Innovating Sales and Planning of Complex Industrial Products
Exploiting Artificial Intelligence

Deliverable 3.2
Semantic Search

Deliverable type:	Document
Deliverable reference number:	ITEA 20054 | D3.2
Related Work Package:	WP 3
Due date:	2024-05-30
Actual submission date:	2024-06-13
Responsible organisation:	Natif
Editor:	Nico Herbig
Dissemination level:	Confidential
Revision:	Submission

Abstract:	The D3.2 Semantic Search deliverable describes (i) the importance of semantic search to optimize sales processes, (ii) the different approaches to semantic search explored within InnoSale, (iii) how the developed software solutions can be used and integrated into the overall InnoSale solution.

Keywords:	Semantic Search, Software Specification, Integration

Table_head	Name 1 (partner)	Name 2 (partner)	Approval date (1 / 2)
Approval at WP level	ERSTE	ERMETAL	27.05.2024
Veto Review by All Partners			13.06.2024

Editor
Dr. Nico Herbig (Natif)

Contributors
Dr. Nico Herbig (Natif)
Rashid Nizamani (Natif)
Sepideh Sobhgol (IFAK)
Mario Thron (IFAK)
Arttu Lämsä (VTT)
Sari Järvinen (VTT)
Mert Daloğlu (DAKIK)
Bilge Özdemir (DAKIK)
Yazmin Andrea Pabon Guerrero (UC3M)

Executive Summary

Semantic search is a crucial tool to speed up sales processes of complex products, as it can help sales engineers in quicker understanding of customer inquiries (often formulated in laymen's terms), finding similar products developed in the past that can be used as a blueprint for the current offer, or searching within meeting notes that discussed customer requirements. Within the InnoSale consortium, the industrial partners provided a variety of use cases that can be simplified with semantic search technology.

This deliverable presents those use cases on high-level, and then focuses on the technical solutions implemented and how they can be integrated within the overall InnoSale solution. The presented approaches differ mainly in the amount of explicit modelling required beforehand, where more explicit modelling (ontologies, knowledge graphs, rules) can drastically reduce the data quantity needed and enhance the explainability of the output, whereas implicitly modelled approaches need to learn this knowledge from data, thus requiring more data, but potentially being able to capture more difficult semantic links. Thus, depending on the use case requirements and data availability, different technical solutions were developed and are presented in the following: (1) Semantic Inquiry Understanding Using Named Entity Recognition (NATIF), (2) Semantic Search Within Automatically Transcribed Meeting Notes (DAKIK), (3) Using an Ontological Database to Search for Semantically Similar Historical Projects (IFAK), (4) SentenceBERT-based Semantic Search for Historical Projects (VTT), (5) Semantic Search Using an Ontology and a Knowledge Graph (UC3M).

Table of Content

Figures

Tables

1 Introduction

Semantic search is an essential tool to speed up sales processes. A common problem of the use case partners is the handling of customer inquiries: Customers formulate their product requirements in their own vocabulary. A sales engineer then needs to semantically understand the inquiry and map it to the wording used within the organization. Then he needs to check if historical projects are similar to the current inquiry, such that parts of the historic planning can be reused, dramatically speeding the time up to offer. Semantic search can (partially) automate this expensive process of understanding and searching.

Another reoccurring problem of the use case partners is the semantic search of decisions taken during meetings, which can also be addressed by indexing a database of meeting transcripts to be searchable for semantic concepts.

This deliverable presents the different solutions developed for semantic search within the InnoSale project, showcases how they contribute to speeding up the use case partners' sales processes, and how the different software components can be used and integrated into the overall InnoSale solution.

The components developed in T3.2 work tightly together with other components developed in Work Packages 3 and 4 and can be assembled into an overall platform as described in D5.1.

2 Semantic Inquiry Understanding Using Named Entity Recognition (NATIF)

For Semantic Inquiry Understanding using Named Entity Recognition (NER), we first present the addressed use case, then the technical solution and the reason why we decided on this approach, before ending with the possibilities to integrate the service.

2.1 Use Case

Customers usually get in contact with the Use Case Partner DEMAG via email / a contact form. In their inquiry, they describe the product specifications. Here, customer expertise differs drastically. Some customers know exactly what they need to specify and which words to use, as they have bought complex products from DEMAG in the past. Others only roughly describe their needs in the language used within their company, sometimes forgetting to specify some relevant information. A sales engineer is therefore needed to understand the customer's intentions and map them to a concrete product configuration. In that process, missing information must be identified, and either must be deducible from other given information or requested from the customer. Once the product configuration is complete, historical projects with similar parameters need to be found, as they can guide the sales engineer in engineering the product and finding a suitable price.

2.2 Technical Solution

The developed solution relies on NER to transform the unstructured inquiries into a structured product configuration. Figure 1 illustrates the approach.

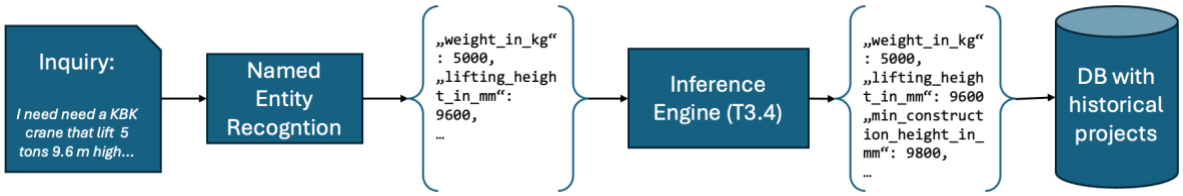


Figure 1: Semantic Inquiry Understanding via NER

A free-form text inquiry (received via email or contact form) is received and sent as is to the NER component. The NER is trained to extract all relevant product specifications from the inquiry and to structure them in a JSON format. The component can work in tandem with the Inference Engine developed in T3.4, which can take the structured output and use its rule base and inferencing to deduce further, not explicitly stated information, thereby enriching the structured output. In the example above, it noticed that a crane that can lift objects up to 9.6 m high, needs to be at least 9.8 meters high, which can guide the project engineering. Finally, the structured JSON, containing all explicitly found requirements and deduced implicit ones can be used to perform normal SQL queries against a database of historical projects. Basically, semantic understanding happens first through modern AI algorithms, such that the search can be kept simple.

This NER-based approach can be seen as an explicit problem modelling. An alternative approach (that was initially explored) would be to learn a proper embedding so that inquiry and semantically similar historical projects are close to each other in a hyperdimensional space (see Sections 3 or 5). However, in the scope of complex engineered products, as we have in this use case, explicit modelling has the advantage that it not only provides similar projects but also can be used to pre-fill configuration masks. Furthermore, the explicit nature is easier to understand (explainable AI) and has drastically reduced data needs because it only requires training data (hundreds of inquiries), and not the whole database of historical projects. Within the legal constraints regarding data sharing within the consortium, it would have been impossible to get access to all historical projects. Without that access, however, implicit modelling could only output semantically similar historical projects that are embedded, so for which data was shared. With the NER approach, the model just needs to structure the unstructured input, so that the use case partner can perform normal SQL-based search, not requiring any AI expertise. Furthermore, the explicit modelling allows the Use Case partner to clearly define what “similar” means based on hard criteria, like “a KBK crane can only be similar to other KBK cranes”, or “the lifting weight must not differ by more than 1 ton”. Such criteria are well-known by sales engineers, but would all need to be learned in implicit modelling, requiring lots of data and being error-prone.

To train our models, we first needed annotation data. For this, we extended our annotation tools, such that relevant fields appearing in cranes can be easily marked on the inquiries, as shown in Figure 2. The process is quite straightforward: select an entity on the left and select all text belonging to this entity on the right.

