TRIBHUWAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
PROJECT REPORT
ON

QUESTION SIMILARITY DETECTION AND ANALYSIS

SUBMITTED BY
MILAN SHRESTHA (075BCT050)
NISCHAL SHAKYA (075BCT055)
NITESH SWARNAKAR (075BCT058)
ROSHAN SUBEDI (075BCT068)

SUBMITTED TO
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

30TH APRIL, 2023

TRIBHUWAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
PROJECT REPORT
ON

QUESTION SIMILARITY DETECTION AND ANALYSIS

SUBMITTED BY
MILAN SHRESTHA (075BCT050)
NISCHAL SHAKYA (075BCT055)
NITESH SWARNAKAR (075BCT058)
ROSHAN SUBEDI (075BCT068)

SUBMITTED TO
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

IN PARTIAL FULFILMENT FOR THE AWARD OF
BACHELOR IN COMPUTER ENGINEERING.

UNDER THE SUPERVISION OF
ASSOC. PROFESSOR SHARAD KUMAR GHIMIRE

30TH APRIL, 2023

Declaration

We hereby declare that the report of the project titled **"Question Similarity Detection and Analysis"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Pulchowk Campus,** in the partial fulfilment of the requirements for the award of the Degree of **Bachelor in Computer Engineering** is a bonafide report of the work by us. The materials in this report have not been submitted to any University or Institute for the award of any degree. We are the only authors of this complete work; no sources other than those listed here have been referenced in the completion of this project.

Milan Shrestha	(075BCT050)	____________
Nischal Shakya	(075BCT055)	____________
Nitesh Swarnakar	(075BCT058)	____________
Roshan Subedi	(075BCT068)	____________

Date: 30th April, 2023

Certificate of Approval

The undersigned certifies that they have read and recommended to the **Department of Electronics and Computer Engineering**, Institute of Engineering for acceptance of a project report titled **Question Similarity Analysis and Detection**, submitted by **Milan Shrestha, Nischal Shakya, Nitesh Swarnakar, Roshan Subedi** in partial fulfilment of the requirements for the degree of **Bachelor in Computer Engineering.**

....	....
Supervisor	Internal examiner
Sharad Kumar Ghimire	
Associate Professor	
Department of Electronics and	
Computer Engineering,	
Pulchowk Campus, IOE, TU.	

....
External examiner

Date of approval:

Copyright

Acknowledgement

We would like to express our deepest gratitude to all those who have contributed to the success of this project.

First and foremost, we extend our sincerest appreciation to our project supervisor, Assoc. Professor Sharad Kumar Ghimire, for his invaluable guidance, insightful feedback, and unwavering support throughout the entire project His advice were instrumental in shaping our ideas and enabling us to overcome various challenges.

We would also like to thank the Rara Labs, which was critical to the completion of our project. Their commitment, hard work, and dedication to excellence were truly inspiring.

Additionally, we extend our gratitude to Naamche for their support and encouragement throughout this project. Their feedback and suggestions were valuable in helping us refine our ideas and achieve our goals. Finally, we would like to acknowledge the support of the Department of Electronics and Computer Engineering, our family and friends, who have been a constant source of motivation and encouragement throughout this journey.

Thank you all for your contributions and support.

Milan Shrestha (075BCT050)
Nischal Shakya (075BCT055)
Nitesh Swarnakar (075BCT058)
Roshan Subedi (075BCT068)

Abstract

The project aims to explore the effectiveness of using the SBERT model and vector database for performing question similarity analysis. The project involves building a vector database by training a sentence transformer model on a large corpus of text data. The vector dataset is then used to perform question similarity analysis by retrieving similar questions and similarity scores to a given search query. The model is trained on a large corpus of ALLNLI datasets, other paraphrase datasets such as MRPC, and PAWS, and the semantic similarity of datasets such as STS and finally adapted on 9,282 custom-prepared engineering datasets. The sentence transformer model is trained using the aforementioned datasets with MNR Loss as the loss function. The effectiveness of the model is evaluated by using the STS test dataset and test set of the MRPC. The result of the project demonstrates that using a sentence transformer model and vector database for question similarity analysis outperforms the baseline method of keyword matching. The approach achieved a spearman correlation value of 0.863 on the STS benchmark and an accuracy of 88.7% on the MRPC test. The Spearman correlation value in the SBERT paper for the NLI-large dataset was below 0.80. These values show that continuous training of the model on other datasets besides NLI helps to increase the performance and performs better for downstream tasks. This suggests that the use of the sentence transformer model and vector database is a promising approach for performing question similarity analysis, which could have significant implications for information retrieval systems.

Keywords: *indexing*, *information retrieval*, *sentence transformer*, *vector database*

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Annoy Approximate Nearest Neighbors Oh Yeah.

API Application Programming Interface.

AWS Amazon Web Services.

BERT Bidirectional Encoder Representations from Transformers.

BLEU Bilingual Evaluation Understudy.

CBOW Continuous Bag of Words.

CDN Content Delivery Network.

CNNs Convolutional Neural Networks.

COBERT COVID-19 Question Answering System Using BERT.

