based on the inputs given. The way that modern models consume and process text data is that words or text segments are encoded into high-dimensional vectors (also known as embeddings), in order to be processed by the computer.

The key concept behind this transformation is that the resulting vectors preserve semantic information about their related word. Words with similar meaning or closely related semantics tend to exist close to each other in the vector space, while semantically irrelevant words are represented by vectors in completely different directions. The example of **Figure 5.1** illustrates a simplified yet accurate example of these relations. The words **'basketball'** and **'football'** are close to each other, having a small angle between them, while the random word **'table'** is in a completely different direction. Of course, the 2D space is used for display purposes and does not have a suitable number of dimensions.

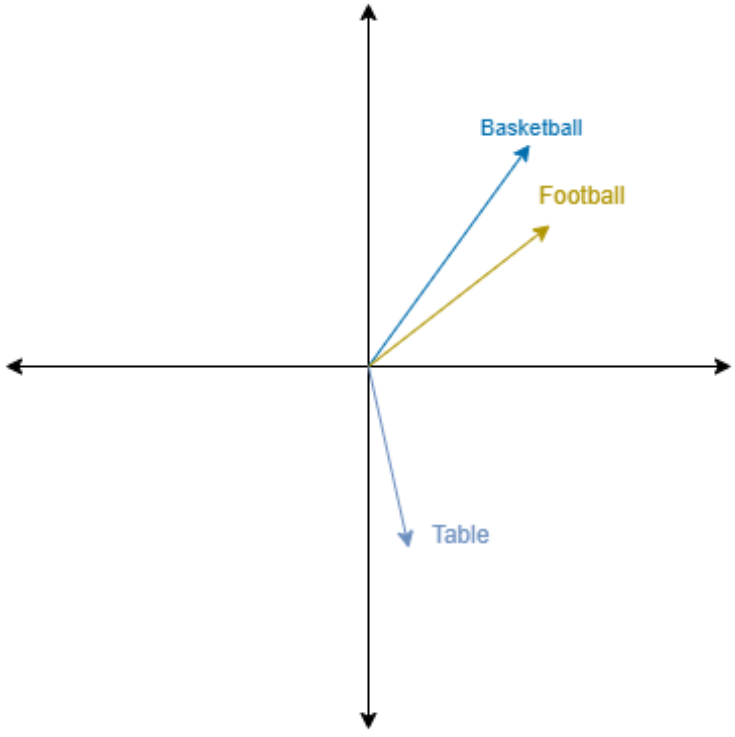

Figure 5.1: Example words in a 2D vector space

Typically, vectors reach over 1000 different dimensions, because this creates room for a vast amount of different directions. In other words, the higher the dimensional space, the richer the encoded data becomes. Data in this form are not interpretable by humans but are very useful for machine learning models designed to perform tasks like semantic search [22]. There are a large number of models that perform this transformation, the most popular being Word2Vec and GloVe.

5.2.2 Contextual Embeddings

Contextual embeddings are a significant improvement over the basic vector representations of words. While models like Word2Vec transform tokens to vectors conveying the global, general meaning of the words, new advanced models based on the transformer architecture and the attention mechanism produce richer representations by taking into account information about the context of each word.

Attention mechanism

The attention mechanism [23] further enhances the embedded vectors with information about the context of each word in a specific case. For example, in the sentence **"The sky is blue"**, Word2Vec would produce a simple embedding for the word **"blue"**, containing information about the word in general, while a more advanced model that uses attention would have created a richer representation, also preserving the information that the word **"blue"** refers to **"sky"**.

In a more formal manner, using the starting embeddings of words in a sentence, the attention block produces 3 matrices: Q (Query), K (Key) and V (Value). If X is the matrix of starting embeddings then these matrices are defined as:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where W_Q, W_K and W_V are matrices of trainable parameters.

Then the attention function is defined as:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V$$

where d_k is a scaling factor.

The output is a new set of vectors which are then added to the original vectors to produce the final embeddings. This process can be repeated multiple times in cascade (many attention blocks) to improve results and also many times in parallel (multi-head attention) to boost the speed of the process. The attention mechanism is also summarized in **Figure 5.2**.

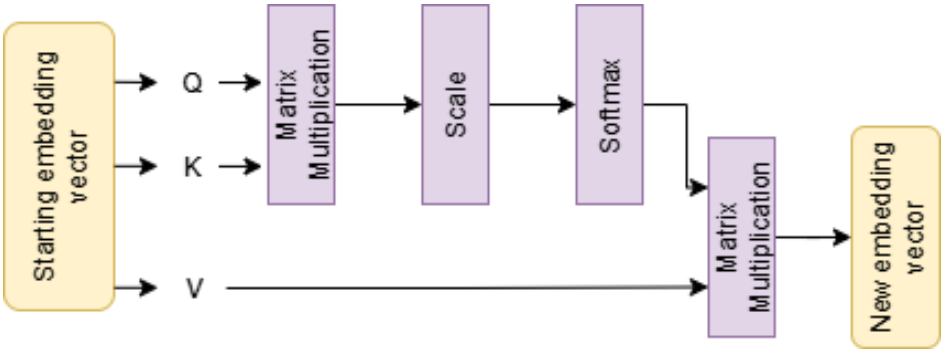

Figure 5.2: Attention mechanism

5.2.3 Embedding Models

Embedding models (also known as embedders) are state-of-the-art transformer models that use the attention mechanism to generate dense vector representations (contextual embeddings) as their output from raw text input. These models enable semantic similarity search tasks, especially in the legal domain, where the texts are formal and convey complex meanings. Embedding models consist of two parts, the tokenizer and the encoder [24].

Tokenizer

The tokenizer is the first module inside an embedder and performs two main tasks. Firstly, it breaks down the input text into small pieces called tokens. These can either be whole words or smaller parts. The most common approach is to use a sub-word tokenization method, meaning that the model splits the text into pieces, smaller than words (subwords). The most popular algorithms for this step are **WordPiece** (used in BERT) and **Byte Pair Encoding (BPE)** (used in GPT). **It is worth-mentioning that the tokenizer is not considered effective if it performs character level splitting.** Converting text at a character level leads to low quality vector representations and thus low model performance. Thus, maintaining relatively large tokens is of great significance for the embedding process.

In the figures below, we display 3 examples of text splitting using the OpenAI's free to

try tokenizer platform [25]. The first **Figure 5.3** is an example of excellent splitting. The tokenizer used is the **o200k_base** and keeps most of the words almost intact, enabling the embedding model to better capture contextual relations and form more useful vectors. The characters-per-token ratio here is 5.69, which is more than adequate for most tasks.

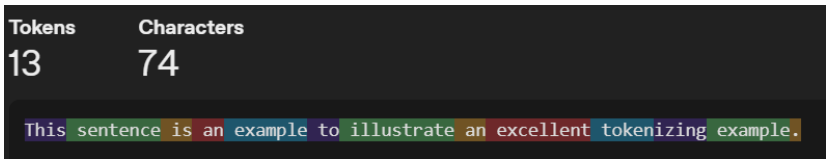

Figure 5.3: Example of excellent tokenizing in English text

In **Figure 5.4**, the same tokenizer is used, and we can already see a significant performance drop in the splitting ability, with most words being broken down to 2 or 3 subwords. This is explained by the use of Greek language instead of English and points out the struggle even the best models face when handling the Greek language. Still though, since the characters-per-token ratio is 3.42, this is a totally acceptable example and could be used in most downstream tasks with satisfactory results.

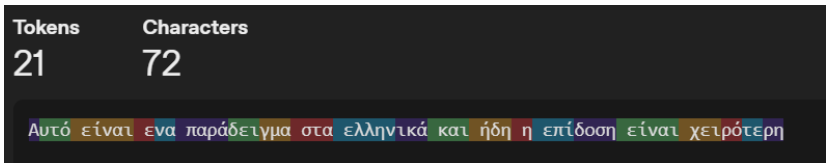

Figure 5.4: Example of acceptable but not great tokenizing in Greek text

Figure 5.5 displays another Greek example but the tokenizer used is an older one provided by OpenAI, the **cl100k_base**. This is an example of the worst performance a tokenizer can have. The text is divided into tokens of 1 to 2 characters (characters-per-token ratio is 1.16) and some symbols seem to not be known at all to the tokenizer. With such a splitting capability, the embedding model is guaranteed to produce low quality embeddings and fail to complete most tasks like semantic search successfully.

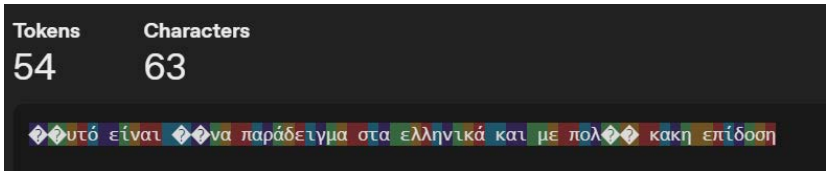

Figure 5.5: Example of unusable tokenizing in Greek text

After splitting the text into tokens, the tokenizer maps all the text tokens to specific numerical ID's. These ID's come from a predefined vocabulary that the tokenizer has, which is built during model training. For example, the sentence **"This sentence is an example to illustrate an excellent tokenizing example."** shown in **Figure 5.3** is converted into the following sequence of ID's after the splitting:

[2500, 21872, 382, 448, 4994, 316, 65836, 448, 7994, 6602, 6396, 4994, 13]

Encoder

The encoder is the core of the embedding model responsible for transforming plain numerical tokens into dense, information-rich vector representations. This is achieved with the aforementioned **self-attention** mechanism which the model uses to determine the relevance of each token to another and blend their contextual information accordingly. This process captures both long and short-term dependencies. Legal texts can greatly benefit from this feature since their semantics are often spread across long periods of text and would otherwise be lost. This process allows the creation of semantic similarity search services using the same model to produce embeddings from the legal documents and the user queries.

The encoder typically starts with an embedding layer that transforms the token IDs into high-dimensional vectors. Then, the main body consists of a multi-head attention block directly followed by a dense, fully connected layers block. Then this set of blocks (also known as a transformer block) is repeated many times in a cascade. In each step, the vectors generated are enhanced with additional information becoming more powerful. The last layer of the encoder is usually a **pooling** layer.

Pooling layers aggregate the results of the previous layer into a single output. In the case of the encoder, the vectors of all tokens are compressed into a single output embedding that contains information about the whole text segment that was given as input. This layer is usually mean pooling which means that the output vector is an average of all vectors in the last hidden layer.

Workflow example

A typical embedder architecture and workflow is described with an example in **Figure 5.6**. Note that the splitting of the tokenizer into a word level is done for simplicity purposes and the values used in the example are random.

Firstly, the raw text is passed into the tokenizer. The tokenizer splits the text into tokens and maps the tokens using its vocabulary to specific numerical ID's. These numbers are passed into the encoder's embedding layer which converts the IDs into multi-dimensional vectors. Then, as described above, these vectors pass through a sequence of transformer blocks (attention and dense layers) and their semantic information is greatly enhanced by the context of the text. Eventually, the vectors produced by the last hidden layer are passed through a pooling layer in order to output the final embedding, which is a vector representing the semantics of the whole input text.

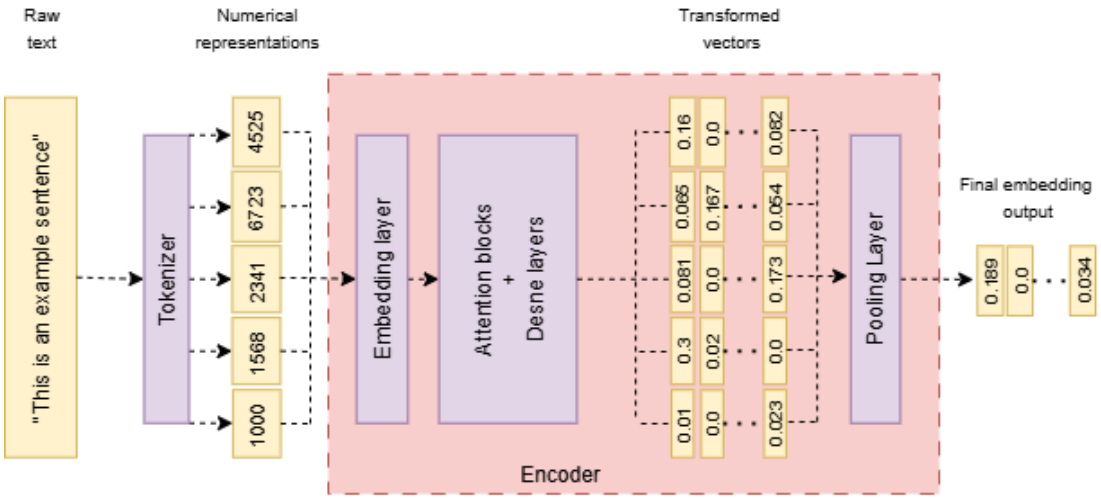

Figure 5.6: Embedding model architecture and workflow

5.2.4 Domain Adaptive Pretraining

Domain Adaptive Pretraining (DAPT) is the practice of further pre-training an already pre-trained language model on text drawn specifically from the target domain of interest. The base model has general knowledge and follows basic language patterns, while the pre-trained one is tuned further on domain-specific language, terminology and structure. This practice can lead to significant improvements on down-stream tasks such as retrieval, question-answering and semantic search on a specialized field like the Greek legal one. A prominent study on this topic is [26], a scientific article that investigates the benefits of continuing pre-training with domain-specific data across multiple domains. Its findings were encouraging enough for us to try this procedure in our model, as domain-adaptive pre-training was shown to consistently improve performance on many downstream tasks.