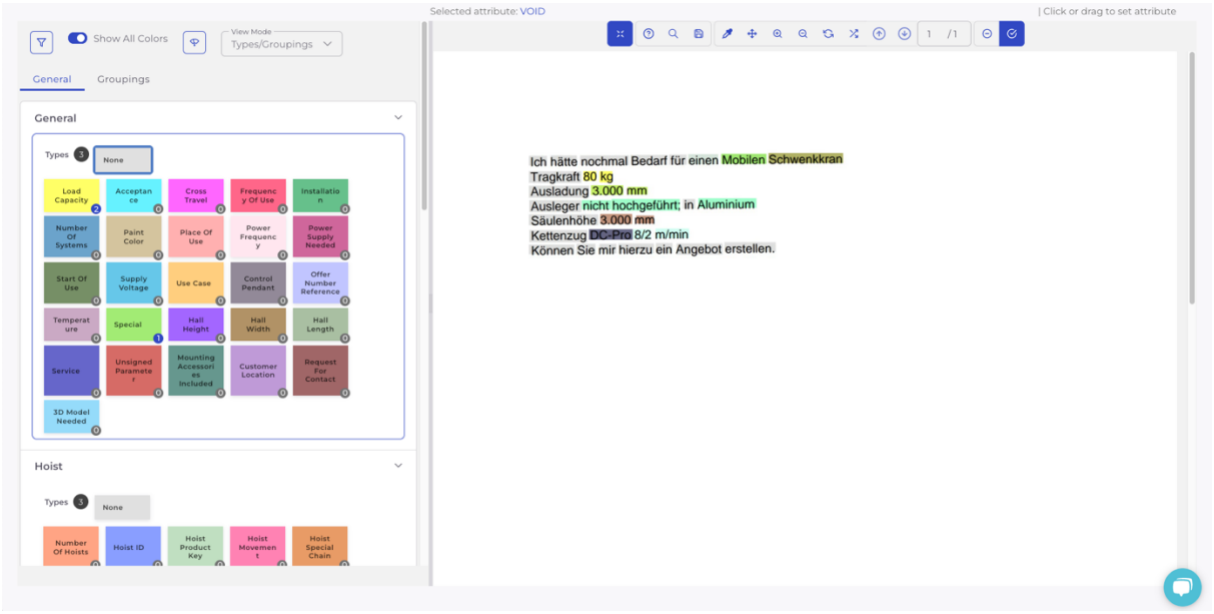


Figure 2: Annotating entities in inquiries

Based on this data, we explored 3 different algorithmic setups to train NER models:

Directly training a text-based NER

To train a text-based NER model, the first approach that we implemented was to directly finetune an already pretrained model. First, since the labelled data provided by DEMAG was in German, we looked for a BERT-based language model that was already pretrained on the German language and decided on the XLM-Roberta-Large model from Facebook. This model has 561 million parameters and is trained on over 100 different languages, so it also allows usage for other use cases in the consortium. Naturally however, it is trained on general German (or other) languages, and not optimized for the vocabulary used in industrial contexts like crane manufacturing.

By “finetuning” what we mean is that we use the labelled data provided by DEMAG to teach this model about the different important entities needed by the product configurator so that in the future the model can automatically extract these entities from the inquiries.

Further pre-training and fine-tuning an NER

To lower the amount of labelled data needed, we also implemented a further pre-training approach. Here, we do not use an out-of-the-box large language model directly for fine-tuning on the NER task, but we first teach it the domain specific language used in the use case. For this, we can use any text, be it public website content, brochures, or technical specifications, that is non-annotated and non-sensitive. In a self-supervised fashion, the generic LLM is trained on that domain-specific unlabelled data, e.g. by hiding certain text areas and asking the model to predict what word was hidden (a so-called pre-training objective). This helps the model learn about domain specific information, such as what the different components of a crane are and how they are connected.

Once optimized on the domain-specific data, we use the same methodology as above to fine-tune the model on the NER task. However, as the general language used in the documents is

already learned, less data is needed for teaching the model to extract entities. Figure 3 visualizes the difference between the two approaches.

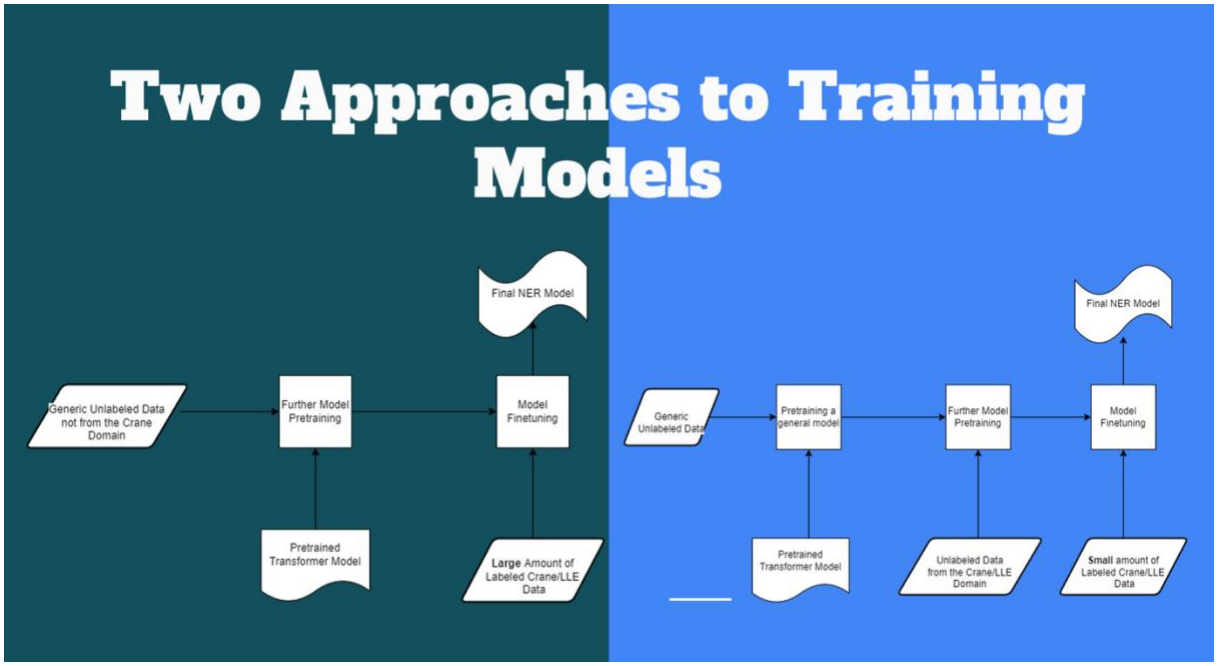


Figure 3: Two approaches to NER training

The first approach involves pretraining a large language model, which in our case was the XLM-Roberta-Large model. This model was pretrained on the CommonCrawl dataset from multiple different languages including German. This pretraining was done by the creators of the model and thus from our perspective it was just a "plug and play model" with no pretraining needed. We only finetuned this model using the annotated dataset provided by DEMAG. Since we don't pretrain our model on domain-specific data in this approach, we need a lot of annotated data to make our model accurate in its extractions.

In the second approach, a large language model is first pretrained on a generic unlabelled dataset, then further pretrained on our domain-specific dataset. DEMAG provided us with some non-annotated, non-sensitive crane data which we use to further pretrain our models. Following the pretraining on the domain-specific data, we fine-tune the model using annotated data. The advantage of this approach is that it requires significantly less annotated data for fine-tuning, thereby reducing the burden of annotating many files. However, this also means that we need a lot of non-sensitive, domain specific data for further pretraining our language models.

Layout-based NER

We noticed that customers also often use text formatting and layouting to specify their requirements. Such information gets lost in normal text-based NER, which works purely on a long character string. Thus, we also implemented and tested a multi-modal layout-based NER approach, that can consider (i) the visual channel (bold/underlined/italic/etc.), (2) the text itself, and (3) the positions of the text in 2D space. For this, a LayoutLM model [1] was trained on the annotated data.

All three approaches yield roughly the same output format, and can thus be integrated in an identical fashion, as discussed in the next section. As too little data has yet been shared for the use case, an in-depth comparison of the three approaches does not yet make sense. All evaluation code was set up but need to be re-run once more data is available.

2.3 Integration

For the integration, an API was developed, that takes an inquiry as input, and outputs the extracted entities, as shown in Figure 4.