CoLA Corpus of Linguistic Acceptability.

CRFs Conditional Random Fields.

FAISS Facebook Artificial Intelligence Similarity Search.

GPT Generative Pre-Trained Transformer.

GPU Graphical Processing Unit.

HMMs Hidden Markov Models.

HNSW Hierarchical Navigable Small World.

HTTP Hypertext Transfer Protocol.

IBM International Business Machine.

JSON JavaScript Object Notation.

LSTM Long short-term Memory.

MEMMS Maximum Entropy Markov Models.

MNLI Multi-Genre Natural Language Inference.

MNR Multiple Negative Ranking.

MRPC	Microsoft Research Paraphrase Corpus.
MVC	Model-View-Controller.
MVP	Minimum Viable Product.
NLI	Natural Language Inference.
NLP	Natural Language Processing.
PAWS	Paraphrase Adversaries from Word Scrambling.
QPS	Queries per second.
RNNs	Recurrent Neural Networks.
SBERT	Sentence Bidirectional Encoder Representations from Transformers.
SDLC	Software Development Life Cycle.
SNLI	Stanford Natural Language Inference.
SQuAD	Stanford Question Answering Dataset.
STS	Semantic Textual Similarity.
STS-B	Semantic Textual Similarity Benchmark.
UI	User Interface.
UKP	Ubiquitous Knowledge Processing.
VCS	Version Control System.
VSCode	Visual Studio Code.

1 Introduction

1.1 Background

Natural Language Processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence that studies how computers and human language interact, with a focus on how to design computers to process and analyse massive amounts of natural language data. Due to the abundance of textual data available on the internet and other digital sources, the field of NLP has been extremely popular. The goal of NLP is to enable computers to recognize and understand the nuances of human language, such as idiomatic expressions, and regional dialects. In recent years, the field of NLP has seen tremendous growth, with the development of more complex and powerful language models. We have also seen the boom of large language models like GPT-3 by OpenAI, Bard by Google, and Sparrow by Deepmind among various others.

This project aims to explore the application of NLP in the field of similarity detection and implement this aspect in determining the repetition of questions in the examination of engineering students. NLP can be implemented to find the similarity between questions and documents, which can be used for semantically and lexically related texts. Its use case can be to prevent the frequency of repetition of questions and maintain the quality of the question paper. There is a huge demand for such implementation in universities around the world as it would greatly reduce the time it takes to examine the quality of question papers as well as reduce the human errors associated with it. The recent cases of examination of TU have highlighted the absence of such mechanisms to prevent the repetition of questions here in Nepal.

The project will focus on creating tools and applications that can be used by researchers and practitioners in various industries.

1.2 Motivation

Similarity detection is a popular NLP technique and is a growing field of research. The news of a complete question paper repetition with the examination paper of the previous year was about 8 months before. This incident highlighted the need for an effective similarity detection tool in the education sector in order to maintain academic integrity. Plagiarism detection tools that are currently available are limited in their ability to detect more sophisticated forms of similarity, such as paraphrasing and rewording. In addition to this, the existing tools are not optimized for detecting similarities in academic contexts, where the use of specialized vocabulary and technical terms is common.

Therefore, this project aims to develop a more advanced similarity detection tool using NLP techniques, which can accurately identify similarities in academic questions and flag potential instances of question repetition. The tool will be trained on a large corpus of academic texts and will leverage the latest advancements in NLP to detect subtle nuances in language and identify patterns of similarity that may not be apparent to the human eye. By providing educators with a more robust tool for detecting similarity, this project can help maintain academic integrity and ensure a fair and level playing field for all students.

1.3 Project Objectives

The main objective of the project is to provide educators with the tool to check the similarity between questions so that they are able to make more informed decisions regarding the setting of examination papers (or question papers). With an NLP-based system built to check for question similarity, problems such as similar questions in subsequent examinations can be minified.

1.4 Project Scope and Applications

The use case of this type of similarity-detecting tool modeled to understand the domain-specific data and assist educators in making better decisions is very large. It can be implemented in various educational bodies to help for better decision-making while setting the questions. This will help to maintain the academic standard in examination question papers.

1.5 Feasibility Analysis

1.5.1 Economic Feasibility

As this project is completely software-based, the economic feasibility of the project includes its economic appropriateness with respect to its presented output. It is not economically feasible if the result it provides cannot justify its capital budget and operational budget. The main expense incurred in this project includes the service fees of the hardware used to train the machine learning model i.e. AWS Sagemaker and in hosting the cloud database.

After the project is completed if we are to deploy the system, then it would incur additional expenses required for the deployment of all the resources. The cost includes using an inference server for generating the result based on the query provided by the user, hosting the database for storing all the embeddings as well as making the proper backups of the database. Also, the cost will be included in expanding the database as more and more questions will be accumulated over time. Since the pattern of the data may change over time so, further resources for training the model either in offline mode or online mode will be required. Further costs will incur on deploying the frontend as well as backend services.