5.3 Available Model Options

To decide on the models for comparison, we mostly used Hugging Face's tier list to sort the models based on their performance on the most relevant category we could find. That category was **GreekLegalCodeClassification**. We chose the top 15 models as shown in **Figure 5.7**.

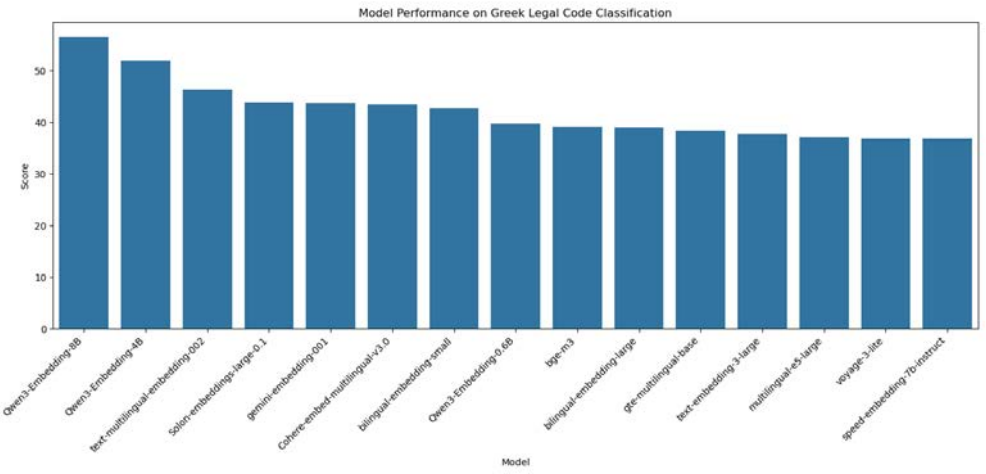

Figure 5.7: HuggingFace's embedding model performance on GreekLegalCodeClassification

Some of the options in this top-15 required paid subscription. Among them, we had access to OpenAI's text-embeddings-3-large so we kept that and eliminated all other premium options. The remaining options were open-source models from hugging face. The high accuracy of the XLM-RoBERTa presented in [4] made us pay close attention to models that had the specific base architecture.

5.4 Method of selection

Having a wide variety of options, we needed a way to filter out the best options available for our specific goal, and then experiment in detail with the remaining options. To achieve this, we split the selection process in two levels.

5.4.1 Level 1 - Filtering

In this phase, we designed a simple experiment for testing the models, having in mind that models which would not pass this simple performance test were not worth experimenting

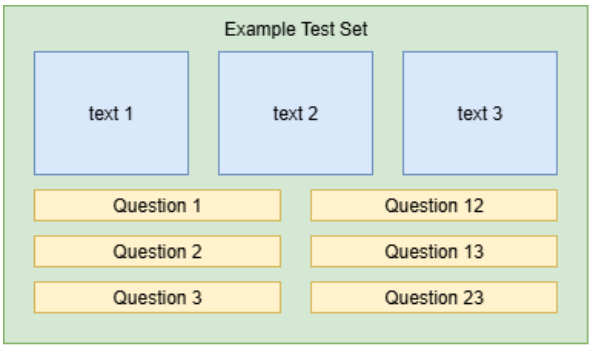

Figure 5.8: Example test set

further with. The test was the following:

We made 2 sets of 3 small Greek legal texts that should not be relevant to each other in the same set. Then we had an LLM like ChatGPT synthetically create 6 questions for each set of texts. The questions followed a strict logic: 3 of them were semantically close to one of the texts, one for each text. The other 3 questions referred to two of the three texts of the set, covering all combinations. A visual representation of a test set is shown in **Figure 5.8**. In the image, Question 1 is relevant only to text 1, while, for example, question 13 refers to both texts 1 and 3 and is irrelevant to text 2.

We then calculated the cosine similarity between each question and the corresponding texts and monitored the distinctive ability of each model between the related texts and the unrelated ones. In parallel, we kept track of the average characters per token that the tokenizer of each model produced, to see how much it affected the model's performance.

The goal of this experiment was to compare the models' capabilities of isolating a single topic from the others and to combine some information while still making a clear distinction from the remaining text. This experiment helped eliminate a lot of models very early in the selection process as either the average characters-per-token was extremely low or the distinctive ability of the model was poor in the Greek legal domain. We also noticed a strong connection between the 2 factors, meaning that the model's performance greatly depended on its tokenizer's ability to generate quality tokens.

For example, the OpenAI's model used the **cl100k_base** tokenizer, which could not handle the Greek characters at all (character to token ratio was 1 and sometimes even smaller, as presented in the example **Figure 5.5**). While other models had better tokenizing performance, their distinctive ability did not increase sufficiently.

The model that stood out and showed the most promising results was **bilingual-embedding-**

large by **Lajavaness** [27], an embedder based on the XML-RoBERTa architecture. Interestingly, the model is not trained in any Greek data, only in English and French, and still performs very well with the Greek legal texts on which we tested it. We believe that this happens partly due to the fact that the tokenizer of the model is XLM-RoBERTa, which is designed for multilingual input and had an average of 2.3 characters per token in our tests. The bilingual embedding-large has a **context window of 512 tokens** and produces 1024-dimensional embeddings.

During our experiments, we discovered that the same profile (Lajavaness) also offered a similar sentence embedding model (**bilingual-document-embedding**) based on the BGE M3 architecture, again using the XLM-RoBERTa tokenizer and having almost identical capabilities with bilingual-embedding-large but with a context window of 8096 tokens [28]. Considering that our documents have significant length (many texts are much larger than 8096 tokens), we decided to also use this model in order to experiment with different context windows and decide the optimal one to capture both long and short-term semantics of the documents.

5.4.2 Level 2 - In-depth testing

Using the two models mentioned above, we converted our entire set of documents into vector database entries using the pipeline analyzed in **Chapter 7**. We made one experiment with the bilingual-embedding-large (512 tokens context window) and four more with the bilingual-document-embedding (1024, 2048, 4096 and 8000 tokens context windows).

After completing the embedding process, we started conducting cosine similarity searches in the database with some generated sample questions and studied the resulting metrics and similarity scores of each search.

The primary issue we faced in this phase was the lack of a ground-truth set to evaluate the performance of the models. Since we do not specialize in the legal field, it is not possible to accurately judge whether the results of the search are legally related to the question asked. Due to this challenge, we based our evaluation on two criteria.

1. **Range of similarity scores**: The range of similarity scores for the database search is a significant indicator of distinctiveness. A small range (e.g., all scores are between 0.59 and 0.6) suggests a limited ability to make distinctions between relevant and irrelevant documents to the query.

2. **Manual inspection of results**: Although we are not capable of determining the correct and faulty legal relations between the documents and the query, we could examine whether the two are connected in a linguistic and semantic way.

The combination of these criteria lead us to decide that the best choice for our semantic search service is the **bilingual-document-embedding with context window size 2048 tokens**, since it showed both the largest similarity score range and returned sufficiently relevant results to the best of our knowledge.

5.5 Domain adaptive pretraining attempt

5.5.1 The procedure

Domain adaptive Pretraining is a procedure that is split into two equally important parts. The Masked Language Modeling (MLM) part and Fine-tuning on a Downstream Task. Let's talk about MLM. At this stage we take our original pretrained model (in our case the **bilingual-document-embedding with context window size 2048 tokens**) and we continue training it on large amounts of unlabeled Greek legal texts. The objective of this stage remains the same as the original pretraining of the model. That is, prediction of randomly masked tokens.

The **Figure 5.9** illustrates the core process of Masked Language Modeling (MLM). Beginning with a sequence of input tokens (t1, t2, t3, t4, t5), one of these tokens is randomly masked (t3 in our case). Then these tokens are fed into the Bidirectional Encoder Representations from Transformers (BERT), which encodes the entire sequence while leveraging both left and right contextual information. The masked token receives special attention, as BERT generates a contextualized vector representation (MASK) informed by its surrounding tokens (t1, t2, t4, t5). This contextualized embedding is then passed through a softmax layer to predict the most likely original word for the masked position. The objective is to maximize the probability of the true token (t3) from the vocabulary distribution. Through repeated exposure to large-scale unlabeled Greek legal text, the model refines its ability to understand domain-specific language usage, thereby improving its embeddings for subsequent fine-tuning on downstream tasks.

The second stage is that the model has to be fine-tuned for a specific task using labeled data. In our case the model had to be fine-tuned on retrieval and semantic search. This stage

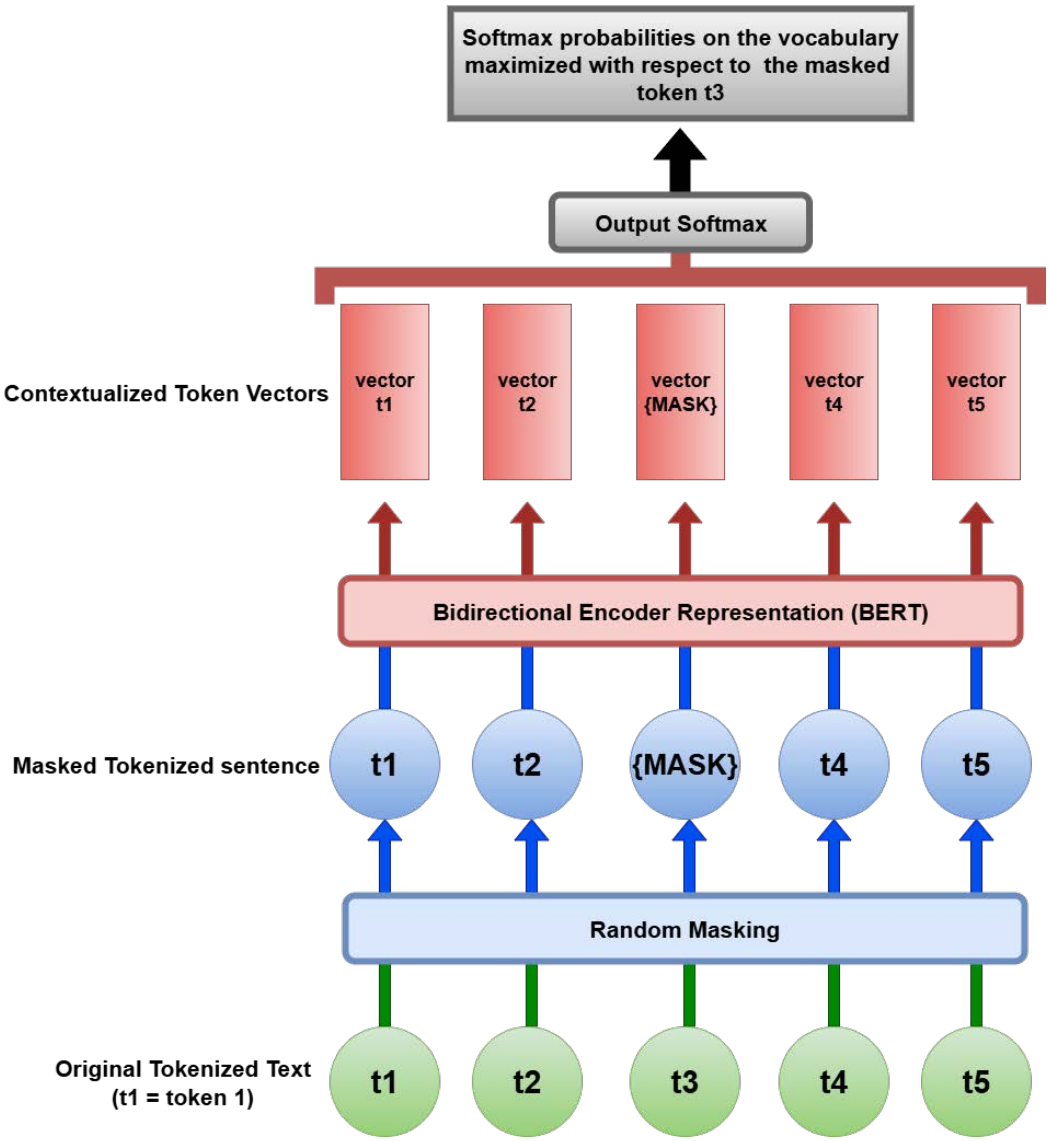

Figure 5.9: Masked Language Modeling (MLM) Procedure

requires a carefully curated labeled dataset consisting of legal queries paired with relevant documents or passages, along with examples of false positives and false negatives to help the model distinguish between semantically similar but legally irrelevant text. During fine-tuning the model learns to encode both queries and documents into a shared embedding space, optimizing their similarity scores such that relevant query-document pairs are placed closer together in the embedding space and irrelevant ones are placed further apart. This process is crucial because it develops the ability of the model to handle complex phrasing and legal synonyms ultimately allowing it to produce better results and increase the retrieval accuracy importantly.