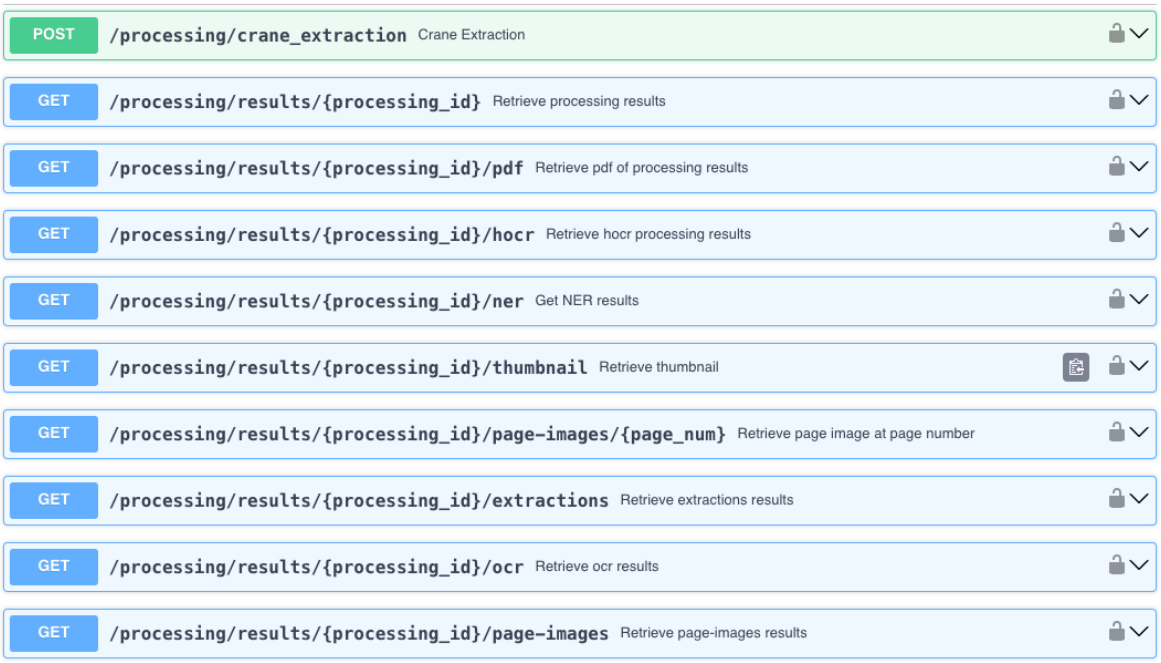


Figure 4: API to transform unstructured inquiries into a structured format

The file simply needs to be POSTed to /processing/crane_extraction. The API can be used synchronously, where the request stays open until the result is ready, or asynchronously, where the request is directly answered with an ID, that can be used to poll the information later on.

In both cases, a structured JSON is returned, containing the entity names as keys, and the extracted values as values.

The API was implemented using a manager-worker pattern, such that as many requests as needed can be submitted, while workers grab the processing tasks and process them asynchronously. This ensures that the API's event loop is never blocked and allows drastically scaling up and parallelizing across workers.

For authentication, API keys or bearer tokens can be used. Furthermore, the API sticks to the openAPI standard, facilitating the integration into the overall InnoSale solution.

3 Semantic Search Within Automatically Transcribed Meeting Notes (DAKIK)

3.1 Use Case

Ermetal employees arrange meetings to provide offers to their customers or discuss completed tasks. When these meetings take place in an online environment, the opportunity to record the meeting arises. Instead of listening to the entire meeting recording when

employees want to recall the topics discussed, they can search using semantic search within automatically transcribed meeting notes. We performed a fine-tuning process on the Whisper model developed by OpenAI for converting meeting recordings to text. This fine-tuning was needed because the support in Turkish of the Whisper model has a lower accuracy rate compared to English not satisfying the use case requirements.

The reason for choosing semantic search over keyword-based search is that employees may have difficulty remembering the words mentioned in meetings organized months ago. With semantic search, employees can reach the desired results with synonymous words even if they do not remember the exact words. Additionally, although we have performed fine-tuning on the Whisper model used for transcribing meeting recordings, it can still make occasional mistakes. Therefore, using semantic search instead of keyword-based search provides significant improvement in search results.

3.2 Technical Solution

Ermetal employees can upload meeting recordings to the server via the interface we developed. Then, they can add the selected meeting recording to the queue to be transcribed with the desired Whisper model through the interface. They can also view the upcoming tasks on the same screen. The reason for allowing the employee to choose the model is that the transcription time varies depending on the size of the model. In recordings where speed and accuracy are not crucial, the tiny model can be used, while in recordings where accuracy is important, the model for which we performed fine-tuning can be used. After selecting the model, the only task the employee needs to do is to use semantic search.

After our Transcribe service finishes processing, it sends a request to our Semantic Search service. With this request, we parse all transcribed meeting recordings stored in the database into structured JSON format, embed the JSON file using the Universal Sentence Encoder Embedding model with the help of LangChain, and create a vector index for use in the FAISS vector database. When the user performs a search, the search term is embedded using the Universal Sentence Encoder, and semantic search is performed via FAISS, presenting the results to the user on the interface.

3.2.1 Fine-Tuning the Whisper Large-v2 Model for Turkish Language

The Whisper model, which is a speech-to-text recognition model, offers multiple model options: tiny, base, small, medium, large, large-v2, and the newly released large-v3 model. The difference between these models lies in their accuracy rate and processing speed, with larger models generally offering higher accuracy and speed. While the English version achieves high accuracy, it doesn't perform as well in Turkish. Therefore, we chose to fine-tune the Whisper Large-v2 model for the meeting recordings we received from Ermetal. We chose the Large-v2 model because it was the largest model available when we began the fine-tuning process. We prioritized accuracy over processing speed, hence opting for the largest model available for fine-tuning.

To be able to use the meeting recordings from Ermetal for fine-tuning, firstly we had to manually transcribe them into text. After transcribing the 14-hour-long recordings, we uploaded them to the TRUBA system and began the fine-tuning process. We tested the obtained models considering the WER/CER metrics. The non-fine-tuned Whisper Large-v2 model had a WER of 59, which decreased to 31 after our fine-tuning process, indicating a noticeable reduction in errors. Figure 5 shows possible Transcribe Model options on Interface

with Finetuned Large-v2 model. Previously, terms like "açınım" (expansion), "bükme" (bending), "kamlı delme" (cam drilling) used by Ermetal were not correctly recognized, but after the fine-tuning process, this issue was resolved.

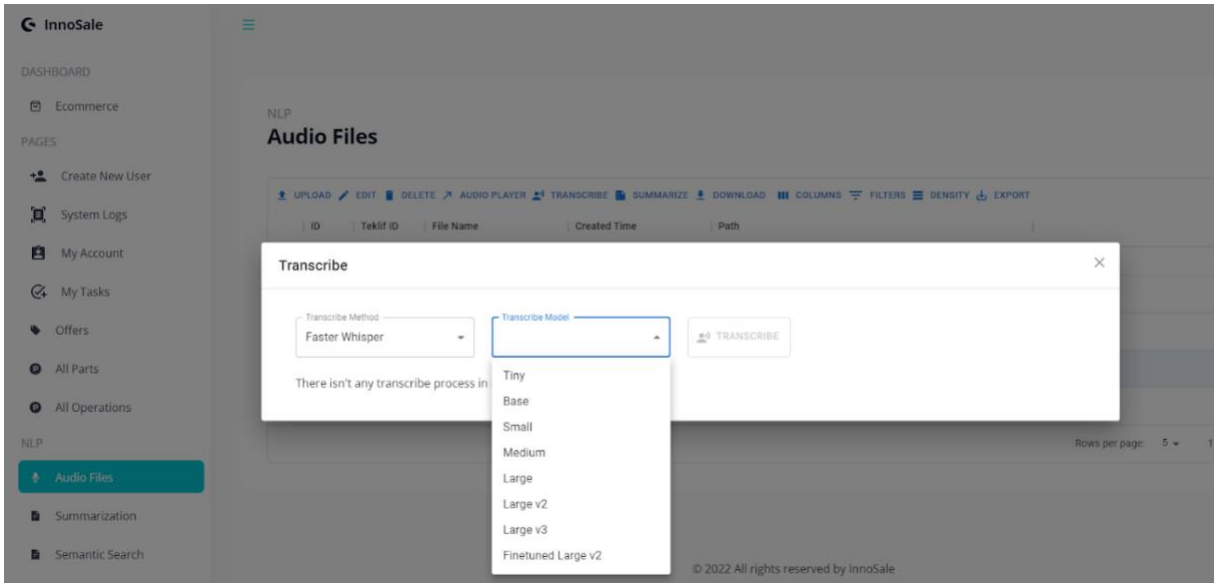


Figure 5 Transcribe screen on Interface with Finetuned Large-v2 model

3.2.2 Embedding Transcript into a Database

The service we created to run the fine-tuned Whisper model sends a request to our Semantic Search service after completing the transcription process and stores the transcription results in MongoDB. Our Semantic Search service retrieves these records from MongoDB to create embedding indexes. We create these indexes specifically for the FAISS vector database by using an embedding model through LangChain. With LangChain's RecursiveTextSplitter, we split the meeting recordings into segments and create indexes using FAISS integration. FAISS compares vectors based on these indexes to return the text segment with the highest score.

