D17 – Middleware search method

Project funded by the European Union's Next Generation EU through the Slovak Republic's Recovery and Resilience Plan under project no. 09I05-03-V02-00049.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
ÚRAD PODPREDSEDU VLÁDY SLOVENSKEJ REPUBLIKY PRE PLÁN OBNOVY A ZNALOSTNÚ EKONOMIKU
VAIA VÝSKUMNÁ A INOVAČNÁ AUTORITA
VÝSKUMNÁ AGENTÚRA

INTRODUCTION

The project **"Automation of Legal Text Analysis based on Machine** Learning" (hereinafter "ALTAML") is a key initiative aimed at integrating innovative approaches in the field of data processing and subsequent analysis, with a focus on the field of law, which also includes information and communication technology law. The aim of this project is to develop and validate effective methods for the automated analysis of legal texts using machine learning techniques. This involves, in particular, the development of tools that facilitate the processing and analysis of large volumes of data in the form of various legal documents, as well as the extraction of relevant information (attributes) from such documents, including the identification of key terms, references to legislation and other attributes.

The aim of the ALTAML project is therefore to contribute to more efficient access to legal information and to the acceleration of legal processes, thereby ensuring a higher degree of legal certainty and improving the availability of legal texts, the results of their analysis and relevant legal information for both professionals and the general public.

As part of work package KPB3**, D17 – Middleware search method**– was developed, the purpose of which is to assist users in formulating search queries for court decisions and legal regulations.

When searching for legal documents, users often formulate queries using specific legal or factual terms and references to legal provisions. However, a problem arises in situations where the user knows only the general topic they wish to search for, but when entering the query, they may not immediately think of all the related terms and legal provisions that could lead to relevant results. The method's input can therefore also be a broadly formulated topic, such as 'cybercrime'.

For such an input, the method first uses semantic search to find semantically related sections of court decisions and legislation. From the semantically identified sections of court decisions, it then proposes search terms that appear directly in these decisions, such as "IP address", "computer network" or "website". At the same time, it selects relevant legal provisions from the retrieved legislation, such as § 360b of the Criminal Code concerning dangerous electronic harassment or § 247a of the Criminal Code governing unauthorised interference with a computer system.

The output of the method is a structured response containing recommended search terms and identifiers of relevant legal provisions. The user can select from the suggested values those terms and legal provisions they wish to use in further searches.

The method has been implemented as a working prototype on a corpus of court decisions and legal regulations.

1. Configuration of search indexes

The intermediary search method uses separate Elasticsearch indices for court decisions and legal provisions. Both indices are designed to support semantic search using vector representations of text. The aim of these indexes is not to cover all attribute filters defined in the D16 query model, but to provide input documents for the intermediary method, which subsequently generates recommended search terms and relevant legal provisions from them.

The sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 model is used to create vector representations, which generates 384-dimensional embeddings. The same model is used for both document indexing and user query processing, so that the documents and the input query are represented in the same vector space. The following sub-sections therefore describe the configuration of the court decision index and the legal provision index.

1.1. Index of court decisions

Court decisions are stored in the index at the level of smaller text segments, hereinafter referred to as 'chunks'. An entire decision may be extensive; therefore, a single vector representation of the entire document may not capture the meaning of its individual parts with sufficient accuracy.

The chunking of court decisions is performed on the main sections of the document identified during pre-processing, specifically the introduction, the operative part, the reasoning and the instructions. The text of each section is first divided into text paragraphs. If any paragraph is longer than the maximum chunk size, it is further divided by words. Subsequently, the shorter text segments are assembled into chunks so that they do not exceed the specified maximum size. In the prototype, the maximum chunk size is set to 400 characters. If the last chunk has fewer than 100 characters, it is merged with the previous chunk, provided the resulting text does not exceed the maximum chunk size.

