UPPSALA UNIVERSITET

UPTEC F 24050

Examensarbete 30 hp

Mars 2024

# Development of a Semantic Search Tool for Swedish Legal Judgements Based on Fine-Tuning Large Language Models

Sebastian Mikkelsen Toth

Civilingenjörsprogrammet i Teknisk Fysik

UPPSALA UNIVERSITET

# Development of a Semantic Search Tool for Swedish Legal Judgements Based on Fine-Tuning Large Language Models

Sebastian Mikkelsen Toth

## Abstract

Large language models (LLMs) are very large deep learning models which are retrained on a huge amount of data. Among the LLMs are sentence bidirectional encoder representations from transformers (SBERT) where advanced training methods such as transformer-based denoising autoEncoder (TSDAE), generative query network (GenQ) and an adaption of generative pseudo labelling (GPL) can be applied. This thesis project aims to develop a semantic search tool for Swedish legal judgments in order to overcome the limitations of traditional keyword searches in legal document retrieval. For this aim, a model adept at understanding the semantic nuances of legal language has been developed by leveraging natural language processing (NLP) and fine-tuning LLMs like SBERT, using advanced training methods such as TSDAE, GenQ, and an adaption of GPL. To generate labeled data out of unlabelled data, a GPT3.5 model was used after it was fine-tuned. The generation of labeled data with the use of a generative model was crucial for this project to train the SBERT efficiently. The search tool has been evaluated. The evaluation demonstrates that the search tool can accurately retrieve relevant documents based on semantic queries and simnifically improve the efficiency and accuracy of legal research. GenQ has been shown to be the most efficient training method for this use case.

**Teknisk-naturvetenskapliga fakulteten**

**Uppsala universitet, Utgivningsort Uppsala**

Handledare: Kristoffer Röshammar  Ämnesgranskare: Ping Wu

Examinator:Tomas Nyberg

## Popularvetenskaplig sammanfattning

Advokater och jurister är ofta i behov av att kunna hitta relevanta domar för att stärka sina fall. Traditionellt har detta inneburit att man förlitat sig på nyckelordsbaserade sökverktyg som kan vara begränsade och ineffektiva, då de inte fångar upp den fulla betydelsen eller sammanhanget i sökfrågorna. De har då behövt justera och anpassa sina sökningar flera gånger för att hitta något relevant. I detta projekt har vi utforskat hur en övergång till semantisk sökning kan förbättra denna process genom att utveckla ett semantiskt sökverktyg särskilt anpassat för svenska domar. En semantisk sökfunktion är även mer användarvänlig då man kan söka med vardagsspråk, och ändå hitta document skrivna I avancerat juridiskt språk.

Semantisk sökning skiljer sig från traditionell nyckelordssökning genom att den strävar efter att förstå avsikten och betydelsen bakom användarens sökfråga, vilket möjliggör mer relevant sökresultat. Till skillnad från en nyckelordssökning kan en semantisk sökfunktion förstå kontext utan att specifika ord matchas mellan sökfras och document. För att åstadkomma detta, har vi tränat en språkmodell (LLM) specifikt för juridisk text, med syftet att förbättra verktygets förmåga att tolka och matcha sökfraser med relevanta domar.

I detta fall har vi en assymetrisk sökning där sökfrasen är mycket kortare än dokumentet vi vill hitta. Detta gör det ännu viktigare att modellen förstår innebörden av texten och kan hitta dokument som har samma innebörd som sökfrasen, men kanske inte har några matchande ord.

Projektet involverade olika metoder för att träna en LLM, som sedan jämfördes. En annan viktig del i projektet var att generera träningsdata. Eftersom vi bara hade PDF dokument att tillgå behövdes sökfraser genereras för att få tillräckligt med data för att träna modellen. En generativ modell från OpenAI finjusterades för att effektivt generera sökfraser som passar till dokumenten. Detta gjorde att bättre träningsmetoder kunde användas när datan var komplett.

Resultaten från utvärderingen av det semantiska sökverktyget visade en märkbar förbättring i förmågan att hitta relevanta dokument baserat på semantisk likhet, vilket bekräftar potentialen hos semantisk sökning i juridik. Denna forskning belyser inte bara vikten av anpassade lösningar för specifika domäner utan öppnar även upp för framtida innovationer inom juridisk informationsteknologi.

Detta arbete avser att förbättra användarupplevelsen samt effektiviteten jämfört med den nuvarande nyckelordssökningen.

# Contents

# Acronyms

| | |
|---|---|
| **AI** | Artificial Intelligens. 5 |
| **API** | Application Programming Interface. 19 |
| **BERT** | Bidirectional Encoder Representations from Transformers. 7 |
| **GenQ** | Generative query network. 13 |
| **GPL** | Generative pseudo labeling. 13 |
| **GPT** | Generative Pre-trained Transformer. iv, 11 |
| **ML** | Machine Learning. 3, 5 |
| **MNRL** | Multiple Negatives Ranking Loss. 17 |
| **MSE** | Mean Square Error. 16 |
| **NJ** | Norstedts Juridik. 1 |
| **NLP** | Natural Language Processing. 1 |
| **RAM** | Random Access Memory. 18 |
| **RNN** | Recurrent Neural Network. 6 |
| **SBERT** | Sentence Bidirectional Encoder Representations from Transformers. iv, 7 |
| **TSDAE** | Transformer-based Denoising AutoEncoder. 13 |

# 1 Introduction

## 1.1 Background

In a lawyer's daily work, they often make use of previous legal judgments. If they can find previous judgments similar to their current case, they will be able to make better decisions.

Norstedts Juridik (NJ) has a tool for searching through thousands of legal documents. However, their text search function is based on keyword search, searching for literal matches of the query keywords without understanding the overall meaning of the words. Therefore, semantic search (i.e., search with meaning) is a better approach than the keyword search.

With semantic search, the lawyer should be able to describe their case and its circumstances and find similar cases, even if the words do not match exactly between the query and the documents.

The fundamental issue with the keyword-based search approach lies in its lack of contextual understanding. Such systems are designed to match exact words or phrases within the database, disregarding the nuanced meanings and contextual relevance that are often crucial in legal texts. This results in several critical shortcomings:

1. Limited Contextual Recognition: Keyword searches do not comprehend the context or the semantic relationships between terms. Legal language is complex and often subject-specific. Without understanding the context, a keyword search might miss documents that are highly relevant but do not use the exact search terms.

2. Inefficiency and Incompleteness: Due to the literal approach of keyword matching, lawyers may need to try multiple variations of search terms to find all relevant documents. This process can be time-consuming and often still fails to retrieve some pertinent cases.

3. Synonym Blindness: Legal terminology often includes synonyms and related terms that a basic keyword search might overlook. For instance, a search for "fraud" might miss documents mentioning "deceit" or "misrepresentation," even though they pertain to similar legal concepts.

4. Rigid Search Queries: Lawyers are forced to tailor their queries to match the anticipated language in the documents, rather than using natural language. This requirement for precise wording can hinder the discovery of relevant documents and make the search process less intuitive.

These limitations underscore the need for a more advanced, semantic-based search tool. Semantic search, with its ability to understand the meaning and context behind words, offers a solution that aligns more closely with the Natural Language Processing (NLP) and interpretation skills of human users. By adopting semantic search approach, NJ's search tool could significantly enhance the efficiency, accuracy, and user experience of legal document retrieval, thus better serving the needs of legal professionals.

## 1.2 Purpose and goals

### 1.2.1 Purpose

The purpose of this project is to develop a semantic search tool for Swedish legal documents, where the user can search for judgments similar to their case. Furthermore, we will train an embedding model specifically on text from legal judgments, and see if it generates satisfying results for a semantic search tool. We will then compare it to an out-of-the-box open-source model that is trained on Swedish text.

Additionally, we will fine-tune a generative model for the purpose of generating labeled data for the training process of the embedding model.

### 1.2.2 Goal

The goals of this project is:

- Develop a search tool
- Train an Embedding model
- Fine-tune a generative model to generate training data

The output of the search tool should show a list of documents relevant to the search query, with the title and an excerpt of the matching text. From the title of the document, the full document can then be picked from the database.

The goal is that the search results will become better after training the model. Additionally, we want the fine-tuned model to generate enough good training data for a low cost.

# 2 Theory

This section will present the theory of the components and methods used in this project.

## 2.1 Natural language processing (NLP)

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics that focuses on the interactions between computers and human languages. It involves applying computational techniques to analyze natural language and speech.

A fundamental step in NLP is tokenization, which involves breaking down of text into smaller components, like words or subwords. This process is essential for converting text into a format that can be processed by Machine Learning (ML) models. Another crucial technique is part-of-speech tagging, which involves assigning parts of speech, such as nouns, verbs, and adjectives, to each word in a sentence. This helps in understanding the grammatical structure of the text.

Transformers are the key of modern NLP advancements. Transformers use self-attention mechanisms to process input data in parallel, rather than sequentially, allowing more efficient and powerful models. Large Language Models (LLMs) such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) have revolutionized NLP by enabling machines to understand and generate natural text with high accuracy. These models are pre-trained on huge amounts of text data and can be fine-tuned for specific tasks, which is done in this project.

NLP has a wide range of applications. In information retrieval, NLP improves search engines by understanding the context and semantics of search queries, leading to more relevant search results. Chatbots and virtual assistants leverage NLP to interact with users in natural language. Text summarization algorithms generate summaries of larger texts. Question answering systems can understand and respond to questions asked in natural language. In this project, we will make use of NLP techniques both for the search function and for text generation.

## 2.2 Semantic search

Often a synonym for some keywords will also match. The order of the words in the search query will also determine how the results are ranked. Basically, the data with text that is written similarly to the query will be the top result.

With semantic search, however, the way the query is formulated should not affect the results. Semantic search signifies a major advancement in how machines understand human language. It does not just recognize words but also grasps the underlying concepts and relationships between them. The results in a semantic search are based on the matches between the semantic meaning in the query and the semantic meaning of the data. This means that the query does not need to be written in a specific way that matches the data to find the best results. In other words, natural language could be used to find the results even if the results are written in an industry-specific language. Another advantage is that results with similar meanings to the search query will appear. For example, in this case with law texts, a lawyer may want to find judgments that are similar to their case, but not necessarily an exact same case. A challenge in this project is that there may be no matching words between the query and the document, so the model has to have an understanding of the meaning. Another challenge for this project is the asymmetry between the search queries and the results. The search queries can be as short as a few words, or a sentence, while the documents in the results can be multiple pages long. This will force the model to understand both the queries and documents, even if they seem very different from a computer's perspective. Additionally, the difference in natural language (that will be used in the search queries) and the legal language (in the documents) will be a challenge for the model.

