

Article

From Fact Drafts to Operational Systems: Semantic Search in Legal Decisions Using Fact Drafts

Gergely Márk Csányi ^{1,*}, Dorina Lakatos ¹, István Üveges ^{1,2}, Andrea Megyeri ³, János Pál Vadász ^{1,4},
Dániel Nagy ¹, Renátó Vági ^{1,3,5}

- ¹ MONTANA Knowledge Management Ltd., Hársalja Street 32, 1029 Budapest, Hungary; lakatos.dorina@montana.hu (D.L.); uveges.istvan@montana.hu (I.Ü.); nagy.daniel@montana.hu (D.N.)
² HUN-REN Centre for Social Sciences, Toth Kalman Street 4, 1097 Budapest, Hungary
³ Wolters Kluwer Hungary Kft., Budafoki Street 187-189, 1117 Budapest, Hungary; andrea.megyeri@wolterskluwer.com (A.M.); renato.vagi@wolterskluwer.com (R.V.)
⁴ Institute of the Information Society, National University of Public Service, Ludovika Square 2, 1083 Budapest, Hungary; vadasz.pal@uni-nke.hu
⁵ Doctoral School of Law, Eötvös Loránd University, Egyetem Square 1-3, 1053 Budapest, Hungary
* Correspondence: csanyi.gergely@montana.hu

Abstract: This research paper presents findings from an investigation in the semantic similarity search task within the legal domain, using a corpus of 1172 Hungarian court decisions. The study establishes the groundwork for an operational semantic similarity search system designed to identify cases with comparable facts using preliminary legal fact drafts. Evaluating such systems often poses significant challenges, given the need for thorough document checks, which can be costly and limit evaluation reusability. To address this, the study employs manually created fact drafts for legal cases, enabling reliable ranking of original cases within retrieved documents and quantitative comparison of various vectorization methods. The study compares twelve different text embedding solutions (the most recent became available just a few weeks before the manuscript was written) identifying Cohere's embed-multilingual-v3.0, Beijing Academy of Artificial Intelligence's bge-m3, Jina AI's jina-embeddings-v3, OpenAI's text-embedding-3-large, and Microsoft's multilingual-e5-large models as top performers. To overcome the transformer-based models' context window limitation, we investigated chunking, striding, and last chunk scaling techniques, with last chunk scaling significantly improving embedding quality. The results suggest that the effectiveness of striding varies based on token count. Notably, employing striding with 16 tokens yielded optimal results, representing 3.125% of the context window size for the best-performing models. Results also suggested that from the models having 8192 token long context window the bge-m3 model is superior compared to jina-embeddings-v3 and text-embedding-3-large models in capturing the relevant parts of a document if the text contains significant amount of noise. The validity of the approach was evaluated and confirmed by legal experts. These insights led to an operational semantic search system for a prominent legal content provider.

Keywords: semantic search; similarity search; legal database; last chunk scaling (LCS); vector-based retrieval; legal fact drafts



Citation: Csányi, G.M.; Lakatos, D.; Üveges, I.; Megyeri, A.; Vadász, J.P.; Nagy, D.; Vági, R. From Fact Drafts to Operational Systems: Semantic Search in Legal Decisions Using Fact Drafts. *Big Data Cogn. Comput.* **2024**, *8*, 185. <https://doi.org/10.3390/bdcc8120185>

Academic Editor: Salvador García López

Received: 3 October 2024

Revised: 20 November 2024

Accepted: 3 December 2024

Published: 10 December 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Finding similar legal cases is a crucial task in the daily work of both lawyers and judges. It ensures that similar facts lead to similar decisions which greatly promotes legal fairness. Traditionally, legal case retrieval focuses on identifying cases using subject matter categories, which is a helpful but limited approach [1]. While filtering cases by subject matter can facilitate the process, it falls short in capturing the semantic relationships between them. As a result, this process remains time-consuming, requiring significant manual effort to ensure relevant precedents are found.

In the legal field, semantic similarity search usually means that for one given document, the user would like to find other similar documents efficiently. In this current work, we deal with semantic similarity search using fact drafts and aim to find other court decisions with similar facts. *Fact drafts* are few-sentence-long drafts shortly describing the facts of one's legal problem. For instance, in a divorce case, a fact draft would briefly describe all the information that might be relevant in a child custody case, like what needs to be known about the parties, the relationship of the children to each parent, and so on, probably ending the draft with a question.

Court decisions are usually very long (ca. 3000–5000 words on average), using specific legal language that makes semantic-based representations harder mainly because the texts do not fit into the transformer-based embedding models' context window. Therefore, creating an in-production semantic search system lays on two important pillars: the proper embedding form used to embed texts that can handle longer texts as well and the ability to evaluate and compare different solutions. Evaluation is not straightforward since even if humans evaluate certain search results for one retrieval scenario, it is not guaranteed that another method of retrieving other documents cannot be acceptable. Hence, human evaluation can only be used for validation but cannot be efficiently used during the method comparison process.

In this paper, we compare a total of twelve embedding methods, out of which eleven different methods are based on transformer architecture [2] and one is based on fasttext [3] to embed text data. Moreover, for each of the transformer-based models having 512 context window or less we compared seven different ways to address the document length vs. context window size problem. We therefore intend this to be a comprehensive comparative study that provides an excellent basis for the development of similar systems and shows whether or not capturing the semantics of the facts is sufficient for searching similar documents. Although the dataset used in our research is in Hungarian, the best-performing models are all multilingual, suggesting that our findings are likely applicable and useful across languages, extending far beyond just the Hungarian context. The described methodology can also be efficiently followed when creating an in-production semantic search system.

The paper is structured as follows. Section 2 summarizes the relevant works. Section 3 describes the dataset used in this study. Section 4 introduces the caveats during the vectorization process and the evaluation metrics. Results are presented and discussed in Section 5. Finally, the conclusions are drawn in Section 6.

2. Relevant Works

Semantic search between long texts is a long-standing problem, and Natural Language Processing has been working on the issue for decades, ever since the advent of Information Retrieval systems (Salton and McGill [4]). The choice over vectorization forms of texts is crucial to the problem, since these vectors will be used to encode the specific semantic similarity and will be used to identify text similarities in later phases by calculating some kind of similarity metric, e.g., cosine distance (Qian et al. [5]).

Compared to historically earlier solutions, e.g., word2vec (Mikolov et al. [6]), one-hot (Boncalo et al. [7]) or Bag of Words representations (Wu et al. [8]), deep-learning and transformer-based solutions have achieved significant performance gains in many NLP tasks, like sentiment analysis (Alaparthi and Mishra [9]) or legal text classification in general (Pal et al. [10]). An illustrative example of this could be Yang et al. [11], where the authors used text similarity calculation based on topic modeling along with transfer-learning-based text vectorization. Their method combines keyword extraction carried out by topic modeling with BERT-based (Devlin et al. [12]) vector representations, although the study did not specifically address the issue of long texts. Their overall comparison shows that, compared to the word2vec method, the BERT method learns semantic information in the text more efficiently, therefore performance is significantly improved when judging text similarity.

Wang et al. [13] propose an efficient long text retrieval model based on BERT (LTR-BERT). The model works by breaking down the text that does not fit into BERT's maximum context window (512 sub-word tokens) into chunks and then interacting with these chunks one by one during semantic queries. Their tests showed that the resulting multi-component model outperformed the other models selected for comparison in terms of mean reciprocal rank at position 10 (MRR@10) and average precision of the top 100 documents (MAP@100), among other metrics.

Limsopatham [14] compared several different approaches to handle long legal texts with transformers, but on a classification task. He compared truncation of long documents from the back and from the front, chunking the documents into 200 token wide chunks and applying average or maximum pooling using BERT models with and without legal pre-training. He also compared BigBird [15] and Longformer [16] based models that can handle longer than 512 token-length sequences but these models were not pre-trained on legal data. The main conclusion was that truncation of the original documents resulted in lower classification results due to the loss of valuable data, while chunking with max and mean pooling achieved remarkably better results. According to his findings, Longformer and BigBird models outperformed all the other approaches that adapted BERT to deal with long documents, even if they were not pre-trained on domain-specific documents.

Vatsal et al. [17] investigated many approaches to handle long legal texts however, not on a similarity search-typed task but on two classification tasks. Hence, not all presented techniques can be transferred to our task (best 512 length chunks, summarization of documents into 512 tokens etc.), but the stride technique is an exception. The stride technique works by providing a window of tokens that are shared/overlapped between two consecutive chunks (64 and 128 tokens were tested in [17]). The best results in both classification tasks could be reached by the stride technique using a 64 token-length stride, reaching 57.5% F_1 score and 60.9% accuracy on 279 categories.

Dong et al. [18] created a recent survey on how to handle long texts using Transformer models. They introduce the simple method of chunking and aggregating the chunk vectors. Our method improves both the chunking mechanism and also the aggregation by using striding and last chunk scaling.

Wan and Yang [19] investigated the effect of using summaries compared to using the full document. In their study, the query document was summarized and similarity searches were performed with the summary instead of the original document.

Vuong et al. [20] created a legal case retrieval system for COLIEE 2023 Competition. To compare the query and a candidate document they used the Longformer model. As the average case pair was longer than the context window of the model they only kept the most important paragraphs from each case.

Ontology-based legal semantic search systems have a deep-rooted history and have also been used in the recent past [21,22]. Ebietomere and Ekuobase [22] introduced an ontology-based semantic retrieval (SR) system designed specifically for case law retrieval. The system was evaluated using 280 Nigerian Supreme Court cases and demonstrated high retrieval performance, achieving 94% precision and 80% recall.