Each chunk stored in the index contains the identifier of the original decision, the section designation, the chunk text and its vector representation (embedding). The `section` field makes it possible to distinguish whether the chunk originates, for example, from the introduction, the operative part, the reasoning or the instructions. During a search, the user query is converted into an embedding and compared with the embeddings of individual chunks.

This information is also reflected in the structure of documents stored in the Elasticsearch index. The `mappings` section specifies the field names, their data types and how they are used in searches. A more detailed description of the Elasticsearch index settings and the data types used is provided in Output D16 – Query model for searching legal texts.

The configuration of the `mappings` section for the court decisions index is available at the Example 1 .

```
{
  "mappings": {
    "properties": {
      "doc_id": {
        "type": "keyword"
```

```
        },
        "chunk_id": {
          "type": "integer"
        },
        "section": {
          "type": "keyword"
        },
        "text": {
          "type": "text"
        },
        "embedding": {
          "type": "dense_vector",
          "dims": 384,
          "index": true,
          "similarity": "cosine"
        }
      }
    }
  }
```

Example 1 - Configuration of the court decisions index.

Table 1 shows the basic structure of a document stored in the court decisions index.

Table 1 - Attributes of the court decision index.

Attribute	**Type**	**Meaning**
`doc_id`	`keyword`	identifier of the original court decision
`chunk_id`	`integer`	sequence number of the text chunk within the decision
`section`	`keyword`	the part of the decision from which the chunk originates
`text`	`text`	text of the chunk used as context for the language model
`embedding`	`dense_vector`	vector representation of a chunk used in semantic search

1.2. Index of legal provisions

Legal provisions are stored in the index at various levels of detail, specifically at the level of a section, subsection or clause. Unlike court decisions, the texts of legal provisions are not further divided into smaller chunks, as individual provisions are generally short enough to allow for the direct creation of a single vector representation.

From the perspective of the retrieval method, three pieces of data are important for a legal provision: the `doc_id` identifier, the text of the provision, and its vector representation. The identifier is provided to the language model and also forms the value returned in the `legal_provisions` array. The text serves as context for the language model, and the embedding is used in semantic search.

Example 2 contains the configuration of the *`mappings`* section for the legal provisions index.

```
{
  "mappings": {
    "properties": {
      "doc_id": {
        "type": "keyword"
      },
      "text": {
        "type": "text"
      },
      "embedding": {
        "type": "dense_vector",
        "dims": 384,
        "index": true,
        "similarity": "cosine"
      }
    }
  }
}
```

Example 2 – Configuration of the legal provisions index.

Table 2 shows the basic structure of a document stored in the legal provisions index.

Table 2 – Attributes of the legal provisions index.

Attribute	Type	Meaning
doc_id	keyword	legal provision identifier
text	text	text of the legal provision used in creating the embedding and as context for the language model
embedding	dense_vector	vector representation of the legal provision used in semantic search

1.2.1. Hierarchical storage of legal text

Legal provisions are stored in the index at multiple levels of detail. The same part of the legal text can therefore be represented by both a more general and a more specific document. A more general document contains a broader textual context, whilst a more specific document contains only the relevant part of the provision and its superordinate context.

At the highest level, the entire section may be stored, including all its subsections and clauses:

```
{ "id": "/SK/ZZ/2005/300/20260701.html#paragraf-10",
  "text": "§ 10 Title of the section or the opening sentence of the section.
            (1) Text of the first paragraph.
            (2) Text of the second paragraph.
                  a) Text of sub-section a).
                  b) Text of subparagraph (b)." }
```

A specific paragraph may be stored at a lower level. In this case, the document contains the title or introductory sentence of the section and the text of the relevant paragraph, including its sub-paragraphs:

{ "id": "/SK/ZZ/2005/300/20260701.html#section-10.**paragraph-2**",
"text": "**§ 10** Title of the section or opening sentence of the section.
(2) Text of the second sub-paragraph.
a) Text of subparagraph (a).
b) Text of subparagraph (b)." }