5.5.2 Limitations

The dataset that we described above is something we had no access to while developing our thesis. So we executed the first part of the DAPT procedure and the results of our training attempt are shown on the figure 5.10.

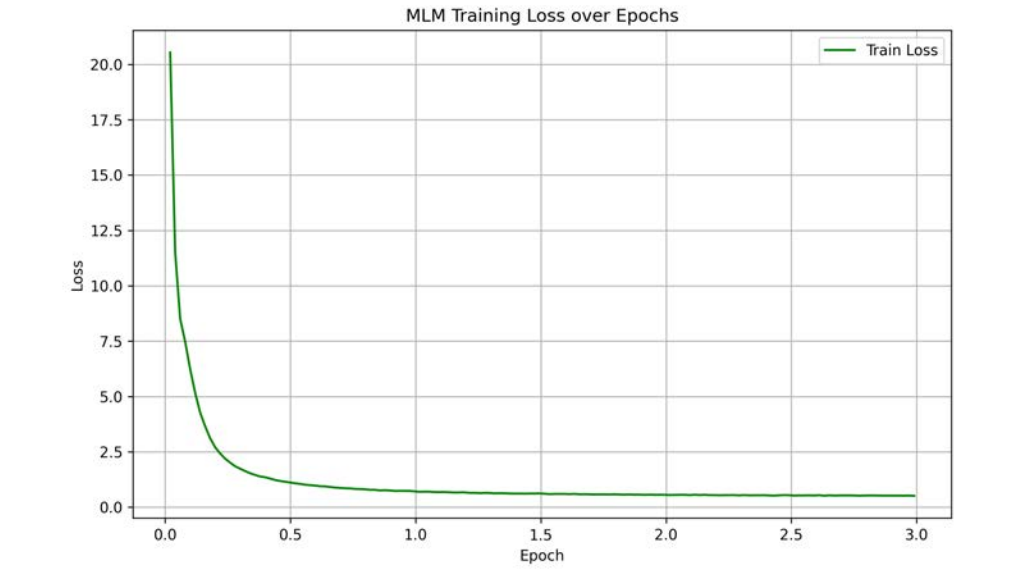

Figure 5.10: Training loss over epochs

As we can see in the plot, the training loss steadily decreased over the first 2 epochs and then plateaued, meaning that the model successfully adapted to the domain-specific language of our corpus. Our associates in the legal domain could produce the required data set in the following years. If this becomes available, we believe that we can achieve more encouraging results in the future and further improve the accuracy and usability of our semantic search application. Although we cannot directly evaluate the downstream task performance of our model since there was no fine-tuning performed, the decreasing of the training loss indicates that the model had successfully learned from the domain-specific documents we used and could achieve better results if fine-tuned in the future.

Chapter 6

Database selection

6.1 Introduction

Semantic search is a service that requires efficient and fast storage and retrieval of documents in their embedded form, in order to achieve high performance. Traditional databases (SQL or No SQL) are not a viable option for the development of such a system, since they lack the ability to handle large-scale semi-structured or unstructured data and are unable to perform efficient insertion and retrieval/search operations on high-dimensional vectors that can reach over 1024 dimensions [29]. Instead, **vector databases** specialize in the management of embedding vectors. They are equipped with algorithms targeted at handling data in this specific, high-dimensional form. This chapter is dedicated to providing foundational information about vector databases while also analyzing the available state-of-the-art options and our choice.

6.2 Background information

6.2.1 Vector Databases

Vector databases are a special type of No SQL databases, that primarily store data in the form of high-dimensional vectors. They support complex unstructured data since they use collections to group data points. Their key feature is their high performance on retrieval and storage, utilizing high-dimensional vectors for indexing and search [22] operations.

In addition to vectors, these databases support the storage of metadata (typically in JSON

format) for each data point. This way vector entries can maintain some useful information in human-readable form. Moreover, most databases provide a hybrid search option. While the primary search key remains the vector point, the retrieval process is enhanced with metadata fields serving as filters.

A typical high-level workflow of a vector database is illustrated in **Figure 6.1**. Firstly, unstructured data are being stored in the database after being converted to vectors by an embedding model. The database optimizes the indexing of the stored entry using various techniques that we analyze in **Subsection 6.2.2**. When a query is made, it must also be embedded by the same model and then passed on to the database. The VDMBS then performs a similarity search between the embedded query and the stored vectors using a variety of Nearest-Neighbor search algorithms, which we describe in detail in **Subsection 6.2.3**.

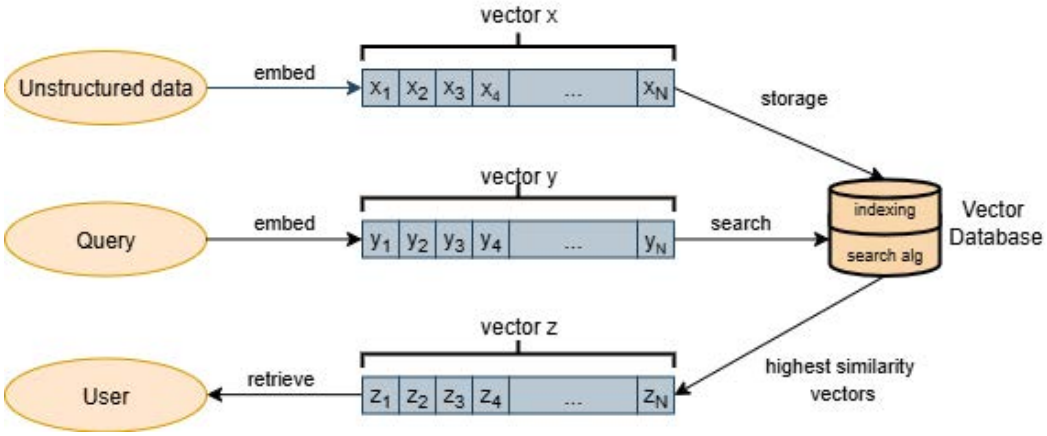

Figure 6.1: High level vector database workflow

6.2.2 Storage optimization techniques

To achieve optimal storage of embeddings while ensuring smooth scalability, continuous support for large-scaled unstructured data and constant high performance, VDBMS use a variety of available methods [30] :

- **Sharding**: It is a method of distributing a vector database system into many machines to boost performance and ensure the availability and scalability of the system. Sharding can be performed either by hashing a specific key from all entries in order to achieve an even distribution of vector entries between shards, or by dividing entries into shards and matching a column of the data points with specific ranges of values.

- **Partitioning**: It is a technique to split the database into smaller parts (partitions). The splitting process is made with criteria such as categories (e.g. shapes) and ranges (e.g. date ranges). The goal of this process is to produce database pieces that are easier to manage and thus enhance the performance and scalability of the database.

- **Caching**: Caching is a common method, that focuses on making the frequently used resources more easily accessible to the system. Its goal is to bypass the retrieval latency from permanent storage by storing data that are used often in a faster memory, like RAM. The most common algorithm is the LRU policy, which drops the **least recently used** data point when the memory serving as cache is full. Caching can greatly enhance the efficiency of the retrieval process, if used correctly.

- **Replication**: This is a technique that creates copies of the data points and stores them in different nodes/clusters of the system. It can either follow a leaderless approach, in which all nodes can process both read and write requests but must be consistently managed, or a leader-follower approach in which a single leader manages follower nodes to perform updates.

6.2.3 Similarity search algorithms

A search query in a vector database is basically a similarity search between the query vector and the vectors stored inside the DB. The similarity search algorithms used by these databases basically try to solve a k Nearest-Neighbors search optimization problem. In other words, they are trying to efficiently determine a set of k points that are considered 'closer' to the query point, given a certain distance metric.

Distance metrics

There are a variety of scores to measure the distance between two vectors in the geometric space. Each score is a function $f : \mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}$ that maps a set of two N-dimensional vectors to a scalar value. Different formulas focus either on the absolute distance of vectors or other attributes like the angle that they form in the vector-space. Depending on the score, higher similarity is associated with larger values or values closer to 0.

- **Euclidean Distance**: It is the L2 norm of two vectors and calculates the straight-line

distance between two points in a Euclidean space:

$$d(x, y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

where x, y are N-dimensional vectors. The range of values is $[0, +\infty)$. A value closer to 0 indicates that the vectors are close to each other.

- **Inner Product**: It is defined as the dot product between two vectors and measures their linear correlation:

$$\langle x, y\rangle = \sum_{i=1}^{N} x_i y_i$$

where x, y are N-dimensional vectors. The possible range of values is $\mathbb{R}$. If x and y are normalized to unit length, the range becomes $[-1, 1]$.

- **Cosine Similarity**: Expresses the angle between two vectors, focusing on orientation rather than magnitude. It is defined as:

$$\cos\_sim(x, y) = \frac{\langle x, y\rangle}{\|x\|\|y\|}$$

where x, y are N-dimensional vectors. The range of values is $[-1, 1]$. A value close to 1 suggests strong similarity, 0 indicates that the vectors are perpendicular, and -1 suggests the vectors have opposite directions.

- **Hamming Distance**: Iτ counts the number of dimensions at which two vectors differ and is commonly used for binary or categorical data:

$$d_H(x, y) = \sum_{i=1}^{N} \mathbf{1}_{\{x_i \neq y_i\}}$$

where $\mathbf{1}_{\{x_i \neq y_i\}}$ is an function that equals 1 if $x_i \neq y_i$ and 0 otherwise. The range of values is $[0, N]$. A value of 0 means that the vectors are identical.

Available optimization algorithms

There are a large number of algorithms designed to optimize the similarity search in vector databases. The most common methods are presented below and belong to the Approximate Nearest-Neighbor search set of algorithms [29]:

- **K-Means**: It is a clustering algorithm that divides vector points into smaller groups (clusters) based on a distance metric. When a query is made, the first step is to match it

to the most relative cluster, and then perform similarity search inside the cluster. This greatly reduces the number of comparisons that need to be performed to complete a search process, thus enhancing the performance of the database.

- **Locality Sensitive Hashing (LSH)**: This is a technique to map similar vector points into the same hash buckets using special hash functions called locality sensitive functions. During a query, only vectors in the same or nearby buckets are compared, significantly reducing the search scope.
- **Hierarchical Navigable Small Worlds (HNSW)**: HNSW is a graph-based algorithm widely used in modern vector databases because of its top performance in similarity search in high-dimensional spaces. It organizes data into multiple layers of graphs where, upper layers provide long-range links between vectors for quick navigation while lower layers offer more accurate connections. During a query, the graph is being accessed from top to bottom, refining the selected k nearest neighbors at each level using a greedy approach.
- **Product Quantization**: It is a compression-based method that works by splitting high-dimensional vectors into multiple low-dimensional subspaces and then quantize each subspace with its own codebook of centroids. The resulting compact codes approximate the original vectors and enable efficient distance computations while preserving sufficient accuracy in search tasks.

6.3 Selecting a vector database

In order to choose the optimal database for our needs we looked into surveys [31] and blogs [32] to see performance metrics, tier lists and other features. The most important criteria we had were:

- **Performance on large scale of data**: Due to the already extensive dataset we had available and given that there is potential to support the full range of legal fields in future work instead of just the Employment Law, selecting a database that can scale easily and still be efficient in storage and retrieval operations was of great importance.
- **Local and cloud host options**: We primarily aimed for a local setup to maintain simplicity and efficiency for our experiments, but we also needed to have the option to

later migrate the database to the cloud, in case the service later gets deployed in the real world. Thus, the database was selected from a collection of free-to-use locally, mostly open-source options.