We chose the Universal Sentence Encoder for its multilingual support and specifically for its Turkish language support, despite being an older embedding model. Through our Semantic Search service built with LangChain, changing the Vector Database or Embedding model is quite straightforward. We plan to compare the results obtained with the currently used model with those obtained from an updated embedding model with Turkish language support and decide on a model change accordingly.

3.2.3 Upon Search Querying, Embed Search Term, and Compare to DB

When a user wants to search within meeting recordings, they enter their query on the page we created in the interface. This query is then forwarded to our Semantic Search service in the background. Our service obtains the vector value of the query using the embedding model we created for indexing. Then, our FAISS Vector Database performs semantic search with this value. The top 4 text segments with the highest scores are sent back to the interface. This allows the user to read the part where the searched term appears or access the entire meeting recording if desired.

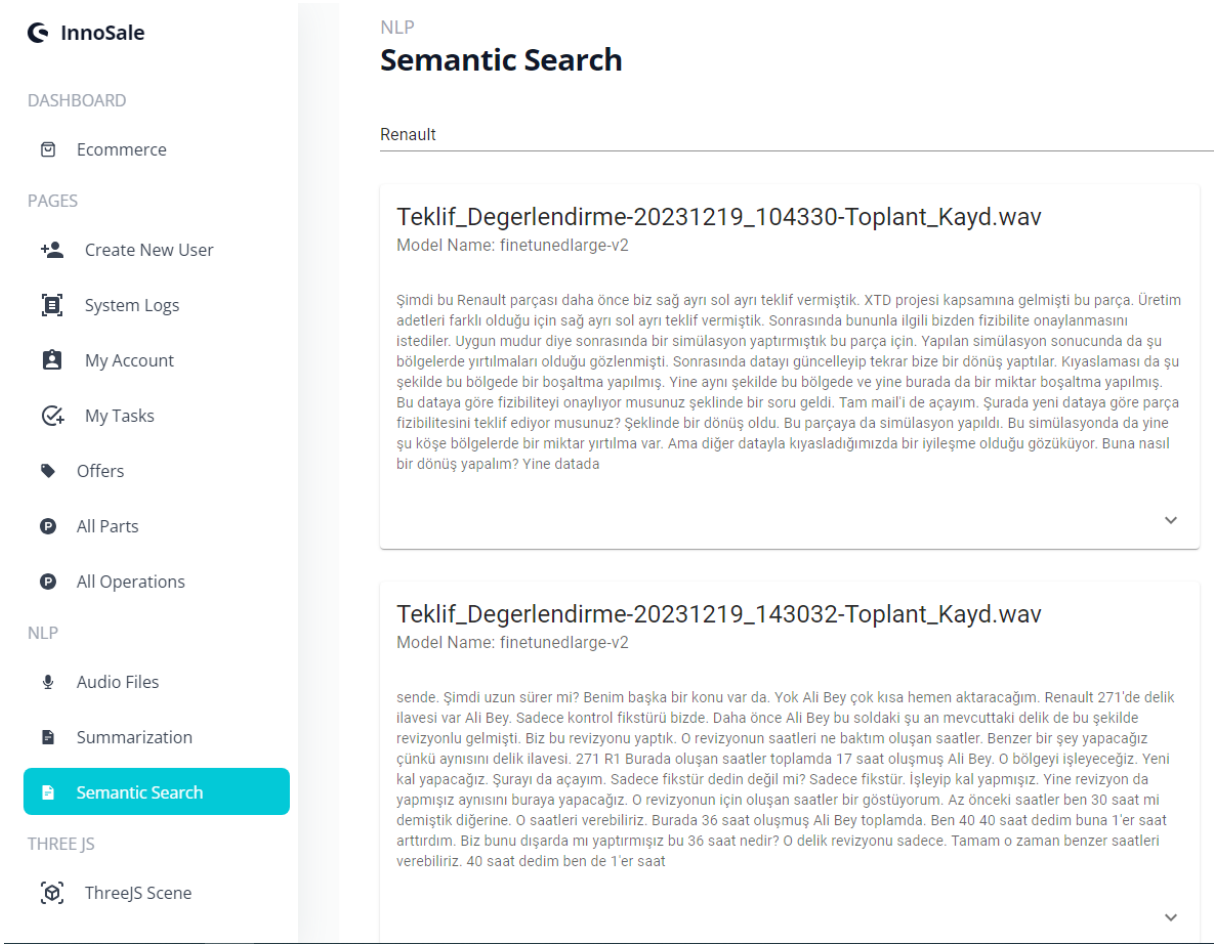


Figure 6 Semantic Search Results on Interface

3.3 Integration

In our Use Case, there are two services in operation. Our Whisper service allows users to upload audio files, and processes them through a queue system using the Whisper model selected by the user for transcription. Figure 7 shows possible API endpoints for Whisper Service. This service also stores the text version of meeting recordings in MongoDB and enables us to perform CRUD operations on these records. Additionally, it sends requests to our Semantic Search service to add newly added meeting recordings to the existing index. This ensures that our Vector Database remains up to date after each transcription process.

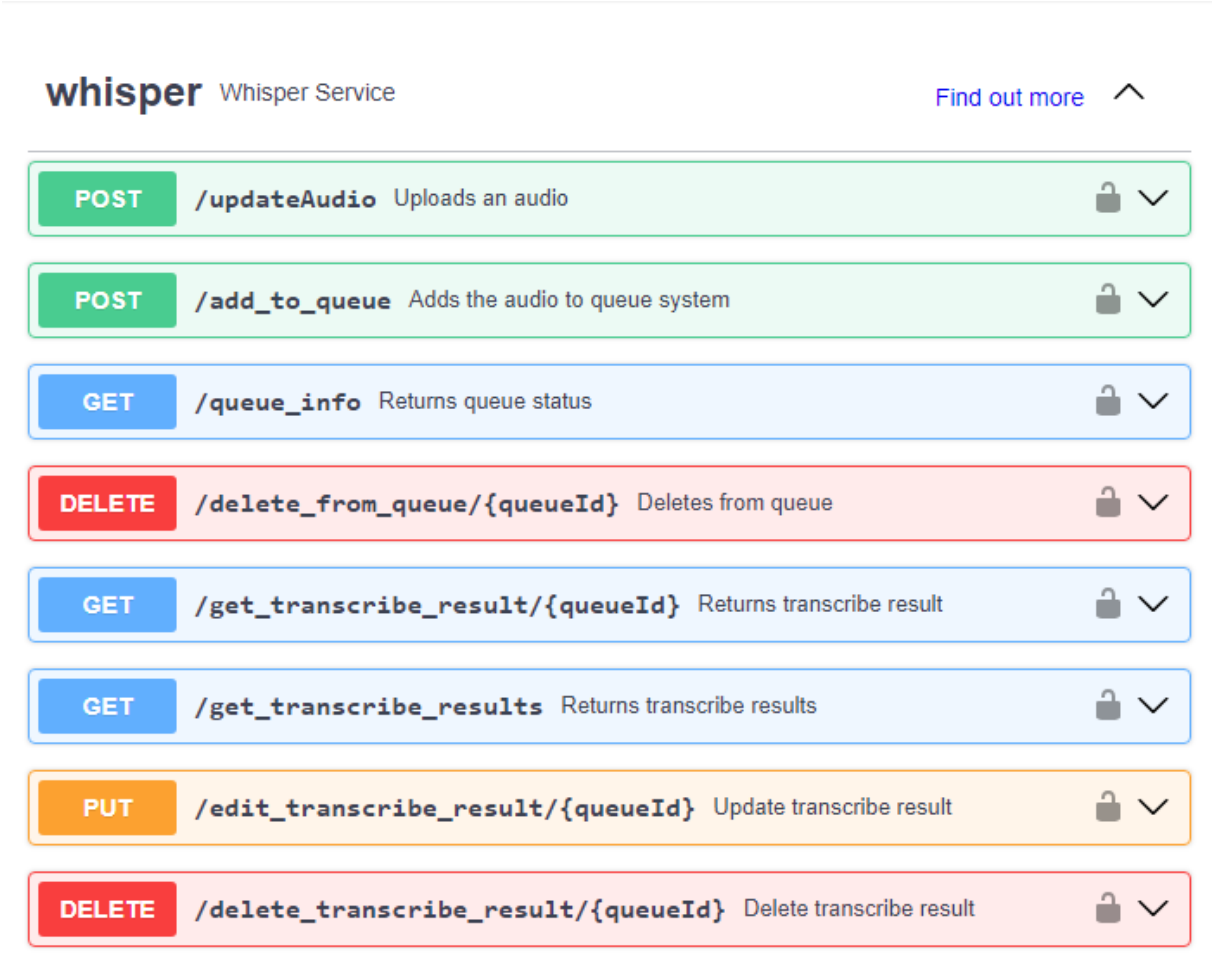


Figure 7 API endpoints for Whisper Service

Our Semantic Search service, on the other hand, has endpoints for creating indexes and returning semantic search results. Our endpoint for creating indexes retrieves all meeting recordings from MongoDB after being called by the Whisper service and splits these recordings into segments using RecursiveTextSplitter via LangChain. The aim here is to speed up semantic search. Figure 8 shows possible API endpoints for Semantic Search Service. We maintain index files generated with FAISS and the embedding model statically on our server. To perform Semantic Search, our endpoint converts the user's query input from the interface into vector format using the embedding model we used to create the index, applies cosine similarity to the vectors found in our Vector Database, and returns the top 4 text segments with the highest similarity. With the page we developed on our interface, users can see search results in real-time and access the entire meeting recording if desired.