1.5.2 Technical Feasibility

The technical feasibility is measured on the basis of the availability of technical resources. We have used Python libraries like Pandas and Numpy for preparing the data. We also used the popular frontend framework React and for backend Django in an attempt to keep technical requirements simple and feasible. The project will require a solid understanding of all these technical components for it to be feasible.

1.5.3 Schedule Feasibility

Our initial study provided us with the confidence that the project can be completed within the allocated time interval, so the project was deemed feasible from the perspective of time constraints.

1.5.4 Legal Feasibility

There are no legal constraints hindering the completion of the project. However, we may need some legal permits to establish the on-premise server system. Furthermore, if the project is decided to be deployed in cloud services, then huge legal constraints may be present as the law does not permit the internal data to be stored outside the country.

2 Literature Review

2.1 Related Work

(Mathematical Structures of Language-1951 or 1968)[7] proposed a mathematical framework for analysing the distribution of words in a text, which paved the way for computational methods in language analysis and influenced the development of computational linguistics and natural language processing.

The paper "A Neural Probabilistic Language Model-2003"[2] introduced a neural network-based language model that effectively captures long-term dependencies between words in a sentence and outperforms traditional n-gram models. It laid the foundation for the development of other neural network-based language models.

(Natural Language Processing (Almost) from Scratch-2011)[3] introduced a neural network-based approach to natural language processing that learned representations of words from raw text data, without requiring extensive feature engineering. This approach achieved state-of-the-art results on several NLP tasks and paved the way for the development of other neural network-based models for NLP, contributing to the rise of deep learning in general.

Another paper from 2013 named "Efficient Estimation of Word Representations in Vector Space"[8] introduced a method for representing words as vectors in a high-dimensional space, using two models called Continuous Bag of Words (CBOW) and Skip-Gram, also called the Word2Vec. This method improved the efficiency of representing words compared to one-hot encoding and captured semantic relationships between words. The learned word embeddings from Word2Vec improved the performance of various natural language processing tasks and has become popular. The paper [9] explained how LSTM networks work for processing sequential data like natural language text and had a significant impact on the development of LSTM-based models for various natural language processing tasks. LSTMs are able to effectively capture long-term dependencies in sequential data, such as natural language text.

In 2017, a team from google published a paper "Attention is all you need"[14] which introduced the transformer architecture based solely on attention mechanisms, avoiding recurrence and convolutions entirely, and is better at handling long-range dependencies, easier parallelization, ability to model bidirectional relationships, and requires less time to train. The proposed model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU.

Deep contextualised word representations - 2018[10] introduced ELMo, a deeply contextualised word representation that generates context-dependent word representations capturing complex linguistic phenomena like polysemy and syntax, which significantly improved performance in downstream natural language processing tasks (like question answering, textual entailments, etc) and outperformed the performance of the pre-trained word vectors that only produced a single context-independent representation for each word. In every task considered, simply adding ELMo establishes a new state-of-the-art result, with relative error reductions ranging from 6 - 20% over strong base models.

[5]"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" introduced a highly effective pre-training method for natural language processing using a transformer-based architecture. BERT is pre-trained on massive amounts of unlabeled text and can be fine-tuned on a variety of downstream NLP tasks. It achieved state-of-the-art results on a wide range of NLP benchmarks, including question-answering, sentiment analysis, and text classification. BERT processes text in both forward and backward directions, enabling the model to better understand the context of each word in a sentence. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 points absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 points absolute improvement).

Measuring Sentences Similarity: A Survey[15] suggests that the sentence similarity methods consist of three main categories: Word-to-word based, structure-based, and vector-based. Word-to-word-based and structure-based focus on lexical and structural similarity whereas vector-based focuses on the semantic word representations generated by deep learning models. Combining different approaches normally gives better results because it considers different aspects (lexical, syntactic, and semantic) of sentences.

[11] introduced a method for learning high-quality sentence embeddings that can capture the meaning and context of entire sentences. SBERT adds a pooling operation to the output of BERT to derive a fixed-sized sentence embedding. The construction of BERT makes it unsuitable for semantic similarity search as well as for unsupervised tasks like clustering.

SBERT drastically reduces the computation overhead for semantic similarity tasks and enables BERT to be used for new tasks like clustering, and information retrieval via semantic search. It reduced the time for finding the most similar pair from 65 hours with BERT/RoBERTa to about 5 seconds with SBERT in a collection of 10,000 sentences while maintaining the accuracy of the BERT.

COBERT: COVID-19 Question Answering System Using BERT - 2021[1] introduces COVID-19 Question Answering System Using BERT (COBERT), a fine-tuned version of DistilBERT on COVID-19-related information and addresses a critical need for a reliable and accurate question-answering system for COVID-19-related information. It outperformed previous pre-trained models obtaining an Exact Match(EM)/F1 score of 80.6/87.3 respectively.

FAQ Retrieval using Query-Question Similarity and BERT-Based Query-Answer Relevance[12] proposes a novel approach to FAQ retrieval that uses both query-question similarity and BERT-based query-answer relevance to improve the accuracy and relevance of retrieved answers. It achieves state-of-the-art results on the FAQ dataset, demonstrating its effectiveness in improving the accuracy of FAQ retrieval.