- **Efficient admin panel**: Since our project was mostly experimental, we required the database to have a user-friendly, simple and easy to learn admin dashboard, in order to manage our system, navigate through and edit the collections when necessary, without having to write commands or scripts to alter the database.
- **Support for hybrid search**: For the needs of our semantic search service, it was important for the database to support the inclusion of filters in the retrieval process. As discussed in **Chapter 9**, the application offers the ability to limit the search year range, to get more relevant documents as a result.

We examined the top five recommended options, focusing on writing down information relevant to the criteria noted above. The features of each database are summarized in **Table 6.1**:

VDBMS	Primary Algorithm	Hybrid Search	Local	Cloud	UI	Scalability
Qdrant	HNSW	Yes	Yes	Yes	Yes	High
Milvus	HNSW	Yes	Yes	Yes	Yes	High
Pinecone	HNSW	Yes	Yes	Yes	Yes	High
Weaviate	HNSW	Yes	Yes	Yes	No	High
Chroma	HNSW	Yes	Yes	Yes	No	Medium

Table 6.1: Comparison of top 5 vector databases.

Pinecone was rejected as an option because it is not open-source, and we preferred an open-source option for our project development to have more freedom. **Weviate** and **Chroma** were ruled out simply because they lack an efficient UI dashboard to manage our database system during experiments. Out of the remaining options (**Milvus** and **Qdrant**) which were both very solid choices, we decided to work with **Qdrant** which is an open-source VDBMS with a user friendly UI, supports both local and cloud host, provides hybrid search with filters, maintains high performance when dealing with large-scale data and uses the HNSW algorithm, which is one of the most popular and widely used similarity search methods available at the moment.

6.4 Navigation in the admin dashboard

To access the admin panel of our Qdrant database we need to use a browser with the url **localhost:6333/dashboard**. Accessing the panel and clicking on the collections icon from the menu on the left loads the collections page shown in **Figure 6.2**. Each collection in the list displays whether its status is healthy, the number of vector points stored in it, the number of segments and shards currently created and basic configurations like the number of dimensions that the vectors have and the type of distance metric that is used to retrieve entries from the collection.

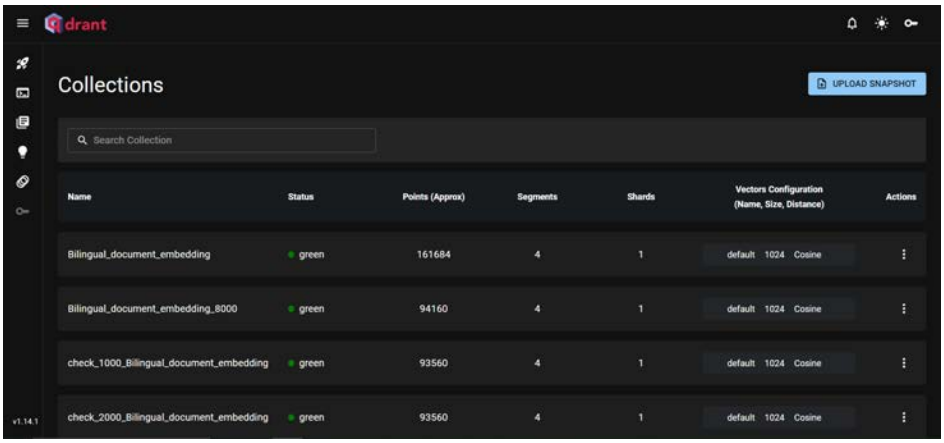

Figure 6.2: Qdrant collections dashboard

Clicking on a collection from the list opens the collection to browse and inspect the points, displaying all metadata and giving the ability to edit them. As shown in **Figure 6.3**, there is also the ability to see some information about the collection's configuration and perform spot filtering on the displayed results with some metadata attributes as keys.

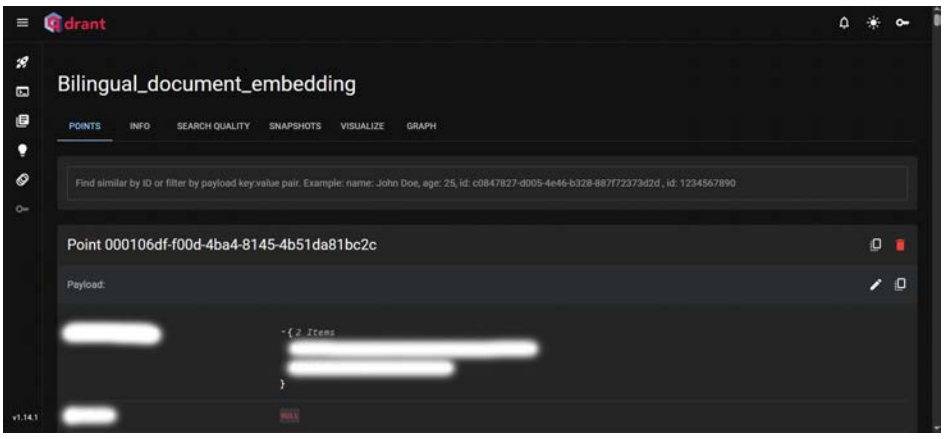

Figure 6.3: Browsing a Qdrant collection

Selecting the **Graph** option loads another interesting page in which the user has the ability to visually inspect the top k nearest neighbors of a particular point. The example in **Figure 6.4** shows a query that illustrates the top 5 nearest neighbors of a random vector point.

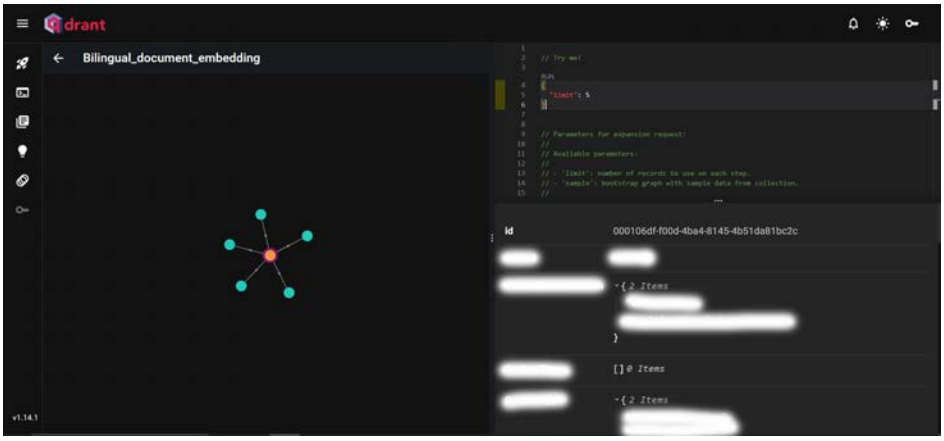

Figure 6.4: Visual presentation of top-k Nearest Neighbors

Lastly, the dashboard offers a console to execute transactions with the database directly from there, without the need to write and execute scripts in a programming language like Python, as shown in **Figure 6.5**. Although we used this feature to perform short tests and inspections in the database, we primarily chose to use the Python client as it was already part of our embedding and information retrieval pipelines.

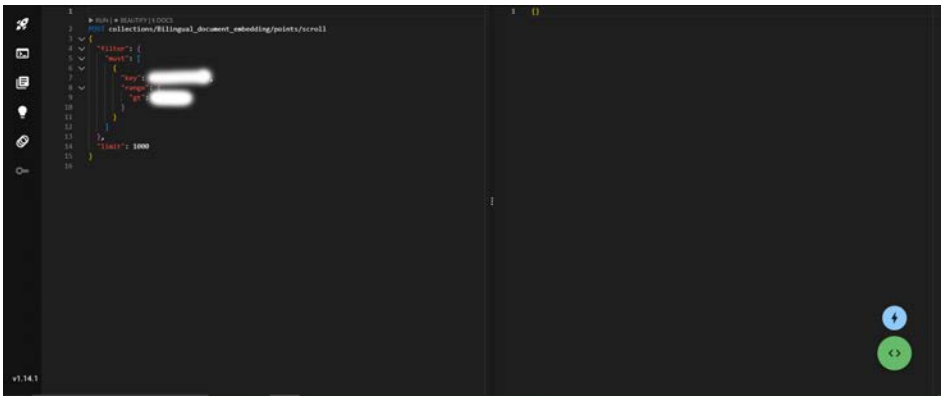

Figure 6.5: Qdrant console

6.5 Setup and configuration

As we already noted, we set up Qdrant locally for our project and experiments. Using the local setup requires the Docker Engine as Qdrant runs inside a container (containers and

Docker are analyzed in detail in **Chapter 9** along with the integration of the database container to a larger docker project). We pulled the Qdrant docker image, mounted a volume of disk storage to keep persistent vector entries and spinned a container exposing two ports (one for REST/JSON and one for gRPC requests) for access from the rest of the system. The database runs on the IP of the host machine (localhost). The collections we made were configured to perform semantic search with the default HNSW algorithm and the **cosine similarity** as distance metric. The configurations used can be examined in the code snippet of our docker-compose.yaml file below:

```
qdrant:
   image: qdrant/qdrant:latest
   ports:
       - "6333:6333"
       - "6334:6334"
   volumes:
       - /opt/qdrant:/qdrant/storage
   networks:
       - app-network
```

For all the transactions we performed with the database through our service we used the **python client** that Qdrant provides.

- **Initializing** the python client:

```
client = QdrantClient(
    url="localhost:6333",
    timeout=60
)
```

  where url is a the timeout is a simple upper time limit in requests that the database processes.

- **Creating a collection** with the python client:

```
point = PointStruct(
    id=sample_id,
    vector=sample_vector],
    payload=sample_metadata
```

```
        )

        client.upsert(
            collection_name=collection_name,
            points=[point]
        )
```

- **Performing a search with filters** with he python client:

```
        condition_list = FieldCondition(
            key=sample_key,
            match=MatchValue(value=sample_value)
        )

        query_filter = Filter( must=[condition_list] )

        search_result = client.query_points(
            collection_name=collection_name,
            query=sample_query_vector,
            query_filter=sample_filters,
            limit=limit
        )
```

These functionalities that the client offers were used to develop both the embeddings pipeline and the information retrieval web service, the two main applications of our system that will be analyzed in the two following Chapters.

Chapter 7

Embedding pipeline

7.1 Introduction

We attempted to develop a scalable and adjustable pipeline for the process of transforming the documents into vector database entries. Although the final model used is from Hugging Face, as we analyzed in **Chapter 5**, we also experimented with OpenAI models. Thus, the goal was for the pipeline to work regardless of the embedding model used, to be able to easily test many models in terms of performance when working with Greek legal data. Moreover, as previously mentioned, we experimented with different chunking methods so we needed this part of the process to be easily editable. For that reason, we designed separate modules for each of the processing steps. Each module is built as a separate Python class.

7.2 Modules

7.2.1 Text Chunker

This module, as its name suggests, is responsible for performing the chunking of documents. The class has a number of parameters and methods. Depending on whether the model is from Hugging Face or OpenAI, some method implementations and parameters are slightly different. This is primarily due to the tokenizer choice depending on the model chosen for embeddings. The class accepts the parameters described in **Table 7.1** and offers the methods described in **Table 7.2**.

Table 7.1: TextChunker Class Parameters

Parameter	Description
max_chunk_size	The maximum chunk size allowed (in tokens)
chunk_overlap	The overlap size (in sentences).
tokenizer_name	The name of the tokenizer that the chunker will use to count chunk sizes

Table 7.2: TextChunker Class Methods

Method	Description
token_count	Counts tokens of a sequence. Depends on the tokenizer
load_sentences	Fetches the sentences of a document from a file
chunk_text	Accepts a document's sentences, creates and returns overlapping chunks. The chunking is performed with the method described in **Chapter 4**.

7.2.2 Text Embedder

Following the logic of the single responsibility that the TextChunker module has, the TextEmbedder's functionality is about using the model to generate embeddings from text sequences. It accepts the parameters and offers the methods described in **Table 7.3** and **Table 7.4** respectively. Some parameters and method implementations again differ depending on whether the model is from Hugging Face or OpenAI.

7.2.3 Database Client

The DBClient is a module responsible for communicating with the database. As explained in **Chapter 6**, the database client used is **Qdrant**. The class accepts the parameters and methods described in **Table 7.5** and **Table 7.6** respectively.

Table 7.3: TextEmbedder Class Parameters

Parameter	Description
max_length	The maximum sequence length allowed in the model (in tokens)
model_name	The name or path of the model to be loaded for creating embeddings
device (hugging face specific)	Determines wether the model will run its inference on GPU acceleration or on CPU

Table 7.4: TextEmbedder Class Methods

Method	Description
remove_paragraphs_markers	Takes as argument a batch of texts and removes the paragraph markers so that the text is cleaned before encoding
embed	Fetches the sentences of a document from a file
chunk_text	Takes as argument a batch of texts, cleans them, creates embeddings using the model and returns them.