Semantic search can also be used together with keyword search, where the search results are the semantic similar results from the data, but they are ordered with respect to the keywords.

Moreover, semantic search plays a critical role in the user experience. It enables more natural and conversational interactions with the search tool, similar to how humans are used to communicate. This enhances user engagement and satisfaction, as people find what they need more efficiently and with less effort. In a world where time is precious, and attention spans are short, this aspect of semantic search is crucial.

In the domain of semantic search, there are two major different cases to consider: asymmetric and symmetric semantic search. These terms refer to the nature of the query and the corpus of information against which the query is matched.

### 2.2.1 Symmetric Semantic Search:

In symmetric semantic search, both the query and the information corpus are of comparable size and structure. This type of search is often employed in scenarios where the objective is to find similarities or connections between two sets of data of similar lengths, such as matching similar questions or comparing paragraphs of text. Symmetric search is characterized by its bidirectional nature, where both the query and the corpus are given equal weight in the analysis.

### 2.2.2 Asymmetric Semantic Search:

Asymmetric semantic search, on the other hand, involves a disparity in the size and structure of the query and the corpus. Typically, the query is much shorter than the information it is being compared against. This approach is prevalent in situations where the goal is to find a detailed piece of information using a brief query. For example, in the context of legal research, a lawyer might use a concise sentence or question to search for a long legal documents. Asymmetric search is particularly challenging as it requires the search algorithm to effectively interpret and match the brief query against a much larger dataset to find the most relevant information.

**Application in this Project:** In this project, we focus on asymmetric semantic search. The nature of legal document retrieval involves searching for extensive documents using concise queries. This approach is more aligned with the typical use case of legal professionals who need to quickly find relevant documents or documents based on short, specific queries. Asymmetric semantic search's ability to interpret and match these short queries against a vast corpus of detailed legal texts makes it an ideal choice for this application.

However, a symmetric search could also be useful, so that the lawyers could exactly describe their case in detail, or copy a police report directly. But that is not a part of this project. Here, we focus on the most popular use case of the tool where the lawyer inputs one or a few sentences.

The challenges in asymmetric search include ensuring that even brief queries can capture the essence of what the user is seeking and that the search tool can effectively navigate through extensive documentation to find the most relevant matches. Another challenge is to capture enough specific details in the document, while still having an overview of the semantic meaning. Addressing these challenges is key to developing an efficient and user-friendly semantic search tool for legal documents.

An especially hard challenge in this use case is that it is very unlikely that the search query has any matching words with the documents. This forces the model to really understand the domain rather than see some patterns.

### 2.2.3 Cosine similarity and cosine distance

Cosine similarity measures how similar two vectors are, and cosine distance measures how different they are. In this thesis, we will mostly use similarity measurements rather than distances. However, in some occasions, it is simpler to talk about distances. There are several ways to measure both similarity and distances, and the metric mostly used in this thesis is cosine similarity. Subhashini et al. (2010) [11] showed that cosine similarity is the most appropriate metric for this kind of application.

Cosine similarity is a measure used to compute the similarity between two non-zero vectors. In the context of semantic search, these vectors often represent text documents and queries. The cosine similarity is given by the cosine of the angle between two vectors.

Mathematically, the cosine similarity between two vectors $A$ and $B$ is represented as:

$$sim(A, B) = cos(\theta) = \frac{A \cdot B}{||A||||B||}, \tag{1}$$

where $A$ and $B$ are vectors and $\theta$ is the angle between them.

In semantic search, documents or text inputs are transformed into vectors where each dimension corresponds to a term or concept in the document. When a query is made, cosine similarity is used to compare the query vector with document vectors in the search space. The cosine similarity scores range from -1 to 1, where represent that the vectors point in the same direction, 0 indicates orthogonality (no similarity), and -1 indicates the opposite. [4].

The cosine distance is given by

$$cosine\_distance = 1 - cosine\_similarity. \tag{2}$$

Other similarity metrics are dot product and Euclidian distance. These metrics also consider the magnitude of the vectors, which is not relevant for this application. Subhashini et al. (2010) [11] concluded that metrics like euclidian distance are not appropriate for this kind of application. However, the dot product is used in some training methods for large language models.

## 2.3 Machine learning

Machine Learning (ML) is a subfield of Artificial Intelligens (AI) focused on building systems that learn from data to make predictions or decisions without explicit programming. Some different cases in ML includes supervised learning, where models are trained on labeled data. Unsupervised learning identifies patterns in unlabeled data. Semi-supervised or self supervised learning combines both.

Deep learning, a subfield of ML, involves neural networks with many layers. These networks are powerful for complex tasks like image, speech or text recognition. In the context of NLP, deep learning models, especially transformers, have enabled significant advancements.

## 2.4 Large language models (LLMs)

Large Language Models (LLMs) are deep learning models trained on vast amounts of text data to understand and generate human language. They are named by their large scale, often containing billions of parameters, and their use of the transformer architecture, which allows for efficient processing of text.

LLMs have various applications, including text generation, language translation, summarization, question answering, and enhancing search engines. In this project, they will be used both for search engines, but also for text generation and categorization. However, training these models requires significant computational resources and raises ethical concerns such as bias and environmental impact. Despite these challenges, LLMs have significantly advanced the field of NLP by enabling more accurate language understanding and generation.

## 2.5 Transformer (a deep learning model)

A transformer is a deep learning model and it was introduced by A. Vaswani et al. (2023) in the paper ”Attention is all you need” [12]. Unlike earlier models that processed data one piece at a time, Transformers look at entire sequences at once. This is a big change from the usual way of using neural networks for these tasks.

The key feature of Transformers is the ’self-attention mechanism.’ This lets the model evaluate and prioritize different parts of the input data. For each word in a sentence, the Transformer can focus on other relevant words in the same sentence to understand it better. This is different from older models, like Recurrent Neural Network (RNN), which read the data in order, like reading a book from start to finish.

Transformers have been crucial in creating LLMs and embedding models. In LLMs, they help the system understand and generate language by considering long-range connections between parts of the text. For example, a word at the beginning of a paragraph can influence the meaning of a word at the end. In embedding models, which are about representing words in a way that machines can understand, Transformers provide a richer sense of a word’s meaning by looking at the entire sentence or text, rather than just nearby words.

This architecture has set new standards in natural language processing tasks. Its ability to process parts of the data in parallel and understand the long-distance relationships between them makes it very effective for large-scale language tasks. Essentially, Transformers have become a cornerstone in the development of advanced language processing models.

## 2.6 Embedding model

An embedding model is a key component in the development of a semantic search tool, as it plays an important role in converting textual information into vectors that capture semantic relationships. In the context of this study, the embedding model is responsible for transforming text into a vector in a vector space, where the position and proximity of vectors correspond to the semantic similarity between texts. This allows us to compare similarities of the meaning of two texts, by looking at the distance between their vectors. The core idea of this is that words, phrases, and documents can be represented as vectors in a multidimensional space. Thus, the vectors are compared, rather than the words in a keyword search.

For this project, the embedding model is tasked with transforming legal judgments into vectors. These vectors then serve as the basis for the semantic search, enabling the tool to identify documents that are contextually and semantically relevant to the user’s query. The ability of the embedding model to understand and represent complex legal paragraphs in a multi-dimensional vector space is what empowers the search tool to deliver precise and contextually relevant results.

In summary, embedding models are the core of semantic search tools. And it is the model that determines where in the vector space to put a vector of a text.

### 2.6.1 Fine-tuning and training of different models

Most of the publicly available embedding models and large language models are trained on a large variety of text data. These data are taken from many different sources in many different fields. Although there might be some legal texts in the training data, they will only represent a small part of the training data. Therefore, the models will perform well in most of the fields, but the models are not specifically adapted to any field. By adapting the model specifically to a type of text, in this case legal texts, it should do a much greater job at tasks in that field.

Another point to notice is that there is no labeled data available in this domain. This forces us to use either unsupervised learning methods, or some form of semi-supervised learning, where we create labeled data, and then use that data for supervised learning methods.

Therefore, in this project, we will fine-tune not only an embedding model for the search tool, but we will also fine-tune a generative model to generate labeled training data. In fact, we will use an expensive model to create some data, to fine-tune an inexpensive model that will generate the rest of the data.

### 2.6.2 Sentence Bidirectional Encoder Representations from Transformers (SBERT)

SBERT, by N. Reimers et al. (2019) [10], or Sentence-BERT, is an innovative adaptation of the well-known BERT model (Bidirectional Encoder Representations from Transformers) designed specifically for generating sentence embeddings. The architecture of SBERT is shown in figure 1. BERT revolutionized the field of natural language processing (NLP) by providing deep contextual representations of words based on their usage in sentences, significantly improving performance across a wide range of NLP tasks. However, BERT's original architecture is not optimized for generating sentence-level embeddings or for comparing the semantic similarity of sentences efficiently. This limitation stems from BERT's requirement to consider sentence pairs jointly, leading to substantial computational overhead for tasks like semantic similarity search and clustering.

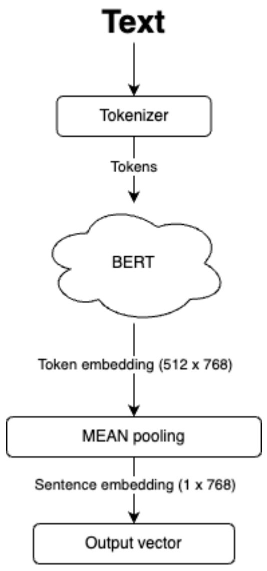

Figure 1: Diagram over how SBERT uses a BERT model when converting text to a vector.

The key innovation of SBERT lies in its adaptation of the BERT architecture into a Siamese and triplet network structure. This modification enables the generation of semantically meaningful sentence embeddings that can be easily compared using simple metrics like cosine similarity. [10]

SBERT utilizes Siamese and triplet network structures to process sentences in parallel, ensuring that the generated embeddings for semantically similar sentences are close in the vector space, and embeddings for dissimilar sentences are far apart. This is achieved through specialized training objectives tailored to sentence-pair similarity tasks.

**Pooling Strategies:** After processing sentences through BERT's transformer layers, SBERT applies a pooling operation to the output to derive fixed-sized sentence embeddings. Various pooling strategies are experimented with, such as using the output of the [CLS] token, averaging all output vectors (MEAN strategy), or taking the maximum value over time (MAX strategy). The MEAN strategy is typically used by default.