Šavelka and Ashley [23] retrieved relevant sentences from a case law database utilizing static word embeddings and topic models using phrases as query expressions. They also took into account the information content of the sentences, addressing the general tendency of models to prefer short sentences during retrieval.

Zhu et al. [24] created a semantic retrieval system for COVID-19 data that is capable of finding the relevant legal cases for a given query. To achieve this, they applied convolutional networks (CNN) as encoders, Chinese Word2Vec word embeddings, and contrastive learning. Interestingly, this custom-tailored solution provided better results than using Sentence BERT (SBERT) [25] representations. Nevertheless, it is important to point out that the exact SBERT model was not specified in the paper.

Louis et al. [26] created a RAG (Retrieval Augmented Generation) system for legal Q&A. They fine-tuned the CamemBERT model (French BERT model) [27] for Q&A where

the question-document pairs appear closer to each other than irrelevant pairs. The same encoder was used for encoding the questions and the relevant documents. Their approach outperformed the multilingual E5 models.

Hu et al. [28] performed similar legal case retrieval on Chinese criminal case facts. Their solution involves threefold encoding: semantic, topical, and legal entity encoding. It handles the long document problem by splitting the documents into paragraphs, as in [29]. For the semantic encoding, they used the Chinese BERT-base model with the [CLS] vector. Each document was split at a paragraph level, and query-candidate paragraph pairs were created by concatenating these paragraphs, using the [SEP] token in between. This setup is used during BERT's pre-training phase for the Next Sentence Prediction task. The evaluation was conducted on the LeCARD dataset [30], which consists of 107 query cases and 10,700 candidate cases selected from over 430,000 criminal judgments. Although the authors state that they used facts during their study, it is not completely clear whether they mean the same by facts as in the present study. Furthermore, the study did not compare their solution with Q&A fine-tuned Sentence-BERT embedding representations. Recently, several such embedding models have emerged, providing state-of-the-art results in capturing semantic similarity or in text retrieval tasks [31–33].

In the experiments detailed in this paper, we have paid particular attention to compare these new embeddings, while also providing firm other baselines using older technologies like fasttext. We have also focused on the difficulties involved in processing long texts and how they can be most efficiently passed to the embedding models that perform the vectorization (cf. Section 4).

Table 1 contains a brief summary about the most relevant works discussed in this section.

Table 1. Comparison of most relevant works.

Authors	Task	Handling Long Texts	Legal	Language
Wang et al. [13], 2023	Long document retrieval	Chunking and sequential chunk interaction with LTR-BERT	No	English
Limsopatham [14], 2021	Classification	Chunking and max/mean pooling, Longformer and BigBird	Yes	English
Vatsal et al. [17], 2023	Classification	Striding	Yes	English
Hu et al. [28], 2022	Retrieval with facts	Splitting to paragraphs	Yes	Chinese
Vuong et al. [20], 2024	Case and statute law retrieval	Important paragraphs retained	Yes	English and French
Louis et al. [26], 2024	RAG using Q&A fine-tuned BERT	Not stated	Yes	French
Current work, 2024	Retrieval with facts	chunking, striding, LCS and max pooling, larger context-windowed S-BERT models	Yes	Hungarian

3. Dataset

The dataset used in this study consists of 1172 Hungarian court decisions from all court levels and served as a test set before [34]. The documents were labeled by an LSTM-based Rhetorical Role Labeler (RLL) neural model trained by the authors. This RLL model splits the documents into sentences and tags each sentence with one of the following labels:

- **Facts:** all sentences describing what the dispute is about.
- **Prior decisions:** as we deal with legal cases from all court levels (Supreme Court, Trial Courts, High Courts, Tribunals, etc.), the documents might contain information about prior decisions. These parts of the text are annotated with this label.
- **Arguments of the parties:** sentences about the arguments and inquiries of the parties.
- **Decision:** sentences describing the judgement.
- **Reasoning of the court/Ratio of the decision:** sentences containing the reasoning of the decision describing the rationale behind it by the given court.
- **Costs:** sentences about how much either side must pay due to the decision.

- **Operative part:** sentences detailing what are the in practice consequences of the decision e.g., the defendant has to pay X amount of money as damages to the plaintiff.
- **OUT:** none of the above-mentioned categories, e.g., signatures, the initial part of the document, section titles, etc.

Since we wanted to create a system that is capable of finding similar legal decisions by entering a brief description of the case, only two rhetorical roles were selected to be retrieved, namely the *Facts* and the *Arguments of the parties*. We compared two scenarios: using only the sentences identified as *Facts*, and combining the sentences identified as *Facts* or *Arguments of the parties*.

3.1. Fact Drafts

To make the evaluation of the similarity search system cost-effective and reproducible, we decided to create search snippets describing the facts of a case as a draft, referred to as Fact drafts in this study. These fact drafts were created from the respective documents, with any specific identifiers, such as places, organization names, dates, etc., removed. Consequently, a forward ranking of this document in the similarity retrieval is expected, which will allow the comparison of different text embedding models by using the appropriate metrics like Mean Reciprocal Rank or Recall@n.

Two approaches were tried to create Fact drafts. At first, we instructed OpenAI's GPT 3.5 turbo model via the API at the completion endpoint to create a summary to the already selected sentences belonging to the Facts part of the document. The used model was gpt-3.5-turbo-1106. For this phase, we used prompts similar to the one presented in Example 1.

Example 1. *In up to 3 sentences, summarise the text between the "" signs as you would if you were a lawyer looking for similar documents in a search engine for a database of legal documents. The summary should not include any specific information like geographical names, exact dates, organizations or one-word search terms. Return only the paraphrased text, and the text should be coherent, 2–3 sentences and shorter than the original and do not copy the original text. The text is: ""... "".*

The most important common elements of the instruction variants were, e.g., instructing the model not to mention specific named entities, and requiring paraphrasing of the text rather than copying parts of it.

Experience showed that in a significant number of cases the model acted contrary to the instructions. Hallucinations were not common, but, for instance, there were several instances of specific company names or places from the original texts were mentioned in the fact drafts. Another typical issue was when the model generated search terms based on the Facts mentioned in the prompt. When searching in a database, such insufficiently general formulations can significantly distort the results. That is why we decided to select 100 documents (20 from each law area: administrative, civil, criminal, economic, and labor) and manually create fact drafts. These manual fact drafts were prepared by 3 annotators by reviewing and, if needed, modifying the GPT-generated answers. In cases where the original summary produced by the model was also correct, it was left intact, but this was only the case in a small minority of the cases.

3.2. Data Statistics

The token-level statistics of the dataset are presented in Table 2. The Facts (draft) column contains the results for the facts of the selected 100 documents, while the Fact drafts column shows the statistics for the drafts created by humans. All other results are calculated on the whole corpus. The percentage of fully covered documents and percentage of tokens covered on documents longer than the context windows by wide (8192) models are shown in Table 3. The exact model names, their respective context window sizes and embedding dimensions are listed in Table 4.

Table 2. Token counts (AVG) and average characters per token measured with different tokenizers. The characters per token ratio was calculated on the full dataset containing 1172 documents.

	Facts	Facts (Draft)	Fact Drafts	Arg. + Facts	Char./tok.
hubert					
sbert_hubert	969.17	697.36	82.02	1996.04	4.694
danieleff					
e5_large					
e5_base					
cohere	1140.09	819.21	108.41	2402.73	3.862
bge_m3					
jina_v3					
mcontriever	1492.35	1091.38	139.82	3159.76	2.908
openai_ada					
openai_3_large	1902.46	1411.87	181.77	4051.84	2.259

Since some embedding models were only available through an API call (Cohere’s embed-multilingual-v3.0, OpenAI’s text-embedding-ada-002, text-embedding-3-large) it was relevant to know the average token counts that should be sent to these APIs. This is because the costs of the mentioned APIs can be calculated in a tokenistic manner. As Table 2. indicates some models share the same tokenizers. For instance, both sbert_hubert [35] and danieleff is a finetuned version of the huBERT model [36], therefore they inherited the original tokenizer from it. It is notable that the average token counts for the same text, when using tiktoken (the tokenizer used for OpenAI models), are twice as high as those with the Hungarian-only huBERT tokenizer. This is an important consideration in any practical application.

Knowing the average token count was also important because, for example, OpenAI embedding models can only operate within a given context window. For instance, in the case of the text-embedding-ada-002 model, this is a maximum of 8191 tokens (as of December 2024) (<https://platform.openai.com/docs/guides/embeddings/embedding-models> (accessed on 2 December 2024)). For exceptionally long texts (which are quite common in the legal domain), if the model’s context window of the model is too small, it may not be possible to send the entire data in a single interaction. This could lead to a deterioration in the quality of the model’s returned result.

To assess how well the models having 8192 token-wide context windows (openai, jina_v3, bge_m3) cover the documents, we have calculated the corresponding statistics shown in Table 3.

Table 3. Percentage of fully covered documents by models having 8192 tokens context window. Percentage of tokens covered on documents longer than the context windows.