7.2.4 Main Module

This is the entry point of the pipeline that loads and uses all other modules. It creates a Python stream that loads sentences from documents and generates chunks from them. When the chunks hit a maximum batch size, it sends the batch to be embedded and stored into the vector database.

It is also responsible for creating the structure of the database entries. It loads the document's metadata and attaches them to every chunk along with some additional chunk-specific metadata fields. The extra specific fields are: the original **chunk text**, the **chunk's number** (in reference to its document), a **flag** indicating that the chunk is part of the document's **summary** and not the main text and the **number of chunks** that the document has.

There is also a configuration file that determines all the parameters described earlier that is loaded into this module.

Table 7.5: DBClient Class Parameters

Parameter	Description
host_ip	The IP that the database is hosted on
port	The port that the database listens on
max_retries	The number of tries that the client will attempt to send a query to the DB
retry_delay	The seconds that the client waits before retrying to send a request
embedder	An embedding module (TextEmbedder in our case) to use in search experiments

Table 7.6: DBClient Class Methods

Method	Description
create_collection	Creates a new collection with the specified name and vector size
insert_to_collection	Attempts to insert a bath of points into the specified collection
search_with_query	Attempts to perform a search with a text query and optionally extra metadata filters. If an output file path is specified an analytic report of the results is dumped into that file.

7.3 Overview of the complete process

The pipeline can work with either the main texts of the documents, or their summaries/titles as separate processes. In each case, the program begins in the main module by setting up the other modules (TextChunker, TextEmbedder, DBClient) and begins to load document texts and metadata as described above. Each time, it sends a document's sentences to the TextChunker and receives the document in chunks. In this step, it also attaches the aforementioned metadata to the chunks. When enough chunks have been created, it sends them as a batch to the TextEmbedder module and gets back the corresponding embeddings. Lastly, the program attempts to insert the data points into Qdrant using the DBClient. This process

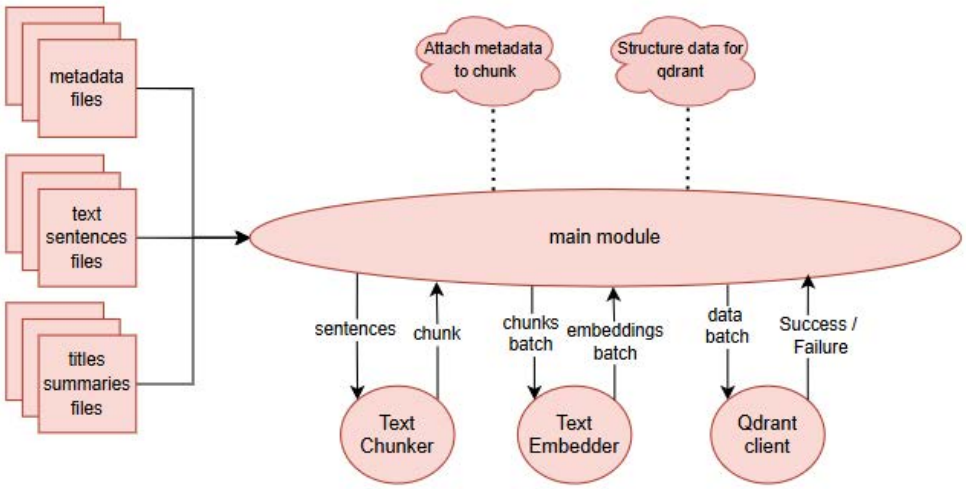

Figure 7.1: Embedding Pipeline

gets repeated until all texts are stored in the database as embeddings. The whole pipeline and process can be visualized in **Figure 7.1**.

Chapter 8

Information Retrieval

8.1 Background Information

8.1.1 Introduction

Information retrieval (IR) is the process of representing, storing and providing access to information items such as documents, web-pages or multimedia objects. The main goal of IR is to provide users with accurate information that satisfies their needs. As the volume of digital information has grown exponentially over the last few years and the demand follows a similar increasing pattern, the need for efficient and effective retrieval techniques has become a cornerstone of modern computing. Historically, the first version of IR was relatively simple pattern matching techniques called Boolean retrieval where a document was retrieved only if it matched the exact terms of a user's query. Over time more sophisticated methods were introduced leading to recent years where machine learning and deep learning IR techniques have developed the ability to capture the semantic and contextual meaning of text. The following sections will review the major categories of AR techniques, discussing their principles, strengths and limitations showing the process from exact matching to semantic understanding.

8.1.2 Classic Information Retrieval Methods

Boolean Model

The Boolean retrieval model is on of the earliest retrieval models based on exact matching using logical operations (AND OR NOT). Documents would be retrieved or not depending

on whether they satisfy the query conditions presented by the user. Even though it is a simple and computationally cheap algorithm, it assumes binary notion of relevance, meaning that documents either match the query or not at all (Salton & McGill, 1983 [33]).

Vector Space Model (VSM)

The Vector Space Model (Salton, Wong, & Yang, 1975 [34]) is designed to represent documents and queries in a high-dimensional-space and each dimension corresponds to a unique term. These terms are weighted using TF-IDF (Term Frequency - Inverse Document Frequency) and are ranked according to their cosine similarity with the query. The VSM model was a great advance from the boolean model because it introduced graded relevance. Its limitation is that it still assumes a bag-of-words representation, so the context and the word order is being ignored.

Probabilistic Model and BM25

Probabilistic models estimate the probability that a model is relevant to a query (Robertson & Sparck Jones, 1976 [35]). The BM25 ranking function derives from probabilistic principles and is one of the most widely adopted and successful methods. It incorporates term frequency saturation and length normalization, making it robust and effective for large scale retrieval ((Robertson, Zaragoza, 2009 [36]). The main problem with this approach is that it still operates on exact term matches which is problematic in domains with rich terminology and many bywords.

8.1.3 Semantic and Topic Models

Latent Semantic Indexing (LSI)

LSI (Deerwester et al., 1990 [37]) is an attempt at overcoming the vocabulary mismatch problem, which is a problem that occurs when the user's query uses different words or expressions than the relevant documents, by applying singular value decomposition (SVD) to the term-document matrix so it can be reduced to a latent semantic space. This allows the retrieval of documents based on conceptual similarity rather than exact term overlap between the document and the query. However, the method suffers from computational inefficiency, especially when dealing with large corpora and it's ability to capture non-linear relationships

is limited.

Probabilistic Topic Models

Probabilistic topic models and especially Latent Dirichlet Allocation (Blei, Ng, & Jordan, 2003 [38]) allow documents to be represented as probabilistic mixtures of latent topics. Each topic encodes a distribution of words capturing thematic structure in the corpus. The LDA method remains limited by the bag-of-words approach so it cannot capture nuanced contextual meaning in specialized domains. Nevertheless, it still provides useful dimensionality reduction and semantic clustering.

8.1.4 Neural and Deep Learning Approaches

Word Embeddings

With the introduction of Word2Vec (Mikolov et al., 2013 [39]) and GloVe (Pennington et al., 2014 [40]) came the rapid advance of IR since now words could be represented as dense, low-dimensional vectors. As discussed in detail in **Chapter 5**, these embeddings can capture semantic similarity (e.g., “man” and “human” appear close in vector space). However they are static, meaning that a word always has the same vector regardless of context, which limits their application when the meaning of a document is heavily dependent on context.

Contextual Embeddings and Transformers

The advent of transformer-based models, most notably BERT (Devlin et al., 2019 [41]), represented a major breakthrough in natural language processing and information retrieval. Unlike earlier embedding methods such as Word2Vec or GloVe, which assign a single static vector to each word regardless of context, BERT produces contextual embeddings (again discussed in **Chapter 5**. This means that the representation of a word dynamically changes depending on the surrounding words in the sentence. For example, the word “bank” will have different embeddings in the sentences “river bank” versus “savings bank”, allowing models to distinguish between multiple senses of a word based on context.

8.1.5 Semantic Search

As we discussed above, classical methods such as Boolean retrieval, VSM, and BM25 have been foundational in IR, but they are limited by their reliance on exact term matching. LSI and LDA provide conceptual similarity but are computationally expensive and don't take the contextual meaning of a sentence into consideration. Using neural methods with static embeddings definitely improved the semantic representation but failed to account for polysemy (the coexistence of many possible meanings for a word or phrase) and context dependence. So we conclude that that the state-of-the-art IR method is semantic search that is powered by contextual embeddings. It combines deep semantic understanding with scalability and provides high precision and recall. That is why we decided that the semantic search method is the most suitable for the Greek legal domain where the documents are lengthy, terminology is specialized and the quality of the results is crucial.

8.1.6 What is Reranking and why it is important

Reranking plays a crucial role in modern information retrieval and question answering systems because initial retrieval methods, whether based on dense or sparse embeddings, often produce results that are relevant but not optimally ordered. By applying a reranker, we can reorder the retrieved items to better align with the user's intent. This step is especially valuable in tasks such as legal document search, open-domain question answering, and retrieval-augmented generation, where subtle semantic differences can determine whether the output is accurate or misleading. Reranking therefore enhances both precision and user trust by ensuring that the most contextually relevant passages are prioritized.

In the legal domain, reranking is especially important because initial retrieval methods often surface passages that are partially relevant but do not fully capture the legal undertone of a query. Legal texts are highly contextual, and two provisions may appear similar on the surface but carry very different implications. By applying a reranker, typically a transformer-based model trained to evaluate semantic relevance at a deeper level, we can reorder candidate results so that legally significant passages appear at the top. This reduces the risk of overlooking key statutory clauses or misinterpreting case law. In practice, reranking has proven valuable in tasks such as case law retrieval, statutory interpretation, and supporting retrieval-augmented generation systems for legal question answering. It ensures that the most contextually and semantically aligned documents are prioritized, thereby improving both the

precision of legal research tools and the reliability of downstream applications.

8.2 Our Approach

8.2.1 Our Semantic search algorithm

In this section we will describe the way we implemented the semantic search information retrieval powered by contextual embeddings. Firstly, we used the **bilingual-document-embedding with context window size 2048 tokens** that we already talked about in **Chapter 5** to turn the chunks (chunking is analyzed in **Chapter 4**) of our texts into contextual embeddings and store them into the qdrant database we set up at **Chapter 6**. When a user asks a question we use the same model to turn the question into an embedding and use cosine similarity to find the most relevant chunks to the question. Then we return the first 5.000 relevant chunks, group them back into documents and rerank the results. This procedure can be summarized by the picture bellow.

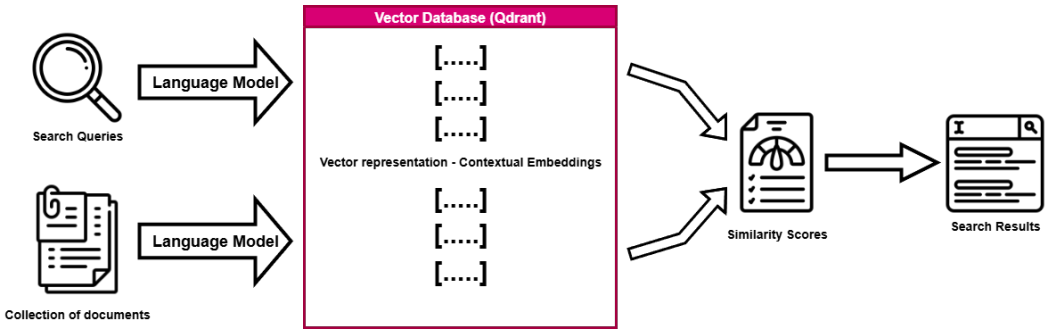

Figure 8.1: Semantic Search Procedure

8.2.2 Challenges of Supervised Reranking

Re-ranking has shown great promise in improving retrieval quality, but training a supervised re-ranker is not always an achievable outcome. Especially in our case, this was proven to be an almost impossible task. The main challenge we had to overcome was the lack of annotated feedback, so we can indicate which retrieved passages are legally relevant and which are not. Constructing such a dataset in the legal domain is especially difficult. It would require extensive input from legal experts and an agreement on what qualifies as the ”correct” answer on a specific question, something that is often ambiguous even for practitioners.

Without reliable ground truth, a machine-learned reranker could risk introducing biases to the system or arbitrary rankings, resulting in lack of trust in the system itself. For this reason, we did not attempt to train a neural reranker and instead explored alternative, unsupervised approaches.