Sentence embeddings for Quora question similarity[6] used SBERT to build a question similarity classifier to detect duplicate questions in Quora Datasets, in order to provide a better Quora user experience. It achieved 91.56% test accuracy

An [13] approach to computing semantic similarity was investigated by mapping terms (concepts) to an ontology and by examining their relationships in that ontology. Some of the most popular semantic similarity methods are implemented and evaluated using WordNet as the underlying reference ontology. Building upon the idea of semantic similarity, a novel information retrieval method is also proposed. This method is capable of detecting similarities between documents containing semantically similar but not necessarily lexicographically similar terms. The proposed method has been evaluated in the retrieval of images and documents on the Web. The experimental results demonstrated very promising performance improvements over state-of-the-art information retrieval methods.

2.2 Related Theory

2.2.1 Natural Language Processing

NLP is a branch of computer science that deals with how computers can understand and analyse human language. It involves developing computer programs that can process, interpret and generate human language. NLP has many practical applications such as language translation, chatbots, sentiment analysis, and question-answering among many others. The recent craze regarding the capabilities of a large language model GPT-3 is also an example of NLP in action. This application was able to gather a user base of more than 1 million in mere 5 days. Over the years, there have been many research papers published on NLP that have contributed to the development of this field. Some of the notable papers include Word2Vec, Attention Is All You Need, Universal Language Model Fine-tuning for Text Classification, BERT, and GPT-3. These papers introduced various techniques and models that have been successful in different NLP applications. NLP is an ever-evolving field, and researchers continue to develop new models and techniques to improve the accuracy and efficiency of natural language processing.

The study of NLP started in the early 1600s by Rene Descartes and Gottfried Wilhelm Leibniz, who proposed codes that could relate words between languages. However, due to technological barriers, it was not until the early 1950s that the study of modern NLP took place. In 1954, the Georgetown-IBM experiment took place which demonstrated the automatic machine translation of 60+ Russian sentences to English. In the 1980s, rule-based models were developed which relied on hand-crafted rules designed to capture the various aspect of languages such as syntax, grammar, and semantics. Soon, statistical models like Hidden Markov Models (HMMs), Conditional Random Fields (CRFs), and Maximum Entropy Markov Models (MEMMS) were developed to overcome the limitations of rule-based models. This was soon followed by the use of deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) which followed the idea of learning hierarchical representations from text data, where lower-level features such as individual words and characters are combined to form higher-level representations such as phrases and sentences.

The substantial jump in the field of NLP which greatly influenced this project was the development of the transformer model in 2017 by a team of researchers from google. The team published the paper titled "Attention Is All You Need"[14] which contained the transformer model architecture. This architecture basically led way to today's advancement in study of Natural Language Processing (NLP). Transformers are based on the attention mechanism, which allows the model to selectively focus on different parts of the input sequence when making predictions. Transformers are said to be the state-of-the-art method in NLP tasks like machine translation, question-answering, and similarity search.

2.2.2 Tokenization

Tokenization is the process of breaking down a piece of text into smaller units called tokens. Tokens are typically words or individual punctuation marks, but they can also be phrases, subwords, or other units depending on the specific application. Tokenization is an important pre-processing step in NLP and machine learning tasks that involve text data. The purpose of tokenization is to convert raw text input into a format that can be easily processed by algorithms, by segmenting the input into meaningful units.

There are several different tokenization strategies used in NLP, including:

- Word tokenization
  This involves breaking up the input into individual words. For example, the sentence "The quick brown fox jumps over the lazy dog" would be tokenized into the words "The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", and "dog".

- Character tokenization
  This involves breaking up the input into individual characters. For example, Input text: "Hello, world!" then character tokens: "H", "e", "l", "l", "o", ",", " ", "w", "o", "r", "l", "d", "!"

- Subword tokenization
  This involves breaking up the input into smaller units such as syllables or subwords. This strategy is useful for handling rare or previously unseen words. For example, Input text: "unbelievable" then Subword tokens (using byte-pair encoding with a vocabulary size of 10): "un", "b", "el", "ie", "v", "a", "bl", "e"

- WordPiece tokenization
  WordPiece tokenization is a type of subword tokenization used in natural language processing, and it is also used in the BERT model. In WordPiece tokenization, words are broken down into smaller units called subwords based on the frequency of occurrence of each subword. The most common subwords are kept as individual tokens, while the less common ones are broken down into smaller pieces. For example: "I just got a funky phone case!" then Tokenized: ["I", "just", "got", "a", "fun", "##ky", "phone", "case"]. The characters ## suggest that this subword should be attached to the previous token.

Tokenization is often followed by additional processing steps such as stemming, lemmatization, or stopword removal, which can further refine the representation of the input data. The tokens thus produced are utilized for Embedding generation, sequence labelling, Language modelling, etc.