Model	Documents Fully Covered [%]		Long Texts Covered Tokens [%]	
	Facts	Arguments and Facts	Facts	Arguments and Facts
openai	97.01	91.89	60.59	68.27
bge_m3, jina_v3	98.55	96.93	63.02	71.54

In the OpenAI models, 97.01% of the documents fit completely into the context window for Facts and 91.89% for Arguments and Facts combined. Looking at long texts only, meaning that the ones over the context window, the average token coverage was 60.59% for Facts and 68.27% for Arguments and Facts together. In the case of the bge_m3 and jina models, 98.55% of the documents fit completely into the context window in the case of Facts and 96.93% in the case of Arguments and Facts combined, while on long texts only, the average token coverage was 63.02% for Facts and 71.54% for Arguments and Facts together. At first glance, having lower coverage on the Facts may seem surprising but this phenomenon occurs

because the number of documents not fitting in the context window is obviously higher when the Arguments and Facts are considered, but also, there are many of these documents that almost fit within the context window, causing the average to increase significantly.

Hence, given the high ratio of covered text, it was unlikely that the chunking would significantly affect the results of the models with 8192 token-long context windows. Therefore, splits were calculated only for the bge_m3 model, while for all other models, only the truncated vector forms were used. It is also important to note that the models with smaller context windows could not cover even 50% of the documents by their context windows. Therefore, addressing this problem was necessary to avoid using suboptimal text representations. To see how we tackled this problem, please refer to Section 4.1.1.

4. Methods

This section briefly describes the vectorization forms used in the study and the methods used to evaluate the results.

4.1. Vectorization

The quality of vectorization plays a crucial role in vector-based retrieval. The vectorization models tested are presented in Table 4. The models for vectorization were selected to cover a wide range of models starting from a word-embedding-based out-dated model to state-of-the-art transformer-based embedding models. First, we give a brief overview and then we provide an in-depth introduction for these models in the next paragraph. As a baseline we chose the *fasttext* model since this is the oldest model and is based on word embeddings. Since our dataset was a legal dataset in Hungarian, we selected the origo of Hungarian BERT models named *huBERT* [36]. There are three different models based on *huBERT* that we tested: a non-finetuned (*hubert*), a semantic similarity fine-tuned (*sbert_hubert*) and a Q&A fine-tuned but on a small dataset (*danieleff*).

The current trend in vector-based retrieval points towards the use of multi-lingual embedding models that are fine-tuned for several tasks including information retrieval and Q&A. We selected models having smaller context windows (*mcontriever*, *e5_large*, *e5_base* and *cohere*, 512 tokens) and ones that have significantly wider context windows (*openai_ada*, *openai_3_large*, *bge-m3* and *jina_v3*, 8192 tokens). Many of these models are at the top of the MTEB leaderboard's [37] non-English Retrieval and Semantic Textual Similarity (STS) sections. The models are described in detail below.

The *cohere*, *openai_3_large*, *openai_ada* and *jina_v3* models were reached through an API. The *fasttext* [3] model was trained by using 157,000 Hungarian legal decisions using the official *fasttext* Python package. The model is a skip-gram model [6], and during training, negative sampling was set to 10, the learning rate was set to 0.05, and the model produced 100-dimensional length vectors. All other parameters were left at their default settings.

Table 4. Vectorization models compared in this study.

Model Name	Short Form	Context Window	Dimension
<i>fasttext</i> [3] model trained on 157,000 Hungarian legal cases	<i>fasttext_montana</i>	-	100
NYTK/sentence-transformers-experimental-hubert-hungarian [35]	<i>sbert_hubert</i>	128	768
danieleff/hubert-base-cc-sentence-transformer	<i>danieleff</i>	512	768
SZTAKI-HLT/hubert-base-cc [36]	<i>hubert</i>	512	768
intfloat/multilingual-e5-base [31]	<i>e5_base</i>	512	768
intfloat/multilingual-e5-large [31]	<i>e5_large</i>	512	1024
Cohere/Cohere-embed-multilingual-v3.0 (https://txt.cohere.com/introducing-embed-v3/ (accessed on 2 December 2024))	<i>cohere</i>	512	1024
facebook/mcontriever-msmarco [38]	<i>mcontriever</i>	512	768

Table 4. Cont.

Model Name	Short Form	Context Window	Dimension
OpenAI text-embedding-ada-002 (https://platform.openai.com/docs/guides/embeddings/embedding-models (accessed on 2 December 2024))	openai_ada	8191	1536
OpenAI text-embedding-3-large (https://platform.openai.com/docs/guides/embeddings/embedding-models (accessed on 2 December 2024))	openai_3_large	8191	3072
BAAI/bge-m3 [32]	bge_m3	8192	1024
jinaai/jina-embeddings-v3 [33]	jina_v3	8192	1024

The oldest technology is *fasttext*, which is similar to the Word2Vec [6,39] word embedding method but handles the out-of-vocabulary words by using subword embeddings for character n-grams. This embedding proved to be especially effective in highly inflecting languages like Hungarian [3]. Generally, a document representation is calculated as the average of the whole words and the character n-grams of the text. All the other models are transformer-based models.

The next group is a group of models that is pre-trained in Hungarian language (hubert) and fine-tuned for providing semantic representations in an S-BERT [25] fashion marked with *sbert_hubert* in this study [35]. This model was fine-tuned on the Hunglish 2.0 parallel corpus [40] to mimic the *bert-base-nli-stsb-mean-tokens*[25] model. The vast majority of the parallel corpus is from the legal domain.

The *danieleff* model is based on the most commonly used huBERT base model [36], which is a Hungarian BERT base language model. The model was fine-tuned using 170 question-answer pairs from the university studies domain. The answers were 1000–5000 characters long chunks from organizational and operational rules of universities.

The next group consists of multilingual models fine-tuned for question answering, semantic similarity and/or passage retrieval tasks: multilingual e5 models [31], cohere multilingual embedding model (<https://txt.cohere.com/introducing-embed-v3/> (accessed on 2 December 2024)), BGE-M3 multilingual embedding model [32], Jina AI' multilingual embedding model [33] and Facebook's mContriever model [38]. These models were fine-tuned on the MSMARCO dataset [41] (that is a Q&A dataset) among other datasets.

The mContriever model is multilingual BERT-based solution[12], trained on the CC-Net dataset [42] using contrastive learning [38]. Contrastive learning means that given the query representation, the goal is to retrieve the representation corresponding to the positive document among all the negatives [38].

The e5 models are multilingual models that are XLM-RoBERTa-based [43] and multilingual extensions of the English E5-based text embeddings [44]. These models have been pre-trained in a weakly supervised contrastive manner on billions of text pairs and fine-tuned in a supervised way on a small quantity of high-quality labeled data [31].

There are no published details on how cohere embeddings are created, likely due to commercial reasons. However, according to a blog post (<https://txt.cohere.com/introducing-embed-v3/> (accessed on 2 December 2024)) it is emphasized that cohere embeddings capture both the topic match and the content quality, in contrast to e.g the *openai_ada* vector that captures only the topic similarity aspect. This leads to higher-quality retrieval results.

The *bge_m3* is a relatively new model, similar to the e5 models. It provides meaningful representations across more than 100 languages, and at multiple depths (word, paragraph, and full text up to 8192 tokens). Additionally, it can handle three common retrieval tasks: dense, sparse, and multi-vector retrieval. Sparse retrieval uses word-level importance scores derived from embeddings to compare and rank documents based on common words, making it suitable for scenarios where precise word matching is important. Multi-Vector retrieval employs multiple vectors to capture different aspects or parts of a document, allowing for more accurate relevance alignment between the query and parts of the document, which is useful for understanding broader context

and semantic meaning. The model is XLM-RoBERTa-based and supports dense retrieval tasks using the [CLS] vector, while token vectors can be applied to sparse and multi-vector retrieval tasks. Jina AI's v3 multilingual embedding model [33] was recently introduced and currently leads in many non-English languages for the Semantic Text Similarity task on the MTEB leaderboard (<https://huggingface.co/spaces/mteb/leaderboard> (accessed on 2 December 2024)) [37]. The model is available both through huggingface (<https://huggingface.co/jinaai/jina-embeddings-v3> (accessed on 2 December 2024)) and via an API (<https://jina.ai/embeddings/> (accessed on 2 December 2024)). In this study, we used the API for vectorization. This model is Jina-XLM-RoBERTa-based and also provides multilingual and multi-task embeddings like the BGE-M3 model and it also has 8192-token-long context window. It can be used for a wide range of tasks, including query-document retrieval, clustering, classification, and text matching. The embedding dimension can be modified from 1024 to as low as 32 without significantly impacting performance, thanks to Matryoshka Representation Learning (MRL).

OpenAI's v3 embedding has also been recently released. These models were trained using a technique that allows the embeddings to be shortened by removing certain numbers from the end without losing their concept-representing properties [45].

4.1.1. Handling Long Texts

Transformer-based models excel at capturing semantic relations. However, one major drawback is that these models can only handle a fixed length of token window. This is not a problem if the text is smaller than the context window of the model. In contrast, questions arise when the texts do not fit into this window. To address this, we developed and compared seven different approaches, including Last Chunk Scaling (LCS), which is particularly useful for texts that are approximately 2–5 times longer than the context window. The approaches are as follows:

- **truncated:** this is the easiest approach, keeping only the first context window length of the document. Note that different models cover different numbers of characters. This is the baseline.
- **chunked:** splits the document into chunks having context window length, ensuring that this split does not cut words into half but only on word ends. The vectors are calculated for these chunks and averaged, resulting in a document vector.
- **stride:** creating chunks similarly compared to the chunked approach but ensuring that there are tokens overlapping between the chunks as described in [17]. These chunks are averaged to get a document vector. We tested two different stride sizes: 25% of the context window and fixed 16 tokens.
- **last chunk scaling (LCS):** this technique can be applied alongside the chunked or stride methods. After chunking the document, the last chunk generally consists of fewer tokens than the context window. Our solution scales the last chunk's vector by the following factor: $\frac{\text{number_of_tokens_in_the_last_chunk}}{\text{context_window_size}}$. This method was tested with the chunked and both stride approaches (25% and fix 16 tokens).

