Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Bachelor of Science in Computer Science - 15.0 credits

SEMANTIC SEARCH IN HISTORICAL DOCUMENTATION

Ivan Hansson
imn21003@student.mdu.se

Edvin Wiklund
ewd20005@student.mdu.se

Examiner: Luciana Provenzano
Mälardalen University, Västerås, Sweden

Supervisor(s): Alessio Bucaioni
Mälardalen University, Västerås

Company Supervisor(s): Martin Prescher
Hitachi Energy, Ludvika

10/06/2024

Abstract

Many organisations face problems with data digitisation and continuous data gathering. They often gather and store this data in outdated systems that are difficult to search through. In our thesis, we utilise the engineering method to investigate the feasibility of incorporating artificial intelligence to search a large corpus of data and find accurate answers.

To achieve the thesis goal, we conducted a literature review, studying existing solutions that enhance flexibility and facilitate artificial intelligence operations to search in databases. This resulted in the choice of utilising OpenSearch. Within OpenSearch, we conducted an experiment investigating which sentence transformer for embedding the contextual meaning of sentences could be best utilised for semantic search in the database. We then evaluated the sentence transformers´s performance with the MS MARCO dataset measuring both speed and accuracy. Through the experiment we found two sentence transformers that outperformed the rest by a slight margin and that all the sentence transformers performed similarly overall. A notable result is that the sentence transformers specifically dedicated to semantic search and sentence transformers with larger dimensions did not perform better. Further, these results showed the easy combination of existing search engines that incorporate artificial intelligence to semantically search in the documentation and showed that this could be used within organisations to handle a large corpus of data.

Keywords: *Semantic search, OpenSearch, dense vector search, sentence transformers*

Acknowledgement

We would like to express our greatest gratitude to our supervisor, Alessio Bucaioni, who guided and supported us throughout our thesis. We also want to thank our manager at Hitachi Energy, Fredrik Antonsson, for hiring us to do this thesis, and our supervisor at Hitachi Energy, Martin Prescher for helping and supporting us throughout our thesis. Without all of you, this thesis would not have been possible.

Contents

1. Introduction **1**

2. Background **2**
2.1 No-SQL . . . 2
2.2 Semantic search . . . 3
2.3 BERT . . . 3
2.3.1 SBERT . . . 4
2.4 OpenSearch . . . 5

3. Related Works **6**
3.1 Sentence transformers . . . 6
3.2 Semantic search engine . . . 6

4. Research process **8**
4.1 Research goal . . . 8
4.2 Research questions . . . 8
4.3 Research method . . . 8
4.4 Threats to validity . . . 9
4.5 Ethical and societal considerations . . . 10

5. Experiment **11**
5.1 Evaluation metrics . . . 11
5.2 Dataset . . . 12
5.3 Experimental process . . . 12
5.4 Validation activity . . . 13

6. Results **14**
6.1 Which No-SQL databases with search-engine interfaces exists and what are their benefits? (RQ1) . . . 14
6.2 Which pre-trained semantic transformer provides the best accuracy and speed for semantic search within a No-SQL database? (RQ2) . . . 15

7. Discussion **17**

8. Conclusions and future work **19**

References **21**

Appendix A Appendix A **22**

List of Figures

1 An illustration of the architecture behind sharding and data storage in a No-SQL database. 3
2 Structural overview of BERT used for single sentence classification. 4
3 Diagram shows an overview of SBERT used for STS. 4
4 Simple demonstration of JSON document(index). 5
5 Metadata representation of an index. 5
6 An overview of the experiment 13

List of Tables

1 Table of benefits of No-SQL databases 14
2 Result from the experiment 15
3 Top four sentence transformers 16
4 Additional information about the transformers 16

1. Introduction

A problem that has been arising with the scaling database capacities is the ability to find relevant information in these databases with semantic search [1], [2], instead of keyword search. The research of each standalone part exists, but the issue lies in the combination of each consisting part that creates the whole system that can handle these queries. However, the issue of creating a tool that can handle the query with enough semantic context is still in the early stages of research [2]. This research field could possibly have an impact on corporations generating vast amounts of documentation. Hitachi Energy is no exception to that. Hitachi Energy's High Voltage Direct Current (HVDC) section alone generates thousands of documents for each project, which gets stored in a manner that makes it hard to find relevant information swiftly. Hitachi's problem becomes a great example highlighting how outdated search engines can create more problems than they provide solutions.

Handling vast amounts of unstructured data is a challenge that has been conquered before. A common theme is to store the data in a database following the schema of SQL or No-SQL.

However, only storing it in an efficient way does not handle the problem of searching through this vast amount of documentation, which is a tedious task due to the keyword search constraint. Keyword search has been the most common type of search in documentation for a long time, where the search engine searches for direct matches against a specific keyword. One issue with keyword searches is that the search has to be very precise, and any misspelling or paraphrasing will turn out to be a faulty search. Another issue is that keyword search does not capture the semantic context of the search query. Recently, great leaps have been taken in this field where semantically embedding of text and queries combined with semantic matching has shown to outperform keyword search [2]. The current state of the art is No-SQL databases layered with search-engine interfaces incorporating semantic transformers to search within the database. [3]. Semantic transformers handle the semantic meaning of the phrase instead of the specific word, and these pre-trained semantic transformers will be the main determinant of our thesis. Different semantic transformers exist, and using such in a search engine could impact the results.

Our thesis aims to research an implementation that incorporates Artificial Intelligence (AI) to search in historical documentation. To achieve this, we have conducted a literature review identifying different search engines, and an experiment using the MS MARCO dataset comparing the execution speed and accuracy between the pre-trained semantic transformers. We have measured the accuracy of how well the search results from a query match the user's intent with $MRR@k$ and $nDCG@k$, while we use milliseconds to clock the speed. Through this work, we aim to contribute to research and the ongoing discussion of semantic search and highlight how easy the current solutions are to implement. Our work will also contribute to the academic literature on dense vector search and advance knowledge in this field of research.

Our thesis is organised in the following order. Section 2 provides background knowledge of No-SQL, Semantic search, BERT, SBERT, and OpenSearch. Section 3 presents state-of-the-art works, their contributions to this field and their relevance to this thesis. Section 4 presents our research process, including research goal, research questions, research method, threats to validity and ethical and societal considerations. Section 5 presents our experiment. Section 6 presents our results for each research question (RQ). Section 7 discusses the implications, limitations and interpretation of the results. Section 8 concludes with final remarks and suggestions for future research.

2. Background

In this section, we present fundamental knowledge about the foundation of our thesis and how it will be used.

2.1 No-SQL

Not Only Structured Query Language (No-SQL) is a non-relational database type that stores its data in flexible structures to accommodate various data types. This follows the basis of the Basically Available, Soft Structure, and Eventually Consistent (BASE) principal [4]. This makes No-SQL databases suitable for storing both unstructured and structured data. It differs significantly from the more conventional storage, Structured Query Language (SQL), which follows a rigid schema, making it effective at storing structured data [5]. In SQL databases data is normalised, and relationships are defined by foreign and primary key constraints. Data is stored in tables rowwise, where the columns are attributes relevant to the objects stored in the table. In a No-SQL database, data storage can be done in different formats. These formats are often categorised into four categories: key-value pairs, graphs, documents, and columns [6]. In a key-value pair-based structure, keys serve as a unique identifier, and the values serve as information that needs to be stored. The key and value can range from simple to more complex objects. In a graph-based structure, data is stored as nodes, and edges represent the relationships between the data, which makes it suitable for describing parent-child relationships, ownership, and similar concepts. In a document-based structure, data is stored in documents such as JSON and XML, which makes the data hierarchically formatted and flexible to match different data types. The last example is the columnar structure, where data is stored in rows and columns, and more similar columns are stored together in a so-called column family. This structure is effective on large datasets that need a lot of expansion room [6].