Mean pooling in the context of SBERT is a technique used to aggregate the information from all the token embeddings produced by BERT's layers into a single, fixed-size sentence embedding. After processing a sentence, BERT outputs a vector for each token representing its contextualized meaning. This vector has a size of $512 \times 768$. Mean pooling involves taking the average of these vectors across all tokens in the sentence. This operation condenses the sentence's information into one vector, of size $1 \times 768$, that captures its overall semantic essence. This method is particularly effective for SBERT as it ensures that the final sentence embeddings retain the rich contextual information encoded by BERT, making them useful for a variety of semantic tasks such as similarity comparison, classification, and clustering. Mean pooling is preferred for its simplicity and effectiveness in capturing the gist of the sentence without necessitating complex transformations or additional model components.

**Applications and Performance** SBERT significantly reduces the computational cost associated with semantic similarity tasks. Traditional BERT requires exhaustive pairwise comparisons, making tasks like finding the most similar sentence pairs in a large dataset computationally infeasible. SBERT, however, allows for pre-computing sentence embeddings and using efficient similarity search algorithms, thereby reducing the time required for such tasks from hours to seconds without sacrificing accuracy.

SBERT has demonstrated superior performance on various semantic textual similarity benchmarks and datasets. It outperforms other state-of-the-art sentence embedding methods, providing more accurate and computationally efficient solutions for tasks such as similarity search, clustering, and information retrieval.[10]

### 2.6.3 Training of SBERT

In the context of SBERT and its training process, Siamese networks are a cornerstone concept that enables the efficient generation of semantically meaningful sentence embeddings. Siamese networks consist of multiple identical subnetworks, meaning they have the same configuration with the exact same parameters and weights. Input to these networks can vary, but the goal is to learn from the comparison between these inputs.

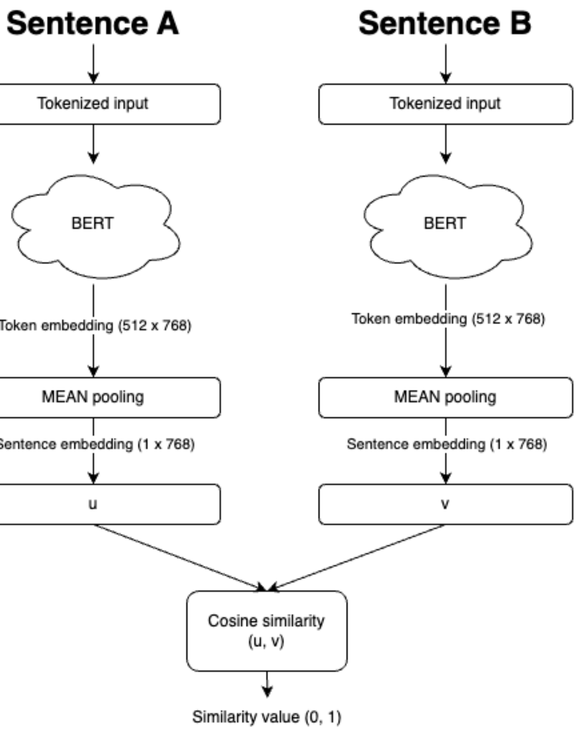

Figure 2: Diagram of how a siamese network can be used to compare two sentences.

For SBERT, Siamese networks are utilized to process pairs of sentences through the same BERT architecture to produce embeddings. See figure 2 for diagram. These embeddings are then compared to understand the semantic similarity between the sentences. The training process involves adjusting the weights of the network so that semantically similar sentences result in closer embeddings in the vector space, and dissimilar sentences result in embeddings that are farther apart.

This architecture is particularly effective for tasks involving sentence similarity, paraphrase identification, and other semantic comparisons. It allows for the pre-computation of sentence embeddings for a large dataset, making it computationally efficient to find the most similar sentences within the dataset by simply computing the cosine similarity between embeddings, rather than having to rerun comparisons through the entire model for each query.

### 2.6.4 KBLab

Holmer et al. (2022) [1] concluded that KBLabs SBERT model had best overall performance on Swedish text. The "kblab/sentence-bert-swedish-cased"[1] model, developed by KBLab, is an adaptation of the Sentence-BERT (SBERT) model specifically tailored for the Swedish language. The model features a maximum sequence length of 384 tokens, and embeds the text to a vector of 768 dimensions.

This model has been developed with a focus on addressing the challenges inherent in non-English language processing, particularly regarding the availability and quality of training data. To overcome these challenges, different methods were explored, including machine translation of English datasets to Swedish and self-supervised training on Swedish Wikipedia data. However, these methods had limitations, such as translation errors or suboptimal performance compared to English models. [6]

[1]https://huggingface.co/KBLab/sentence-bert-swedish-cased

However, this model is the most downloaded model on Huggingface[2] for Swedish text.

### 2.6.5 Vector representation of text

In the vector space, each word, sentence, or document is represented as a high-dimensional vector. For sentence-BERT, the dimension of the vector space that BERT maps the vectors to is 768. This representation is learned through the training process, where the model refines its parameters to optimize the encoding of semantic information. Words or documents with similar meanings are mapped close to each other in the vector space, facilitating the identification of semantic relationships during the search process.

## 2.7 Other models

### 2.7.1 Cross-encoder

Cross-encoders represent a significant evolution in the field of natural language processing, particularly in tasks requiring deep understanding and comparison of text pairs. These models, which build upon the foundation of transformer architectures like BERT, are designed to simultaneously analyze pairs of texts (such as a query and a document), enabling a more nuanced and detailed understanding of their relationship.

The key characteristic of a cross-encoder is its ability to take two separate pieces of text as input and process them jointly. This joint processing allows the model to consider the interplay and dependencies between the two texts, leading to a more accurate assessment of their semantic similarity or relevance. Unlike models that process texts independently (such as SBERT), cross-encoders can capture subtleties in how the meanings of two texts relate to each other. [3]

In the context of semantic search, particularly for legal documents, cross-encoders can play a pivotal role. They can be used to evaluate the relevance of a document (or a section of it) to a specific query. For instance, when a lawyer searches for precedent cases using a brief description, a cross-encoder can assess how closely each case in the database matches the query, taking into account the complex legal terminology and context.

However, the use of cross-encoders comes with certain challenges. They are generally more computationally intensive than independent encoding models, as they require processing each query-document pair separately. This can lead to longer processing times, particularly when dealing with large legal databases. Balancing the need for deep semantic analysis with operational efficiency is a key consideration in employing cross-encoders in a semantic search tool [3]. Also, at this moment in time, there is no cross-encoder publicly available that is trained on Swedish text.

Due to that, the cross-encoder is very computationally expensive, so it's not appropriate for use in this semantic search tool. Instead, it can be used during the training process of SBERT, since it can provide valuable data for the training by labeling the similarity between a query and a document. This is useful as labeled data can make the training more efficient.

### 2.7.2 Zero-shot classification model

Zero-shot classification models are particularly good in their ability to understand and categorize text based on previously unseen labels or classes. This adaptability to unseen classes is useful when labeled data is unavailable och in dynamic fields.[15]

The essence of zero-shot classification lies in its ability to make predictions for categories or classes that were not present in the training data. This is achieved by leveraging deep learning

[2]https://huggingface.co/models?pipeline_tag=sentence-similaritylanguage=svsort=trending

and language models trained on a diverse range of texts and topics. These models develop an understanding of language and context that can be generalized to new, unseen categories.

In zero-shot classification, the model is typically provided with a description of the category or label and then uses this description to classify new instances. The model relies on its learned semantic understanding to infer the most likely category for each new instance, even if it has never seen an example of that specific category during training.

However, zero-shot classification models are not without challenges. Their predictions might be less accurate than those of models trained on specific categories, especially when dealing with highly specialized or nuanced legal terms. The quality of the model's predictions depends heavily on the breadth and depth of its training data and its ability to generalize from that training.

Zero-shot classifiers take two text strings as input and generate a similarity score as output. This is in the same form as the cross-encoder.

### 2.7.3 Generative language model

A generative LLM (large language model) is a model that generates text based on an input of text. When an LLM generates text, it considers the probability of the next set of tokens, i.e. the next words. The probabilities are set by the training data on which the model is trained on, as well as the fine-tuning, prompts and hyperparameters.

Choosing the next word based on their probability can sometimes make the model "hallucinate", which means that the answer it gives is not the correct fact, but just that the sequence of words is reasonable according to its training data.

The temperature of a model determines the randomness of the answers. The temperature is set by the user and is a number between 0 and 1. A higher temperature is more random and gives a more creative answer, whilst a lower temperature is more deterministic and is more suitable for factual answers. A generative model can be fine-tuned to generate text in a certain format, which will be used in this project.

### 2.7.4 OpenAI's Generative Pre-trained Transformer (GPT)

OpenAI's GPT3.5 model is a generative model that was seen to outperform the other models in OpenAI (2020) [8], until GPT4 was released [7]. However, direct use of the GPT3.5 model can be insufficient in more specific use cases and might need to be fine-tuned to perform well [5]. At the time of this thesis, GPT4 seems to be the model that outperforms all others. However, the cost of using GPT4 (during the time of this thesis) is $\$0.01/1Ktokens$ for the input and $\$0.03/1Ktokens$ for the output[3]. The way OpenAI's models are used in this project will be a lot of tokens as input, but very few tokens as output since we will generate shorter queries to longer text. Therefore, for the rest of the cost analysis, the output price will be neglected and all tokens (input and output) will be counted for the input price. A fine-tuned GPT3.5 is $\$0.003/1Ktokens$, and the fine-tuning process is $\$0.008/1Ktokens$. Therefore, for use cases such as this, where we will generate a lot of data, a fine-tuned GPT3.5 will be cheaper than GPT4, while it still might provide good results.

GPT4 has a context window of 128,000 tokens, while GPT3.5 can handle 16,385 tokens[4]. 16,385 tokens are not enough for some of the documents if we want to provide two examples for every call. Therefore, GPT4 can be used to produce the training data for fine-tuning the GPT3.5 model. About 50-100 sets of data are suggested for the fine-tuning according to OpenAI[5]. After the fine-tuning, we do not need to provide examples anymore, so then 16,385 tokens will be enough.