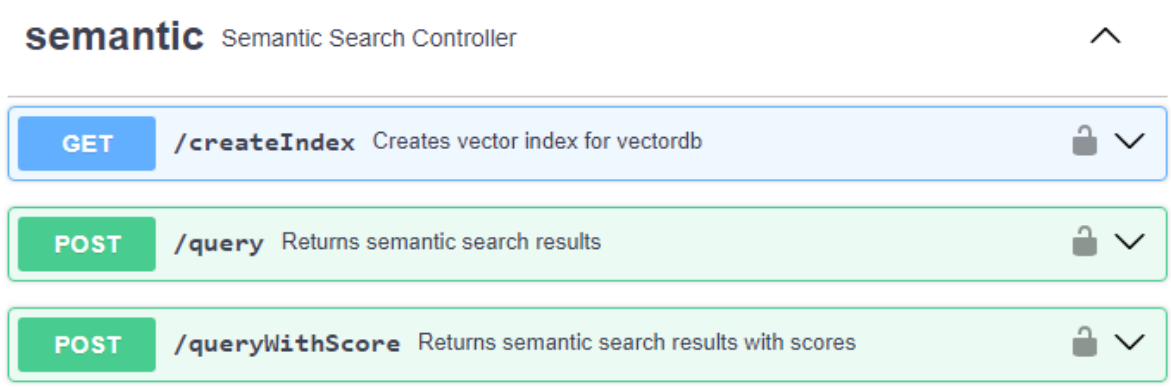


Figure 8 API endpoints for Semantic Search Service

4 Using an Ontological Database to Search for Semantically Similar Historical Projects (IFAK)

4.1 Use Case

Clients generally reach out to DEMAG, the Use Case Partner, through email or a contact form, outlining their product requirements. Due to divergent levels of customer knowledge, some offer comprehensive descriptions, while others may overlook pertinent details, requiring a sales engineer to decipher and finalize the product configuration process, including identifying absent information and consulting past projects for guidance in product development and pricing. For efficiency, the sales engineer looks for similar projects in the past to derive new technical solutions from proven ones. This use case aligns closely with the description provided in Section 2.1

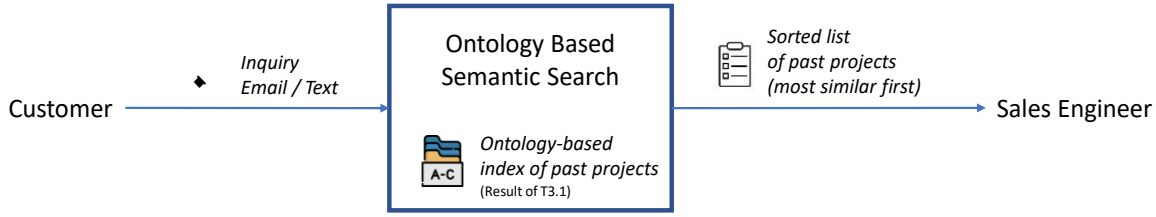


Figure 9: Application of Ontology Based Semantic Search for sales cases

4.2 Technical Solution

In project documents, the technical terms commonly used by experts can differ significantly from the layman's terms used in customer requests. This disparity necessitates the use of an ontology that establishes synonyms or other types of relationships between these terms. By incorporating an ontology, semantic search techniques can yield more accurate and relevant results than a simplistic word indexing approach, as shown in Figure 9.

There are several approaches to semantic search, which are based e.g. on word embeddings or ontologies. In the InnoSale project, firstly, we explore an ontology-based semantic search approach that can retrieve results when synonyms or generalisations of the search terms out of the query text (e.g. the inquiry email text) are present in the target documents (past project documents). Secondly, we contrast the outcomes of an ontology-based semantic search with

those of a word-embedding approach to examine similarities or discrepancies in the retrieved results.

The connection between documents and ontologies plays a crucial role in semantic search. There are two main approaches: tight coupling and loose coupling. In tight coupling, documents explicitly refer to concepts in the ontology, making it easier to resolve homonymies. However, it requires significant effort to annotate documents with semantic information. On the other hand, in loose coupling, documents are not bound to a specific ontology, which presents the challenge of selecting the appropriate ontology. While loose coupling provides flexibility, it has limitations in terms of semantic resolution, especially in scenarios like the World Wide Web. Ontology-based semantic search engines rely on the structure of ontologies, which consist of concepts, properties, constraints, and axioms.

Standard properties, including synonym-of, hypernym-of, meronym-of, instance-of, and negation-of, are used to capture relationships in semantic search based on common sense. These properties enhance the search capabilities but also introduce dependencies on the structure of ontologies [2]. Regarding the tight coupling approach, document annotation is the process of identifying and marking up specific elements or information within a document for various purposes, such as information retrieval, data extraction, or semantic understanding [3]. The specific types and methods of annotation can vary depending on the context, purpose, and tools used for annotation. Some types of document annotation include:

- NER: Involves identifying and categorizing named entities in a document [4]
- Sentiment Analysis: Sentiment analysis aims to determine the sentiment or opinion expressed in a document, often categorized as positive, negative, or neutral [5]
- Topic Modelling: Topic modelling identifies latent topics or themes within a document collection and assigns documents to those topics [6]
- Text Classification: Text classification involves categorizing documents into predefined classes or categories based on their content [7]
- Named Entity Linking (NEL): NEL involves linking named entities mentioned in a document to their corresponding entities in a knowledge base or reference source [8]
- Semantic Role Labelling (SRL): SRL aims to identify and classify the semantic roles of entities and predicates in a sentence or document [9]
- Coreference Resolution: Coreference resolution deals with identifying and linking expressions that refer to the same entity within a document [10].

Hence, when engaging in ontology-based semantic search, it is imperative to link the entities within documents to those in a knowledge base. Given that we have already established concepts within our ontology database, we opt for a different approach by linking the shared tokens or entities between documents and the concepts already present in the ontology. This process constitutes the annotation of documents.

The details of the structure of the ontology to be used can be seen in D3.1. There, we explain how a group of terms (synonyms) - which form an ontological concept - will be assigned to the same conceptID. Further on, the concepts can form different relations such as an abstraction-specification relation with one another.

Table 1 and Table 2 show the relations between concepts and existing project files. Existing project files need to be indexed to get search results with high performance. Hence, we initially store the path of each project file, along with a generated ID, in the database. To extract entities from documents, we utilize NER and also perform simple tokenization of the documents. Subsequently, we compare these entities and tokens with concepts in the ontology to establish their linkage.

Table 1 : document_index_table

File Name (str)	File ID (int)
DemagCleanedData\DE-262-00467107_01\kfm\Anfrage\dE-262-00467107 OFFER-301645 Hans Mustermann.txt	1
DemagCleanedData\DE-262-00467107_01\kfm\Angebot\AW Ihre Angebotsnummer DE-262-00453901-00 DC-ProCC 2-250 11 H4 V82 380-41550.txt	2

Table 2 : concept_document_table

Concept ID (List(int))	File ID (int)
[1]	1
[1, 2]	2
[3 , 5 , 6]	3
[4]	4

To determine the documents within project files that are most similar to our customer inquiry, we propose a method termed "concept frequency". This method calculates the frequency of concepts within each document. Unlike traditional "term frequency", this approach considers all terms related to a specific term, resulting in the same frequency for terms with different relations. Consequently, terms in different languages that are synonyms to specific terms within our customer inquiry receive the same frequency. Thus, when seeking the most similar project files, those containing terms from other languages are also considered. Once concept frequency is calculated, we identify project files containing these concepts. Given that we have already linked concepts and documents in our ontology database, locating them is straightforward. Subsequently, we assign a score to these project files based on the number of concepts they contain. The score is the sum of the frequencies of the concepts within them. The following formula summarizes the aforementioned process.