No-SQL databases can be scaled up and expanded amongst several servers due to the structure of the previously mentioned storage formats. A common theme amongst No-SQL databases is the concept of sharding, where a set of data is divided into smaller subsets, called shards, that can be distributed amongst multiple nodes. A node is a server running the database service. A collection of nodes is called a cluster, and within a cluster, a specific node can be specialised on a sub-task for queering the data. This is illustrated in Figure 1, which shows a cluster of three nodes. There are also three datasets, A, B and C, which have been divided into shards A1, A2, A3, B1, B2, C1 and C2. The shards have been distributed amongst the nodes in the cluster and the nodes then communicate with each other to complete tasks handed to the cluster. This helps to distribute the load and makes for faster executions on queries and scaling the storage capacity of the cluster [6]. In our thesis, we will use a No-SQL database encapsulated with a search engine to store and manage the documents. This will make for faster retrievals and modularity in our work.

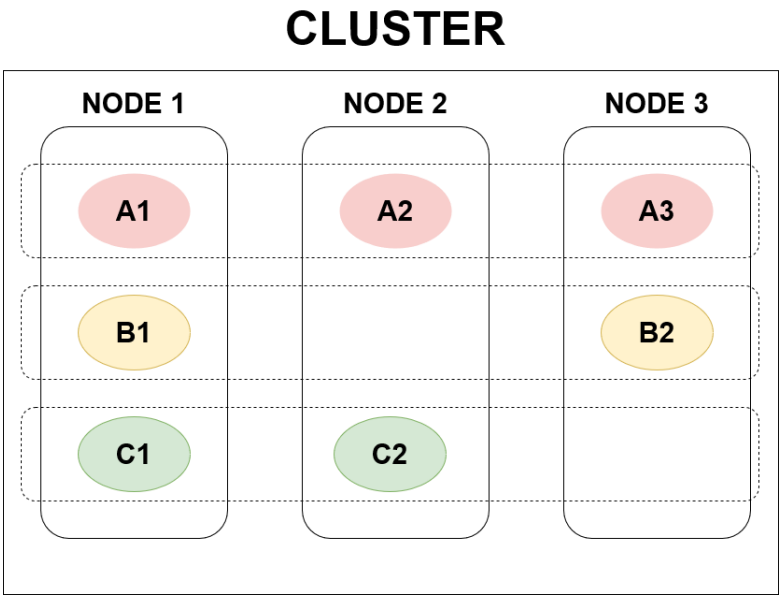


Figure 1: An illustration of the architecture behind sharding and data storage in a No-SQL database.

2.2 Semantic search

Semantic search is a technology that interprets the meaning of search queries to retrieve answers that match their contextual meanings [7]. Semantic search consists of two parts, semantic and search. The semantic portion refers to contextualising the query, and the search portion signifies the retrieval of top candidates matching the query. There are two categories of semantic search, asymmetric search and symmetric search [8]. In asymmetric search the query and the result can be of different lengths, such as a short question to match a full document. Symmetric search is the opposite, where the query and the result are of similar lengths. Semantic search achieves its ability to understand the contextual meaning of words and sentences through a combination of machine learning and AI. Commonly used is a vectorisation of the documents or texts that then is compared with a vector representing the search query. From this comparison, the top candidates are then selected based on their similarity. This approach differs from the more commonly used keyword search which aims to match keywords in the query to retrieve a result. We will use semantic search to retrieve and find relevant documents in the No-SQL database.

2.3 BERT

Bidirectional Encoder Representation from Transformers (BERT) is a bidirectional neural network that is used for sentence embedding and transformation [9]. It is an evolution of Transformers, a state-of-the-art work done by Google that utilises a self-attentive network with recurrence and convolutions [10]. Unlike more traditional sentence embedders, BERT doesn't generate a single consolidating embedding vector for the sentence but a set of embeddings where each part of the sentence, also referred to as a token, is inherently tied to the context of the surrounding. This results from the bidirectional nature of BERT where the network looks at the surrounding tokens to give contextual meaning to a specific token. This makes BERT good at understanding context and nuances in the text and answering questions. A BERT model can be specified to perform better at specific tasks through the training process. The training process consists of two main parts, *pre-training* where the model learns contextual meaning and *fine-tuning* where it is specialised with an input and output layer according to the special needs [9]. The vector dimensions of the outputs from different models can vary in size depending on the output layer. BERT has become a state-of-the-art sentence embedder due to its versatility [11], and numerous amounts of pre-trained and fine-tuned models have been created [12].

Figure 2 illustrates how the BERT architecture embeds a sentence. To begin, a sentence is fed through an initial embedder which translates the words into tokens. A special classification token [CLS] is added, this token is an embedding representing the full sentence. The tokens are then fed through encoders, which give them their contextual meaning. This encoding can be used for multiple natural language processing tasks such as semantic search.

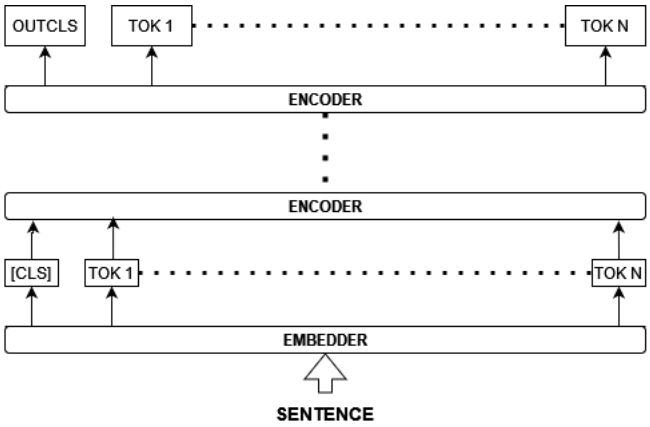


Figure 2: Structural overview of BERT used for single sentence classification.

2.3.1 SBERT

Sentence Bidirectional Encoder Representation from Transformers (SBERT) is an algorithm that can be used for semantic search [11], [13]. SBERT is a Siamese neural network as illustrated in Figure 3. Figure 3 shows how SBERT combines pooling, the neural network BERT and combinatorial function and the flow of a task handed to the algorithm. All of the parts in a SBERT algorithm can be swapped out to perform better on specific tasks, such as Semantic Textual Similarity (STS), a metric used to measure the similarity between texts and phrases. SBERT together with BERT is also the foundation for the pre-trained sentence transformers, which are SBERT or BERT networks that have been trained and modelled for different use cases. These sentence transformers are what we will evaluate in our thesis.

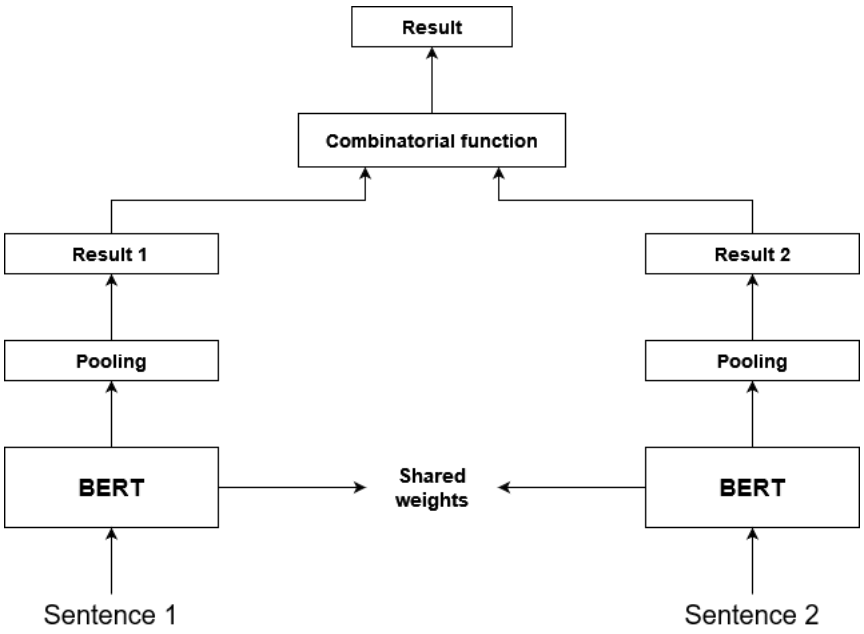


Figure 3: Diagram shows an overview of SBERT used for STS.