If a paragraph is divided into sub-sections, a document may also be stored for a specific sub-section. In this case, it contains the parent context of the section and paragraph, but no longer contains the other sub-sections at the same level:

{ "id": "/SK/ZZ/2005/300/20260701.html#section-10.paragraph-2.**point-a**",
"text": "**§ 10** Title of the section or introductory sentence of the section.
(2) Text of the second paragraph.
a) Text of subparagraph (a)."}

This method of storage is necessary to ensure that even the more specific parts of a legal provision remain semantically complete. The legal text is thus represented in the index at multiple levels: more general documents capture the broader context of the provision, whilst more specific documents represent a narrower and more precisely defined part of the legal text. In semantic search, this makes it possible to find either a more broadly related provision or a more detailed part of the provision, depending on the meaning of the user's query.

If the letter were stored on its own, without the section title, introductory sentence or text of the relevant paragraph, its meaning might not be clear. For example, the letter itself:

{ "id": "/SK/ZZ/2005/300/20260701.html#paragraph-10.section-2.letter-a",
"text": "**a)** Text of letter a)." }

2. Procedure of the intermediary method

The intermediary method processes user input in the form of a topic or a briefly formulated query. The method's output is not a final Elasticsearch DSL query, but a structured proposal of values that the user can subsequently use in further searches. The method returns two groups of results: recommended search terms and identifiers of relevant legal provisions.

The overall procedure of the method is illustrated inFigure 1 . The method consists of several steps. First, the user input is converted into an embedding. This is used for a semantic search of related chunks of court decisions and legal provisions. Textual contexts are then created from the search results, which are fed into two separate calls to the language model. The first call generates recommended search terms and the second call selects relevant legal provisions. The results of both calls are finally merged into a single JSON output.

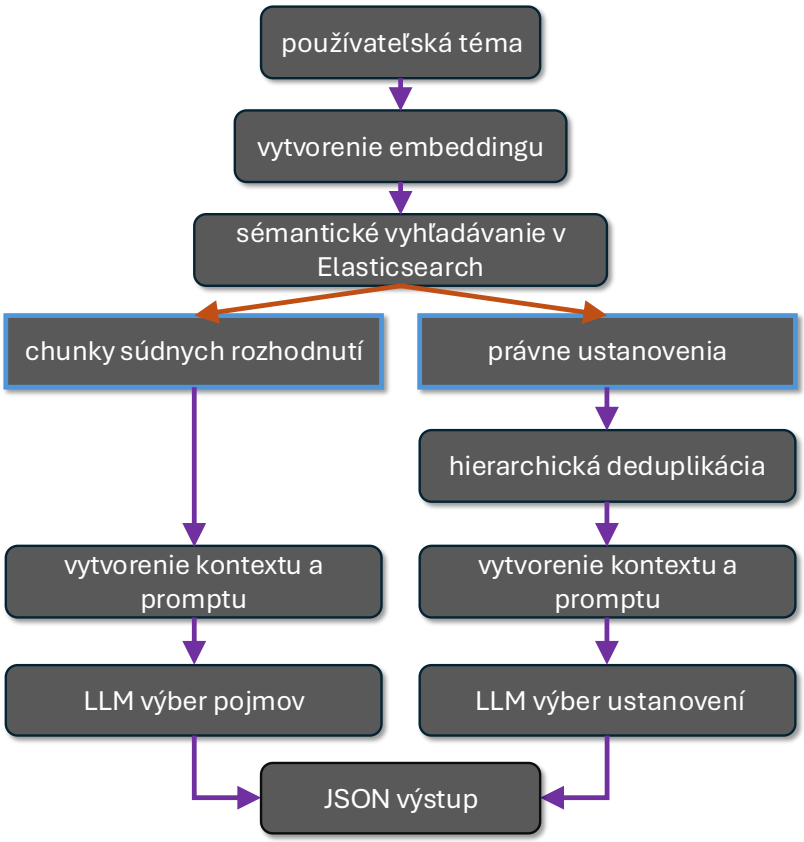


Figure 1 - Procedure of the intermediary search method.