Let:

- $\boldsymbol{CF(d,c)}$ be the concept frequency of concept ***c*** within document ***d***.
- $\boldsymbol{Count(c,d)}$ be the number of times concept ***c*** appears within document ***d***.
- $\boldsymbol{TotalTokens(d)}$ be the total number of tokens (words or terms) in document ***d*.**
- $\boldsymbol{Score(d)}$ be the score assigned to document ***d***.
- $\boldsymbol{Concepts(d)}$ be the set of concepts contained in document ***d***.

Then, the score $\boldsymbol{Score(d)}$ for document $\boldsymbol{d}$ is calculated as the sum of the concept frequencies for all concepts within the document:

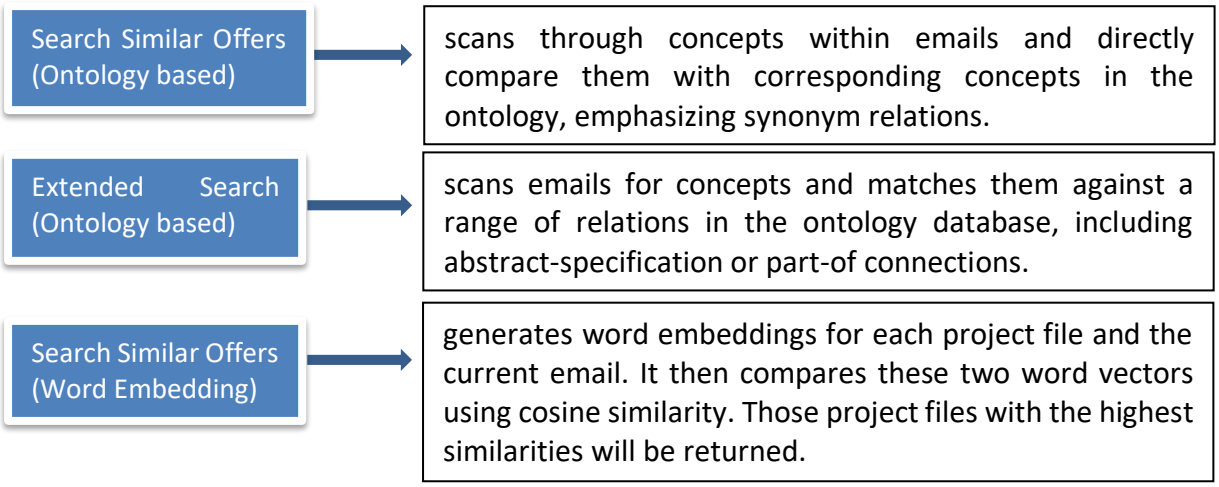
$$\boldsymbol{CF(d,c)} = \frac{Count(c,d)}{\text{TotalTokens(d)}}$$

$$\boldsymbol{Score(d)} = \sum_{c \in Concepts(d)}^{\infty} CF(d,c)$$

This formula represents the aggregation of concept frequencies for all concepts within a document, providing a measure of similarity between the document and the customer inquiry based on the shared concepts.

We have created three distinct search functions. The initial two adhere to ontology semantic search principles, while the final functions employ word embedding techniques. The aim is to contrast our ontology-based methodology with at least one alternative approach to observe

disparities in result retrieval. As of composing this deliverable, we possess only three project folders encompassing 70 files for testing purposes. To better comprehend the outcomes, additional data is required to solidify our conclusions.



Project File	Document Score
DemagCleanedData\DE-202-00469403_00\kfm\Anfrage\WG Anfrage B DE-202-00469403_00.txt.txt	0.099
DemagCleanedData\DE-202-00469403_00\kfm\Angebot\DE-202-00469403_00_Black.pdf.txt	0.099
DemagCleanedData\DE-202-00469403_00\techn\Klärung und Notizen\AW Anfrage B DE-202-00469403_00.txt.txt	0.099
DemagCleanedData\DE-202-00469403_00\kfm\Angebot\DE-202-00469403_00_KundenAnonym.rtf.txt	0.086
DemagCleanedData\DE-202-00469403_00\techn\Entwurf\00469403_00_Entwurf_KundeAnonym.rtf.txt	0.086

Figure 10: Similar Offers Emphasizing Synonym Relations

Project File	Document Score
DemagCleanedData\DE-202-00469403_00\kfm\Angebot\DE-202-00469403_00_Black.pdf.txt	0.284
DemagCleanedData\DE-202-00469403_00\kfm\Anfrage\SalesCAD.pdf.txt	0.173
DemagCleanedData\DE-262-00467107_01\kfm\Anfrage\RE Ihre Angebotsnummer DE-262-00453901-00 DC-ProCC 2-250 11 H4 V82 380-41550.txt.txt	0.173
DemagCleanedData\DE-262-00467107_01\kfm\Angebot\AW Ihre Angebotsnummer DE-262-00453901-00 DC-ProCC 2-250 11 H4 V82 380-41550.txt.txt	0.173
DemagCleanedData\DE-262-00467107\Clear Order\03_Angebot\AW Ihre Angebotsnummer DE-262-00453901-00 DC-ProCC 2-250 11 H4 V82 380-41550.txt.txt	0.123

Figure 11: Extended Search Utilizing Other Relation such as abstract-specification relation

Project File	Similarity
DemagCleanedData\DE-202-00469403_00\techn\Kalkulation\00469403_Stï¿½li_Black.pdf.txt	0.919
DemagCleanedData\DE-202-00469403_00\kfm\Anfrage\WG Anfrage B DE-202-00469403_00.txt.txt	0.914
DemagCleanedData\DE-262-00467107_00\kfm\Anfrage\Auftrag 651531_Black.pdf.txt	0.834
DemagCleanedData\NL-003-00451690_00\kfm\Angebot\ABK_Print_00451690_00_Black.pdf.txt	0.833
DemagCleanedData\DE-262-00467107_00\techn\Entwurf\AW DE-262-00467107 OFFER-301645.txt.txt	0.827

Figure 12: Similar Offers Using Word Embeddings

Figure 10, Figure 11 and Figure 12 show the outcomes of the refined similar offers provided in response to customer inquiries. As previously mentioned, we currently only possess three projects to assess the results, and our viewpoint may evolve with new data. We selected one of the existing files from **"\DE-202-00469403_00\kfm\Anfrage\ WG Anfrage B DE-202-00469403_00.txt"** as a customer inquiry and subsequently conducted three distinct searches. In the initial search, which prioritized synonym relations, we did not retrieve the exact file but found a nearly identical one in another directory**, "DE-202-00469403_00\techn\Klärung und Notizen"**. Generally, this search predominantly returned files from the same project folder. Conversely, the final search, based on word embedding, located the exact file along with several others from different folders. Although the second search does not catch the exact file but there was greater diversity in the returned offers when utilizing alternative term relations (second search based on different term relations) and word embeddings compared to relying solely on direct relationships. However, if the primary objective is to retrieve the appropriate project file without specific regard to a particular file, all three searches yielded nearly identical project files, with project **DE-202-00469403** being the common result among them. In the first two images, document scores are computed following the explanation provided earlier. Each project file is assigned a score, and we retrieve the top five highest scores. In the third image, similarity is determined using cosine similarity, utilizing the sklearn library in Python. To calculate the word embedding for each project file, we utilized the sentence transformers library in Python.

In the future, if we acquire more data, we intend to explore various approaches, such as leveraging large language models or Sentence-BERT, to compare our ontology-based semantic search with them.

4.3 Integration

We've developed an Angular web application that takes in customer inquiries and returns the path to similar project files along with a score indicating their similarity to the customer inquiry. The email processing and calculations are performed in Python, and the results are sent to an Angular application via a REST API. If integration with the current Angular application is not feasible, we have the option to send the results directly from Python to the main application using the REST API.

5 SentenceBERT-based Semantic Search for Historical Projects (VTT)

5.1 Use Case

When a sales expert at KONECRANES receives a customer inquiry for equipment that cannot be configured, a product support ticket is created. Product support manually prepares the offer with necessary custom options and provides it back to the sales expert, aiming to reduce lead time in these cases. When a sales expert creates a product support ticket with offer details, including most technical features already set, the product support person uses the semantic search tool to find historical support tickets with similar content. The aim is for the tool to understand the meaning of the support ticket to identify similar tickets and enable "search with meaning". The search engine transforms the text to a word-embedding format, compares it with historical data, and identifies relevant tickets using clustering algorithms. The results are presented in a user interface with case information, drawings, relevancy estimation, and voting options for validation.