2.2.3 Dense Vector

In NLP, a dense vector is a numerical representation of a word, sentence, or document that captures its semantic meaning in a high-dimensional space. Dense vectors are also known as word embeddings, sentence embeddings, or document embeddings. Unlike sparse vectors which contain sparsely distributed bits of information, and only represent the presence or absence of a feature, dense vectors that are information-rich and densely packed contain continuous values and can capture the similarity and relationships between different features. They are learned through unsupervised training of neural network models, such as Word2Vec, GloVe, or BERT, on large corpora of text. Dense vectors have become a popular tool in NLP for tasks such as document classification, sentiment analysis, machine translation, and question-answering, among others. They allow NLP models to capture the semantic meaning of words, sentences, and documents, and to generalize to new or unseen data. Due to the nature of the vector, they can also be stored in specialized databases called vector databases for performing various NLP tasks such as semantic searching, and sentiment analysis. Since these text representations are numbers or floating points, different mathematical operations can also be applied to them to perform transformations that result in actual transformations in the semantics of the words. Dense vector embeddings for semantically similar entities are close to each other in vector space which can be measured by cosine similarity and distance.

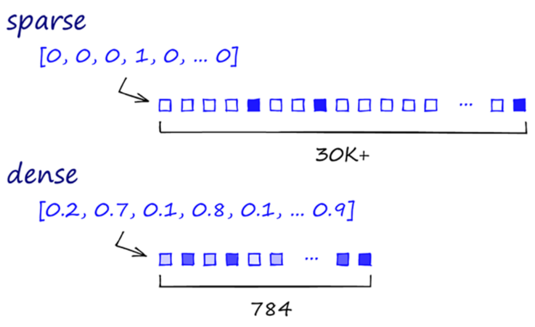


Figure 1: (From pinecone) Sparse Vectors vs Dense Vectors

2.2.4 Transformer

(Attention Is All You Need, 2017) [14] presented the transformer architecture which has revolutionalized the field of Natural Language Processing (NLP). [14] presents transformers as the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention. The architecture of the transformer is as shown below:

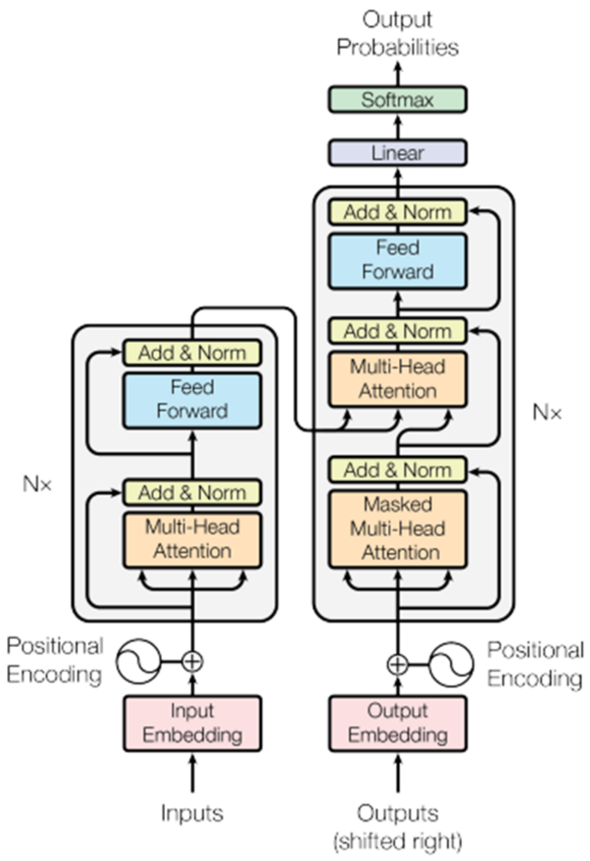


Figure 2: The Transformer-Model Architecture[14]

Encoder Stack and Decoder Stack

The architecture consists of two main sections divided into the Encoder stack and the Decoder stack.

- Encoder Stack
  The encoder is composed of a stack of 6 identical layers which has two sub-layers. The first sublayer is a multi-head self-attention mechanism and the second is a simple, position-wise fully connected feed-forward network.
- Decoder Stack
  Similar to the encoder layer, the decoder layer also consists of a stack of 6 identical layers. In addition to the two sub-layers of the encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder layer.

Attention

Attention refers to a mechanism in the Transformer model that allows the model to selectively focus on different parts of the input sequence when producing the output. The attention mechanism assigns weights to each element in the input sequence, based on its relevance to the current output element. These weights are then used to compute a weighted sum of the input sequence elements, which is used to produce the output.

Self-attention

As proposed by [14], self-attention allows the model to compute weights based on the similarity between different positions or "tokens" within the input sequence, unlike traditional attention mechanisms, which compute weights based on the similarity between a query and a set of key-value pairs.

Multi-head Attention

Multi-head attention is a type of attention mechanism used in Transformer models, as explained in the paper "Attention is all you need" [14]. It builds upon the basic self-attention mechanism by performing multiple attention computations in parallel, each with its own set of query, key, and value weights. By doing so, the model is able to capture various relationships between the tokens in the input sequence. This enables the model to represent the input more effectively and produce better results in a variety of natural language processing tasks.