2.1. Creating a topic embedding from the input

The user's input topic is first converted into a vector representation using the same embedding model as that used for indexing court decisions and legal provisions. The prototype uses the *sentence- transformers/paraphrase-multilingual-MiniLM-L12-v2,* which generates 384-dimensional vector representations.

Example 3 demonstrates the creation of an embedding for the input topic 'cybercrime'.

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-
MiniLM-L12-v2")
query_vector = model.encode("cybercrime").tolist()
```

Example 3 – Creating a topic embedding from the input.

2.2. Semantic search for related documents

The same embedding (**`query_vector`**) is then used to search two indexes: the court decisions index and the legal provisions index. In both cases, this involves k-nearest neighbours (kNN) search, i.e. searching for the vectors closest to the input vector of the user's topic.

In the case of court rulings, the embedding (**`query_vector`**) is compared with the embeddings of individual chunks of the rulings.Example 4 shows an Elasticsearch query for finding the most similar chunks of court rulings.

```
                        {
                          "knn": {
                            "field": "embedding",
                            "query_vector": [0.12, ..., 0.44], /*384-dimensional vector
*/
                            "k": 10,
                            "num_candidates": 100
                          },
                          "_source": [
                            "doc_id",
                            "chunk_id",
                            "section",
                            "text"
                          ]
                        }
```

Example 4 – Elasticsearch KNN query for searching for chunks of court decisions.

The `field` specifies which vector field to search against. In this case, it is the `embedding` field. The `query_vector` value represents the embedding of the user's input query. The k parameter specifies the number of results to be returned. In the prototype, the default value is 10, meaning that the ten most similar results are retrieved from each index. The `num_candidates` parameter specifies the number of candidate vectors from which Elasticsearch selects the closest results.

The `_source` section specifies which document fields are to be returned from the index. For court decisions, the identifier of the original decision, the chunk sequence number, the decision section and the chunk text are returned. The embedding itself is not returned in the result, as it is only required for calculating similarity.

The same type of KNN query is also used for the legal provisions index. The only difference is that legal provisions are not divided into chunks, so only the provision identifier and its text are returned from the index.Example 5 shows a simplified query for the legal provisions index.

```
                       {
                        "knn": {
                         "field": "embedding",
                         "query_vector": [0.12, ..., 0.44], /*384-dimensional vector
*/
                            "k": 10,
                            "num_candidates": 100
                          },
                          "_source": [
                            "doc_id",
                            "text"
                          ]
                        }
```

Example 5 – Elasticsearch KNN query for searching legal provisions.

This step yields two sets of candidate documents: semantically similar chunks of court decisions and semantically similar legal provisions. These results do not yet constitute the final output of the method. They serve as input for the subsequent steps, in which the language model proposes search terms and selects relevant identifiers for legal provisions.

2.3. Generation of recommended search terms

The results of the semantic search of court decisions are used as input for the language model, whose task is to suggest related search terms that expand on the user's original topic. For the topic 'cybercrime', the model might suggest terms such as 'IP address', 'computer network', 'website' or 'computer data'.

A textual context is first created from the retrieved chunks of court decisions. Each chunk is marked with an identifier in the form [Decision #] and contains the text of the relevant chunk. Individual chunks are combined into a single text context and separated by the --- tag. The prototype uses a value of 10, so the context includes a maximum of ten of the most relevant chunks from court decisions. A shortened example of such a context is shown in theExample 6 .

```
[Decision 1]
The defendant unlawfully gained access to user data via a website...
---
[Decision 2]
The court considered unauthorised interference with a computer system and the manipulation of
computer data...
---
• • •
---
[Decision 10]
The IP address was used to identify the device used in the attack...
```

Example 6 - Context created from extracted sections of court decisions.

The language model call is divided into a system prompt and a user prompt. The system prompt defines the general role of the model, as shown in theExample 7 .