5.2 Technical Solution

The core of the technical implementation is based on SentenceBERT [11]. SentenceBERT is a language model that is based on deep neural networks and it can be used to transform a sequence of text into a numerical vector representation. An especially useful fact in vector representation is that it contains semantic information about the text. Semantic similarity of two pieces of texts can be analyzed by calculating the cosine similarity of two transformed vectors. This opens the possibility to utilize SentenceBERT for semantic search.

The semantic search solution developed by VTT uses SentenceTranformers framework[1] for the utilization of SentenceBERT. The implementation of the semantic search components was done with Python programming language.

On a high level, the solution consists of two main functionalities:

1. Conversion of text into vector representations with SentenceBERT
2. Searching the most similar texts by using cosine similarity from the set of transformed texts

Since the semantic search component's objective is to find the closest matching text from a large set of historical texts, the historical data must be transformed into vector representations beforehand. This is due to the fact that transforming a large amount of data requires some computation time and it would not be feasible to execute this task while the user is performing the search. The resulting vectors from the transformation process (using SentenceBERT) are stored in a file which is then used by the actual search functionality.

When the user executes the search, they provide a search phrase text as an input. This input is transformed into a similar vector representation as the texts in the historical data. This vector is then used to find the closest match from the historical data by calculating the cosine similarity between the query vector and the vectors of the historical data. The text(s) corresponding to the most similar vectors are then returned as a result and displayed to the user.

5.2.1 Current implementation

As at the time of this version of the semantic search component's development, the real data from a use case provider was not available, an open data set was used instead. The demonstrator shown in Figure 13 is implemented using a credit card complaint dataset[2]. The user interface was developed solely for demonstration purposes, but the underlying mechanism implements the SentenceBERT vector transformation and cosine similarity-based search described in the previous section.

[1] https://sbert.net

[2] https://www.consumerfinance.gov/data-research/consumer-complaints/

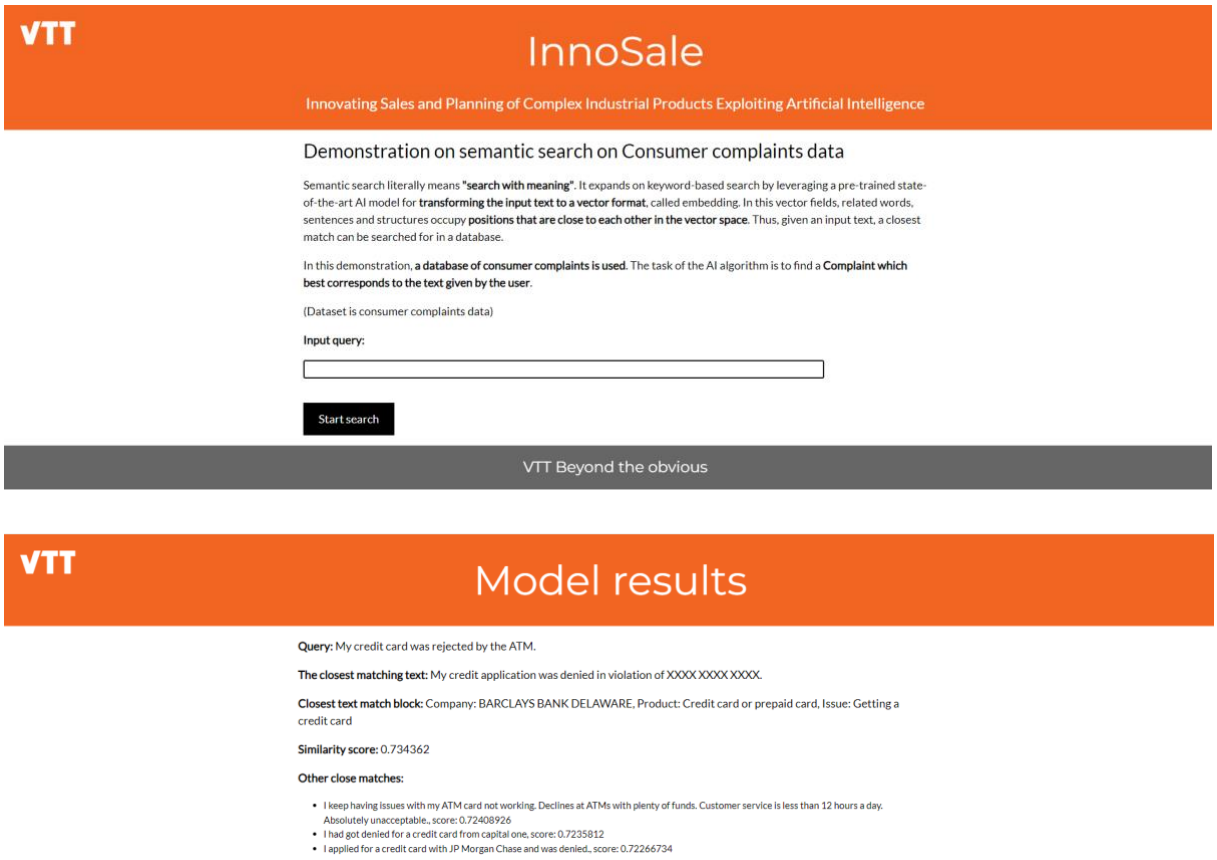


Figure 13. Screenshots of a demonstrator of the VTT semantic search component.

In addition to this also two other datasets have been experimented with to support the development of semantic search component; movie synopsis dataset[3] and arXiv dataset (dataset containing e.g. abstracts and titles of scientific articles published in arXiv.org[4]).

Based on the arXiv dataset, also functionality to train a custom SentenceBERT model was developed. The model can then be used as a basis for semantic search. This enables adaptation of the semantic search component to new contexts and vocabularies.

5.3 Integration

The component will be wrapped as a Docker container. The component can be called by using HTTP POST requests with JSON payloads, as shown in Figure 14. The responses are also in JSON format. When writing this document, the API is not yet implemented but the initial specification is presented below in the scope of the credit card complaint dataset.

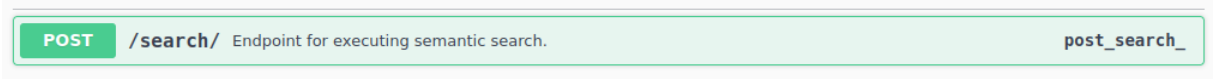


Figure 14. API to perform semantic search with the VTT component.

Example JSON payload for HTTP request to the semantic search component:

{"query": "I have lost my credit card"}

[3] https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

[4] https://www.kaggle.com/datasets/Cornell-University/arxiv

Example JSON payload for HTTP response:

{

"response":

[

{

"document_id": 123,

"matching_text_block": "May have lost my card or it was stolen is it way a to get a new card",

"long_text": "May have lost my card or it was stolen is it way a to get a new card please help me. My XXXX XXXX XXXX credit card # : XXXX and my XXXX XXXX XXXX XXXX XXXX credit card # : XXXX is lost/stolen, and all the transactions on my credit card accounts is fraudulent transactions.",

"title": "Visa credit card is lost/stolen",

"score": 0.765

}

]

}

6 Semantic Search Using an Ontology and a Knowledge Graph (UC3M)

6.1 Use Case

Customers often require software solutions for their businesses but typically lack the technical expertise necessary to navigate traditional software marketplaces. Platforms like GitHub, Stack Overflow, and Microsoft AppSource usually require users to input specific keywords to search for software, which can be challenging for those without a technical background. Our aim with InnoSale, a digital products marketplace, is to revolutionize this process by enabling users to express their needs in natural language. This approach allows customers to articulate their requirements in their own words, simplifying the search process and ensuring they find the most relevant software solutions without needing specialized knowledge. The decision to implement semantic search based on ontologies in InnoSale is driven by the need to effectively bridge the communication gap between non-technical user queries and technical software solutions. By enhancing the search process in this way, InnoSale significantly improves accessibility and user satisfaction, catering to a broader audience with diverse technical backgrounds.