8.2.3 Our Custom Unsupervised Approach

To address this, we developed a custom reranking method that does not rely on labeled data, but instead combines mathematical signals derived directly from the retrieval process. In addition to the maximum cosine similarity that a chunk has with the user's query, our implementation considers two other signals: (i) *coverage*, defined as the number of fragments that were recovered from a legal document in response to a query, and (ii) *mean similarity score* of the document, meaning the mean of the cosine similarity scores of the chunks that were relevant to the query inside a document. To combine these three measures, we introduced a weighted scoring function.

$$\text{score}(d) = \alpha \cdot \text{mean\_sim}(d) \; + \; \beta \cdot \text{coverage}(d) \; + \; \gamma \cdot \text{max\_sim}(d)$$

where α, β, and γ are tunable weights that determine the importance of each component. We give our users the ability to adjust these weights according to their preferences. For example, prioritizing *coverage* (β) favors documents that are consistently relevant across many chunks, even if individual similarities are moderate. Increasing the weight of *maximum similarity* (γ) highlights documents where at least one chunk is highly relevant, which can surface highly specific matches but is also prone to give outlier results, meaning that it can focus on a specific small part of a legal document that refers to the question while the document itself talks about something entirely different. Finally, focusing on *mean similarity* (α) results in rankings that focus primarily on overall semantic closeness, without regard to how many chunks were retrieved or whether one chunk was an outlier. This flexibility allows the reranker to adapt to different use cases within legal document retrieval.

Chapter 9

Full Stack Application

9.1 Introduction

In order to test our service, create demos and provide a friendly user experience, we developed a simple full stack web application. Our goal was to create an efficient user interface with a chat box to make the questions and a panel to display the related results enhanced with rich information regarding the similarity and reranking scores that the model used to sort the retrieved documents. Additional features include links to the original documents, sorting based on different keys and simple search filters. This chapter is dedicated to exploring and explaining the available tools for full stack development, highlighting our choices and describing in detail the web application we built.

9.2 Background Information and selection of tools

Modern full stack web applications typically follow a complete separation logic between the frontend and the backend components. Usually, these are developed as different applications, are often hosted by different servers and communicate through an API exposed by the backend.

9.2.1 Backend tools

The backend application is the server-side logic of a web system. It is responsible for processing requests from the frontend and handling all necessary transactions with the system's

databases. It provides URL endpoints that frontend applications can reach to request for data or other actions like login from the server.

Choosing a programming language

There is vast amount of programming languages support the development of backend services, including Python, JavaScript, Java, and C#. Due to the fact that our backend service required the hosting and use of a deep learning model, we chose **Python** for our development. This choice was the simplest way to achieve direct integration with the machine learning libraries and frameworks (which all are available in Python) that support the functionality of our model:

- **Scikit-learn**: Famous library for performing a significant amount of data pre-processing tasks and training basic machine learning models [42].
- **PyTorch**: It is one of the most popular machine learning frameworks, providing the ability to train custom or existing models and manage resource allocation to CPU and GPU, having direct support for cuda drivers [43].
- **Transformers**: This is a framework offered by **HuggingFace** and it is used to fetch, train and inference free-to-use models from their repository [44].

Choosing a backend framework

Frameworks are reusable foundations of code, built to help developers speed up their work. They provide higher level abstractions and pre-built tools that boost the efficiency to a large extend. There is a great amount of options when it comes to choosing a backend framework in Python. According to [45], the best options at the moment are **Django**, **FastAPI** and **Flask**.

- **Django** is an open-source, extremely popular web framework. By providing a range of built-in tools revolving around Object-Relational Mapping (ORM) security, admin panels and scalability, it boosts productivity while ensuring the creation of a robust application. On the downside, it has a steep learning curve and can be too complex when there is need for a small and simple API design.

- **FastAPI** is a modern, high-performance, web framework for building APIs with Python. It enables rapid code development, has a low learning curve and its applications are robust for production [46]. It also has native support for async programming, a powerful tool for API development. Compared to Django, it has a lot less built-in features which limit its capabilities but also ease the development of simpler services.

- **Flask** is a micro-framework for Python. Its key features are simplicity and flexibility, following a minimalistic approach and having a relatively small learning curve. There is also a large ecosystem of extensions that support the framework. Its limitations include the requirement of a complex setup and configuration for large applications and the lack of pre-built components and tools (external libraries are required).

Out of the three options, we decided to work with **FastAPI** because of its simplicity and as it is a fairly common choice for developing endpoints that serve requests to machine learning models [47]. The next step was to choose the type of API we were going to develop for our similarity search service.

Choosing an API type

An API (Application Programming Interface) is an intermediate layer that connection and data exchange between different software applications and systems. By defining a set of protocols and rules, the API abstracts most of the functionality included in the data transfers [48]. Choosing the right API protocol for each application according to its needs is a crucial task. While there is a vast number of options, we focused on the top 3 API protocols we found [48] [49].

- **RESTful ((Representational State Transfer)** API is an API that conforms to the design principles of the representational state transfer (REST) architectural style. The key points of a REST API are: statelessness (it does not require messages to be in a strict format like other protocol [48]), client-server architecture, representation, uniform interface, resource-based, cache ability and a layered system [49]. This type of API communicates through HTTP requests to perform standard database functions like creating, reading, updating and deleting records (also known as CRUD) within a resource [50]. The main HTTP methods associated with CRUD operations are summarized in **Table 9.1**.

Table 9.1: Backend routes

HTTP Method	CRUD Operation	Description
POST	CREATE	POST requests are usually made to create a new resource in the database, whether it is a new user, a new login connection, a new product or any other similar transaction.
PUT	UPDATE	PUT requests are made when the user wants to perform an update transaction with a resource, modifying existing data.
GET	READ	GET requests are used when the user wants to fetch data from a server. In other words, they request access to information of existing resources.
DELETE	DELETE	DELETE requests, as their name suggests, are usually sent when the user wants to delete an existing resource from a server. The common practice is to request the deletion of a single resource and never multiple.

- **SOAP (Simple Object Access Protocol)** [48] [49] API is a relatively old but still used set of rules to exchange data between sources. It strictly uses the XML (Extensible Markup Language) as a format for both requests and responses and can be sent over both HTTP and SMTP protocols. Its key benefits include interoperability, security, reliability and handling of complex data. While still applicable as a method, it can be a bit more complex to use compared to more modern alternatives like REST.

- **GraphQL** API [48] [49] is a strongly typed query language developed by Facebook. It offers the ability to request specific data in a strict, client-defined format, which reduces unnecessary data exchange. Its unique characteristics offer flexibility and efficiency and is a solid choice when working with mobile apps (limited bandwidth), complex

data structures or micro-services.

Out of the three options, we decided to go with a **RESTful API**. There is a number of reasons that justify our choice. Firstly, our web application had a minimal amount of requests that needed to be implemented (we only needed a login and a search service as we explain later in this Chapter). Also, REST API offers a simple and lightweight solution and is directly implementable through Python's FastAPI framework. Lastly, we rejected SOAP as REST is considered a more modern and popular choice nowadays and GraphQL is a less ideal, more complex solution for a simple, straightforward web application like the one we intended to build.

Authorization and Authentication

Authentication and authorization are the two key factors that establish the security and access control in a web application or server resources. While the two terms are closely related, they actually have a different role in an application. Authentication is the process of verifying the identity of a user or a service. Typically, in a web application this is achieved through a username and password request. Authorization is the security process of granting a corresponding access to requested resources, based on the verified identity that the authentication produced. There is a number of ways to implement this security measure for a web application. To keep things simple, we present two simple and lightweight methods from which we made a selection [51]:

- **Session based Authentication** tries to solve the issue that the user needs to authorize himself with every new HTTP request he makes. After successful login, the server creates a session in memory or a database and gives the user a session ID usually stored in a cookie. The session ID is automatically sent with every request and the server verifies the credential with his stored session and then executes the request. It is a simple and scalable way to establish security in a web application.

- **JSON Web Token (JWT)** is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. The key advantage is that the server does not need to store any information about each client. After successful login, the server generates a JWT with an expiration date that has encrypted inside some client information (signature) and sends it to the

client, which stores it in localStorage or a similar temporary space. All subsequent requests are authorized by the server if they include a valid JWT. JWTs are a widely used option for authentication and authorization within HTTP requests [52].

Since we wanted to keep things simple in the server side (avoid extra database or data structures for sessions) and establish a secure way to access the page we decided to use **JSON Web Tokens** as our security feature.

9.2.2 Frontend tools

The frontend is the client side of a web system. It provides a user interface in the form of a web page and exchanges information with the backend server by making requests to its exposed API. The core components of any frontend application are HTML for defining elements on the page, CSS for styling the elements and JavaScript for making the page interactive and dynamic. Opposed to the backend development, where the language options are many, the frontend development relies almost exclusively in the combination of the above 3. Using a framework simplifies the development process, as it further limits the code writing in only JavaScript.

Choosing a framework

There is a variety of JavaScript library / framework options to speed up and boost productivity of frontend development like React, Vue, Angular and more. The choice for our application was **React**, as it has a relatively low learning curve for its basic features, which are sufficient to built the simple UI we aimed for to support our semantic search service. React is also the most popular tool to build websites right now.

React is a JavaScript library for building user interfaces, allowing developers to create reusable UI components that manage their own state. These components are used to build complex, robust UI applications [53]. It is also a library widely supported by the community, as there is a huge amount of prebuilt components that can be used to significantly decrease code development.

Other development tools

- **Material UI** is the most popular component library for react, providing prebuilt customizable UI elements like buttons, boxes and navigation bars. Using these components ensures responsiveness and consistency of the UI [54].

- **Vite** is a modern build tool and development server for JavaScript applications. It provides fast module hot-reloading during development. It is a popular choice for the development of React applications [55].

- **Nginx** is a high-performance web server that also works as a reverse proxy, load balancer, and HTTP cache. It is a very popular tool choice when deploying a web application [56].

9.2.3 Containerization with Docker

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. It isolates the runtime environment completely from the host system. Using containers ensures that the application runs in all hosting environments while still being a lightweight approach compared to virtual machines. [57]. The most popular and widely used container engine is Docker.

Docker Engine is the industry's standard container runtime that runs on various Linux and Windows Server operating systems. Docker Engine enables containerized applications to run consistently on any infrastructure, eliminating dependency and compatibility issues for developers and operations teams. The classic container architecture is illustrated in **Figure 9.1** [57]. All containers run ("spin") on top of the Docker Engine, which runs directly on the host operating system.

We decided to use wrap our application in Docker containers and create a robust, complete web service. That way we got closer to real-world application development and ensured our website would not face compatibility issues in possible future extensions, while establishing its current stability and performance.

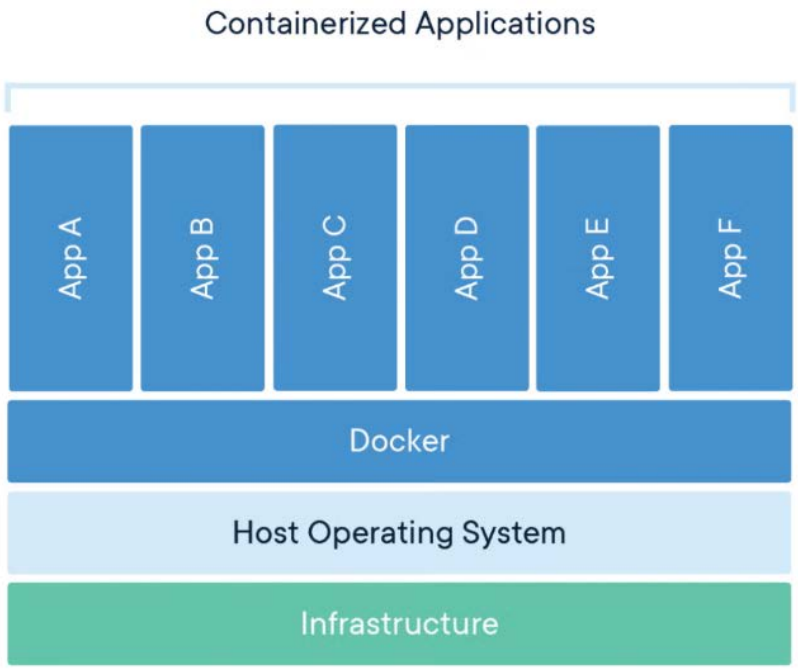

Figure 9.1: Docker container setup architecture

9.3 Backend setup and configuration

We setup the backend application to listen for requests on port 8000 and run it using **Uvicorn**, a lightweight tool for hosting Python web applications, especially when using the FastAPI framework.