2.2.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a deep learning-based pre-training technique for NLP developed by Google. BERT is based solely on the encoder component of the Transformer architecture, which enables it to generate high-quality contextualized embeddings for words in a sentence that capture the meaning of words in context. This is achieved by training a large-scale neural network on a diverse range of language modelling tasks, such as masked language modelling and next-sentence prediction. The pre-training concept in BERT involves training the model on a large dataset in an unsupervised manner using language modelling. This allows the model to understand the context of the input sentence. After pre-training, the model can be fine-tuned on a task-specific supervised dataset to achieve good results on that particular task. Pre-training on a larger dataset helps the model to learn more general language representations that can be effectively used in different downstream tasks. The name itself reflects the key features of the model as

Bidirectional

BERT is a bidirectional model, meaning that it takes into account both the left and right context of each word in a sentence. This allows the model to capture a more nuanced understanding of the meaning of words, as well as the relationships between words within a sentence.

Encoder

BERT is based on the transformer architecture, which is an attention-based neural network that can process sequences of variable length. The encoder component of BERT is responsible for generating contextualized embeddings for each word in the input sequence, based on the surrounding context.

Representations

BERT generates rich, multi-layered representations for each word in the input sequence, which capture both its syntactic and semantic meaning. These representations can be fine-tuned for a wide range of downstream NLP tasks.

Transformers

BERT is based on the transformer architecture, which uses self-attention mechanisms to allow the model to selectively focus on different parts of the input sequence, based on their relevance to the current task. This allows BERT to generate highly accurate and robust representations for a wide range of NLP tasks.