```
It is a component of the legal RAG system designed to extract search terms.
```

Example 7 – System prompt for generating recommended search terms.

The user prompt contains the user's original topic, the specific task, the rules for generating search terms, and the context of the retrieved court decisions. The context is inserted at the end of the prompt between tags marking the start and end of the retrieved documents, as shown at the Example 8 .

```
User's original question:
cybercrime

Task:
Suggest 5 to 10 relevant search terms for the user's question based on the court decisions found.

Instructions:
- Use only the court decisions found.
- All search_phrases must be in Slovak.
- Do not translate Slovak terms into English.
- If a term appears in Slovak in the retrieved decisions, retain the Slovak wording.
- Only include English terms if they appear exactly as such in the retrieved decisions and have no
```

```
suitable Slovak equivalent.
- Derive search terms from the meaning and terminology of the retrieved decisions.
- Each search term should be a single word or a short phrase of no more than four words.
- Do not return duplicate terms or terms with identical meanings.
- Do not include the names of legal acts, sections, provision numbers, file references or citations.
- If the retrieved documents do not contain terms relevant to the query, return an empty list of
search_phrases.
- Do not invent search terms without support in the retrieved documents.

--- START OF SEARCHED DOCUMENTS ---
{documents_context}
--- END OF SEARCHED DOCUMENTS ---
```

Example 8 - User prompt for generating recommended search terms.

To support structured output, a JSON schema is used when calling the model. The JSON schema defines the expected structure of the response, i.e. which fields the output should contain and what data type their values should have. It only allows the `search_phrases` field, whose value is a list of strings. The schema used for this step is shown at the Example 9 .

```
{
  "type": "object",
  "properties": {
     "search_phrases": {
        "type": "array",
        "items": {
           "type": "string"
         }
      }
   },
   "required": ["search_phrases"],
   "additionalProperties": false
  }
```

Example 9 – JSON schema of the output for recommended search terms.

Once a response is received, any empty values are removed from the result and the number of returned terms is limited to 10. An example of the resulting output is shown at the Example 10 .

```
{
  "search_phrases": [
     "IP address",
     "computer network",
     "website",
     "computer data"

 }
```

Example 10 - Example of the output generated for recommended search terms.

2.4. Selection of relevant legal provisions

The second output of the method is a list of relevant legal provisions. This step follows on from the semantic search for legal provisions described in the sub-section2.2 and works with the query results displayed atExample 5 .

For legal provisions, hierarchical deduplication is performed after the semantic search for candidates. This addresses situations where both a more general and a more specific provision are found in the results. If the results contain, for example, both a section identifier and a subsection identifier, the more general section identifier is removed and the more specific provision is retained.

The legal provisions amended in this way form the textual context for the language model. Each provision contains a serial number, an ID, and the text of the provision. An example of an abridged context can be viewed at the Example 11 .

```
[Legal provision 1]
ID: /SK/ZZ/2005/300/20260701.html#paragraph-247a.section-2.letter-c
Text:
Unauthorised interference with a computer system
(1) The offender shall be punished by imprisonment for a term of one to five years if he commits the act referred to in paragraph 1
  c) by misusing another person's personal data with the aim of gaining the trust of a third party,
---
[Legal provision 2]
ID: /SK/ZZ/2005/300/20260701.html#section-360b.paragraph-1.letter-a
Text:
Dangerous electronic harassment
(1) Whoever intentionally, through an electronic communications service...
  a)  humiliates or intimidates them over a prolonged period, or acts on their behalf without authorisation...
---
. .
```

Example 11 – Context created from the legal provisions searched for.

As with the generation of search terms, this step also uses a separate system and user prompt. The system prompt defines the role of the model as a component designed to select relevant legal provisions from the searched context, as shown in theExample 12 .