2.4 OpenSearch

OpenSearch is an open-sourced search and analytics suite used to ingest, search, visualise and analyse data. It was originally a fork of Elasticsearch and Kibana to provide a fully open-source search and analytics suite. OpenSearch consists of a data store, a search engine, a visualisation and user interface, and a server-side data collector. Users can extend the functionality of OpenSearch with a selection of plugins that enhance search, analytics, observability, security, machine learning, and much more[1]. OpenSearch works under the Apache 2.0 license and is built on Apache's open-source Lucene library, which is utilised to build, create and manage indexes in the database. Several other search engines similar to OpenSearch are built on the Lucene library.

OpenSearch distributed design means that you interact with OpenSearch clusters. Each cluster is a collection of one or more nodes. A node is a server which runs OpenSearch for storing data and processing search requests[2]. The system requirements of OpenSearch are minimal, you can run or host it on a single laptop, but the flexibility of OpenSearch makes it easy to scale up with multiple servers. OpenSearch organises the data into indexes and is built on a No-SQL document format storage where each index is a JSON document collection. If the documents you want to add are in raw text format, they must be pre-processed and converted to JSON format. A simple JSON document for the book "1984" can look like this:

```
{
    "title": "1984"
    "release Date": "1949-06-08"
}
```

Figure 4: Simple demonstration of JSON document(index).

For each index, OpenSearch itself adds multiple parameters of metadata as well, such as the unique document ID and many more properties, like this:

```
{
    "_index": "<index_name>",
    "_type": "_doc",
    "_id": "<id>",
    "_version": 1,
    "_source": {
        "title": "1984"
        "release_date": "1949-06-08"
    }
}
```

Figure 5: Metadata representation of an index.

OpenSearch splits the indexes into shards for an even distribution across the nodes in a cluster. For example, if an index is 100GB, the index might be too large for a single node in the cluster to handle, and then OpenSearch splits it into ten shards, which results in each shard being 10GB. OpenSearch can then redistribute the shards between the existing nodes to work with each shard individually[2].

[1] https://opensearch.org/about.html
[2] https://opensearch.org/docs/latest/intro/#indexes-and-documents

3. Related Works

In this section, we describe state-of-the-art research and what they provide for our thesis.

3.1 Sentence transformers

The use of sentence transformers has rapidly increased in recent years, and the applications are numerous. Several works are fundamental to sentence transformers. The first significant work is by Delvin et al. [9], which introduced the BERT model. This model was created to pre-train bidirectional representations of text by jointly considering both the left and right context to understand the meaning of tokens. The model demonstrated state-of-the-art results on eleven natural language processing tasks. This work is foundational for the SBERT model later developed by Reimers and Gurevych [13], which addressed the challenges of the BERT model for STS tasks, where the BERT model was slow but delivered accurate results. The work resulted in a Siamese network that maintained the accuracy of BERT but significantly increased speed. Both of these works aimed to solve previous weaknesses and, in doing so, create new solutions. Our work aims to build upon their achievements and evaluate the sentence transformers derived and trained through SBERT and BERT. Our research deepens the understanding of how the training corpus of sentence transformers can affect its performance on test data outside the training corpus and how they differ across various performance categories.

The paper by Zhang et al. [14] compared the efficiency and robustness of their HLATR (Hybrid List Aware Transformer Reranking) module to different retrieval and reranking models. They concluded that their module outperformed existing models like BERT and RoBERTa. Their experiment is comparable to ours because they utilised the MS MARCO document and passage dataset to compare the models and employed MRR@10 and MRR@100 as their metric scores. We used similar Mean Reciprocal Ranking (MRR) values, but instead of the full MS MARCO document and passage datasets, we conducted our experiment with the MS MARCO top 1000 passage dataset due to its similarity with the data from Hitachi Energy. With this in mind, the experiment for our thesis will evaluate the sentence transformers used in OpenSearch instead of the HLATR.

3.2 Semantic search engine

Finding information in a database based on a query is a research topic popularised due to AI's rapid development. However, this specific topic is fairly new in the field of science and research, so we have not found many papers on it. The papers we found approach this topic in different ways, and new solutions are constantly being developed, which makes it more difficult to differentiate which approach is the most feasible.

One work, conducted by Andre et al. [1] in 2021, focusing on identifying COVID-19 publications, was particularly interesting for us due to the similarity in approach and motive to our own. Their evaluation was concentrated on the TREC-COVID document dataset combined with the CORD-19 dataset. This dataset comprises five sets of topics, each represented by a tuple of query, question, and narrative. They also employed several information retrieval (IR) metrics to enhance their approach, including Normalized Discounted cumulative gain (nDCG), top-N precision, and MAP. A critical limitation they acknowledged was the dependency of effectiveness on complete relevance judgement across all topic-document pairs. Their paper primarily explores the performance of the CO-search search engine in comparison to top contenders in their field of research. The outcome of their study indicated that their system, CO-search, did not outperform the other systems evaluated. This is in contrast to our work, which focuses on evaluating existing transformers within search engines, rather than on the development and evaluation of a complete system.

The work by Xian et al. [2], published in March 2024 demonstrates that state-of-the-art vector search can be achieved using existing tools. The authors highlight the issue of slow development within the field, a problem acknowledged by many. Despite this, several search engine software providers, such as OpenSearch and ElasticSearch, continue to incorporate and embed Lucene into their software, suggesting optimism for the future of such solutions. The comparison done by Xian et al. [2] between Lucene and Faiss (Facebook's vector search software) revealed significant differences in speed, query latency, and throughput, with the authors concluding that dense vector

manipulation is essential for search functionality overall. While there are no definitive answers on how to implement these solutions in a production system environment, the possibility of combining Lucene with AI embedding offers a hopeful pathway for deployment within such systems. This research is foundational for our work due to its evidence of Lucene's performance in dense vector searches, a critical component of semantic search. It has also provided us with a framework to further evaluate existing semantic transformers and semantic search methodologies. The approach of our work aligns closely with that of Xian et al. [2] where both adopt an engineering philosophy focused on creating, developing, and evaluating existing solutions.

4. Research process

In this section, we describe the research process that we used to answer our goal and questions.

4.1 Research goal

Our thesis investigates the feasibility of implementing a solution that incorporates AI to search historical documentation. This goal derives from Hitachi Energy´s need to overcome the obstacle of searching a data set containing thousands of documents, which can also be modified and applied in several other fields handling a large corpus of data.

4.2 Research questions

With this goal in mind, we began a literature review to find which current solutions there are to the problem and how they could help us. Based on the results from the literature study we developed two RQs to guide us through the work. In the literature review, we identified two primary objectives for the study: to identify the NoSQL databases with search-engine interfaces that are currently available and to determine which pre-trained semantic transformers for semantic search perform best in retrieving relevant results.