[3]https://openai.com/pricing

[4]https://platform.openai.com/docs/models/overview

[5]https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset

## 2.8 Vector store

Instead of storing and querying from a traditional database, we use a vector store. A vector store is a kind of database specifically designed for storing and querying vectors efficiently. The vector store used for this project is Milvus. The vector store maps the vectors into a vector space of a dimension determined by the embedding model, in this case 768 dimensions for the SBERT model.

A vector store, such as Milvus, is engineered to handle the storage, indexing, and querying of high-dimensional vector data. Unlike traditional databases that are optimized for scalar data types (e.g., integers, strings), vector stores are optimized for vector operations, making them ideal for tasks involving machine learning models and embeddings.

**Key Features of Vector Stores:**

- Efficient Storage: Vector stores utilize storage mechanisms that are optimized for high-dimensional data. This allows for compact storage of vectors, reducing the amount of space required and improving I/O operations.
- Indexing and Retrieval: To facilitate fast retrieval, vector stores implement various indexing strategies such as KD-trees, HNSW (Hierarchical Navigable Small World graphs), and IVF (Inverted File Index). These indices help in reducing the search space for query vectors, significantly improving query performance.
- Scalability: Most vector stores are designed to scale horizontally, supporting distributed systems architecture. This scalability is crucial for handling large datasets and complex queries in production environments.

### 2.8.1 The vector space

All the available text data will be converted to vectors by the embedding model, and inserted in the vector store. The vector store will arrange the vectors in the vector space. The vectors are then grouped into clusters. An illustration of how the vectors can be mapped is shown in figure 3. This illustration is in 2 dimensions, while the real case is in 768 dimensions, but the illustration shows how the search query finds the closest document as a result of the search.

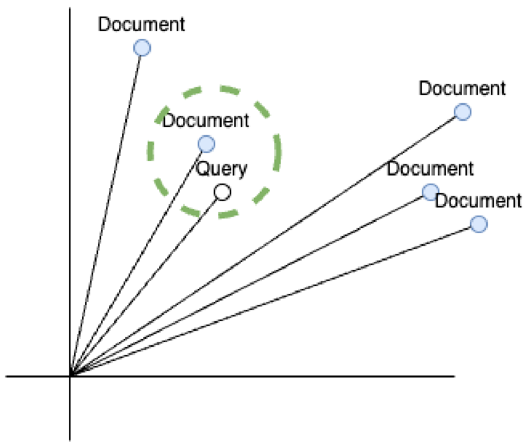

Figure 3: Illustration in 2D of how the vector space can look like and that the document closest to the query will be the result of the search

### 2.8.2 Parameters

Two parameters that is set for the vector store is *nlist* and *nprobe*. How these are set is a trade-off between performance and accuracy. *nlist* determines how many clusters that will be available for the vectors in the vector store. *nprobe* is a parameter that decides how many of the closest clusters that will be searched through when searching with a query. For example, if *nlist* is 1024, all the document vectors will be categorized in one of those 1024 clusters. With $nprobe = 32$, for example, the 32 closest clusters will be searched through when searching with a query.

## 2.9 Data set

The data set consists of 57000 judgment documents. The part ”domskäl” of a Swedish judgment document contains the court’s description and motivation of the case. This is the part of the document that is relevant for a semantic search, since the users want to search for cases. The ’domskäl’ is the text part that is extracted from the PDFs and put in a JSONL-file for this project.

## 2.10 Training and fine-tuning of embedding model

The training phase of the semantic search tool involves preparing the embedding model to effectively capture semantic relationships within legal texts. This section outlines the key components of the training process, including unsupervised learning, the Transformer-based Denoising AutoEncoder (TSDAE), Generative pseudo labeling (GPL) and Generative query network (GenQ).

### 2.10.1 Fine-tuning

Pretraining a model will teach it the language, the words and the structure of the sentences of the training data. However, the model can be further fine-tuned for a specific task. In this case, that specific task is to compare the similarity between a query and a document. The model can therefore be fine-tuned, but labeled data is needed. To fine-tune a model for semantic search, one approach would be to have a query, together with a matching document, and possibly a document that does not match, to provide a positive and a negative example. Figure 4 illustrates how the query vector gets closer to the positive query and further from the negative document during training.

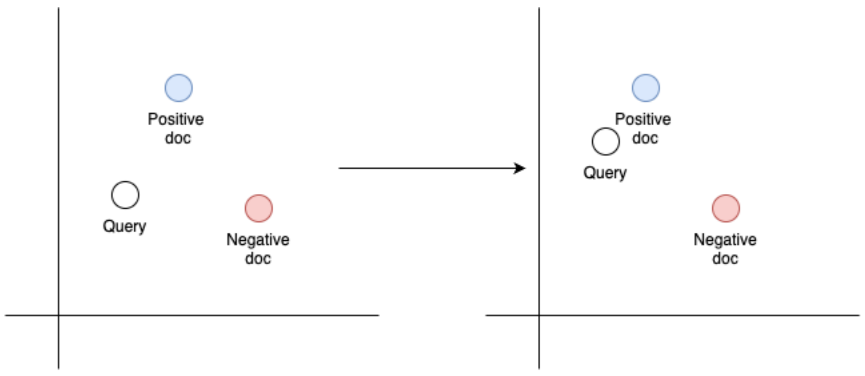

Figure 4: illustration in 2D of how the arrangement of the vectors can change when the model is trained. The query and the positive document will be closer to each other after the training, while the negative document will be further away.

### 2.10.2 Unsupervised learning

Since our data consists of PDF documents with unstructured data, we have a lot of unlabeled data. To train on unlabeled data, we need to rely on unsupervised learning methods for the larger parts of the training. One method that can be used here is TSDAE (Transformer-based Denoising AutoEncoder).

### 2.10.3 Semi-supervised training

The data can be labeled manually. In this case that would be that someone manually writes a search query to some of the documents. This takes time and requires people with knowledge of the field, so we will only have a limited set of labeled data. However, with this (small amount of) labeled data, we can generate more by the usage of a generative large language model like GPT from OpenAI.

**Prompt engineering** Prompt engineering is a simple way to use a generative model by providing examples as well as describing what to do. By providing all our examples of documents with queries, we can teach the generative model to provide data in the same way. This way, the generative model will write example queries to the texts we provide.

However, this can be expensive due to the large amount of input tokens needed. A cheaper way is to fine-tune a generative model. Prompt engineering will be used for the queries generated by GPT4, that will be used to fine-tune GPT3.5.

**Supervised learning** With enough labeled data, we can use supervised methods to further fine-tune the embedding model.

### 2.10.4 Transformer-based Denoising AutoEncoder (TSDAE)

Tranformer-based Denoising AutoEncoder (TSDAE) is a state-of-the-art unsupervised method for pretraining of, for example, embedding models. TSDAE is an innovative approach in the field of unsupervised sentence embedding learning, introduced by Kexin Wang, Nils Reimers, and Iryna Gurevych from the Ubiquitous Knowledge Processing Lab at the Technical University of Darmstadt [13]. This method is significant as it addresses the challenge of learning sentence embeddings often requiring extensive labeled data, which is usually expensive and scarce for most tasks and domains.

TSDAE's success is particularly noteworthy in the field of unsupervised learning. Its ability to learn effective sentence embeddings without relying on labeled data makes it a valuable tool, especially considering the high costs and limited availability of labeled datasets in many domains.

**Overview** TSDAE is a state-of-the-art method that utilizes pre-trained Transformers in combination with a Sequential Denoising Auto-Encoder. This approach significantly surpasses previous methods, achieving up to 93.1% of the performance of in-domain supervised approaches. TSDAE has been proven effective as both a domain adaptation and pre-training method for sentence embeddings, outperforming other approaches like the Masked Language Model. [13]

**How TSDAE works** The TSDAE approach involves an encoder-decoder architecture. During training, it encodes 'corrupted' sentences into fixed-sized vectors. These corrupted sentences are generated by adding specific types of noise, such as deleting or swapping words, to the input sentences. The decoder is then tasked with reconstructing the original sentences from these sentence embeddings. For effective reconstruction, the sentence embeddings generated by the encoder must capture the semantics well. It's important to note that, in contrast to the original transformer encoder-decoder setup [12], the TSDAE decoder only decodes from a fixed-size sentence representation produced by the encoder, and does not have access to all contextualized word embeddings from the encoder. This change is significant because it creates a sort of "bottleneck" in the process. So, the need to be selective increases. This forces the encoder to be very efficient and smart about how it represents the sentence. It has to capture the essential meaning and nuances of the entire sentence in a much smaller, more condensed form.

In the article about TSDAE, K. Wang et al (2021) [13] made an analysis showing that no more than 10000 sentences are needed for training in their 4 different cases. They also investigated the optimal configuration of TSDAE. They found that the best noise type is deletion of tokens. Additionally, with deletion as noise, they found that a noise ratio of 60% is the optimal ratio. They also found that CLS pooling was the optimal pooling method for TSDAE.

DenoisingAutoEncoderDataset adds noise to the sentences in the form of deleting random tokens from a sentence [9]. By default, it deletes 60% of the words in each sentence.

The loss function used for TSDAE is called DenoisingAutoEncoderLoss and compares the reconstructed sentence with the original one.

#### 2.10.5 Generative pseudo labeling (GPL)

K. Wang et al. (2022) [14] describe and compare GPL (generative pseudo labeling) to the other top-performing methods for unsupervised training of LLMs. It turns out that GPL outperforms all of the others, and training a model with both TSDAE and GPL gives the best results. These training methods also do not require much data compared to other methods.

Some (theoretical) advantages of GPL include [14]:

1. Enriched Training Data: GPL addresses the challenge of scarce or expensive labeled data, especially relevant in the context of legal documents where manual labeling is impractical due to the volume and complexity of the data.

2. Enhanced Model Precision: By incorporating both positive and negative examples, along with similarity scores, GPL trains models to discern fine-grained differences in text, leading to more precise semantic search results.

3. Efficiency in Data Utilization: GPL maximizes the utility of available data, transforming unlabeled datasets into a valuable resource for model training, thereby enhancing the efficiency of the learning process.

There are three main steps in GPL to prepare the data for the training of the model:

- Query generation
- Negative mining
- Pseudo labeling

After these three steps, we will have a large amount of labeled data to train the model with.

In the first and the last step, we need to use other LLMs. The LLMs needed are one *query generation model* (commonly used a T5 model), and one *cross-encoder*. However, at this point in time (the time of this project), there are not any publicly available query generators or cross-encoders. Therefore, in this project, we step out a bit from the GPL method, but the general outline is followed.