```
You are a component of the legal RAG system designed to select relevant provisions from the retrieved list.
```

Example 12 - System prompt for selecting relevant legal provisions.

The user prompt for selecting legal provisions is displayed at theExample 13. The model should select only from the provisions presented in the context displayed at theExample 11 and return only the values listed under **ID**: in the output.

```
User's original question:
cybercrime

Task:
Select the relevant legal provisions relating to the user's question from the attached list of legal
```

```
provisions.

Instructions:
- Return only the values listed under "ID:" to the legal_provisions field.
- Do not return the text of the legal provision.
- Do not invent IDs for legal provisions.
- If the legal provisions found do not contain any relevant provisions for the query, return an empty list of legal_provisions.

--- START OF SEARCHED LEGAL PROVISIONS ---
{laws_context}
--- END OF SEARCHED LEGAL PROVISIONS ---
```

Example 13 - User prompt for selecting relevant legal provisions.

To support structured output, a JSON schema is used with a `legal_provisions` field, whose value is a list of strings. The schema is displayed at the Example 14 .

```
{
  "type": "object",
  "properties": {
    "legal_provisions": {
        "type": "array",
        "items": {
           "type": "string"
         }
     }
  },
  "required": ["legal_provisions"],
 "additionalProperties": false
}
```

Example 14 – JSON output schema for relevant legal provisions.

Once a response is received, any empty values are removed from the result. An example of the output for the selection of relevant legal provisions is shown at the Example 15.

```
{
  "legal_provisions": [
    "/SK/ZZ/2005/300/20260701.html#paragraf-247a.odsek-2.pismeno-c",
    "/SK/ZZ/2005/300/20260701.html#paragraf-247b.odsek-3.pismeno-b",
    "/SK/ZZ/2005/300/20260701.html#section-247b.paragraph-2.point-c",
    "/SK/ZZ/2005/300/20260701.html#section-360b.paragraph-1.point-b",
    "/SK/ZZ/2005/300/20260701.html#section-360b.paragraph-1.letter-a"
  ]
}
```

Example 15 – Example of the output for selecting relevant legal provisions

The legal provision identifier contains information about the legal act, its version and the specific part of the provision. The section /SK/ZZ/2005/300/20260701.html identifies the legal act and its version – in the example given, 2005/300 represents the identifier of the Criminal Code and 20260701 indicates the version of the legal act. The part following the #

symbol then specifies the specific provision, for example Section 247a, paragraph 2 and subparagraph (c).

2.5. The final output of the intermediary method

The outputs from the previous steps are combined into a single JSON object at the end. The `search_phrases` array contains recommended search terms obtained from the of court decisions, and the `legal_provisions` array contains the identifiers of relevant legal provisions selected from the retrieved list of legal provisions.

An example of the resulting output of the intermediary method is shown in the example 16.

```
{
  "search_phrases": [
    "IP address",
    "computer network",
    "website",
    "computer data"
  ],
  "legal_provisions": [
    "/SK/ZZ/2005/300/20260701.html#section-247a.paragraph-2.letter-c",
    "/SK/ZZ/2005/300/20260701.html#section-247b.paragraph-3.letter-b",
    "/SK/ZZ/2005/300/20260701.html#section-247b.paragraph-2.point-c",
    "/SK/ZZ/2005/300/20260701.html#section-360b.paragraph-1.point-b",
    "/SK/ZZ/2005/300/20260701.html#section-360b.paragraph-1.letter-a"
  ]
}
```

Example 16 - Resulting JSON output of the intermediary method

3. Running the demo project

A working prototype of the intermediary method is available in the project's GitLab repository: https://gitlab.science.upjs.sk/altaml/middleware-search-method.

The repository also includes the demo script **demo_cybercrime.py**, which is used to verify the method's functionality on the topic of 'cybercrime'.

Before running the demo script, you must follow the instructions in the **README.md** file, which contains information on setting up the environment, configuring the connection to Elasticsearch, connecting to the language model, and creating search indexes. The demo data consists of a subset of court decisions and legal regulations intended for quick verification of the prototype's functionality.