The pseudo code for all of the chunking strategies is shown in Algorithm 1.

Both the stride and the LCS approaches aim to mitigate the chunking method's major drawback, namely that the chunks are completely separated from each other. While the stride method addresses this issue by sharing tokens between the chunks, the LCS approach works differently: it prevents the unwanted overweighting of the last, often smaller chunks when averaging the chunk vectors. This is important because such overweighting can easily distort the direction of the average vector, particularly when there are 2–5 chunks per document. Our case fell into this category, as the Facts documents consisted of 2–8 chunks on average, depending on the vectorizers. An example showing the effect of the Last Chunk Scaling is shown in Figure 1.

Algorithm 1 Chunking strategies.

```

1: function GETVECTOR(tokens, max_len, stride, lcs)
2:   Input: tokens: list of tokens
3:   Input: max_len: maximum tokens per chunk
4:   Input: stride: striding window size, 0 means no striding
5:   Input: lcs: whether to use last chunk scaling
6:   slices  $\leftarrow$  GetChunks(document, max_len, stride)
7:   vectors  $\leftarrow$  VectorizeSlices(slices)
8:   if lcs then
9:     scale_factor  $\leftarrow$  len(slices[-1]) / max_len
10:    vectors[-1]  $\leftarrow$  vectors[-1] * scale_factor
11:   end if
12:   return Average(vectors)
13: end function
14: function GETCHUNKS(tokens, max_len, stride)
15:   Input: tokens: list of tokens
16:   Input: max_len: maximum tokens per chunk
17:   Input: stride: number of overlapping tokens (0 means no overlap)
18:   slices  $\leftarrow$  []
19:   word_ends  $\leftarrow$  GetWordEndPositions(tokens)
20:   start_pos  $\leftarrow$  0
21:   while start_pos < len(tokens) do
22:     end_pos  $\leftarrow$  max( $x \in$  word_ends :  $x < \text{start\_pos} + \text{max\_len}$ )
23:     current_slice  $\leftarrow$  tokens[start_pos:end_pos]
24:     slices.append(current_slice)
25:     start_pos  $\leftarrow$  min( $x \in$  word_ends :  $x > \text{start\_pos} + \text{max\_len} - \text{stride}$ ) + 1
26:      $\triangleright +1$  is needed for the word start position
27:   end while
28:   return slices
29: end function
30: function GETWORDENDPOSITIONS(tokens)
31:   Input: tokens: list of tokens
32:   Output: word_ends: list of indices marking the ends of complete words
33:   word_ends  $\leftarrow$  []
34:   for each  $i$  in  $[0, \text{len}(\text{tokens}) - 1]$  do
35:     if tokens[ $i$ ] is the end of a word or punctuation then
36:       word_ends.append( $i$ )
37:     end if
38:   end for
39:   return word_ends
40: end function
41: function VECTORIZESLICES(slices, vectorization_model)
42:   Input: slices: list of token chunks
43:   Input: vectorization_model: model used for vectorization
44:   Output: vectors: list of vectors corresponding to the slices
45:   vectors  $\leftarrow$  []  $\triangleright$  Initialize an empty list for vectors
46:   for each slice in slices do
47:     vector  $\leftarrow$  vectorization_model.vectorize(slice)
48:     vectors.append(vector)
49:   end for
50:   return vectors
51: end function

```

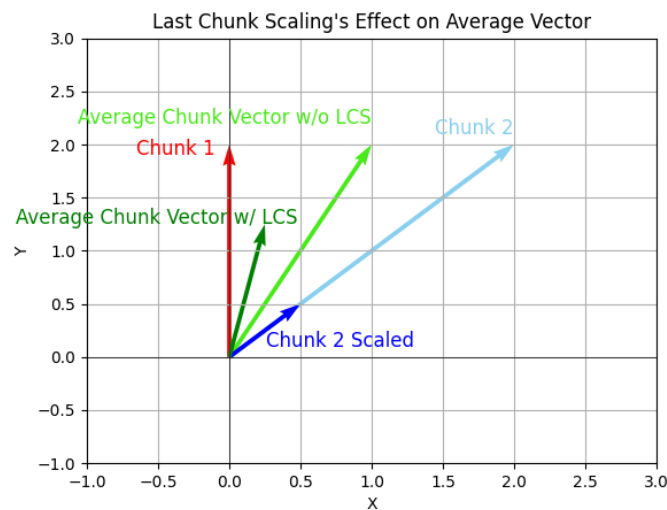


Figure 1. Effect of Last Chunk Scaling. Chunk 2 is the last chunk having only 128 tokens in a 512 token wide context window. Notice how much the direction of the average (document) vector alters by applying LCS (from light green to green vector). This mitigates overweighting of the shorter sequence.

All in all, seven approaches were tested on the models having at most 512 tokens: truncated, chunked, chunked+LCS, stride (25%), stride (25%)+LCS, stride (fix 16), and stride (fix 16)+LCS.

4.1.2. Vectorization Architecture

The vectorization architecture diagram is shown in Figure 2. First the chunks are generated from the documents. These chunks are then fed into the vectorizer (embedding) model. Vectorization can be done either by calling the vectorizer on a single chunk or on a batch. However, vectorizing batches is more computationally intensive. The vectorizer used Sentence-BERT [25] in the majority of the models except for the *fasttext* and *hubert*. The *fasttext* is not transformer-based and in case of *hubert* we used the [CLS] embeddings instead of the S-BERT default token averaging. The document vector is finally gained by averaging the chunk vectors.

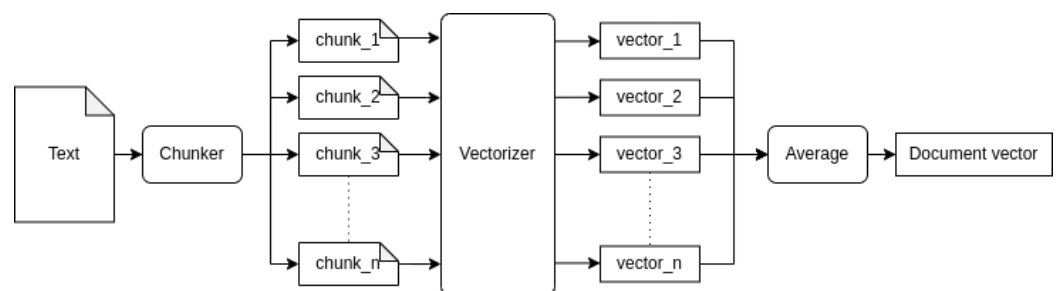


Figure 2. Architecture diagram of vectorization

4.2. Evaluation Metrics

As evaluation metrics, we applied two widely used measures: Mean Reciprocal Rank (MRR) and Recall at n ($R@n$), as well as another metric, namely the Cosine Similarity Difference (CSD). The exact calculations of these are described below.

4.2.1. Mean Reciprocal Rank

The mean reciprocal rank is calculated as follows:

$$MRR = \frac{1}{n} \sum_{i=0}^n \frac{1}{r_i} \quad (1)$$

where n is the number of documents used during the evaluation (in our case 100 documents) and r_i is the retrieved position of the good document. Mean reciprocal rank is a metric that can be applied when for a given query the expected document is known. And this was the case with our created dataset described in Section 3.1.

4.2.2. Recall at n (R@n)

Recall at n is a metric used to compare different solutions by measuring whether the relevant document was retrieved within the top n documents. It is calculated by dividing the number of relevant items retrieved by the total number of relevant items. For example, if the ground truth document is A and the retrieved list of documents is [C, D, A, E, F] the R@1 score will be 0, but the R@3 score will be 1. Note that Precision@1 is the same as the Recall@1 value in our setting since if the first retrieved document is not the ground truth it can be considered either as a false positive or a false negative.

4.2.3. Cosine Similarity Difference (CSD)

A major drawback of the MRR method is that it does not take into account if a similar but not the same document is retrieved as the expected one. Cosine similarity is a widely used metric to compare two embedded texts semantically. To measure how well the retrieved documents compare to the ground truth one, we calculated the cosine scores for the given query and calculated the difference of these scores between the first retrieved and the ground truth documents.

Example: The retrieval for a given query retrieved the following order of documents: [C, B, A] with cosine scores [0.98, 0.96, 0.95], respectively. The ground truth document for the query is A. The difference is calculated as $0.98 - 0.95 = 0.03$. Note that if the first document is the ground truth one, this score is 0. To make the comparison easier, we also multiplied the results by 100, so in the example above the result would have been 3. Another important point is that CSD scores cannot be compared across different vectorization models, as the retrieval scores vary between methods. However, this measure can still be used to compare different chunking methods.

5. Results and Discussion

In this section we share the results of the study. The best results in each row are highlighted with bolding. The chunking row declares the chunking methods used:

- **trunc** means truncated,
- **0** means striding with 0 tokens that is equivalent to the chunked option,
- **16** means the stride setting with 16 overlapping tokens,
- **25%** means striding with 25% of the context window,
- **LCS** shows whether the last chunk scaling has been applied or not.