RQ1: *Which No-SQL databases with search-engine interfaces exist and what are their benefits?* *RQ1* was designed to address the first obstacle we identified in the primary literature review and the research goal, which was to store and, in a unified manner, make the data searchable. By identifying the current databases and their benefits, we could conclude which solution was most beneficial to implement.

RQ2: *Which pre-trained semantic transformer provides the best accuracy and speed for semantic search within a No-SQL database?* By answering this *RQ* we evaluate the feasibility and the performance of using AI to search in documentation.

Even if the goal and questions are derived from Hitachi Energy´s problem, they are still relevant to the ongoing research. To our knowledge, no research has combined these techniques to evaluate their combined performance. Our thesis also showcases the ease of use when combining these techniques. Therefore, our thesis will be relevant for other researchers to refine and further develop.

4.3 Research method

Due to the time constraint, we would not have time to train and implement our own model. Instead, we needed to find, implement and connect existing solutions that could satisfy the needs of our thesis, thus making the engineering method the most appropriate and feasible method to use when investigating this problem. The engineering method is an experimental model that fits our work since the steps in an engineering method [15] are: observe existing solutions, build/develop, measure, analyse, and repeat the process until no more improvement appears possible. This paradigm is an evolutionary improvement-oriented approach which focuses on currently existing models and modifies the existing model to improve the thing being studied [15].

We used different methods to fulfil the different steps within the engineering method. To observe existing solutions, we conducted a literature review with the goal of finding suitable tools that we could use for our purpose and answer *RQ1*. To limit the amount of data collected, we used three scientific databases: ACM Digital Library[3], IEEE Explore[4] and Google Scholar[5]. To search in the databases, we used three queries "*Semantic search in No-SQL databases*", "*Open-source search engine performance evaluation*" and "*Vector search*". To ensure the validity of the texts and the integrity of our work we established some requirements for the texts to be included in our study.

- Written in English

[3] https://dl.acm.org/
[4] https://ieeexplore.ieee.org/Xplore/home.jsp
[5] https://scholar.google.com/

- Peer reviewed
- Longer than 4 pages
- Available as full text
- Provide information applicable to the thesis

Based on the outcome of the literature review, we iterated and refined the *RQs* to improve their clarity and relevance. We also answered *RQ1* by identifying the current solutions and their benefits related to our work. The papers we identified and used for the literature review are listed in section 6.1.

The next step we took was to build and implement the initial solution found from the literature review with minor modifications to suit the data we were using. We initially began with a solution that did not involve a No-SQL database but instead embedded the sentences with the sentence transformers each time it did a search. Through some testing and feedback from our supervisor at Hitachi Energy, we quickly concluded that this solution was too slow, and we went back to do some more research. In the next iteration, we built a more refined solution using a No-SQL database layered with a search engine and sentence transformers. To measure the proposed solution, we conducted an experiment that produced quantitative results from which we could analyse and draw conclusions. The implementation was further validated at a workshop with employees of different ranks and roles to strengthen our results further. Based on the results from the experiment, we refined our solution and deployed the sentence transformer that performed the best in the experiment.

4.4 Threats to validity

In this subsection, we will present the threats to validity that we identified throughout the research conducted in our thesis [16].

Construct validity threats relate to how well the research measures the concept it intends to measure. In our work, we have identified one primary construct validity threat, the MRR and nDCG metrics, which we chose as our primary evaluation scores. As stated earlier, this field of research is relatively new, so there are no established rules dictating which metrics to use. We acknowledge that there might be other metrics that could perform better. However, due to the limited scope of this thesis, we do not have the time to test all alternative metrics. To mitigate this validity threat, we used the knowledge gained from our literature review and picked metrics that we found were useful and complied with the requirements for accuracy. The metrics were carefully picked based on what had been tested before, these works are mentioned in section 2.1. MRR is also widely used specifically for the MS MARCO dataset which can be found on their GitHub[6].

Internal validity threats examine whether the investigated causal relationship is affected by unknown factors. We have identified two internal validity threats in our work, the dataset used in our experiments and the choice of sentence transformers. Due to time constraints, we have limited the number of sentence transformers evaluated to the ones already provided by OpenSearch, which means we may have missed a better solution among all the existing sentence transformers. These transformers is placed amongst the most liked multilingual and English transformers on the HuggingFace community website[7], which makes the most appropriate to test. The choice of the dataset represents the second internal validity threat. To reduce this threat, we compared several datasets with the one received from Hitachi Energy and picked the dataset that most closely resembled the variety of search queries and passages within the Hitachi Energy dataset. This dataset is also publicly available for research[6] and is commonly used for benchmarking in STS tasks.

External validity threats assess how general the paper's findings are. We have identified one main external validity threat related to the confidentiality of our test data. To maintain the confidentiality of Hitachi Energy's documents, we needed a dataset that could mimic their documentation. Consequently, we chose a dataset from MS MARCO to simulate Hitachi Energy's

[6] https://microsoft.github.io/msmarco/
[7] https://huggingface.co/models?pipeline_tag=sentence-similarity&sort=likes

documents, which are substantially different from each other. This also makes the experiment more relevant for other research and future work. With this in mind, the solution might perform differently depending on the dataset we are evaluating.

4.5 Ethical and societal considerations

Since our thesis revolves around sentence transformers and semantically searching through documentation, we have not been in contact with any critical personal data. Therefore, we do not have to take ethical precautions regarding that kind of data. However, the data we use to test our solution within Hitachi Energy is highly confidential. This data consists of technical specifications, electrical drawings and a lot more data. Because of this we have signed a contract with Hitachi where we are bound not to share confidential information for three years. To maintain the confidentiality of Hitachi Energy´s documents and keep our thesis's scientific value, we will use the MS MARCO dataset to simulate the real data and show how our solution performs. Because of the nature of company-specific vocabulary, there might be slight differences in the results of our solution, which can alter the conclusion of our thesis.

5. Experiment

In this section, we present our experiment to investigate *RQ2*. In the experiment, we have evaluated all the pre-trained semantic transformers provided by Huggingface via OpenSearch[8]. We conducted this evaluation through an experiment where we assessed the sentence transformers using the Mean Reciprocal Rank (MRR), Normalised Discounted Cumulative Gain (nDCG) and search speed in milliseconds on the MS MARCO Top 1000 Evaluation set. Together, these metrics satisfied Hitachi Energy's product requirements and made the experiment repeatable for others.

The experiment was conducted with 100,000 pairs of questions and answers. The original dataset has 8.8 million queries and answers, which, with our computer power, would take an estimated 370 hours to run. With this in mind, we had to limit our experiment to the first 100,000 pairs in the dataset due to time constraints. The result will be better or worse based on where the expected answer reveals itself in the ranking. The design of this experiment has been made to explicitly answer *RQ2*. The best-performing sentence transformers were then implemented in the final solution.

We chose to look at both $k = 10$ and $k = 100$ for MRR and nDCG to determine how often relevant answers would present themselves. A high score when $k = 10$ indicates that relevant answers present themselves in the top 10 answers more often than not. A high score when $k = 100$ means the user might need to dig deeper to find relevant answers in the documents.

5.1 Evaluation metrics

In this subsection, we present the evaluation metrics used in the experiment. The first two measurements described aim to investigate the first part of *RQ2*, which is to evaluate the accuracy of the sentence transformers. The last measurement presented aims to investigate the second part of *RQ2*, which is the speed of the search using the different sentence transformers.