**Query generation** The first step in GPL is to generate queries for the documents' unlabeled text data, which in this project is the 'domskäl' from the judgments. K.Wang et al. (2022) investigated the optimal number of queries per document, and found that the optimal number of queries increases as the size of the data set decreases. Around 5 queries per document were good for a dataset of 57k documents. After generating queries for each document the data will consist of query-document pairs $[(Q, D), (Q, D), ...]$, where at this point there are multiple of each document.

The queries can be generated with a generative model. There are some parameters to consider when generating queries. The temperature of a generative model decides how "creative" results the model will give. K.Wang et al. (2022) found that the queries were most useful at a temperature of 0.1.

**Negative mining** When we have a few queries per document, the next step is negative mining. The queries that are generated are considered a good match to the document. Negative mining is collecting, for example, 10 more documents for each query. These 10 documents are found when semantically searching with the query, so they can contain information that is similar to what the query is asking for, but maybe not relevant. The data now consists of a query, a positive document (same as before), and a negative document (newly mined) $[(Q, D+, D-), (Q, D+, D-), ...]$.

By having documents that are similar but not necessarily the best matches, the model can learn very nuanced differences.

**Pseudo labeling** In this step, a cross-encoder should be used to determine the similarity between the queries and their documents. A cross-encoder is most often used because, similar to a zero-shot model, they are good at handling unseen data and new categories. This will in the end give fully labeled data, with a query, a document, and a similarity score between them, which will be beneficial for training the model. This pseudo-labeling step is what makes GPL perform better than other methods. Without it, it would be assumed that the negative documents would be irrelevant, and the positive would be the optimal. Now, we are not as limited, because the positive document might not be optimal since a bad query could have been generated in the first step. So, this step makes the data more nuanced and fault-proof. Now the data consists of the query, positive and negative document, and a label $[(Q, D+, D-, label), (Q, D+, D-, label), ...]$.

**MarginMSE loss** The training process is now a standard training process for bi-encoders, where we use the margin MSE (mean square error) loss:

$$L(\theta) = -\frac{1}{M}\sum_{i=0}^{M-1} |\hat{\delta}_i - \delta_i|^2, \tag{3}$$

where $\hat{\delta}_i$ is the predicted score by the model we're training, and $\delta_i$ is the score labeled by the cross-encoder.

Generative Pseudo Labeling stands out as a highly effective method for training semantic search models, especially in domains like legal research where the availability of labeled data is limited, and the need for precision is paramount. By employing GPL, we can significantly advance the capabilities of semantic search tools, making them more efficient, accurate, and tailored to the specific needs of legal professionals.

### 2.10.6 Generative query network (GenQ)

GenQ (Generative query network) is a training method that has some similarities with GPL. The method description can be easily followed in the diagram in figure 5. For genQ, we are generating Queries, similar to for GPL. Each query is paired with the related document. This makes up the dataset. Thereafter, Multiple Negatives Ranking Loss (MNRL) is used to organize the data. This step is described in detail below. The SBERT model is fine-tuned with the loss function Multiple Negatives Ranking Loss (MNRL).

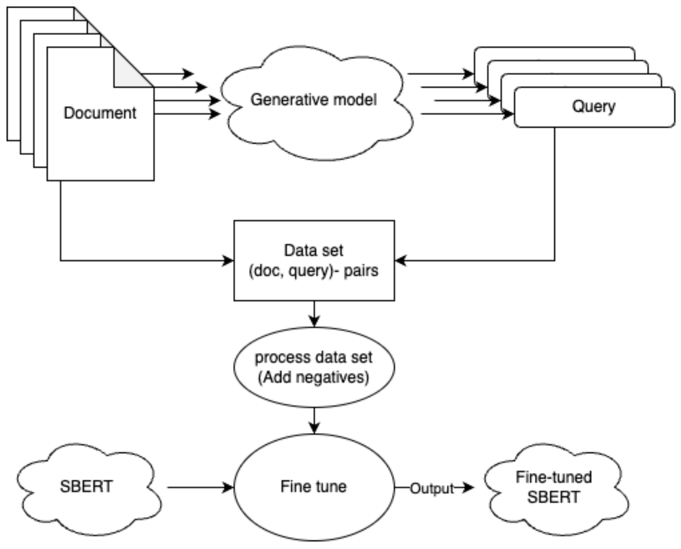

Figure 5: Diagram of the structure of the genQ training process.

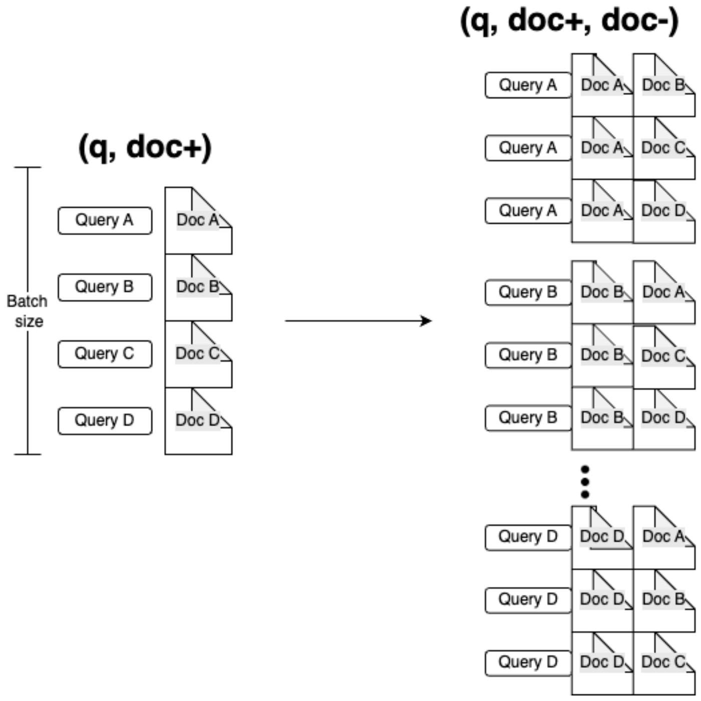

Figure 6: Example of how data is arranged when using Multiple Negative Ranking Loss

**MultipleNegativesRankingLoss** Multiple Negatives Ranking Loss (MNRL) is a loss function, used to train sentence transformers models to create a clearer distinction between positive and negative documents for a query. The purpose is to maximize the similarity between the query and the positive document, as well as minimize the similarity between the query and the negative document.

The loss function is given by

$$Loss = -\frac{1}{K}\sum_{i=1}^{K}[S(q,p_i) - log\sum_{j=1}^{K}e^{S(q,n_j)}] \tag{4}$$

where K is the number of query-document pairs, $q$ is a query, $p$ is a positive document and $n$ is a document considered negative. Both $p$ and $n$ are documents in the batch, but they have separate variables for simplicity. $S$ is a similarity function. For all query-document $(q_i, p_i)$ pairs in each batch, the related document is the positive, and all other documents in the batch are considered negative for that query. That is $n_j$ where $j \neq i$. See figure 6 for an illustration. [2]

The model will be trained in batches. For a batch size $B$, we will have $B - 1$ negative document and 1 positive document for every query. An increased batch size usually leads to an increased performance [2]. However, a larger batch size also requires more resources and time.

## 2.11 Techniques for processing data

Since this project involves large amounts of data, we need to utilize some techniques to be able to handle the data as efficiently as possible, on as little hardware as possible. Since we're working on cloud instances, we can choose the Random Access Memory (RAM) and processors depending on needs, but it comes with a price. By making the data processing more efficient, we can use cheaper resources and keep the costs down.

### 2.11.1 Batch processing

It requires lots of resources to process big arrays with lots of data at once. By not processing all data simultaneously, and instead processing the data in smaller batches, we will consume less RAM.

### 2.11.2 Streaming

Another way to keep the resources down is by streaming the data. This is done by, for example, looking at one line at a time while reading jsonl files.

# 3 Methods and implementation

## 3.1 Training of the models

The implementation of all steps was made on a cloud instance in Google Cloud Platform with 16GB RAM and 100GB disk memory. The application and vector store is run using docker containers. An overview of the training process is shown below in fig 7.

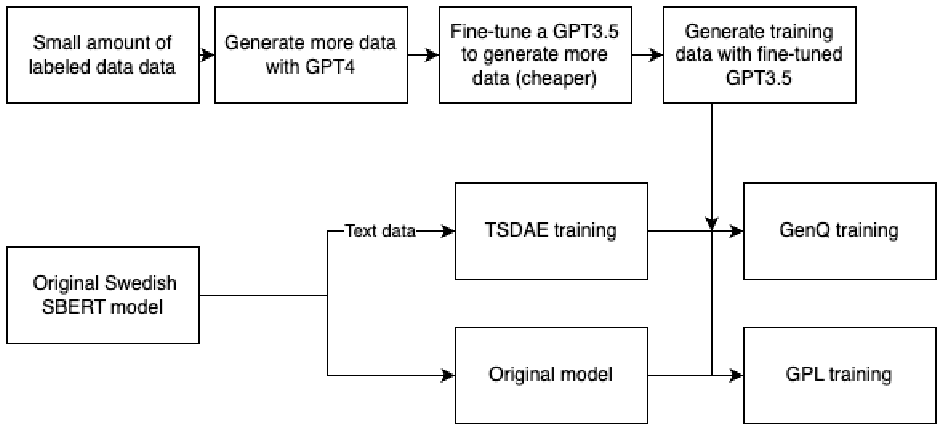
Small amount of labeled data data
Generate more data with GPT4
Fine-tune a GPT3.5 to generate more data (cheaper)
Generate training data with fine-tuned GPT3.5
Text data
TSDAE training
GenQ training
Original Swedish SBERT model
Original model
GPL training

Figure 7: overview of the training method.

### 3.1.1 Data

The text from the 57000 judgment documents was extracted and the part 'domskäl' was inserted together with the file name of the PDF into a JSONL-file. Some of the available PDFs did not contain a recognizable 'domskäl'. Those were not used. Also, some of the 'domskäl' are short and do not provide any relevant info. These are not useful and were sorted out. So, we ended up with 54000 documents as our data to develop this semantic search tool. And this amount of data is more than enough for the purpose of developing this application.

### 3.1.2 Sentences for unsupervised training

For training with TSDAE, 10 000 random sentences were extracted from the documents. Furthermore, 1500 other sentences were also extracted for evaluation. These 1500 sentences were copied and the copies were perturbed by deleting 60% of the words. So, we now have 1500 sentence pairs, where one of the pairs is the original sentence, and the other is the same sentence but with 60% of the words deleted.