The **_F** postfix means that only the Facts have been used during the retrieval and **_A_F** means that both the Facts and the Arguments of the parties. Due to the large context window in the case of OpenAI and Jina embedding models, only the truncated vectors have been calculated.

5.1. MRR Results

The MRR results are shown in Table 5. The MRR scores have been multiplied by 100, so the theoretical maximum value is 100. From now on we refer to this multiplied score as MRR. The best result in each row is highlighted with bolding and the results are color coded: the better values are marked with red color and the worse ones with blue.

Table 5. MRR results (multiplied by 100).

Chunking/Stride LCS	Trunc False	0 False	0 True	25% False	25% True	16 False	16 True
bge_m3_A_F	88.21	88.03	88.04	88.04	88.04	88.03	88.04
bge_m3_F	93.67	93.67	93.67	93.67	93.67	93.67	93.67
cohere_A_F	91.18	87.36	88.45	87.29	88.29	87.37	89.35
cohere_F	92.99	93.23	94.87	90.69	92.21	94.27	95.03
danieleff_A_F	73.01	57.61	61.65	56.63	58.34	58.07	61.13
danieleff_F	77.09	77.09	79.88	76.46	78.35	78.23	79.88
fasttext_montana_A_F	41.57	41.57	41.57	41.57	41.57	41.57	41.57
fasttext_montana_F	53.72	53.72	53.72	53.72	53.72	53.72	53.72
hubert_A_F	1.94	4.98	4.66	4.88	4.28	5.92	4.68
hubert_F	7.69	12.12	8.79	8.01	7.69	10.65	8.79
e5_base_A_F	82.69	74.97	82.80	78.07	81.32	77.63	83.15
e5_base_F	85.13	86.81	88.01	86.67	87.45	87.42	88.77
e5_large_A_F	87.15	76.84	82.89	81.05	83.74	78.72	84.68
e5_large_F	87.82	89.00	92.45	87.83	89.37	88.51	91.81
jina_v3_A_F	73.21	-	-	-	-	-	-
jina_v3_F	93.08	-	-	-	-	-	-
mcontriever_A_F	83.92	81.70	84.86	82.45	83.63	82.55	85.23
mcontriever_F	81.46	89.53	90.62	90.34	91.14	89.23	91.42
openai_3_large_A_F	67.80	-	-	-	-	-	-
openai_3_large_F	93.05	-	-	-	-	-	-
openai_ada_A_F	77.54	-	-	-	-	-	-
openai_ada_F	83.65	-	-	-	-	-	-
sbert_hubert_A_F	30.76	45.95	46.28	47.90	48.11	47.16	47.32
sbert_hubert_F	34.81	60.43	57.31	57.17	57.91	58.56	58.11

The worst-performing vectorization method was the hubert approach, which provided by far the lowest results. This highlights the importance of task-specific fine-tuning, as using only the CLS vector from a non-finetuned BERT model did not yield meaningful representations for this task.

The next group consisted of the small context sentence BERT (S-BERT) sbert_hubert model, and the domain-trained fasttext model. The sbert_hubert model was fine-tuned to represent semantic information for sentences. Therefore, it is not that surprising that the models performed better than the non-finetuned hubert solution. The fasttext model stands out among the compared vectorization methods as the only non-transformer-based model. Despite this, it managed to outperform the majority of the different settings of the small S-BERT model, although it underperformed when compared to models with larger context windows.

The next level was the danieleff model, which is a model fine-tuned in a Q&A fashion but on fewer than 200 examples. It has a context window that is four times larger (512) than the small context sbert_hubert model (128). The results suggest that this level of context window size difference and the task-specific fine-tuning are far more significant than all the chunking, striding and LCS methods combined, pushing the danieleff model 20 points ahead compared to the sbert_hubert model. This is a huge development considering that the MRR is not a linear metric: a 20-point increase in Mean Reciprocal Rank (MRR) from a value of 60 is significant—MRR equals 1 only with a perfect match, while the second-ranked result contributes to the average with just 0.5, highlighting the diminishing returns of lower-ranked correct answers.

A significant increment (ca. 8–12 points) in MRR was observed in the mBERT-based mcontriever model and the multilingual E5-based models compared to the danieleff model, reaching 92.45 maximum MRR scores, respectively. The significant increase verifies that purely semantic-meaning-oriented fine-tuning is not enough when searching for similar facts of legal cases. Passage retrieval-like fine-tuning brings the solution to another level.

The best performing OpenAI model was the recently published `openai_3_large`, achieving 93.05 MRR on Facts, which represents a 10-point advantage compared to the prior `openai_ada` model, proving its superiority. Interestingly, on Arguments and Facts the `openai_ada` model performed better than the `openai_3_large` model (77.54 vs. 67.80).

The `jina_v3` model reached 93.08 MRR score on Facts narrowly surpassing the `openai_3_large` model. While on Arguments and Facts, the model performed a lot worse, reaching only a 73.21 MRR score.

The `bge_m3` model reached the second best MRR results for both Facts and Arguments and Facts, with scores of 93.67 and 88.21, respectively.

The best results were reached by the `cohere` embeddings reaching 95.03 on Facts and 91.18 on Arguments and Facts, despite having only 512 token wide context window.

On average, 76.54 was reached on the Facts, while a significantly lower score of 66.05 was reached on Arguments and Facts across all models. This indicates that the Facts are more useful when the task is finding similar cases to a Facts draft. This is probably because the query texts have been created using the sentences with the Facts label therefore, this result is not surprising. However, the difference between the results reached using Facts and Arguments & Facts highlights the ability of the embedding to focus on the relevant parts. This is relevant for the long models since the majority of the texts could fit in the context window. Therefore, we can compare these models in terms of how well they represent smaller parts of the texts, in other words, how well they perform on noisy data. In the case of the `bge_m3` model, this difference is about 5 points, just like the `openai_ada` model. However, this difference is way bigger in `jina` and `openai_3_large` models: approximately 20 and 26, respectively. This suggests that `jina` and `openai_3_large` models may struggle to capture only the relevant parts of the text compared to the `bge_m3` model. Alternatively, this might mean that the `bge_m3` and `openai_ada` models capture better the initial part of the texts since the facts always preceded the arguments in the texts.

Comparing the Long Text Handling Approaches

The results on Facts showed that any chunking strategy can significantly boost the performance of the models compared to the truncated approach. This result is in line with the previous findings [17]. Generally, when LCS was applied, the results increased compared to the results without applying it. On average, this resulted in a 1.40 increase in MRR, excluding the `openai`, `bge_m3`, `jina` and `fasttext` models, since chunking methods either did not apply to these solutions or showed no improvement with chunking. For Facts, applying LCS increased the average MRR by 0.84 (72.01→72.85), and for Arguments and Facts, the increase was 2.37 (62.15→64.52). However, focusing only on the best-performing `cohere` and `e5` models, LCS improved the average MRR by 1.81 (90.29→92.10) for Facts and by 2.87 (82.81→85.68) for Arguments and Facts on average. On Facts, pure chunking was the best approach only for the `hubert` and `sbert_hubert` methods. Pure chunking combined with LCS yielded the best results for the `danieleff` and `e5_large` models. Three models achieved the best results using the stride 16 + LCS setting (`cohere`, `mcontriever`, `e5_base`, `danieleff`). Additionally, we also observed that the stride 16 approach outperformed the stride 25% setting, which had been found to work best in previous research [17]. The best approach proved to be the `cohere` embeddings with stride 16 + LCS setting, reaching a 95.03 MRR score. The second best was the `bge_m3` embedding model with 93.67 MRR score where the chunking did not have an effect on the results on Facts. The OpenAI embeddings also worked well, especially the recently published `openai_3_large` model reaching a 93.05 score. The close fourth is the `e5_large` model reaching 92.45 with chunked+LCS approach. It is important to point out that the top four approaches (`cohere`, `bge_m3`, `openai_3_large` and `e5_large`) performed significantly better than all other approaches having small differences between these methods. Given the small differences between the methods it is less possible to discriminate between these models using the current dataset.

The results for the Facts and Arguments of the parties parts show a different picture. Five models provided the best results with truncated vectorization, excluding the OpenAI models (cohere, bge_m3, danielleff, e5_large, and mcontriever). The stride 16, stride 16 + LCS, stride 25% + LCS approaches proved to be the best once, respectively. This outcome is highly unexpected compared to the results on Facts alone. The reason for this is that the query fact drafts were created using only the Facts data from the documents, so adding the Arguments of the parties introduces noise into the retrieval process. In addition, the sentences with Fact labels usually precede the ones with Arguments of the parties labels in legal decisions. Therefore, the most relevant part of the A_F compound texts appears at the beginning of the document, resulting in the best retrieval result.

To see how the chunking techniques influenced the MRR scores on smaller models, Table 6. shows the average results of the different chunking methods on all of the vectorization models except the ones having 8192 context windows (bge_m3, openai and jina) and fasttext and showing the best three 512-sized models (cohere, e5_large and mcontriever).

Table 6. MRR average results for different chunking methods (multiplied by 100).

Chunking/Stride LCS	Trunc False	0 False	0 True	25% False	25% True	16 False	16 True
all_F	66.71	72.60	73.13	71.02	72.02	72.41	73.40
all_A_F	64.38	61.34	64.51	62.61	63.96	62.49	65.08
best_3_F	87.42	90.59	92.65	89.62	90.91	90.67	92.75
best_3_A_F	87.42	81.97	85.40	83.60	85.22	82.88	86.42

On average, the best chunking method was the stride 16 + LCS approach, with chunked + LCS coming in a close second (73.40 vs. 73.13 on the Facts, on Arguments and Facts 65.08 vs. 64.51). The same order was observed in the best-performing three models (92.75 vs. 92.65 on Facts). Interestingly, the 25% stride did not improve the results on Facts, but on the Arguments and Facts (on longer texts) it did, compared to stride 0, which is pure chunking. In contrast, stride 16 improved the performance on all texts compared to pure chunking.