We have defined accuracy based on how well the answers to a query match the user's intent. This can be measured by the ranking of the retrieved results. We have chosen two different metrics that consider the ranking of the retrieved results to measure accuracy. The first one is the $MRR@k$. Equation 1 demonstrates the summation to achieve the $MRR@k$ and the average result for the whole dataset. It sums up the values of the first correct answer to query one to j and divides it by its rank in the list of retrieved answers. The result is between one and zero, and a higher value resembles better accuracy, where a correct result is ranked higher in the list of answers. This will show if the retrieved results closely match the user's intent. The k limits how long the list of retrieved answers is. If the list does not contain a result, this gives that query a reciprocal rank of zero.

$$MRR@k = \frac{1}{j}\sum_{j=1}^{j}\frac{1}{rank_i} \tag{1}$$

The $MRR@k$ ranking evaluation method measures the quality of the recommendations retrieved by a search engine and how quickly a user will find a relevant answer. This suits the requirements of our work, where our goal is to evaluate if the implementation can be answered accurately and whether it is feasible to implement in a real-life scenario.

In order to strengthen our result further, we utilised the $nDCG$ metric to support our experiment and to give us a further grasp of the performance. $nDCG$ not only looks at the first correct result to measure accuracy but also looks further down the order of results and compares the ideal result against the actual result. This is done by dividing the Discounted Cumulative Gain (DCG) with the Ideal Discounted Cumulative Gain (IDCG) as can be seen in equation 2.

$$nDCG@k = \frac{DCG@k}{IDCG@k} \tag{2}$$

The Discounted Cumulative Gain (DCG) at rank k is given by equation 3. In the equation rel_i is the relevance of the document at rank i, which is assigned one if the result is relevant and zero if

[8] https://opensearch.org/docs/latest/ml-commons-plugin/pretrained-models/

it is irrelevant.

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{3}$$

$IDCG$ at rank k is calculated by sorting the relevant documents in descending order of relevance and computing the DCG for the sorted documents, as seen in equation 4.

$$IDCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{4}$$

Note that for both MRR and $nDCG$ the results are presented between zero and one, a higher score indicates that correct answers are ranked higher in the list of answers.

The second part of *RQ2* is the speed of the search. We have defined speed as how quickly an answer can be presented to the user. To evaluate the speed for each query we have chosen to time the execution of each query in milliseconds to achieve a precise score. These timed results are then summed up and averaged by the total number of queries. This can be seen in equation 5 where k is the number of queries and t is the time in milliseconds.

$$Average\ Speed = \frac{\sum_{i=1}^{k} t_i}{k} \tag{5}$$

5.2 Dataset

The experiment was conducted on the first 100,000 lines in the MS MARCO Top-1000 Evaluation dataset for passages [17]. The choice of the dataset was made through an observation and division of Hitachi Energies data and by observing the type of search queries our supervisor used to query the initial product. The documents were split and uploaded through an automation process using a Python script. This resulted in passages of different lengths which closely matches the MS MARCO top-1000 evaluation dataset, which contains different-sized passages. Through the observation of our supervisor's queries, we concluded that he often used short queries to unconditionally match against both long and short passages. This property is also found in the MS MARCO dataset. These two observations together made the MS MARCO the most suitable for our experiment. We also think that for other organisations a similar division of the data will be made which further motivates our choice of dataset and that it is generalised. Another remark that also was crucial when choosing the dataset was the repeatability and generalisation of the experiment. To address the repeatability of the experiment we limited our search for datasets to only publicly available, which the MS MARCO dataset is. The dataset contains of an question with an ID paired with an answer also with an ID. An example of a line from the dataset is the question, "*what class was the uss caine*" paired with the answer "*The Navy's Most Expensive Aircraft Carrier Ever Just Got Even More Expensive. And surprise, surprise – the third ship of the class will be the most expensive yet. Meet the USS John F. Kennedy (CVN 79). One day, it will be a beautiful boat – if we can afford it.*"

5.3 Experimental process

The experiment is illustrated in Figure 6 and was run on our personal laptop with an AMD Ryzen 7 4800U with Radeon Graphics CPU and 16 GB of RAM. We used a Docker environment to conduct the experiment and set up a cluster of OpenSearch nodes (version 2.13.0) within a container using a docker-compose file. This file is set to create four nodes: two data storage nodes, one machine learning node dedicated to running sentence transformers and other AI algorithms, and a node running the dashboard for interacting with the cluster. After the initialisation of this file, a sentence transformer was uploaded to OpenSearch, and an index for the data was created. The index contained the answer, its ID and the sentence embedding. Secondly, the MS MARCO dataset answers were inserted into the OpenSearch database through a pipeline utilising the uploaded sentence transformer to embed the sentence. Subsequently, a semantic search was performed with the queries from the MS MARCO dataset, which was sent through a search pipeline for sentence embedding before the OpenSearch engine did a k-NN search between the query and

the uploaded dataset and then a cosine similarity between the results embedding and the query embedding. This part captures the closest semantic results to the query and lists the retrieved results in descending order based on their cosine similarity. This process was timed and averaged over all queries. The retrieved answers were then measured against the expected results in the MS MARCO dataset using the nDCG and MRR scoring metrics. Steps two, three and four in the process were automated using Python to ensure no human errors were made during the testing. The results were documented for later analysis, and the experiment was repeated for all chosen sentence transformers.

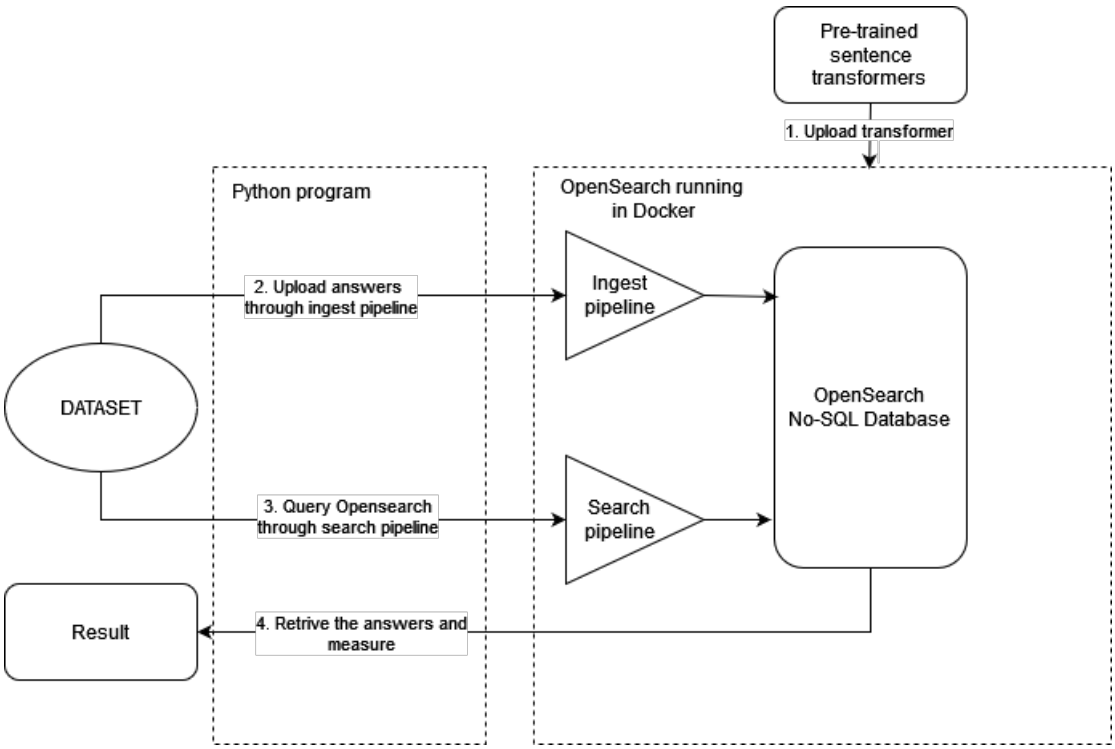


Figure 6: An overview of the experiment

5.4 Validation activity

To validate the work, we hosted a workshop at Hitachi Energy that included three global managers, one line manager and one preeminent. During the validation activity, we presented the work process to implement the product, some background knowledge, and an insight into the tools used and how they are combined. To receive feedback on the product, validate it, and get propositions for future work, we let the participants test the solution with a small sample dataset from Hitachi Energy. For easier integration and use of the product, we added a GUI created with PyQt6 to provide better visibility of search results and a more intuitive use of the product. The GUI queries OpenSearch on an index loaded with text and embeddings from the Hitachi Energy corpus. The validation activity also aimed to give feedback on the work processes and identify weaknesses in it.