Connection with the database

As mentioned in **Chapter 6** our vector database choice to store the embeddings we created was Qdrant. As explained in **Chapter 6** Python offers a client to execute transactions with the database. We used a copy of the same wrapper client module we built for the embedding pipeline described in **Chapter 7**. As we already noted, The service we developed was configured to search the database collections with the **cosine similarity**, which focuses more on the angular distance of vectors and seems to be the optimal choice when dealing with high-dimensional text data [29].

Embedding model host

In order for the semantic search service to work, the embedding model we chose (bilingual-document-embedding) needed to be pre-loaded onto the backend application. To achieve that

we used a copy of the TextEmbedder module we created in **Chapter 7**. Currently, due to limited resources, the inference of the model for the questions made by the users is being performed on the CPU of the host machine.

Reranker module

As we described in great detail in **Chapter 8**, the server also contains a reranking module to rearrange the results based on several criteria. The fetched document chunks from the database query are passed through a long but necessary procedure. Firstly, the chunks are grouped into objects based on the document they belong to. Then, these objects are also grouped based on whether they represent different versions of the same document and are sorted according to their publishing date. Finally, the reranker produces the metrics **max_score**, **average_score** and **coverage**. From them it calculates the **weighted_score** and sorts the objects based on it.

CRUD Functionalities and Routes

The goal of the web application is to test and make demos of the semantic search service. For that matter, there was no need to make any CREATE, DELETE or UPDATE functionalities. The only required actions are the user login and the ability to search the database for documents similar to the semantics of the question asked. Thus, the application was built only with READ operations.

The **/login (POST)** was built in order to limit the access to the service. More specifically, a password login is required. We decided to not create a more complex, account based setup, since the interface is built for testing and demo purposes for now. After logging in, the authorization of the requests on the protected endpoints is achieved with JSON Web Tokens, as described in a previous section. It expects a **password** string field and returns a **valid JSON Web Token** if the authentication was successful.

The **/search (POST)** route is the main endpoint of the application that the user must reach to get document results from a specific question. It expects a **query** string field, a **filters** dictionary and a valid authorization token, and returns the semantic search **results** in the form of a dictionary.

The above endpoints that the backend application exposes are summarized in **Table 9.2**.

Table 9.2: Backend routes

Route	Method	Request	Response
/login	POST	• password (string)	• token (JWT)
/search	POST	• query (string) • filters (dictionary)	• results (dictionary)

Architecture and workflow

The **Figure 9.2** illustrates the complete structure we created for our full stack web application focusing on the backend modules and the workflow that is being executed when a search request is made on the **/search** (POST) endpoint.

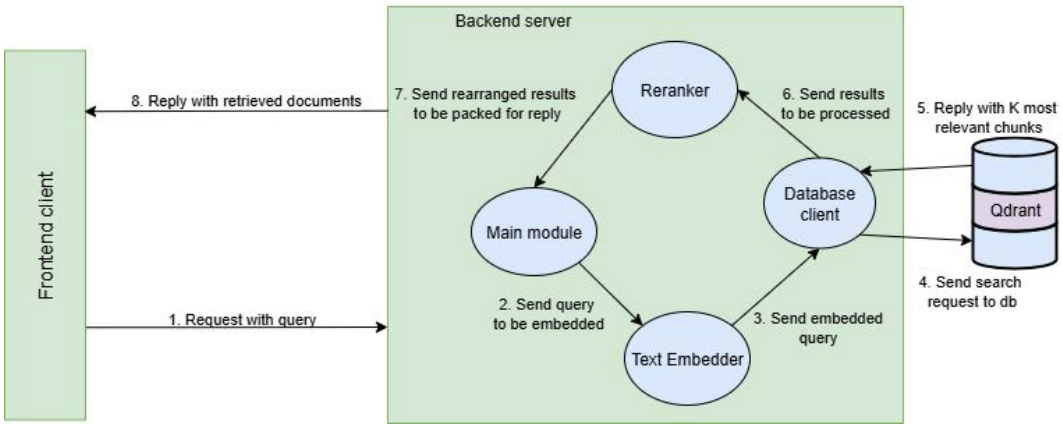

Figure 9.2: Full stack architecture and workflow

As we can observe, the process is completed through a set of 7 steps:

1. The frontend client sends a request with a query for search, that gets accepted by the main module of the backend server.

2. The main module extracts the query and passes it to the TextEmbedder module to be embedded by the model.

3. When the embedding is produced, the TextEmbedder passes the query vector to the database client.

4. The database client handles the transaction with the database. It sends the requested query along with any existing filters to perform similarity search.

5. The database returns the top K most relevant chunks, where K is predefined in the servers parameters.

6. The database client passes the results to the reranker module for further processing, restructuring and sorting.

7. The reranker sends the refactored results back to the main module to be packed for a response.

8. The main module replies to the frontend client with the packed results.

9.4 Frontend configuration and UI

As previously stated, the frontend UI was built for testing and displaying demos. Thus, we designed a simple, easy to use interface. Since the service is addressed to mainly Greek users (the semantic search is performed on Greek legal documents), the majority of the web application displays content in Greek.

The application runs with **Vite** for development and **nginx** for production inside a Docker container (the Docker setup will be explained in a later section of this chapter). In each case, the app server is set to listen on port 3000.

Login Page

The login page, as shown in **Figure 9.3** contains a minimal login form, requiring only a password to access the service.

Home Page

The home page is the landing page shown in **Figure 9.4** when the user logs in successfully. It contains the chat tab to submit questions and a large panel to display results grouped in categories.

Figure 9.3: Login Page

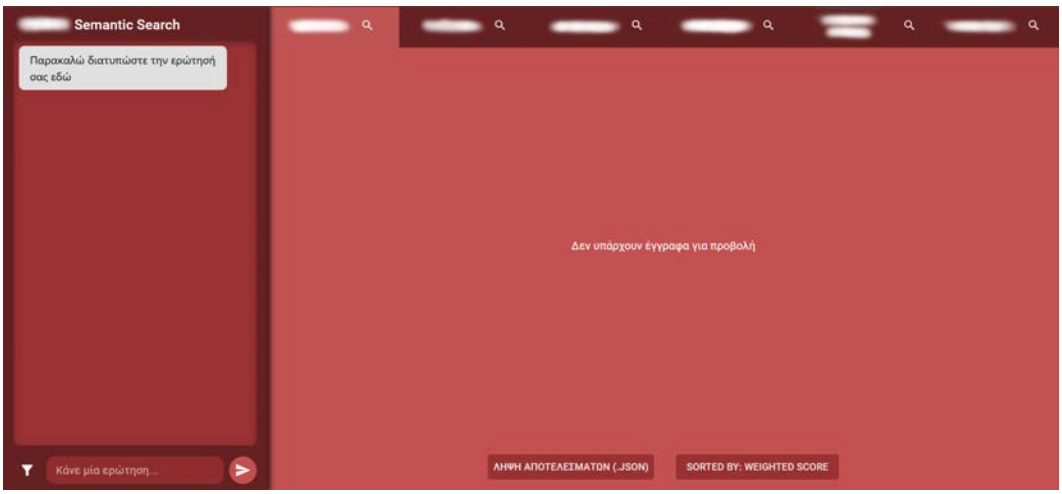

Figure 9.4: Home Page

Chat Tab

The chat tab is not an AI chatbot. It has default prompts and is only for the user to submit questions in free text. There is also a filter icon that allows the user to limit the year range that the search will be performed on.

Results Tab

An example of fetched results is displayed in **Figure 9.6**. The results are always grouped in categories and displayed in the form of cards. Each card corresponds to a specific document. Cards are grouped in pages and are clickable to reveal more details about the returned document. Also, clicking the icon on the right of a card instantly redirects the user to the original document. Lastly, there is a button to download the results in JSON format and another

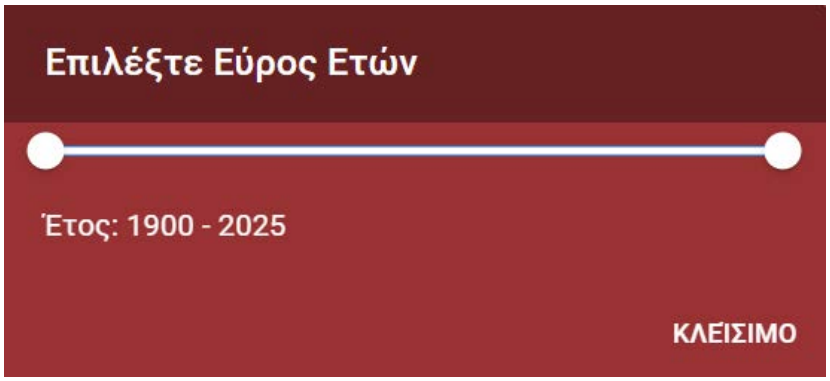

Figure 9.5: Year range filter

one to change the sort key from weighted score to high score (these scores are analyzed in detail in **Chapter 8** where the results reranking process is being explained).

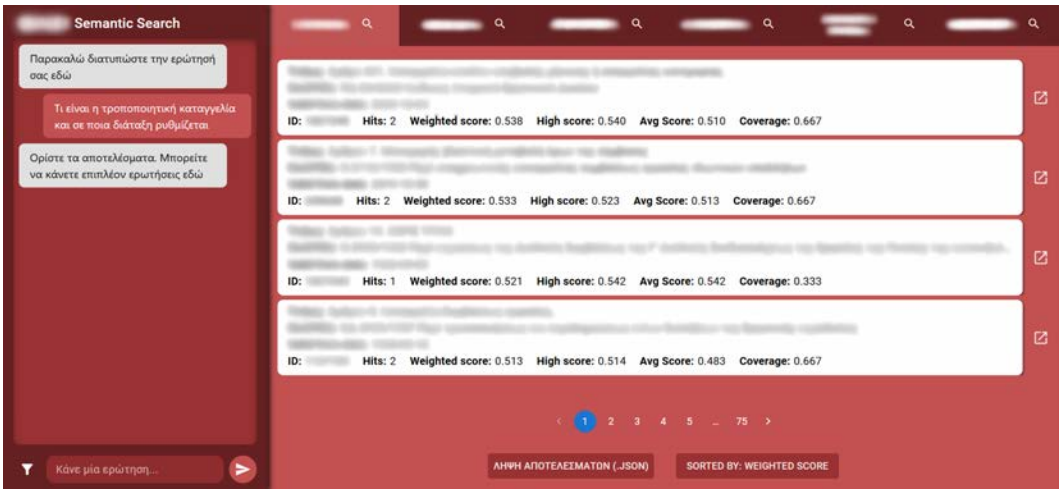

Figure 9.6: Results Page

Result Card

Every result card contains some basic information about the document retrieved and the scores / metrics that were calculated and used to rerank the document in the results list. These scores are: **num of hits weighted score**, **high score**, **average score**, **coverage**.

Result Details

Clicking on a card opens the result details modal (example shown in **Figure 9.7**, that on top of displaying again the scores and basic info of the retrieved result, it also includes more thorough details about the document. A crucial piece of info displayed, is the older versions

of the document available, with links showing the date of publishing for each version and navigating to the corresponding original files. Lastly, the modal has a link button to navigate to the original document's newest version.

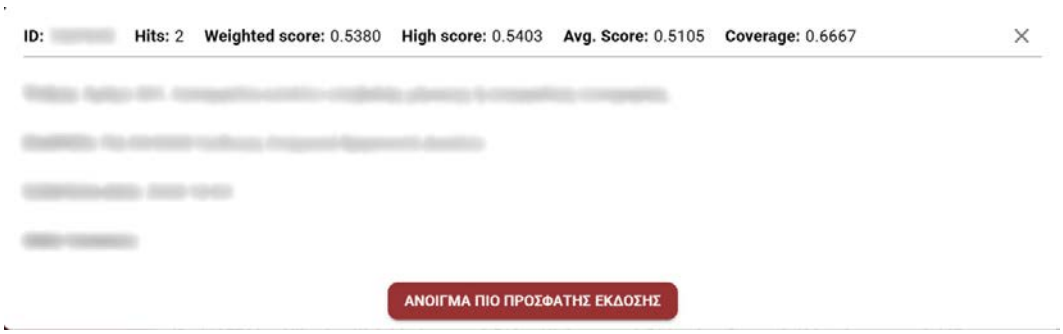

Figure 9.7: Result Details Modal

9.5 Docker setup and configuration

As we described in an earlier section, in order to ensure smooth runtime, isolation and eliminate compatibility issues we setup docker containers for the whole application. More specifically we created three images, one for the backend, one for the frontend and one for the qdrant database. All three images are spinned into containers with the Docker Engine, and communicate with each other through Docker's network bridge driver. The backend runs with **uvicorn** while the frontend runs with **nginx**. The structure of our Docker containers is summarized in **Figure 9.8**.