Figure 3 summarizes the average MRR points gained by applying the different chunking strategies compared to vectorizing with truncated text.

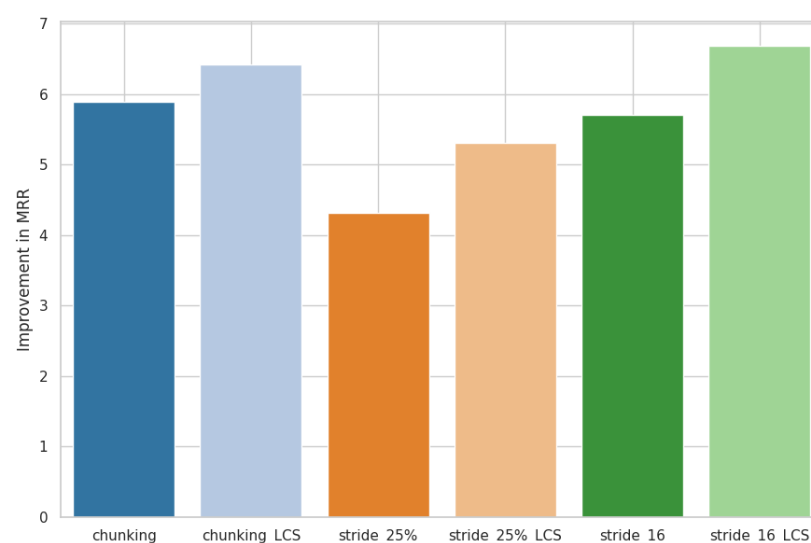


Figure 3. Average MRR improvement of the different chunking strategies compared to the truncated vectors (multiplied by 100) on Facts.

It can be clearly seen that all chunking strategies improved the quality of the embeddings. Also, in each case applying LCS improved the results. These results suggest that striding can improve the quality of the representations, but its effect varies depending on text length, with different impacts on shorter and longer texts.

5.2. Recall at n Results

The Recall at n results have been calculated for three n settings: 1, 3 and 5. The reason for choosing these settings is that according to the Click-Through Rates (CTR) of Google ranking positions (<https://firstpagesage.com/reports/google-click-through-rates-ctrs-by-ranking-position/> (accessed on 2 December 2024)), summed up to the 1st, 3rd and 5th ranks are 39.8%, 68.7%, and 81%, respectively. We assumed that the users of the legal database would behave similarly.

The $R@n$ results show a pattern very similar to the MRR results discussed above (see $R@1$ in Table 7, and $R@3$ and $R@5$ in Appendix A Tables A1 and A2). At first glance, the integer values may seem unusual, but remember that there are 100 examples, and averaging and multiplying the numbers with 100 ends up with having the number of documents appearing in the top n positions. The $R@1$ results show that 93% of the documents were retrieved in the first position by the cohere stride 16 + LCS method on Facts. The e5_large, jina and openai_3_large models closely followed, each retrieving 90% of the documents. The bge_m3 model reached 89%. The mcontriever model ranked sixth with 87%, followed by e5_base with 83%, openai_ada with 79%, and danielleff with 73%.

Figure 4 shows the 100-MRR@ n results in case of the best five methods, namely: the cohere, e5_large, jina_v3, openai_3_large, and bge_m3 methods. Note that the e5_large and openai_3_large models are completely identical in the plot. All methods improved by increasing n values, nevertheless, the clear winner was the cohere approach, probably due to the content quality awareness of the embedding.

Table 7. $R@1$ results (multiplied by 100).

Chunking/Stride LCS	Trunc False	0 False	0 True	25% False	25% True	16 False	16 True
bge_m3_A_F	83	83	83	83	83	83	83
bge_m3_F	89	89	89	89	89	89	89
cohere_A_F	90	82	84	82	82	83	85
cohere_F	91	89	92	86	89	92	93
danieleff_A_F	66	47	53	47	48	48	52
danieleff_F	72	70	73	68	71	70	73
fasttext_montana_A_F	33	33	33	33	33	33	33
fasttext_montana_F	45	45	45	45	45	45	45
hubert_A_F	1	3	2	2	2	4	2
hubert_F	6	8	7	6	6	7	7
e5_base_A_F	78	67	78	72	77	72	78
e5_base_F	80	82	83	82	83	83	83
e5_large_A_F	82	69	76	73	76	71	78
e5_large_F	82	85	90	84	86	84	89
jina_v3_A_F	65	-	-	-	-	-	-
jina_v3_F	90	-	-	-	-	-	-
mcontriever_A_F	79	73	78	74	76	74	78
mcontriever_F	76	84	85	85	87	84	87
openai_3_large_A_F	57	-	-	-	-	-	-
openai_3_large_F	90	-	-	-	-	-	-
openai_ada_A_F	73	-	-	-	-	-	-
openai_ada_F	79	-	-	-	-	-	-
sbert_hubert_A_F	24	37	38	41	42	39	38
sbert_hubert_F	28	50	46	46	49	49	49

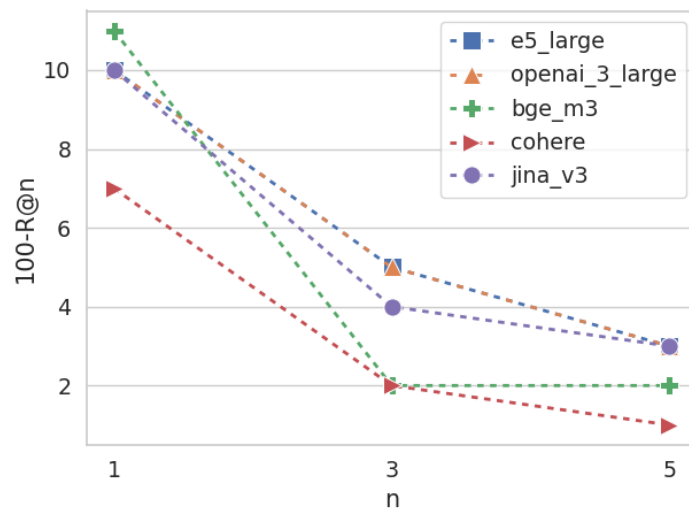


Figure 4. 100-R@n values for the best 5 approaches.

5.3. Cosine Similarity Difference Results

This metric measures the difference between the cosine scores of the first retrieved document and the ground truth document. Therefore, the best possible score is 0, and lower scores indicate better performance. Here we also followed the same color coding and bolding: the best scores by rows are highlighted with bolding the colors are changed from blue to red by going from bad to good scores. Table 8 contains the CSD results multiplied by 100.

Table 8. Cosine similarity difference (CSD) results (multiplied by 100)

Chunking/Stride	Trunc	0	0	25%	25%	16	16
LCS	False	False	True	False	True	False	True
bge_m3_A_F	0.48	0.49	0.49	0.49	0.49	0.49	0.49
bge_m3_F	0.20	0.20	0.20	0.20	0.20	0.20	0.20
cohere_A_F	0.43	0.41	0.38	0.49	0.41	0.42	0.36
cohere_F	0.28	0.25	0.20	0.36	0.29	0.26	0.21
danieleff_A_F	1.47	1.51	1.24	1.61	1.52	1.48	1.27
danieleff_F	1.14	1.05	0.67	0.76	0.73	0.86	0.68
fasttext_montana_A_F	0.98	0.98	0.98	0.98	0.98	0.98	0.98
fasttext_montana_F	0.96	0.96	0.96	0.96	0.96	0.96	0.96
hubert_A_F	10.94	6.39	7.81	7.55	8.14	6.62	7.81
hubert_F	11.13	6.93	9.07	8.10	8.94	7.04	8.97
e5_base_A_F	0.24	0.28	0.19	0.24	0.21	0.24	0.19
e5_base_F	0.20	0.19	0.12	0.16	0.15	0.17	0.11
e5_large_A_F	0.19	0.25	0.18	0.20	0.17	0.24	0.17
e5_large_F	0.15	0.19	0.12	0.17	0.14	0.17	0.12
jina_v3_A_F	2.37	-	-	-	-	-	-
jina_v3_F	0.55	-	-	-	-	-	-
mcontriever_A_F	0.90	0.73	0.55	0.67	0.54	0.77	0.63
mcontriever_F	1.57	0.56	0.67	0.74	0.76	0.68	0.68
openai_3_large_A_F	1.66	-	-	-	-	-	-
openai_3_large_F	0.32	-	-	-	-	-	-
openai_ada_A_F	0.35	-	-	-	-	-	-
openai_ada_F	0.31	-	-	-	-	-	-
sbert_hubert_A_F	12.19	3.43	3.37	3.38	3.36	3.48	3.37
sbert_hubert_F	10.69	3.57	3.53	3.61	3.66	3.55	3.52

In general, it can be observed that the average of the CSD scores obtained using the Arguments of the parties text were higher than those obtained using the Facts alone. This

also suggests that the noise in our data increased with the addition of the Arguments of the parties text. More specifically, the average CSD score of Facts was 1.48, while for Arguments of the parties, it was 1.78. Of course, this can only be considered noise if we assume that the Arguments of the parties section is not of interest to the users' search, which was the case in our study. For the `sbert_hubert` model with its small context window, it became clear that only the first context window alone was often insufficient to provide a good answer and that any chunking solution significantly improved performance.