6. Results

In this section, we provide the results we attained from our literature review and our experiment.

6.1 Which No-SQL databases with search-engine interfaces exists and what are their benefits? (RQ1)

We identified these three No-SQL databases layered with a search engine interface:

- ElasticSearch is built upon the Lucene library and is a full-text search engine with a distributed search and analysis engine at the centre of the elastic stack. ElasticSearch utilises the Elastic License.
- OpenSearch is originally a fork from version 7.10.2 of ElasticSearch. This search engine also builds upon the Lucene Library and supports full-text search. OpenSearch utilises the Apache License Version 2.0 (ALv2).
- Apache Solr is also built upon the Lucene library and is a full-text indexing and search engine. The Apache Solr operations have three main steps: indexing, querying, and ranking. Apache Solr utilises the ALv2 license.

All of the listed search engine databases are built upon Apache Lucene, which is an open-source software library dedicated to building search engines. This software is built to index and manage documents within a No-SQL database. This similarity also makes the different options similar in performance.

Table 1 visualise the benefits of the different identified search engines, and is based on the literature review and identification of these papers: [2], [18]–[25] where different identified factors for the search engine databases have been identified. We found six categories that we looked at: *document storage* which is how the data is stored, *free of charge* which is if all functionalities within the software are available without extra charge, if the solution supports *Dense vector search* which is the technique used when doing semantic search, if the software has *built-in sentence transformers*, *supports third party embedding models* and if you need to *pre-embed sentences before inserting* them in the database.

Capability	OpenSearch	ElasticSearch	Apache Solr
No-SQL storage type	Document	Document	Document
Free of charge	X		X
Dense vector search (Semantic search)	X	X	X
Built-in sentence transformers	X	X	
Supports third party embedding models	X	X	
Pre-embedding of sentences before insertions			X

Table 1: Table of benefits of No-SQL databases

Table 1 and the literature review resulted in a decision to implement the open-source architecture called OpenSearch. OpenSearch fulfils all the needs of our solution. It is a No-SQL database with a search engine interface that can handle the semantic transformers used for semantic search. It is also open-source, which means it is free. With these factors in mind, Hitachi Energy and future researchers could deploy a similar solution within their system for further development or utilise one of the other mentioned databases.

6.2 Which pre-trained semantic transformer provides the best accuracy and speed for semantic search within a No-SQL database? (RQ2)

This subsection presents the results related to *RQ2* and additional information about the tested transformers. The goal of answering this RQ is to clarify which of the sentence transformers would perform the best on the earlier mentioned MSMARCO dataset, thus being the model implemented in the solution presented to Hitachi Energy. In Table 2 we present the metrics documented during the experiment. A low score in MRR and $nDCG$ indicates that the correct answer was ranked lower than their expected rank in the list of answers. A low average time per query indicates that the transformer executes a query quickly. Table 2 shows that four-sentence transformers stand out, displaying superior performance compared to others across the different metrics. Through analysis of the result, we saw that the fluctuation of the results is low and that there are two transformers, distiluse-base-multilingual-cased-v1 and paraphrase-multilingual-MiniLM-L12-v2 v1, which are anomalies and performed significantly worse than the others regarding both MRR and $nDCG$.

Sentence transformer	MRR@10	MRR@100	$nDCG$@10	$nDCG$@100	Avg. $ms/query$
alldistilroberta-v1	0.3377	0.3424	0.2546	0.4039	203
all-MiniLM-L6-v2	0.3617	0.3654	0.2876	0.4484	73
all-MiniLM-L12-v2	0.3572	0.3611	0.2824	0.4422	53
all-mpnet-base-v2	0.3470	0.3514	0.2640	0.4220	162
msmarco-distilbert-base-tas-b	0.3540	0.3582	0.2650	0.3867	94
multi-qa-MiniLM-L6-cos-v1	0.3566	0.3605	0.2810	0.4360	67
multi-qa-mpnet-base-dot-v1	0.3223	0.3278	0.2341	0.3722	169
paraphrase-MiniLM-L3-v2	0.3580	0.3618	0.2817	0.4192	74
paraphrase-multilingual-MiniLM-L12-v2	0.3060	0.3115	0.2202	0.3458	55
paraphrase-mpnet-base-v2	0.3341	0.3389	0.2459	0.3829	177
distiluse-base-multilingual-cased-v1	0.2531	0.2617	0.1648	0.2930	94
Average	0.3352	0.3401	0.2528	0.3957	111

Table 2: Result from the experiment

Table 3 shows the top four sentence transformers and was created to enhance the readability. When looking at Table 3, we see that it is close between which transformer could be declared as the best from this experiment, as they perform similarly in all metrics. We found that there are two winners, one in the category of best accuracy and one with the best speed. The all-MiniLM-L6-v2 performs the greatest regarding MRR and $nDCG$, while the all-MiniLM-L12-v2 has the best average time per query. Notably, the time difference in speed between all-MiniLM-L6-v2 and all-MiniLM-L12-v2 is at a low 20ms.

Sentence transformer	MRR@10	MRR@100	$nDCG$@10	$nDCG$@100	Avg. $ms/query$
all-MiniLM-L6-v2	0.3617	0.3654	0.2876	0.4484	73
all-MiniLM-L12-v2	0.3572	0.3611	0.2824	0.4422	53
paraphrase-MiniLM-L3-v2	0.3580	0.3618	0.2817	0.4192	74
multi-qa-MiniLM-L6-cos-v1	0.3566	0.3605	0.2810	0.4360	67

Table 3: Top four sentence transformers

Table 4 shows additional information[9] for the transformers tested in the experiment. Table 4 and Table 2 show no correlation between the embedding dimensions and the accuracy. The worst transformer in the category of speed was the alldistilroberta-v1, which with its 203ms, took 92ms more than the average transformer to search. We also found that paraphrase-mpnet-base-v2 is an anomaly with 512 dimensions and is the worst performer in MRR and $nDCG$, it also performs badly when looking at the speed. Notably, there is also no correlation between significantly better accuracy and the transformers optimised for semantic search. In Table 4, we list the transformers, version number, the dimension of the embedding the transformer creates, and if the transformer has been specifically trained on question and answer pairs to optimise for semantic search.