The same approach was also done where we extracted 10 000 text chunks of 1-5 sentences each, and 1500 text chunks for evaluation.

### 3.1.3 Generation of labeled training data with GPT-models

13 of the 54000 judgment documents were read manually by a lawyer, who manually wrote 2 search queries that could be relevant for each of these 13 documents. As this is time-consuming, we would never get enough labeled data manually. However, with these labeled data, we can generate more data of similar characteristics by using a generative model. For this, OpenAI's ChatGPT was used via their API. Since GPT4 generates more qualified results and can take more tokens as input than the other models available, this would be the best choice. However, GPT4 is 10 times more expensive than a fine-tuned GPT3.5, so a more economical choice, which still gives good data, is to fine-tune a GPT3.5-model. By fine-tuning a model, we will not only

be able to use a cheaper model, We will also not need to send examples as input every time as the fine-tuned model will give output according to how it's tuned.

Generating all data with only GPT3.5 would not be possible because the context window can be too short for providing examples and a new document.

### 3.1.4 Generation of data by GPT4

To generate data for fine-tuning of a GPT3.5-model, we used GPT4. As input messages to the model, as shown in figure 8, we described the system as well as 2 random documents with its 2 queries out of the 13 documents with queries as examples of how the model will generate the output. Together with these 2 examples, we gave a new document, for which the model should give 2 queries. This was done to generate 2 queries each to 100 documents. The cost for this step is shown in table 3 in the results section.

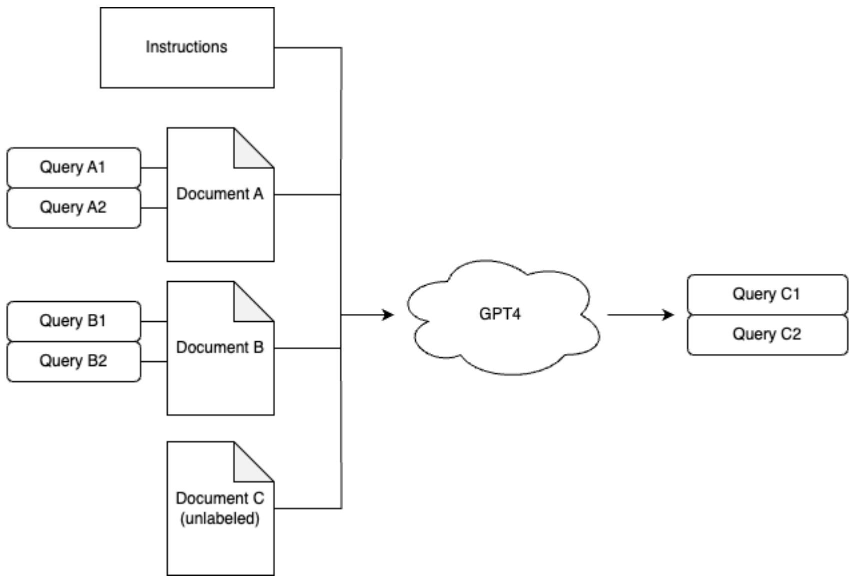

Figure 8: Diagram of the process of generating queries with GPT4.

### 3.1.5 Fine-tuning of GPT3.5

The 100 documents with 2 queries each were split so that we got 90 sets of training data and 10 sets of validation data. The 90 sets of training data were used together with the validation data to fine-tune the GPT3.5 model according to OpenAIs' guide for fine-tuning their models[6]. The cost of this step is shown in table 3.

### 3.1.6 Generation of full training data with a fine-tuned GPT3.5

Now with the fine-tuned model, it knows how to structure the output as well as the tone and the way of writing relevant queries. 10k random documents was drawn from the data, and sent as input to the model, which generated 2 queries for each document, as illustrated in figure 9. The cost of this step is shown in table 3.

[6]https://platform.openai.com/docs/guides/fine-tuning

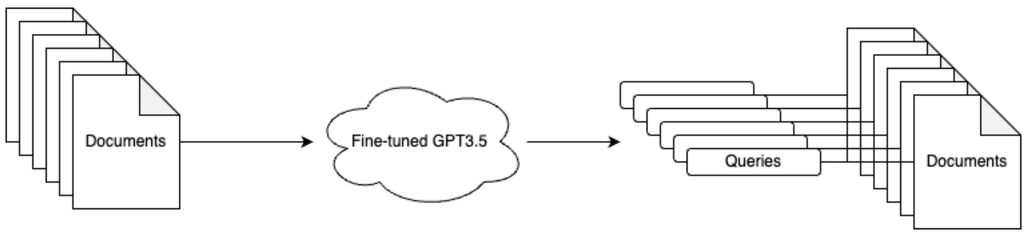

Figure 9: Diagram of how the fine-tuned GPT3.5 now can generate queries, with just the documents as input.

### 3.1.7 TSDAE training

The original model from KBlab model was trained with TSDAE using 10 000 sentences for 1 epoch, with the learning rate set to $3e^{-5}$ and a batch size of 8, as suggested from the inventors. $DenoisingAutoEncoderLoss$was used as loss function, and the $fit()$-function provided by the sentence-transformers library was used to perform the training. The 1500 sentence-pairs were then used for evaluation as described in section 3.3.1. The same was done for the text chunks.

### 3.1.8 Generative pseudo labeling (GPL)

We have now generated 2 queries each for 10k documents. That means that we have 20k query-document pairs.

**Negative mining** All the documents were first embedded (by the original model) into vectors and inserted in the vector store. Then, for each query, the 5 most semantically similar (according to the original model) documents from the vector store were taken.

**Pseudo labeling** Instead of a cross-encoder which is proposed with the method, a zero-shot classification model was used to label the similarity between the documents and the query. The model MoritzLaurer/mDeBERTa-v3-base-mnli-xnli[7] was used to label the query-document pairs with a similarity value.

The model was then trained on the data using the $fit()$ method from the sentence-transformers package and the margin MSE loss function.

### 3.1.9 GenQ

The same 20k query-document pairs were used. With the Multiple Negatives Ranking Loss function, described in the theory section, the model was trained using the $fit()$-function.

## 3.2 Application

A web application was made to demonstrate the search tool. Below, in figure 10, is an overview of the application. The application is opened in the browser by the user. The front end is run via Flask. When a query is made by the user, the query will be embedded into a vector by the trained embedding model. From the Vector store, the 100 closest document vectors will be returned to the user via the browser.

Already in the vector store are the documents. The text was first extracted from the PDFs before the text in the document was embedded into vectors and inserted into the vector store.

[7]https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-mnli-xnli

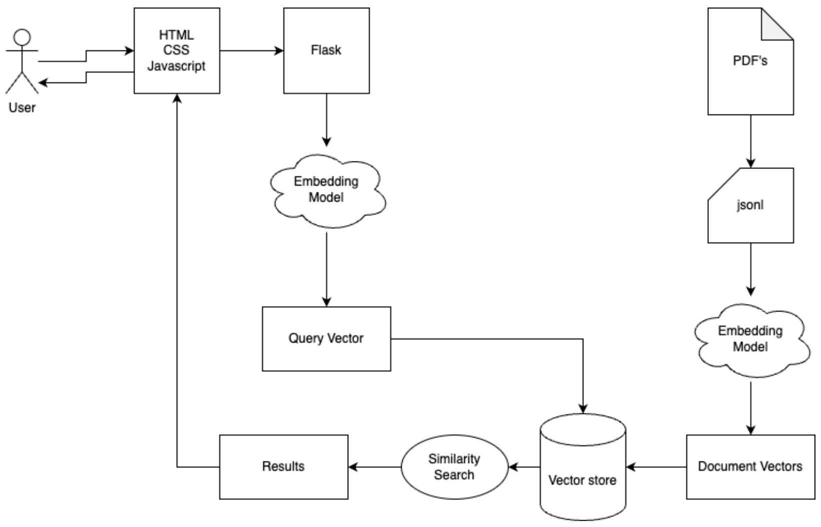

Figure 10: Diagram of the application

### 3.2.1 Web interface

The front end was made with HTML and CSS, with JavaScript to handle the dynamics and results. Flask is used to connect the semantic search application with the web interface.

### 3.2.2 Vector store

Milvus was used to store the vectors because it's easy to use with good documentation. It is also fast in retrieving information[8] and has some filtering options (which will not be used in this project, but is useful for further development of the application). Also, it's open source, but it also allows for a managed option if that would be preferred in the end.

A collection was created to store an ID, the name of the document, and the embedding vector. The distance metric was set to Cosine distance to not let the size of the vectors affect the distance. The size will be off since the query is much smaller than the data.

The index type was set to IVF_FLAT, which divides the stored data into *nlist* clusters. *nlist* were set to 1024. By using IVF_FLAT, the distance between the search query vector and the center of each cluster will be compared. Only the vectors in the *nprobe* most similar clusters will be returned, which reduces the query time[9] .*nprobe* was set to 32, making each query search the 32 most similar clusters.

## 3.3 Evaluation

The optimal way of evaluating this product would be by feedback from users. However, there are not resources available for that at this moment. Instead, we have used a few other methods for evaluation, both for the model separately and for usage of the complete application.

### 3.3.1 Understanding of pertubated sentences

To evaluate the model after training with TSDAE, and to determine the optimal training parameters, we use 1500 random sentence pairs from the data (not included in the training data). The sentence pair consists of an original sentence and the same sentence perturbed by the same methods as in the training. We let the model embed the two sentences in the

[8]https://milvus.io/
[9]https://milvus.io/docs/index.md

pair. Then we check the cosine similarity between the two sentences. The same is done for all sentences in the validation set, and the results will be compared. To compare the trained model to the original one, we check if the similarities between the two sentences in a pair have increased. With increased similarity, it means that the model has gotten a better understanding of the specific way of formulating the sentences.

### 3.3.2 Backwards search

With 500 randomly selected documents, we are generating two queries for each (but only using one of them for this part). So, these queries will be a query where we expect to get the related document in the search results. However, there might be other documents in the database that are better or just as good match for that query. Therefore, the expected documents might not be in the top result. But this is a good approach for comparing different models if we use many results.

Out of the 500 query-document pairs, we will look at the amount of searches where the matched document appears in top 100, top 200 and top 250.

### 3.3.3 User feedback

The ideal evaluation would be to get feedback from users. However, this is not possible in this project due to limited resources.