Interestingly, the `bge_m3`, `cohere`, `e5`, and the `openai` models achieved CSD scores below 0.5, reflecting minimal differences. Regarding chunking strategies, two main approaches stood out: the stride 16 + LCS solution, and the chunked+LCS solution. However, these were only slightly outperformed by other solutions using the above-mentioned stride splitting method (see the `danielleff` model). The stride 25% approach also generally improved the results compared to the plain chunking for models with 512-token wide context windows. However, this trend was less consistent for the `sbert_hubert` model, similar inconsistencies were observed for the stride 16 approach.

On average, applying LCS lead to an increase in CSD scores from 1.62 to 1.72, a difference of 0.10. This result means that applying LCS increased the average distance from the best-ranked document. This result is the opposite compared to the previously experienced results.

However, when focusing only on the best-performing `cohere` and `e5_large` models, a different pattern emerges: by applying LCS on Facts, the CSD decreased by 0.05, and on Arguments and Facts by 0.06. This highlights that the CSD measure is highly dependent on vectorization methods, which limits its general application.

Overall, the results suggest that the best vectorization approach is the `cohere` embedding, followed by the `bge_m3`, `openai_3_large`, `jina`, and `e5_large` models as the next best-performing approaches, in order. In terms of handling the long texts, last chunk scaling generally improved in case of all chunking strategies. Chunking improved the embedding quality as well however, we have also revealed that striding can become contra-productive. In our study, the best setting was striding with 16 tokens for the top-performing `cohere` and `e5` models.

5.4. Validation with Experts

In order to validate the results, we involved five legal expert editors to compare the embedding models. Since models with longer context windows were not available at the time, we were only able to compare the smaller models. For the first part of the validation we downloaded 20 real use cases from a legal forum and compared only the `hubert`, `sbert_hubert` and `fasttext` representations, evaluating the retrieved documents using the Facts sections. The legal experts had to choose the best representation for each draft. In 13 cases, the `sbert_hubert` proved to be the best, while in 7 cases, none of the models were deemed acceptable.

For the second evaluation round, 43 custom-composed fact drafts were used, composed by the editors and the retrieved documents were evaluated using the Facts. We compared the previous best `sbert_hubert` and the new best `e5_large` and `cohere` embeddings using the best chunking method in each case. Similarly, the legal experts had to choose the best representation for each fact draft from the top 5 retrieved documents. The `cohere` model was selected 25 times, the `e5_large` 9 times and the `sbert_hubert` only once. In 8 cases, no model was deemed acceptable.

These results aligned with our MRR findings, further validating our approach.

6. Conclusions

In this study, we have investigated a semantic similarity search use case in the legal domain on a corpus containing 1172 Hungarian court decisions. We have established the foundations of an in-production semantic similarity search system for searching for cases with similar facts using only a draft of a legal fact.

Evaluating a retrieval system is not straightforward, as reliably assessing retrieval performance requires reviewing all retrieved documents. However, involving humans in this task is expensive, and the evaluation's reusability may be limited. We addressed this challenge by using OpenAI's gpt-3-turbo to create fact drafts for the legal cases. This allowed us to measure the rank of the original case among the retrieved documents, making the different vectorization methods quantitatively comparable. However, the automatically generated fact drafts contained many errors (keeping specific information, not responding properly, etc.) despite the explicit prohibition in the prompt. Therefore, for 100 documents (20 from each law area), the fact drafts were corrected by humans containing 3–4 sentences describing the facts briefly, not containing explicit information about numbers, geolocation information, etc. We compared one domain-trained fasttext model and eleven transformer-based vectorization solutions for vector-based retrieval. The best-performing models were in order: Cohere's embed-multilingual-v3.0, Beijing Academy of Artificial Intelligence's bge-m3, Jina AI's jina-embeddings-v3, OpenAI's text-embedding-3-large, and Microsoft's multilingual-e5-large. The Cohere model performed the best, likely because it produces a vector that captures both topic and content quality information. We also demonstrated that among the top-performing models with 8192 token context windows, the bge-m3 model was the most effective at capturing relevant details in texts with significant noise, surpassing by a significant 20-point MRR margin the jina-embeddings-v3 and text-embedding-3-large models, respectively. This result could be significant for other retrieval use cases, such as Retrieval Augmented Generation.

Transformer-based models have a context window that typically does not cover the entire document. We investigated and compared how chunking (splitting documents into context-window-sized chunks), striding (splitting documents into overlapping context-window-sized chunks), and last chunk scaling (downscaling the last chunk's vector by the ratio of the last chunk's size in tokens to the context window size) affect the quality of the embeddings compared to using only the first context window to generate embeddings. Our findings revealed that last chunk scaling efficiently improves the embedding quality, and all chunking methods outperformed the truncated setting. Striding results varied depending on the stride length, so this method should be evaluated with different stride values depending on the use case. The best results were achieved with a stride of 16 tokens, which represented 3.125% of the context window size for the best-performing Cohere and Microsoft (E5) models. These findings were used to implement an in-production semantic search system for a major legal content provider.

Author Contributions: Conceptualization, G.M.C., D.L., I.Ü. and R.V.; methodology, G.M.C., D.L. and R.V.; software, G.M.C., D.L. and I.Ü.; validation, R.V. and A.M.; investigation, G.M.C., D.L. and R.V.; resources, A.M.; data curation, G.M.C., D.L. and I.Ü.; writing—original draft preparation, G.M.C., D.L. and I.Ü.; writing—review and editing, D.N., I.Ü., J.P.V., G.M.C. and D.L.; visualization, G.M.C., D.L. and I.Ü.; supervision, A.M., R.V., J.P.V. and D.N.; project administration, D.N., R.V. and A.M.; funding acquisition, A.M. and J.P.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to being privately owned by Wolters Kluwer Hungary Ltd.

Conflicts of Interest: Author Gergely Márk Csányi, Dorina Lakatos, István Üveges, János Pál Vadász, Dániel Nagy and Renátó Vági were employed by the company MONTANA Knowledge Management Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A. Recall@3 and Recall@5 Tables

Table A1. R@3 results (multiplied by 100).

Chunking/Stride LCS	Trunc False	0 False	0 True	25% False	25% True	16 False	16 True
bge_m3_A_F	93	92	92	92	92	92	92
bge_m3_F	98	98	98	98	98	98	98
cohere_A_F	92	93	94	92	95	91	94
cohere_F	94	97	98	95	95	96	97
danieleff_A_F	79	64	69	61	66	63	67
danieleff_F	80	84	86	81	83	84	84
fasttext_montana_A_F	48	48	48	48	48	48	48
fasttext_montana_F	59	59	59	59	59	59	59
hubert_A_F	2	5	3	5	2	6	4
hubert_F	9	14	10	9	9	12	10
e5_base_A_F	84	83	85	81	83	83	86
e5_base_F	89	91	92	91	91	92	94
e5_large_A_F	92	83	88	87	90	84	88
e5_large_F	92	92	95	91	92	93	94
jina_v3_A_F	80	-	-	-	-	-	-
jina_v3_F	96	-	-	-	-	-	-
mcontriever_A_F	87	88	89	90	91	88	90
mcontriever_F	86	94	96	96	96	95	96
openai_3_large_A_F	76	-	-	-	-	-	-
openai_3_large_F	95	-	-	-	-	-	-
openai_ada_A_F	81	-	-	-	-	-	-
openai_ada_F	87	-	-	-	-	-	-
sbert_hubert_A_F	35	52	50	49	50	51	54
sbert_hubert_F	39	68	64	63	62	65	63

Table A2. R@5 results (multiplied by 100).

Chunking/Stride LCS	Trunc False	0 False	0 True	25% False	25% True	16 False	16 True
bge_m3_A_F	96	95	95	95	95	95	95
bge_m3_F	98	98	98	98	98	98	98
cohere_A_F	93	93	96	95	96	93	95
cohere_F	95	99	99	97	97	99	98
danieleff_A_F	82	73	77	73	74	76	79
danieleff_F	85	85	90	87	86	88	90
fasttext_montana_A_F	52	52	52	52	52	52	52
fasttext_montana_F	66	66	66	66	66	66	66
hubert_A_F	3	7	5	7	7	8	7
hubert_F	10	17	11	11	10	15	11
e5_base_A_F	89	86	89	85	90	84	89
e5_base_F	90	92	96	93	95	93	95
e5_large_A_F	93	88	93	93	94	89	95
e5_large_F	97	94	97	95	95	94	96
jina_v3_A_F	82	-	-	-	-	-	-
jina_v3_F	97	-	-	-	-	-	-
mcontriever_A_F	90	94	96	94	95	95	95
mcontriever_F	91	97	98	97	96	97	97
openai_3_large_A_F	83	-	-	-	-	-	-
openai_3_large_F	97	-	-	-	-	-	-
openai_ada_A_F	82	-	-	-	-	-	-
openai_ada_F	88	-	-	-	-	-	-
sbert_hubert_A_F	39	57	55	57	58	59	60
sbert_hubert_F	45	75	71	72	71	70	68