6.2 Technical Solution

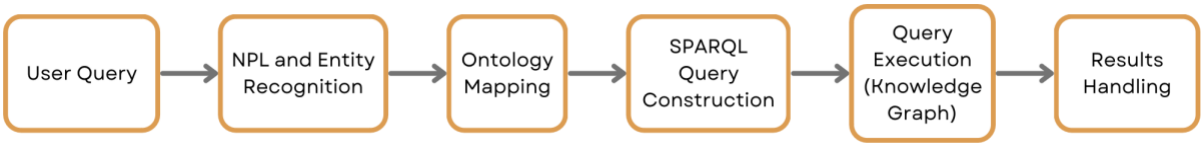


Figure 15 Semantic Search Using an Ontology and a Knowledge Graph Process

The semantic search process begins when a user inputs a natural language query expressing their software needs. Using advanced NLP techniques, the system extracts not only keywords

but also contextual meanings that are then mapped onto specific ontology concepts and relationships defined in the ontology, as shown in Figure 15.

In the initial phase of our semantic search system, we utilized the spaCy NLP library to handle the processing of natural language queries. spaCy provided robust support for various NLP tasks such as tokenization, part-of-speech tagging, and NER. Our approach primarily relied on spaCy's efficient processing pipeline and a set of heuristics we developed to identify and extract key information relevant to our ontology, such as software features, categories, and licensing details.

The heuristics-based method involved rules that helped in mapping common phrases or synonyms to standard terms defined in our ontology. For example, the term "free" was automatically associated with the "Open Source" or "Freemium" categories under the License class. While this approach was effective to a certain extent, it sometimes struggled with understanding more complex queries or capturing the full context of user requirements, particularly when the queries contained ambiguous terms or were phrased in ways not anticipated by our initial rule set.

The ontology, structured in RDF format and described using OWL, includes a variety of classes and object properties that represent different aspects of software solutions. This rich structure allows for detailed mappings and more accurate query formulations. For example:

- Software: This is the central class in our ontology. It represents any software product listed in the marketplace. Each instance of the Software class can be associated with various attributes like name, description, version, and other specific details that describe the software product comprehensively.
- Category: The Category class categorizes software into logical groups based on its functionality, use case, or industry applicability. For example, categories could include "CRM software", "Web Application", "Mobile Application", "Graphic Design Software", etc. This classification helps in narrowing down search results based on the functional domain or intended use expressed in the user's query.
- License: This class details the licensing model under which the software is distributed. Examples of license types include "Open Source", "Freemium", "Commercial", etc. Understanding the licensing requirements is important for users who have specific compliance needs or budget constraints.
- Requirement: Often, users specify particular needs or constraints that should be met by the software. The Requirement class captures these as attributes that software products might possess, such as "supports multi-user", "cloud-based", "offline operation", etc.

Each query is analyzed to identify which aspects of the ontology it pertains to. For instance, if a user searches for 'CRM software with a flexible license', the system identifies 'CRM' as a Category, 'software' as a Software, and 'flexible license' as a License type.

Using the ontology, natural language queries are decomposed and mapped to these specific classes and their attributes. Here's how this mapping facilitates effective search:

- Entity Recognition and Classification: When a user inputs a query, the NLP component first identifies key nouns and phrases. For example, terms like "photo editing", "monthly subscription", or "desktop app" are recognized and classified under Category, License, and Requirement respectively.
- Ontology-Based Query Formulation: After classification, these terms are used to formulate structured queries. If a user asks for "photo editing software with a free license," the system uses this mapped data to create a query that looks for software categorized under "Photo Editing" that is also "Free" under the License class.
- Enhanced Query Precision: The ontology's relationships, such as categorizedAs, licensedUnder, and meetsRequirement, further refine the search. These relationships help in linking software products not just by direct category matches but also by associated features or requirements mentioned in the user query.

These mappings facilitate the dynamic construction of SPARQL queries, tailored to search the underlying knowledge graph effectively. The knowledge graph serves as an interconnected repository of data that includes detailed software descriptions and attributes as defined by the ontology. The knowledge graph is regularly updated with new software entries and user feedback, which not only refines current product categorizations but also enhances the system's learning, making the ontology richer and more aligned with current market trends.

The search algorithm then uses these ontology mappings to execute queries against the knowledge graph. The search process leverages the ontology's object properties like categorizedAs, licensedUnder, and meetsRequirement to find software that matches the user's criteria. For example, the categorizedAs property helps link software items to their respective categories, and the licensedUnder property relates software to their license types. By traversing these relationships in the knowledge graph, the system can retrieve and recommend software products that precisely match the user's expressed needs.

This integration of the ontology with the knowledge graph not only enhances the accuracy of search results but also enables a deeper understanding of user queries, leading to more personalized and relevant software recommendations. The knowledge graph serves as a dynamic repository of all software-related data, allowing complex queries to be handled more effectively. As a result, InnoSale provides a user-friendly interface that significantly reduces the complexity of finding suitable software solutions for non-technical users.

6.3 Integration

An API was developed to process natural language queries from users and return a list of matching digital products, as shown in Figure 16. The API endpoints output structured JSON

containing information about these digital products. Like other use cases within the InnoSale solution, we adhere to the OpenAPI standard to ensure seamless integration.

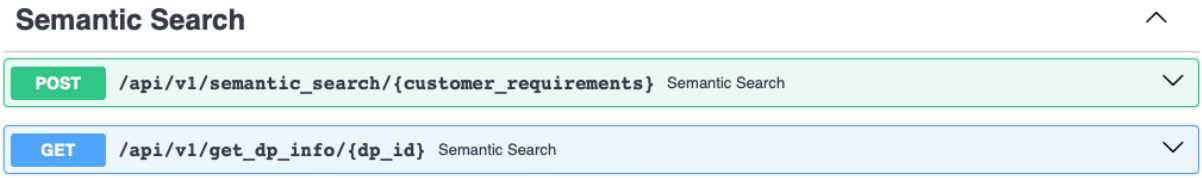


Figure 16 API endpoints for DPM Semantic Search

7 Conclusion

Semantic search is a crucial component to reduce the time-to-offer in sales processes of complex products. Without it, sales engineers spend a vast amount of time deciphering and understanding inquiries, looking at potentially irrelevant meeting notes, and inefficiently searching for similar offers from the past that can be reused for the current inquiry.

This deliverable showed a variety of approaches for semantic search implemented within the InnoSale project. Some approaches rely on explicit modelling of concepts and their relations in an ontology or knowledge graph (see Sections 4 and 6), while others rely solely on implicit modelling (see Sections 3 and 5) and yet others use an intermediate path with implicit modelling for inquiry understanding but explicit knowledge deduction based on rules (see Section 2). This diversity results from the different use case requirements within InnoSale and the legal constraints regarding data sharing, as the approaches trade off data quantity against the need to manually model aspects of the use case (via ontologies, knowledge graphs, or rules). Once the data sharing issues have been resolved for all use cases, the approaches can be retrained, evaluated, and better compared against each other.

Overall, the semantic search component can be considered one of the most crucial components supporting the sales engineer in his day-to-day operations, saving manual effort and leading to higher quality offers.

8 References

[1] Huang, Yupan, et al. "Layoutlmv3: Pre-training for document ai with unified text and image masking." *Proceedings of the 30th ACM International Conference on Multimedia*. 2022.

[2] Hotho, Andreas, et al. "BibSonomy: A social bookmark and publication sharing system." (2006).

[3] Corcho, Oscar. "Ontology based document annotation: trends and open research problems." International Journal of Metadata, Semantics and Ontologies 1.1 (2006): 47-57.

[4] Tkachenko, Maksim, and Andrey Simanovsky. "Named entity recognition: Exploring features." KONVENS. Vol. 292. 2012.

[5] Vinodhini, G., and R. M. Chandrasekaran. "Sentiment analysis and opinion mining: a survey." International Journal 2.6 (2012): 282-292.

[6] Nikolenko, Sergey I., Sergei Koltcov, and Olessia Koltsova. "Topic modelling for qualitative studies." Journal of Information Science 43.1 (2017): 88-102.

[7] Mirończuk, Marcin Michał, and Jarosław Protasiewicz. "A recent overview of the state-of-the-art elements of text classification." Expert Systems with Applications 106 (2018): 36-54.

[8] Al-Moslmi, Tareq, et al. "Named entity extraction for knowledge graphs: A literature overview." IEEE Access 8 (2020): 32862-32881.

[9] He, Luheng, et al. "Deep semantic role labeling: What works and what's next." Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2017.

[10] Ng, Vincent, and Claire Cardie. "Improving machine learning approaches to coreference resolution." Proceedings of the 40th annual meeting of the Association for Computational Linguistics. 2002.

[11] REIMERS, Nils; GUREVYCH, Iryna. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084, 2019.