## 4 Results and discussion

### 4.1 TSDAE training

#### 4.1.1 Sentences

After training the original model with TSDAE on 10000 sentences, it became better at understanding the sentences in this kind of text. When computing the cosine distance between the vector of the original and the reconstructed sentence, the trained model performed better. The results are visualized in figure 11, where it's seen that more of the sentence pairs got a higher similarity after the training. And fewer sentence pairs got a low similarity after the training. The values are presented in table 1, where it's seen that after training, the mean and median of the similarity scores increase while the standard deviation decreases after training.

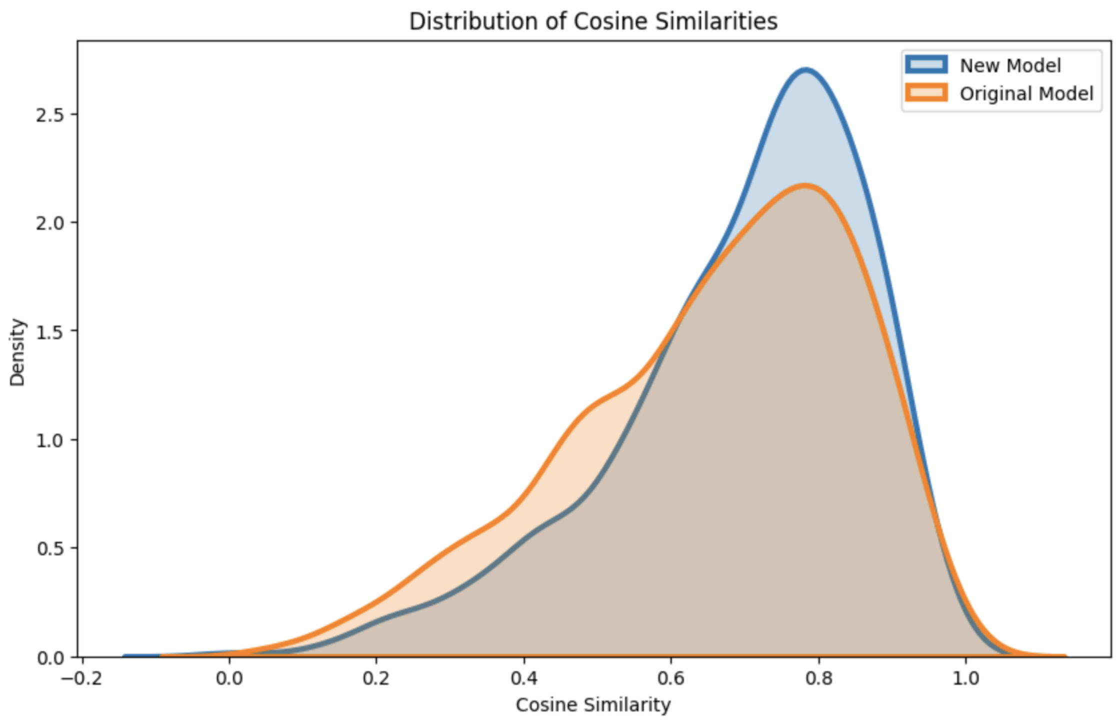

Figure 11: Cosine similarity scores between perturbed and the original sentences, for the original model compared to after the model was trained with TSDAE.

Table 1: Statistics of the cosine similarity based on 1500 reconstructed sentences compared to the original sentences. 1 is the highest similarity, 0 is the lowest.

| Statistic | Before TSDAE | After TSDAE |
|---|---|---|
| Mean | 0.6571 | 0.6952 |
| Standard Deviation | 0.1928 | 0.1730 |
| Median | 0.6906 | 0.7322 |

#### 4.1.2 Textchunks

Similar as for the sentences, the model improved after the training on the 10000 text chunks, as shown in figure 12 and in table 2. The similarity scores improved when the model computed the distance between the perturbed text chunks and the original text chunks.

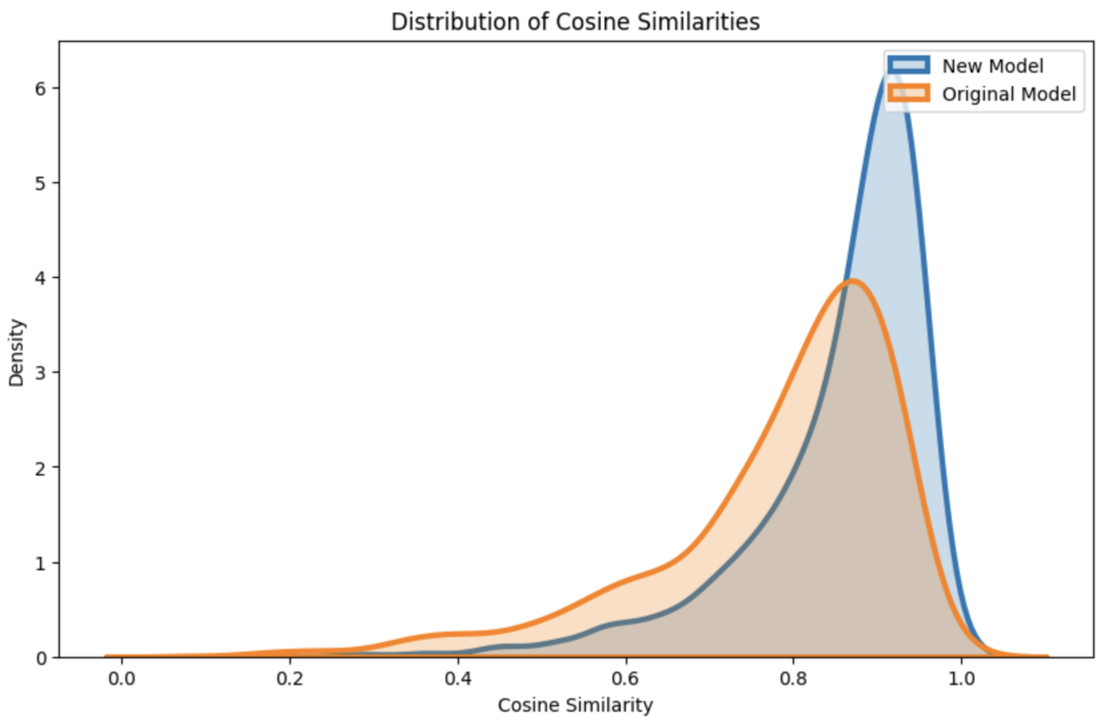

Figure 12: Cosine similarity scores between perturbed and the original text chunks, for the original model compared to after the model was trained with TSDAE on text chunks.

Table 2: Statistics of the cosine similarity based on 1500 reconstructed text chunks compared to the original sentences. 1 is the highest similarity, 0 is the lowest.

| **Statistic** | **Before TSDAE** | **After TSDAE** |
|---|---|---|
| Mean | 0.7821 | 0.8547 |
| Standard Deviation | 0.1460 | 0.1069 |
| Median | 0.8242 | 0.8881 |

## 4.2 Fine-tuning of GPT3.5

The GPT-3.5 model was fine-tuned with 100 documents, using queries made by GPT-4, to improve its query generation for Swedish legal judgments. This fine-tuning enabled the model to produce relevant queries for an additional 10,000 documents, demonstrating significant enhancements in generating contextually appropriate queries for the data for the semantic search tool.

During the fine-tuning process, which lasted for 3 epochs and 90 steps each, we monitored training loss and accuracy as key performance indicators. Initially, the training loss was high, indicating the model's unfamiliarity with the task-specific data, but it generally decreased over time, despite showing significant fluctuations which suggests variability in the training process and data. On the other hand, training accuracy began high and remained above 80%, reflecting the model's capability to understand the underlying patterns in the data, despite occasional volatility.

By the end of the training, the model achieved a training loss of 0.20 and an accuracy of 91%, indicating effective learning despite the observed fluctuations in training metrics.

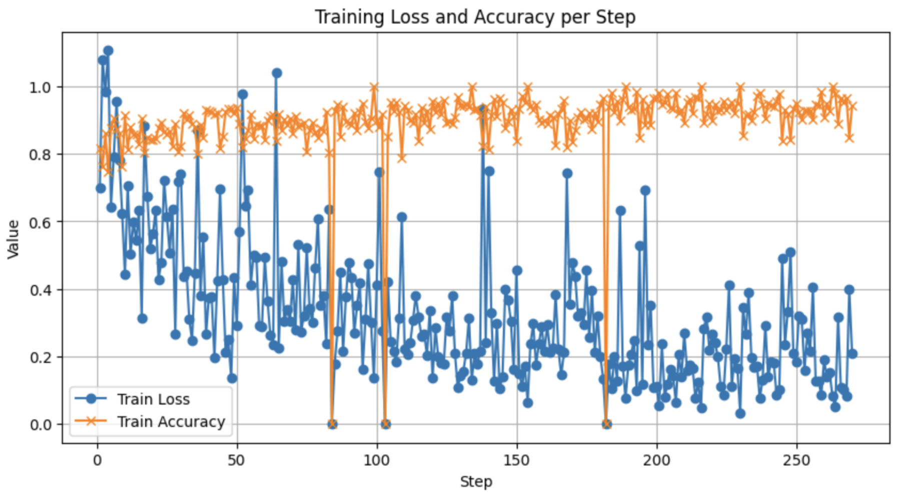

Figure 13: Training loss and accuracy while fine-tuning GPT3.5

### 4.2.1 Cost

The cost for generating all labeled data is shown below in table 3. The total cost for using openAI's API was $56.6. This approach of fine-tuning a GPT3.5 model was $632 less than the calculated cost of generating 10k queries directly with GPT4.

Table 3: costs for the use of OpenAI's API for this project.

| **Task** | **Model** | **$/1k tokens** | **tokens** | **total cost** |
|---|---|---|---|---|
| Queries for 100 documents | GPT4 | 0.01 | 690431 | $6.9 |
| fine-tuning job | GPT3.5 | 0.008 | 425667 | $3.4 |
| 10k queries for training | fine-tuned GPT3.5 | 0.003 | 14624299 | $43.9 |
| 500 queries for evaluation | fine-tuned GPT3.5 | 0.003 | 792292 | $2.4 |

## 4.3 Searches

Table 4: Amount of times the expected documents appeared in the top results when searched with 500 queries from query-document pairs

| **Model** | **Top 100** | **Top 200** | **Top 250** |
|---|---|---|---|
| original | 21% | 28.75% | 32.25% |
| Trained on TSDAE | 14.25% | 20.5% | 24.25% |
| Trained on TSDAE and GPL | 9.25% | 13.25% | 16% |
| Trained on TSDAE and GenQ | 33.27% | 42.28% | 46.29% |
| Trained on TSDAE-textchunks and GenQ | 43.69% | 55.31% | 60.32% |
| Trained on only GenQ | **46.89%** | **57.72%** | **61.72%** |

Table 4 shows the number of times the expected document appeared in the search results for 500 queries. The best results were given when the original model was trained with GenQ.