References

1. Csányi, G.M.; Vági, R.; Nagy, D.; Üveges, I.; Vadász, J.P.; Megyeri, A.; Orosz, T. Building a Production-Ready Multi-Label Classifier for Legal Documents with Digital-Twin-Distiller. *Appl. Sci.* **2022**, *12*, 1470. [\[CrossRef\]](#)
2. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5998–6008.
3. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [\[CrossRef\]](#)
4. Salton, G.; McGill, M.J. *Introduction to Modern Information Retrieval*; McGraw-Hill, Inc.: New York, NY, USA, 1986.
5. Qian, G.; Sural, S.; Gu, Y.; Pramanik, S. Similarity between Euclidean and cosine angle distance for nearest neighbor queries. In Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 14–17 March 2004. [\[CrossRef\]](#)
6. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781. [\[CrossRef\]](#)
7. Boncalo, O.; Amaricai, A.; Savin, V.; Declercq, D.; Ghaffari, F. Check node unit for LDPC decoders based on one-hot data representation of messages. *Electron. Lett.* **2015**, *51*, 907–908. [\[CrossRef\]](#)
8. Wu, L.; Hoi, S.C.H.; Yu, N. Semantics-Preserving Bag-of-Words Models and Applications. *IEEE Trans. Image Process.* **2010**, *19*, 1908–1920. [\[CrossRef\]](#)
9. Alaparthi, S.; Mishra, M. BERT: A sentiment analysis odyssey. *J. Mark. Anal.* **2021**, *9*, 118–126. [\[CrossRef\]](#)
10. Pal, A.; Rajanala, S.; Phan, R.C.W.; Wong, K. Self Supervised Bert for Legal Text Classification. In Proceedings of the ICASSP 2023—2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes Island, Greece, 4–10 June 2023; pp. 1–5. [\[CrossRef\]](#)
11. Yang, X.; Yang, K.; Cui, T.; Chen, M.; He, L. A Study of Text Vectorization Method Combining Topic Model and Transfer Learning. *Processes* **2022**, *10*, 350. [\[CrossRef\]](#)
12. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
13. Wang, J.; Huang, J.X.; Sheng, J. An efficient long-text semantic retrieval approach via utilizing presentation learning on short-text. *Complex Intell. Syst.* **2023**, *10*, 963–979. [\[CrossRef\]](#)
14. Limsopatham, N. Effectively leveraging BERT for legal document classification. In *Proceedings of the Natural Legal Language Processing Workshop 2021*; Association for Computational Linguistics: Punta Cana, Dominican Republic, 2021; pp. 210–216.
15. Zaheer, M.; Guruganesh, G.; Dubey, K.A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; et al. Big bird: Transformers for longer sequences. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17283–17297.
16. Beltagy, I.; Peters, M.E.; Cohan, A. Longformer: The Long-Document Transformer. *arXiv* **2020**, arXiv:2004.05150. [\[CrossRef\]](#)
17. Vatsal, S.; Meyers, A.; Ortega, J. Classification of US Supreme Court Cases using BERT-Based Techniques. *arXiv* **2023**, arXiv:2304.08649.
18. Dong, Z.; Tang, T.; Li, L.; Zhao, W.X. A survey on long text modeling with transformers. *arXiv* **2023**, arXiv:2302.14502.
19. Wan, X.; Yang, J. Document similarity search based on generic summaries. In Proceedings of the Information Retrieval Technology: Second Asia Information Retrieval Symposium, AIRS 2005, Jeju Island, Republic of Korea, 13–15 October 2005; Proceedings 2; Springer: Berlin/Heidelberg, Germany, 2005; pp. 635–640.
20. Vuong, T.H.Y.; Nguyen, H.L.; Nguyen, T.M.; Nguyen, H.T.; Nguyen, T.B.; Nguyen, H.T. NOWJ at COLIEE 2023: Multi-task and Ensemble Approaches in Legal Information Processing. *Rev. Socionetwork Strateg.* **2024**, *18*, 145–165. [\[CrossRef\]](#)
21. Vadász, J.P. Case Study for Measuring the Feasibility of a Semantic Search System. *Mil. Eng.* **2012**, *7*, 405–415.
22. Ebietomere, E.P.; Ekuobase, G.O. A semantic retrieval system for case law. *Appl. Comput. Syst.* **2019**, *24*, 38–48. [\[CrossRef\]](#)
23. Šavelka, J.; Ashley, K.D. Legal information retrieval for understanding statutory terms. In *Artificial Intelligence and Law*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 1–45.
24. Zhu, J.; Wu, J.; Luo, X.; Liu, J. Semantic matching based legal information retrieval system for COVID-19 pandemic. *Artif. Intell. Law* **2024**, *32*, 397–426. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Reimers, N.; Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv* **2019**, arXiv:1908.10084.
26. Louis, A.; van Dijk, G.; Spanakis, G. Interpretable long-form legal question answering with retrieval-augmented large language models. In Proceedings of the AAAI Conference on Artificial Intelligence, Vancouver, BC, Canada, 20–27 February 2024; Volume 38, pp. 22266–22275.
27. Martin, L.; Muller, B.; Suárez, P.J.O.; Dupont, Y.; Romary, L.; de La Clergerie, É.V.; Seddah, D.; Sagot, B. CamemBERT: A tasty French language model. *arXiv* **2019**, arXiv:1911.03894.
28. Hu, W.; Zhao, S.; Zhao, Q.; Sun, H.; Hu, X.; Guo, R.; Li, Y.; Cui, Y.; Ma, L. BERT_LF: A similar case retrieval method based on legal facts. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 2511147. [\[CrossRef\]](#)
29. Shao, Y.; Mao, J.; Liu, Y.; Ma, W.; Satoh, K.; Zhang, M.; Ma, S. BERT-PLI: Modeling paragraph-level interactions for legal case retrieval. In Proceedings of the IJCAI, Yokohama, Japan, 7–15 January 2020; pp. 3501–3507.
30. Ma, Y.; Shao, Y.; Wu, Y.; Liu, Y.; Zhang, R.; Zhang, M.; Ma, S. LeCaRD: A legal case retrieval dataset for Chinese law system. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Online, 11–15 July 2021; pp. 2342–2348.

31. Wang, L.; Yang, N.; Huang, X.; Yang, L.; Majumder, R.; Wei, F. Multilingual E5 Text Embeddings: A Technical Report. *arXiv* **2024**, arXiv:2402.05672.
32. Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; Liu, Z. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. *arXiv* **2024**, arXiv:2402.03216.
33. Sturua, S.; Mohr, I.; Akram, M.K.; Günther, M.; Wang, B.; Krimmel, M.; Wang, F.; Mastrapas, G.; Koukounas, A.; Wang, N.; et al. jina-embeddings-v3: Multilingual Embeddings With Task LoRA. *arXiv* **2024**, arXiv:2409.10173.
34. Csányi, G.M.; Vági, R.; Gadó, K.; Üveges, I.; Nagy, D.; Bajári, L.; Megyeri, A.; Fülöp, A.; Vadász, J.P. Building a Production-ready, Hierarchical Subject Matter Classifier for Legal Decisions. *Mach. Learn. Knowl. Extr.* **2024**, Under review.
35. Osváth, M.; Yang, Z.G.; Kósa, K. Analyzing Narratives of Patient Experiences: A BERT Topic Modeling Approach. *Acta Polytech. Hung.* **2023**, *20*, 153–171. [[CrossRef](#)]
36. Nemeskey, D.M. Introducing huBERT. In Proceedings of the XVII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2021), Szeged, Hungary, 28–29 January 2021 ; pp. 3–14.
37. Muennighoff, N.; Tazi, N.; Magne, L.; Reimers, N. MTEB: Massive text embedding benchmark. *arXiv* **2022**, arXiv:2210.07316.
38. Izacard, G.; Caron, M.; Hosseini, L.; Riedel, S.; Bojanowski, P.; Joulin, A.; Grave, E. Unsupervised dense information retrieval with contrastive learning. *arXiv* **2021**, arXiv:2112.09118.
39. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 3111–3119.
40. Varga, D.; Halácsy, P.; Kornai, A.; Nagy, V.; Németh, L.; Trón, V. Parallel corpora for medium density languages. *Amst. Stud. Theory Hist. Linguist. Sci. Ser. 4* **2007**, *292*, 247.
41. Bajaj, P.; Campos, D.; Craswell, N.; Deng, L.; Gao, J.; Liu, X.; Majumder, R.; McNamara, A.; Mitra, B.; Nguyen, T.; et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv* **2016**, arXiv:1611.09268.
42. Wenzek, G.; Lachaux, M.A.; Conneau, A.; Chaudhary, V.; Guzmán, F.; Joulin, A.; Grave, E. CCNet: Extracting high quality monolingual datasets from web crawl data. *arXiv* **2019**, arXiv:1911.00359.
43. Conneau, A.; Khandelwal, K.; Goyal, N.; Chaudhary, V.; Wenzek, G.; Guzmán, F.; Grave, E.; Ott, M.; Zettlemoyer, L.; Stoyanov, V. Unsupervised cross-lingual representation learning at scale. *arXiv* **2019**, arXiv:1911.02116.
44. Wang, L.; Yang, N.; Huang, X.; Jiao, B.; Yang, L.; Jiang, D.; Majumder, R.; Wei, F. Text embeddings by weakly-supervised contrastive pre-training. *arXiv* **2022**, arXiv:2212.03533.
45. Kusupati, A.; Bhatt, G.; Rege, A.; Wallingford, M.; Sinha, A.; Ramanujan, V.; Howard-Snyder, W.; Chen, K.; Kakade, S.; Jain, P.; et al. Matryoshka representation learning. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 30233–30249.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.