BERT uses WordPiece tokenizer which produces tokens that ultimately get converted to token IDs and corresponding segment IDs and attention masks. Token IDs represent indices of the tokens on the pre-trained vocabulary. The tokenizer also adds special tokens to the input sequence, including a [CLS] token at the beginning of the sequence to indicate the start of a new input, and a [SEP] token between segments or at the end of the sequence. The [CLS] token is used as the input for classification tasks, while the [SEP] token is used to separate multiple sentences or segments in the input. In addition to the token IDs, the tokenizer also generates segment IDs and attention masks for each token. The segment IDs indicate which segment each token belongs to, which is useful for tasks that involve processing multiple sentences or segments in a single input. The attention mask is a binary mask that indicates which tokens should be attended to and which should be ignored, based on the presence or absence of padding tokens. The padding tokens [PAD] are used to represent paddings to the sentence to make up the required length.

```
# Original Sentence
Let's learn deep learning!

# Tokenized Sentence
['Let', "'", 's', 'learn', 'deep', 'learning', '!']

# Adding [CLS] and [SEP] Tokens
['[CLS]', 'Let', "'", 's', 'learn', 'deep',
'learning', '!', '[SEP]']

# Padding
['[CLS]', 'Let', "'", 's', 'learn', 'deep',
'learning', '!', '[SEP]', '[PAD]']

# Output of BERT tokenizer
Token IDs:[101, 2421, 112, 188, 3858, 1996, 3776, 106, 102, 0]
Segment IDs:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Attention mask: [1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

In the above snippet, it should be noted that the [101] and [102] tokens correspond to the special [CLS] and [SEP]. A segment ID of 0 corresponds to the first sentence. An attention mask of 1 represents a valid token and 0 represents a [PAD] token.

The output of the pre-trained BERT tokenizer is used to create input embeddings for the BERT model. Once the input embeddings are created, they are passed through the various layers of the BERT model to extract the relevant features for the given NLP task. The output of the model is then used to make predictions or perform other downstream tasks, such as generating a summary, answering a question, or classifying a text into a category. During Fine tuning of the BERT the output of the tokenizer remains the same, only the parameters of the model are changed to align for the specific task.

2.2.6 SBERT

Sentence Bidirectional Encoder Representations from Transformers (SBERT) is a variant of BERT that is specifically designed to generate high-quality sentence embeddings. SBERT uses a siamese network architecture, in which two identical copies of the BERT network are used to process two input sentences. The two-sentence embeddings are then compared using a similarity measure such as cosine similarity or Euclidean distance. The siamese architecture allows SBERT to learn high-quality sentence representations that capture the semantic meaning of the sentence. The sentence embeddings generated by SBERT have several advantages over the embeddings generated by BERT. For example, they can be used in various NLP tasks such as semantic search, text classification, and information retrieval. In addition, they have been shown to outperform traditional approaches like averaging word embeddings or using pre-trained word embeddings like GloVe in sentence similarity tasks. Before sentence transformers, the approach to calculating accurate sentence similarity with BERT was to use a cross-encoder structure. This meant that we would pass two sentences to BERT, add a classification head to the top of BERT - and use this to output a similarity score. The cross-encoder network does produce very accurate similarity scores (better than SBERT), but it's not scalable i.e. as the collection of sentences increases searching through them requires huge time. Unlike BERT, SBERT is fine-tuned on sentence pairs using a siamese architecture. The token embeddings are pooled to generate sentence embeddings.

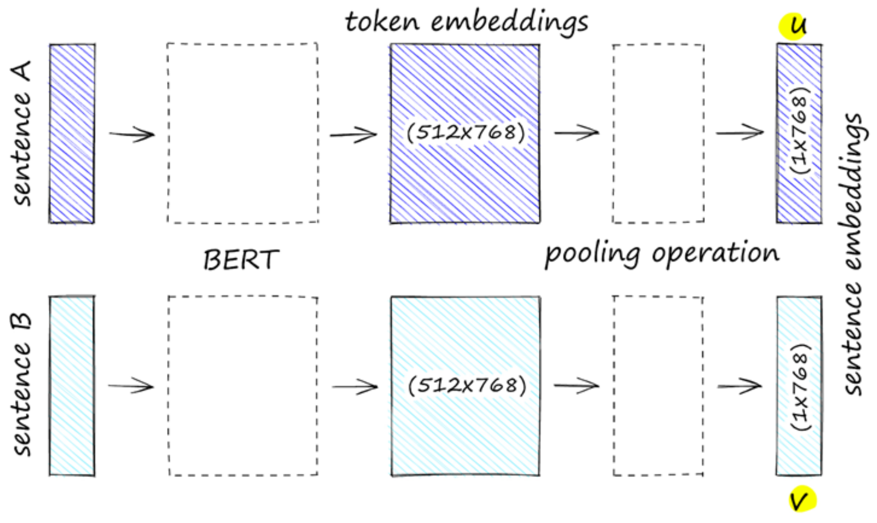


Figure 3: (From pinecone) An SBERT model architecture

In the above figure, the SBERT model is applied to sentence pair A and B. The BERT model outputs token embeddings(consisting of 512 * 768-dimensional vectors). We then compress that data into a single 768-dimensional sentence vector using a pooling function.

2.2.7 Spearman Correlation

Spearman correlation is a statistical measure of the degree and direction of the relationship between two variables. It is a non-parametric measure that evaluates how effectively a monotonic function can explain the relationship between two variables. Basically, it determines how well a straight line with an upward or downward slope can depict the relationship between the two variables.

$$r_s = 1 - \frac{6\sum_{i=1}^{n} D^2}{n(n^2-1)} \tag{1}$$

The Spearman's correlation coefficients range between -1 and +1. The coefficient's sign tells us whether the relationship is positive or negative monotonic relationship. A positive correlation indicates that the two variables tend to rise together as one rises. When two variables are negatively correlated, it means that if one variable rises, the other tends to fall. The values close to -1 or +1 represent stronger relationships than values closer to zero.

2.2.8 Vector Database

A vector database is a type of database that stores and retrieves vector-based data. A vector is a mathematical representation of a quantity that has both magnitude and direction, such as velocity or acceleration. In the context of databases, vectors are commonly used to represent high-dimensional data, such as images, audio, and text.

In a vector database, each data point is represented as a vector, with each dimension of the vector corresponding to a specific feature or attribute of the data point. For example, in an image database, each image may be represented as a vector of pixel values, with each pixel being a dimension of the vector.

One of the main advantages of a vector database is that it allows for efficient and flexible retrieval of similar data points. This is accomplished through the use of similarity search algorithms, which can quickly identify data points that are similar to a given query point.

There are several different types of similarity search algorithms that can be used with vector databases, including

Euclidean distance

This measures the straight-line distance between two vectors in Euclidean space. It is commonly used in image and audio databases.

Cosine similarity

This measures the cosine of the angle between two vectors, which is a measure of their similarity. It is commonly used in text databases.

Jaccard similarity