Sentence transformer	Version	Embedding dimensions	Optimised for semantic search
all-distilroberta-v1	1.0.1	784	
all-MiniLM-L6-v2	1.0.1	384	
all-MiniLM-L12-v2	1.0.1	384	
all-mpnet-base-v2	1.0.1	768	
msmarco-distilbert-base-tas-b	1.0.2	768	X
multi-qa-MiniLM-L6-cos-v1	1.0.1	384	X
multi-qa-mpnet-base-dot-v1	1.0.1	384	
paraphrase-MiniLM-L3-v2	1.0.1	384	
paraphrase-multilingual-MiniLM-L12-v2	1.0.1	384	
paraphrase-mpnet-base-v2	1.0.0	768	
distiluse-base-multilingual-cased-v1	1.0.1	512	

Table 4: Additional information about the transformers

[9] https://opensearch.org/docs/latest/ml-commons-plugin/pretrained-models/

7. Discussion

The primary objective of our thesis was to investigate the feasibility of a solution incorporating AI to search historical documentation. Due to the time constraint and our decision to embrace the engineering method, we explored existing software with built-in search engines rather than building our own software. The findings from section 6.1 show three search engines that fit our purpose and our needs for the solution. The benefits of the search engines found in section 6.1 led to our choice for the final solution, OpenSearch. The main reason for this choice was the flexibility that OpenSearch provides and that it is completely free to use all the functionalities. We still acknowledge that our findings do not look at the drawbacks of each solution, mainly because our thesis aims to look at the feasibility of a solution, rather than the most optimal solution. We concede that the other solutions mentioned could have performed better regarding speed, memory usage and redundancy than ours, based on some of the papers mentioned in section 6.1, which claims that ElasticSearch is faster in execution. Notably, all of the identified search engines are built upon the Apache Lucene library. This would indicate that they would still perform fairly similarly and prove easy to switch in between due to the underlying architecture of the software. The latter assumption is also strengthened by the switch from ElasticSearch to OpenSearch at CERN [18].

In the result of the experiment in section 6.2, we establish that the best sentence transformer for our solution is ”all-MiniLM-L6-v2”, which outperformed all of the other sentence transformers in all of our chosen metrics, except for the average time per query. We overlooked the fact that the sentence transformer was slower than the second-best. We see the difference, 20ms, as negligible and focus more on the other scores instead due to them indicating that a correct answer could be found higher in the list of answers. Finding an answer earlier would impact the total search speed, from when the user enters a prompt to finding the correct answer. An interesting discovery was that the sentence transformers specifically optimised for semantic search were not the sentence transformers performing the best accuracy. Only one of them, ”multi-qa-MiniLM-L6-cos-v1”, was placed in the top four. That said, the scores are very close in the top four and only differ in the hundredth decimal, making us believe that all of the top four sentence transformers could perform well in our solution. However, the result could have been different because the sentence transformer could have performed better with more experimental time and computer power. We only tested the first 100,000 queries and answers for our experiment since performing the experiment for one transformer would have taken around 370 hours. This may have negatively impacted the result, but with the amount of queries tested, we argue that we got close to a correct result. Another not so surprising remark in the results is that the transformers that create larger dimensional embedding also perform worse at speed. This indicates that the k-NN utilised by OpenSearch for semantic search, as expected, performs worse on larger embedding vectors. What is more notable is when we look at the dimensions correlated to the accuracy result. Here, we find that larger dimensions don’t necessarily mean more accurate results, which, to us, is counterintuitive. The larger dimension should reasonably capture more small differences in the passages and sentences.

Similar to Xian et al. [2], our work demonstrates the easy combination of existing tools to create and use vector search as a powerful solution. We also built on their work with our experiment, where we tested part of the AI tools provided by OpenSearch. It also demonstrates that it is most definitely feasible to achieve search with AI in historical documentation through the validation activity at Hitachi Energy.

It’s also worth mentioning that our work doesn’t focus on developing the performance of the sentence transformers tested. A proper fine-tuning, training, and testing process dedicated to a specific dataset such as the one from Hitachi Energy would be interesting. While validating with Hitachi Energy, we found indications that such work would prove beneficial and that a model dedicated to a specific dataset should far outperform the pre-trained models.

After finding out which sentence transformer performed best, we implemented that sentence transformer in our final solution, which we presented to Hitachi Energy HVDC at our planned workshop with Hitachi Energy. The final solution involves a graphical user interface (GUI) tool shown in appendix A: where you can enter a query, change how many answers you want to receive, change the k-value, see the score of each answer, see which document each answer lays in and the number of answers from each document. Originally, we wanted to use a command line interface,

but after discussing the matter with our supervisor from Hitachi Energy, we came to the conclusion that it would be more intuitive to use a GUI, which people easier could become accustomed to. While developing and evaluating the tool, we found that some answers were duplicates. The reason is that the answers appear several times in the documents, sometimes in the same document and sometimes in different documents. This is why we wanted to include the document from which the answer came from.

The common theme of the workshop was their enthusiasm for us showing the possibilities of this kind of solution. And we were later invited to present our work and process to another team at Hitachi Energy Energy. The workshop also provided good discussion to promote future work from the people who participated. It also gave insight into how large corporations could use this tool. The test we conducted with their dataset presented good results whereas queries, written by the participants, presented relevant results within a respectable span in their eyes. The participants were satisfied with the work process, and that the work combined with the test showed that it is feasible to use AI to search the documentation. The workshop we held led to another presentation for a different team at Hitachi Energy, which means that our findings are significant and important.

Finding a solution to semantic search has not been an easy task, we have been investigating different solutions and decided to invest our time and research in a solution involving dense vector search. Usually when investigating a problem like this, the dataset that is being tested is created for a specific task in mind. However, our approach, targeting real companies with real documentation has not been explored yet. Each part of our solution has its own research behind it, however, dense vector search, using a No-SQL database together with sentence transformers has not been tested on real documentation before. In order to use our work as an out-of-the-box solution some steps are required. The configuration files can be found on GitHub[10].

1. Download docker
2. Initiate the OpenSearch in the docker environment.
3. Choose the sentence transformer and take all steps initiating it and the pipelines within OpenSearch.
4. Identify fields and post an index containing the fields.
5. Upload the files using the Python script, modifications to the script for specific data to correctly parse it are necessary.
6. GUI will need minor changes to accept and present the answers. Fields need to match those identified in step three.

The advantage of our solution is the time needed to find the correct answer and document. With our solution, you can query and retrieve the list of answers in less than a second. The potential of this kind of solution is great, the research is in an early phase and as AI grows stronger, these solutions will to. If more companies and researchers decided to put effort into solving this kind of problem, more powerful solutions would arise, which could out-compete the more traditional keyword search.

[10] https://github.com/edde-w/Semantic-Search

8. Conclusions and future work

Our thesis has investigated the feasibility of incorporating AI to search with semantical meaning through a large amount of historical documentation. To achieve this, we have used the engineering method, where we, through a literature review, looked at current solutions, implemented one of them and through an experiment, evaluated the tools used for searching with AI in the solution and lastly analysed the results. Through the literature review, we identified three different search engines, OpenSearch, ElasticSearch, and Apache Solr, that could be used to store and enable search in the documentation. Notably, is that all of them are built upon the Apache Lucene library for managing the storing and search for data. For this work, we found OpenSearch most suitable for our needs. We proceeded with the experiment evaluating sentence transformers within the OpenSearch environment using the first 100,000 rows in the MS MARCO top 1000 Eval set, using MRR and $nDCG$ as our metric scores. We found that the best-performing sentence transformer for semantic search regarding accuracy measured with MRR and $nDCG$ was all-miniLM-L6-v2, and with regards to speed, all-miniLM-L12-v2. The most surprising finding was that the transformers specially trained for semantic search and transformers with bigger dimensions didn't perform significantly better than the rest. While our work doesn't present a finished product, it indicates the feasibility of incorporating AI to search historical documentation and how easy it is to use current tools. This aligns closely with earlier work done by Xian et al. [2].

Looking ahead at future work, our thesis suggests that training and evaluating a dedicated sentence transformer for organisation-specific language and data would be interesting to see if better accuracy and speed could be achieved. We suggest that such work should incorporate labelling and gathering a dataset with question-and-answer pairs that are specifically made for the use case.

References

[1] A. Esteva, A. Kale, R. Paulus *et al.*, 'Covid-19 information retrieval with deep-learning based semantic search, question answering, and abstractive summarization,' *npj Digit. Med.*, vol. 4, p. 68, 2021. DOI: 10.1038/s41746-021-00437-0.