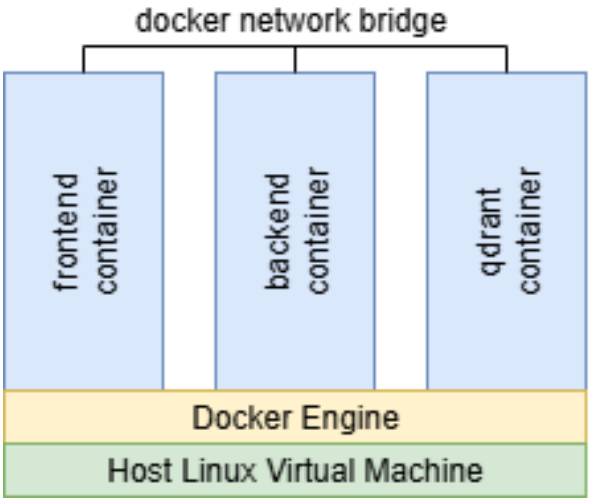

Figure 9.8: Docker Setup

Chapter 10

Conclusion and Future Work

In this chapter we provide a final overview of all the research, experiments and software development that were performed as part of our thesis. Moreover, we discuss potential extensions as well as subsequent research and development ideas that could be conducted based on the foundations of our work.

10.1 Conclusions

The subject of this thesis was the design and implementation of a semantic search service for Greek legal data. The lack of related work in this area (as noted in **Chapter 2**), made the task highly challenging, though it enabled us to explore original ideas in almost every step of the process. Another factor that made this project demanding was the multiple different parts it contained. Over the course of our thesis we had to devote significant attention and time to a vast amount of topics including data cleaning, Natural Language Processing, sentence splitting, chunking, embedding models, semantic search, vector databases, reranking and full stack development.

Our research and development project showed promising results as with the use of a pretrained model, the service functions with a sufficiently high accuracy. Apart from showing that semantic search in Greek legal data has the potential to become a production level service in the future, we also proposed original, effective solutions to many known yet unsolved issues in the area. For instance, our custom -built sentence splitting model is a highly accurate and detailed solution to handle the complexity of Greek legal sentences, while our reranking formula offers a lightweight, straightforward alternative to AI reranking models that do not

have a guaranteed performance.

10.2 Future Work

Though our thesis covered a large range of topics and was an end-to-end research and development project, in our opinion there is still plenty of space for further research and extensions in several parts of the project.

- **Creating a solid ground truth dataset**: As discussed, one of the main challenges we encountered throughout the project was the lack of a labeled dataset to serve as ground truth for both the development of custom embedding models and the evaluation of the semantic search service. Constructing such a collection of data is a highly demanding task, as it requires a significant amount of time and collaboration between machine learning engineers and experts in Greek legal matters. The existence of such a dataset is of paramount importance, as it would greatly enhance any ongoing research and also encourage more researchers to contribute the field.

- **Improving sentence splitting**: Although our sentence splitting model is an already reliable and effective tool for processing Greek legal text, there is still room for improvement. Firstly, the current approach could be extended to include additional rules or be combined with other algorithms to further increase the accuracy. Moreover, the creation of a labeled dataset enables the experimentation with different machine learning models, in order to achieve a more generalized and robust sentence splitter.

- **Training the existing or a new embedding model with Domain Adaptive Pretraining**: As mentioned in **Chapter 5**, due to the absence of a labeled dataset, we were unable to perform a successful Domain Adaptive Pretraining procedure on our model. In future projects, and given that a dataset will be constructed, two paths should be explored. Researchers could either attempt to complete the training attempt we started on the bilingual-document-embedding model or design a new embedder and train it with the same method. A custom-made model should be the best approach in the long term, as it would adapt to work accurately specifically with Greek legal data.

- **Experiment with AI reranking approaches**: Even though our reranking formula is a simple and straightforward solution that functions with satisfactory accuracy, we be-

lieve there is still room to experiment with machine learning models that could achieve higher precision and eliminate the need for manually tuning weight parameters.

- **Investigating the accuracy of the existing model with Greek legal text**: As noted in **Chapter 5**, an interesting observation we made while selecting our model was that the bilingual-document-embedding handled Greek legal data surprisingly well, despite only being trained on English and French texts. A future project could be to investigate and attempt to explain this behavior with targeted experiments.

- **Scale and optimize the full stack service to be deployed**: Although our full-stack web application is fully functional, it was created for demos and testing purposes. While some features for production are included, such as packing the application in Docker containers, a lot of work is required in order for the application to be ready for deployment in the real world. Ensuring security, enhancing responsiveness, scaling to support a large number of users and optimizing the computationally heavy backend (because it hosts the embedding model) are essential tasks for future projects.

Bibliography

[1] Juli Bakagianni, Kanella Pouli, Maria Gavriilidou, and John Pavlopoulos. A systematic survey of natural language processing for the greek language. *Patterns*, page 101313, 2025.

[2] Stamatis Outsios, Christos Karatsalos, Konstantinos Skianis, and Michalis Vazirgiannis. Evaluation of greek word embeddings, 2020.

[3] Charalampos M. Liapis, Konstantinos Kyritsis, Isidoros Perikos, Nikolaos Spatiotis, and Michael Paraskevas. A hybrid ensemble approach for greek text classification based on multilingual models. *Big Data and Cognitive Computing*, 8(10), 2024.

[4] Christos Papaloukas, Ilias Chalkidis, Konstantinos Athinaios, Despina-Athanasia Pantazi, and Manolis Koubarakis. Multi-granular legal topic classification on greek legislation, 2021.

[5] VAMVOURELLIS EFSTRATIOS. Comp-bert-ition: Which bert model is better for greek legal text classification?, 2021.

[6] Vasileios Saketos, Despina-Athanasia Pantazi, and Manolis Koubarakis. The large language model greeklegalroberta, 2024.

[7] Andrea Ferraris, Davide Audrito, Giovanni Siragusa, and Alessandro Piovano. *Legal Chunking: Evaluating Methods for Effective Legal Text Retrieval*, pages 275–281. IOS Press Ebooks, 12 2024.

[8] Brandon Smith and Anton Troynikov. Evaluating chunking strategies for retrieval. Technical report, Chroma, July 2024. Accessed: 2025-04-30.

[9] Jagadeesan Ganesh. Understanding chunking algorithms and overlapping techniques in natural language processing. `https://medium.com/@jagadeesan.`

ganesh/understanding-chunking-algorithms-and-overlapping-techniques-in-natural-language-processing-df7b2c7183b2. Accessed: 2025-04-30.

[10] Derek Wong, Lidia Chao, and Xiaodong Zeng. isentenizer-μ: Multilingual sentence boundary detection model. *TheScientificWorldJournal*, 2014:196574, 04 2014.

[11] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

[12] Nipun Sadvilkar and Mark Neumann. PySBD: Pragmatic sentence boundary disambiguation. In Eunjeong L. Park, Masato Hagiwara, Dmitrijs Milajevs, Nelson F. Liu, Geeticka Chauhan, and Liling Tan, editors, *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*, pages 110–114, Online, November 2020. Association for Computational Linguistics.

[13] Mehwish Fatima and Mark-Christoph Mueller. HITS-SBD at the FinSBD task: Machine learning vs. rule-based sentence boundary detection. In Chung-Chi Chen, Hen-Hsen Huang, Hiroya Takamura, and Hsin-Hsi Chen, editors, *Proceedings of the First Workshop on Financial Technology and Natural Language Processing*, pages 115–121, Macao, China, August 2019.

[14] Explosion AI. *Linguistic Features: Sentence Segmentation*. spaCy Documentation, 2025. Accessed June 2025.

[15] George Sanchez. Sentence boundary detection in legal text. In Nikolaos Aletras, Elliott Ash, Leslie Barrett, Daniel Chen, Adam Meyers, Daniel Preotiuc-Pietro, David Rosenberg, and Amanda Stent, editors, *Proceedings of the Natural Legal Language Processing Workshop 2019*, pages 31–38, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[16] Yang Liu, Andreas Stolcke, Elizabeth Shriberg, and Mary Harper. Using conditional random fields for sentence boundary detection in speech. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 451–458, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[17] Chenglin Xu, Lei Xie, Guangpu Huang, Xiong Xiao, Eng Siong Chng, and Haizhou Li. A bidirectional lstm approach with word embeddings for sentence boundary detection. *Journal of Signal Processing Systems*, Jul 2018.

[18] Unstract Documentation. Chunk size and overlap, 2025.

[19] Rohan Paul. What are different types of chunking?, 2022. Accessed: 2025-09-01.

[20] Yimeng Zhuang, Jinghui Xie, Yinhe Zheng, and Xuan Zhu. Quantifying context overlap for training word embeddings. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[21] Harshad Suryawanshi. How document chunk overlap affects a rag pipeline, 2024.

[22] Toni Taipalus. Vector database management systems: Fundamental concepts, use-cases, and current challenges. *Cognitive Systems Research*, 85:101216, 2024.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[24] Edgar Bermudez. Understanding embedding models in the context of large language models. `https://medium.com/about-ai/understanding-embedding-models-in-the-context-of-large-language-models-02da706ee9b3`, January 2025. Accessed: 2025-08-30.

[25] OpenAI. Tokenizer, 2025. Accessed: 2025-09-12.

[26] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics.

[27] La Javaness. bilingual-embedding-large. `https://huggingface.co/Lajavaness/bilingual-embedding-large`. Accessed: 2025-09-01.

[28] La Javaness. bilingual-document-embedding. `https://huggingface.co/Lajavaness/bilingual-document-embedding`. Accessed: 2025-09-01.

[29] Xingrui Xie, Han Liu, Wenzhe Hou, and Hongbin Huang. A brief survey of vector databases. In *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*, pages 364–371. IEEE, 2023.

[30] Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023.

[31] James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems. *The VLDB Journal*, 33(5):1591–1615, 2024.

[32] Eswara Sainath. Top 5 vector databases in 2025. `https://www.cloudraft.io/blog/top-5-vector-databases`, 2025. Accessed: 2025-04-20.

[33] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[34] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[35] Stephen E. Robertson and Karen Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[36] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[37] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[38] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[40] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[42] Scikit learn Developers. Scikit-learn: machine learning in python. `https://scikit-learn.org/stable/`, 2025. Accessed: 2025-09-12.

[43] PyTorch Contributors. Pytorch. `https://pytorch.org/`, 2025. Accessed: 2025-09-12.

[44] Hugging Face. Transformers documentation. `https://huggingface.co/docs/transformers/index`, 2025. Accessed: 2025-09-12.

[45] Cubode Team. What's the best backend framework for ai development in 2024? django, fastapi or flask. `https://medium.com/@cubode/whats-the-best-backend-framework-for-ai-development-in-2024-django-fastapi-or-flask-d52c165ea20c`, January 2024. Accessed: 2025-09-12.

[46] FastAPI. Python web framework. `https://fastapi.tiangolo.com/`. Accessed: 2025-08-30.

[47] Grigor Khachatryan. Serving ml models with fastapi: A production-ready api in minutes. *Medium*, June 2025. Accessed: 2025-08-30.

[48] Vinayak Singh. What is an api? its types and protocols used. `https://medium.com/@vinayak-singh/what-is-an-api-its-types-and-protocols-used-bfaad3fa8e77`, November 2023. Accessed: 2025-09-12.

[49] Olatunde Emmanuel. Types of apis. `https://medium.com/@mohbohlahji/types-of-apis-32bb2f9f40a8`, March 2024. Accessed: 2025-09-12.

[50] IBM. What is a rest api (restful api)? `https://www.ibm.com/think/topics/rest-apis`, April 24 2025. Accessed: 2025-08-30.

[51] Shiva Rrad. Understanding the different methods of authentication and authorization. `https://shiva-rrad.medium.com/understanding-the-`

different-methods-of-authentication-and-authorization-2534de1a77f6, 2023. Accessed: 2025-09-12.

[52] Json web tokens. https://auth0.com/docs/secure/tokens/json-web-tokens. Accessed: 2025-08-30.

[53] React. A javascript library for building user interfaces. https://reactjs.org/. Accessed: 2025-08-30.

[54] Material UI Contributors. Material ui: React components that implement material design. https://mui.com/material-ui/, 2025. Accessed: 2025-09-13.

[55] Vite. Next generation frontend tooling. https://vite.dev/. Accessed: 2025-08-30.

[56] Igor Sysoev and the NGINX Project. Nginx: High-performance web server, reverse proxy, load balancer, and http cache. https://nginx.org/, 2025. Accessed: 2025-09-12.

[57] Docker. What is a container? https://www.docker.com/resources/what-container/. Accessed: 2025-08-30.