The GenQ method outperformed GPL in training the semantic search model, as evidenced by the search results accuracy. The application of GenQ resulted in a higher percentage of expected documents appearing in the top search results, significantly improving the tool's precision in document retrieval. Specifically, the percentage of searches where the expected document appeared in the top 100 results increased to 46.89% with GenQ, compared to 21% with the original model.

This result suggests that the Swedish zero-shot model is not suitable as a substitute for a cross-encoder. Since the biggest difference between GPL and GenQ is that GPL gives the data a label for how the close the query is to the document, this is most likely the part that ended up giving the decreased results for GPL.

Evaluations using 500 queries showed that the trained model significantly improved in identifying relevant documents within the top search results. The model trained with GenQ demonstrated the best performance, with a noticeable increase in the instances where the expected document was ranked within the top 100, 200, and 250 search results. This enhancement in search precision underscores the effectiveness of GenQ methodology for semantic search applications in the legal domain.

## 4.4 Summary of results

The results of this thesis project illustrate the challenges and potential of employing advanced NLP techniques to improve semantic search tools for specialized domains like Swedish legal judgments. While TSDAE training enhanced linguistic understanding, it was the combination of fine-tuning GPT-3.5 and employing GenQ for model training that yielded the most significant improvements in search accuracy and relevance. These findings highlight the importance of tailored training methodologies and the potential for fine-tuned LLMs to significantly enhance semantic search capabilities in domain-specific applications.

## 4.5 Further discussion

### 4.5.1 Overview of Key Findings

The development and evaluation of the semantic search tool underscored the importance of specialized training and fine-tuning methods for enhancing the tool's ability to accurately retrieve Swedish legal documents based on shorter queries. The employment of TSDAE for model training improved the system's understanding of the Swedish legal language. However, this did not directly translate into improved search performance, indicating a nuanced challenge in applying natural language understanding (NLU) techniques to domain-specific search tasks.

The fine-tuning of the GPT-3.5 model for generating query data and the application of GenQ for model training were pivotal in achieving significant improvements in search relevance and precision.

### 4.5.2 Fine-tuning GPT-3.5 for Data Generation

The success of using a fine-tuned GPT-3.5 model for generating query data illustrates the potential of using existing large language models (LLMs) for domain-specific tasks. This approach not only proved cost-effective but also underscored the adaptability of LLMs to specialized requirements through fine-tuning. The generated queries were important for creating a labeled dataset that significantly improved the search tool's performance.

**Interpretation of Training Dynamics** The observed training dynamics, characterized by a high degree of variability in both loss and accuracy, suggest a complex learning process. While the general decrease in training loss and high training accuracy indicates that the GPT-3.5 model was capable of learning from the training data, there were some fluctuations that can be because of the variability of the data.

The high variability in training loss could be because of a high learning rate that causes the model's parameters to overshoot optimal values at certain steps. The volatility in accuracy might also imply that the model encountered difficulties in generalizing from certain batches of training data, potentially due to the presence of outliers or noise within the dataset. Since the dataset consists of texts and search queries, which are highly unstructured data, these fluctuations seem reasonable.

### 4.5.3 Efficacy of GenQ over GPL

The superiority of GenQ in training the semantic search model, as compared to GPL, points to the significance of the training methodology in achieving optimal model performance. The limitations of GPL, possibly due to the absence of a suitable cross-encoder for Swedish text, emphasize the need for innovative training and labeling strategies in non-English language contexts.

Another factor for the low performance of GPL can be the data. Due to the large amount of judgment documents, the negative documents might be very similar to the positive one, which will in the end not give the model any negative examples. Because a lot of the crimes are similar, the documents that get retrieved in the negative mining might be as good or better than the 'positive' document. This can have made GPL perform badly on this data.

### 4.5.4 TSDAE

Even though the TSDAE training was successful and made the model better at predicting these specific texts, the improvements did not seem to translate to the bigger purpose of the model. The end results in the semantic searches did not improve, and therefore, TSDAE was not a useful training for this purpose.

While TSDAE enhanced the model's linguistic understanding, the translation of this understanding into effective document retrieval highlighted the complex nature of semantic search. As mentioned in the theory, this use case forces the model to understand the meaning of the text, and cannot rely on that some words might be matching. Thus, it's not enough to only train on predicting deleted words. This suggests that while foundational NLU capabilities are crucial, they must be complemented by training strategies that directly target the search retrieval mechanism.

### 4.5.5 Batch size for GenQ

The batch size in the training step of genQ determines how many negative documents that are seen for each query. If, for example, the batch size is 4, then we have 3 negative documents and 1 positive for each query. With a larger batch size, the model would see more examples of negative documents, which might make the model adapt better. However, the larger batch size also means that we need more resources and time for the training.

### 4.5.6 Asymmetric search

This use case of the semantic search tool is at the very extreme of an asymmetric semantic search. The queries are very short compared to the documents, and can be written in a very different style from the document. This requires a lot of domain understanding from the embedding model, which we seem to have gotten in this project.

# 5 Conclusions and further work

This thesis has successfully addressed the challenge of developing a semantic search tool tailored for Swedish legal documents, leveraging the capabilities of large language models (LLMs) and employing advanced training methodologies. The project's core objectives were to enhance the search tool's ability to understand and process Swedish legal language and improve its precision in retrieving relevant legal judgments based on semantic queries.

While TSDAE effectively improved the model's understanding of legal language, its direct impact on semantic search performance was limited. However, GenQ had a positive impact on the performance of the search tool.

## 5.1 Generation of labeled data

The fine-tuning of the GPT-3.5 model, followed by the application of Generative Query Network (GenQ) training, significantly enhanced the search tool's precision. By generating more relevant queries and effectively training the model to identify pertinent documents, the tool achieved considerable improvements in returning accurate search results. This would not be possible without the usage of GPT to generate this large amount of data. The insights gained from this task contribute to a better understanding of the fine-tuning process for large language models and offer a foundation for further research.

Also, by fine-tuning a GPT3.5 reduced the cost significantly compared to generating everything with GPT4.

## 5.2 GPL

The comparative efficacy of GenQ over Generative Pseudo Labeling (GPL) provided valuable insights into the importance of selecting appropriate training methodologies. This comparison underscored the potential limitations of GPL depending on the data and the absence of a suitable cross-encoder for Swedish text and highlighted GenQ's effectiveness in overcoming these challenges.

## 5.3 Further work

### 5.3.1 Development of Custom Cross-Encoders

The limitations encountered with the Generative Pseudo Labeling (GPL) method highlighted the need for a cross-encoder suited for Swedish text. Further work could focus on developing and training custom cross-encoders tailored to the Swedish language and legal terminology. Such models could improve the labeling accuracy for training data, thereby enhancing the overall performance of the semantic search tool.

### 5.3.2 Adapt to user feedback

User feedback would give a better insight of the performance. So, for future work, it would be valuable with some feedback from users on how the application satisfies their needs.

Also, integrating user feedback mechanisms into the semantic search tool could provide valuable data for continuous improvement. By allowing users to rate the relevance of search results, the system could adapt and fine-tune its algorithms based on real-world usage patterns. Machine learning techniques could then be employed to analyze feedback and adjust the model's parameters to enhance search accuracy over time.

### 5.3.3 Performance Optimization and Scalability

As the semantic search tool evolves, by more users and data, addressing performance optimization and scalability becomes crucial. Future work should explore architectural improvements, efficient indexing strategies, and cloud-based deployment options to ensure the tool can handle large datasets and high query volumes efficiently.

### 5.3.4 Incorporate more functionalities and features to the application

Another improvement to the search tool is to incorporate more functionalities. Some of these functionalities could be filtering or sorting by date, location or other specifics of the documents.

Also, allowing the user to paste a complete police report or similar long text, would require a symmetric search approach, which could also be a useful feature for this tool.

## References

[1] A. J. Daniel Holmer, “Comparing the performance of various swedish bert models for classification,” 2022. [Online]. Available: https://spraakbanken.gu.se/sites/default/files/2022/SLTC-2020_paper_12.pdf

[2] M. Henderson, R. Al-Rfou, B. Strope, Y. hsuan Sung, L. Lukacs, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, “Efficient natural language response suggestion for smart reply,” 2017. [Online]. Available: https://arxiv.org/pdf/1705.00652.pdf

[3] S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston, “Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring,” 2020.

[4] J. P. Jiawei Han, Micheline Kamber, *Getting to Know Your Data*, 3rd ed. In The Morgan Kaufmann Series in Data Management Systems, 2012.

[5] E. Latif and X. Zhai, “Fine-tuning chatgpt for automatic scoring,” *Computers and Education: Artificial Intelligence*, vol. 6, p. 100210, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666920X24000110

[6] M. Malmsten, L. Börjeson, and C. Haffenden, “Playing with words at the national library of sweden – making a swedish bert,” 2020.

[7] OpenAI, “Gpt-4 technical report,” 2023. [Online]. Available: https://arxiv.org/pdf/2303.08774.pdf

[8] N. R. M. S. OpenAI (Tom B. Brown, Benjamin Mann, “Gpt-4 technical report,” 2020. [Online]. Available: https://arxiv.org/pdf/2005.14165.pdf

[9] N. Reimers, “Datasets,” https://www.sbert.net/docs/package_reference/datasets.html#denoisingautoencoderdataset, 2022, accessed: Dec 31, 2023.

[10] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” 2019. [Online]. Available: https://arxiv.org/pdf/1908.10084.pdf

[11] R. Subhashini and V. J. S. Kumar, “Evaluating the performance of similarity measures used in document clustering and information retrieval,” in *2010 First International Conference on Integrated Intelligent Computing*, 2010, pp. 27–31.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[13] K. Wang, N. Reimers, and I. Gurevych, “Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning,” 2021. [Online]. Available: https://arxiv.org/abs/2104.06979

[14] K. Wang, N. Thakur, N. Reimers, and I. Gurevych, “Gpl: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval,” 2022.

[15] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, “Zero-shot learning – a comprehensive evaluation of the good, the bad and the ugly,” 2020.