[2] J. Xian, T. Teofili, R. Pradeep and J. Lin, 'Vector search with openai embeddings: Lucene is all you need,' in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, ser. WSDM '24, , Merida, Mexico, Association for Computing Machinery, 2024, pp. 1090–1093. DOI: 10.1145/3616855.3635691.

[3] C. Ni, J. Wu, H. Wang, W. Lu and C. Zhang, *Enhancing cloud-based large language model processing with elasticsearch and transformer models*, 2024. arXiv: 2403.00807 [cs.IR].

[4] N. Banothu, S. Bhukya and K. Sharma, 'Big-data: Acid versus base for database transactions,' Mar. 2016, pp. 3704–3709. DOI: 10.1109/ICEEOT.2016.7755401.

[5] M. Muniswamaiah, T. Agerwala and C. C. Tappert, 'Comparison of sql, nosql, and newsql database technologies,' in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 6230–6232. DOI: 10.1109/BigData59044.2023.10386522.

[6] E. Tang and Y. Fan, 'Performance comparison between five nosql databases,' in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, 2016, pp. 105–109. DOI: 10.1109/CCBD.2016.030.

[7] H. Li and J. Xu, 'Semantic matching in search,' *Foundations and Trends® in Information Retrieval*, vol. 7, no. 5, pp. 343–469, 2014. DOI: 10.1561/1500000035.

[8] N. Muennighoff, 'Sgpt: Gpt sentence embeddings for semantic search,' *ArXiv*, vol. abs/2202.08904, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246996947.

[9] J. Devlin, M. Chang, K. Lee and K. Toutanova, 'BERT: pre-training of deep bidirectional transformers for language understanding,' *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805.

[10] A. Vaswani, N. Shazeer, N. Parmar *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[11] T.-T. Ha, V.-N. Nguyen, K.-H. Nguyen, K.-A. Nguyen and Q.-K. Than, 'Utilizing sbert for finding similar questions in community question answering,' in *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, 2021, pp. 1–6. DOI: 10.1109/KSE53942.2021.9648830.

[12] B. Li, H. Zhou, J. He, M. Wang, Y. Yang and L. Li, 'On the sentence embeddings from pre-trained language models,' in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online: Association for Computational Linguistics, Nov. 2020, pp. 9119–9130. DOI: 10.18653/v1/2020.emnlp-main.733.

[13] N. Reimers and I. Gurevych, 'Sentence-bert: Sentence embeddings using siamese bert-networks,' en, in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Event Title: The 2018 Conference on Empirical Methods in Natural Language Processing, Hong Kong, China: Association for Computational Linguistics, Aug. 2019. [Online]. Available: https://tubiblio.ulb.tu-darmstadt.de/117723/.

[14] Y. Zhang, D. Long, G. Xu and P. Xie, *Hlatr: Enhance multi-stage text retrieval with hybrid list aware transformer reranking*, May 2022. DOI: 10.48550/arXiv.2205.10569.

[15] V. R. Basili, 'The experimental paradigm in software engineering,' in *Experimental Software Engineering Issues*, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:14982809.

[16] P. Runesson and M. Höst, 'Guidelines for conducting and reporting case study research in software engineering,' *Emperical Software Engineering*, pp. 131–164, 2 2009. DOI: 10.1007/s10664-008-9102-8.

[17] P. Bajaj, D. Campos, N. Craswell *et al.*, *Ms marco: A human generated machine reading comprehension dataset*, 2018. arXiv: 1611.09268 [cs.CL].

[18] S. Papadopoulos, P. Saiz and S. Ulrich, 'Architecting the opensearch service at cern,' in *26th International conference on computing in high energy and nuclear physics (CHEP2023)*, 2023.

[19] Z. Parker, S. Poe and S. V. Vrbsky, 'Comparing nosql mongodb to an sql db,' in *Proceedings of the 51st ACM Southeast Conference*, ser. ACMSE '13, Savannah, Georgia: Association for Computing Machinery, 2013, ISBN: 9781450319010. DOI: 10.1145/2498328.2500047.

[20] J. Hansen, K. Porter, A. Shalaginov and K. Franke, 'Comparing open source search engine functionality, efficiency and effectiveness with respect to digital forensic search,' 2018. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2584227/577-Article%2BText-1167-1-10-20181009.pdf?sequence=2&isAllowed=y.

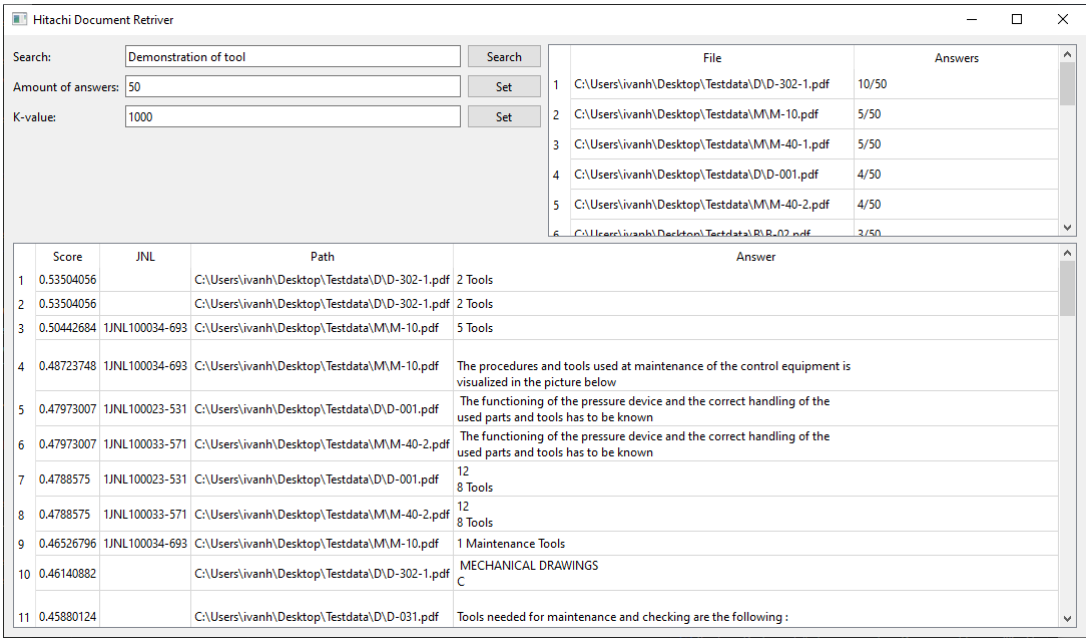
[21] X. Chang, 'The analysis of open source search engines,' *Highlights in Science, Engineering and Technology*, vol. 32, pp. 32–42, Feb. 2023. DOI: 10.54097/hset.v32i.4933.

[22] N. Kathare, O. V. Reddy and V. Prabhu, 'A comprehensive study of elasticsearch,' *International Journal of Science and Research (IJSR)*, 2020.

[23] S. Gupta and R. Rani, 'A comparative study of elasticsearch and couchdb document oriented databases,' in *2016 International Conference on Inventive Computation Technologies (ICICT)*, vol. 1, 2016, pp. 1–4. DOI: 10.1109/INVENTIVE.2016.7823252.

[24] G. Liu, C. Li, W. Tian and Z. Li, 'Distributed geospatial data service based on opensearch,' in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016, pp. 100–104. DOI: 10.1109/CompComm.2016.7924673.

[25] R. LeVan, 'Opensearch and sru: A continuum of searching,' *Information Technology and Libraries*, vol. 25, no. 3, pp. 151–153, Sep. 2006. DOI: 10.6017/ital.v25i3.3346.

A Appendix A

A.1 Graphical user interface for Hitachi