This measures the similarity between two sets of data points by comparing their intersection and union. It is commonly used in graph databases.

Hamming distance

Hamming distance measures binary data strings. The distance between two strings of equal length is the number of bit positions at which the bits are different.

In addition to similarity search, vector databases can also be used for other tasks such as clustering, classification, and regression. For example, clustering algorithms can group similar data points together, while classification algorithms can assign new data points to predefined categories.

Overall, vector databases are a powerful tool for storing and retrieving high-dimensional data. They can be used in a wide range of applications, from image and audio processing to natural language processing and graph analysis.

2.2.9 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors. This concept is frequently applied in information retrieval and natural language processing. It calculates the cosine of the angle formed by two vectors in a multidimensional space. Cosine similarity yields a value between -1 and 1, where -1 denotes entire dissimilarity, 0 denotes no similarity, and 1 denotes total similarity.

To calculate the cosine similarity, we initially normalize the vectors to unit length

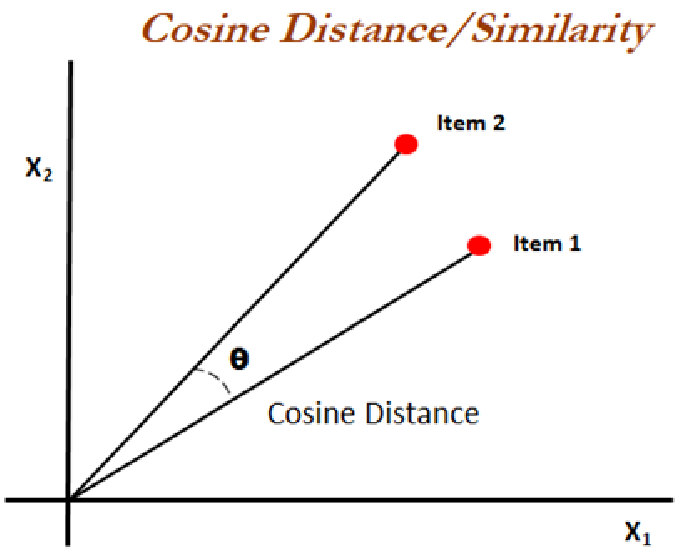


Figure 4: Cosine Similarity[4]

by dividing the vectors by their magnitude. Then, using the sum of the products of the respective components of the two vectors, we compute the dot product of the two normalized vectors. The dot product is finally divided by the product of the magnitudes of the two vectors. This yields a value between -1 and 1, which represents how similar the two vectors are to one another.

$$similarity(A,B) = \frac{A.B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

2.2.10 Inverted File Index (IVF)

Information retrieval systems employ the inverted file index as a data structure to effectively retrieve data from huge document collections. A unique phrase in a group of documents is mapped to the set of papers that include it via this index structure.

An inverted file index's main concept is to compile a dictionary of terms that are used in the documents and then produce a list of all the documents that use each keyword. In this approach, the system may swiftly seek up a list of papers that include a certain phrase when a user queries it and deliver the appropriate results.

Consider a group of papers that contain the words "apple", "orange," and "banana," for instance.

A dictionary with three entries for each phrase and a list of documents that use that term would be produced using the inverted file index. For instance:

Apple: Documents 1, 3, and 5.
Orange: Docs. 2, 4, and 5.
Banana: Docs. 1, 2, and 4.

The algorithm would simply consult the definition of "apple" in the dictionary to do a search for that term, returning a list of publications that do (in this case, Doc 1, Doc 3, and Doc 5).

Search engines and other information retrieval systems frequently employ inverted file indexes to effectively extract pertinent data from big document collections. It effectively links search phrases to papers that include those terms, enabling quick searches.

2.2.11 MNR Loss

Multiple Negative Ranking (MNR) is a great loss function for datasets having positive pairs, for example, only pairs of similar texts like pairs of paraphrases, pairs of duplicate questions, pairs of (query, response), or pairs of (source_language, target_language).

This loss expects as input a batch consisting of sentence pairs (a_1, p_1), (a_2, p_2)..., (a_n, p_n) where we assume that (a_i, p_i) are a positive pair and (a_i, p_j) for i!=j a negative pair.

For each a_i, it uses all other p_j as negative samples, i.e., for a_i, we have 1 positive example (p_i) and n-1 negative examples (p_j). It then minimizes the negative log-likelihood for softmax normalized scores.

This loss function works great to train embeddings for retrieval setups where you have positive pairs (e.g. (query, relevant_doc)) as it will sample in each batch of n-1 negative docs randomly.

The MNR loss is given by:

$$\begin{aligned} L(x, y, \theta) & \\ &= -\frac{1}{K}\sum_{i=1}^{K} log P_{approx}(y_i|x_i) \\ &= -\frac{1}{K}\sum_{i=1}^{K}\left[S(x_i, y_i) - log\sum_{j=1}^{K} e^{S(x_i, y_j)}\right] \end{aligned} \tag{3}$$

where
K = no of sentences or batch size
x_i and y_j are the duplicate sentences for i = j and negative for i != j
θ represents word embeddings
S(x,y) is the similarity score

3 Methodology

3.1 Requirement Analysis

3.1.1 Functional Requirements

From the perspective of functional requirements, the system must provide users with the ability to enquire about various questions related to the domain of engineering studies. The system must enable users to view the pattern of question repetition in past papers in order to help users improve the mix of questions in papers. The users should be able to enter the question they want to analyse and the system should show them all the similar questions and their appearance year along with the similarity index indicating the level of repetition.

In addition to this, the system should also allow users to upload a file containing the questions of a subject and receive the analysis related to the questions in the file. The analysis contains the question-wise similarity as with a single question, as well as the overall similarity of uploaded files with historical papers.

3.1.2 Non-functional requirements

Non-functional requirements are essential characteristics of a system that is not directly related to its functionality but to other aspects such as usability, performance, maintainability, scalability, security, and reliability. These requirements focus on how the system should behave, and how it should interact with its users. So, with that in mind, the following non-functional requirements are considered for the project:

1. The system shall be optimized for minimum response time regarding similarity analysis.
2. The user interface has been kept as simple and minimalistic as possible, allowing for a seamless and intuitive user experience.
3. The system has been designed as a web application for the system to be platform-independent.
4. All exceptions arising from incorrect user input shall be handled and all output shall be provided with minimal data loss.

3.1.3 Other Requirements

This system development demands several software components for programming and hardware accessories for training and processing.

Languages, Libraries, and Frameworks

- Python
- Javascript/Typescript
- Reactjs - Frontend Web
- Django - Backend Web
- Zilliz - Cloud Database
- Numpy and Pandas

Hardware Requirements
The training of large datasets requires a large amount of computing and memory resources. For this use case, we have used the ***g5.xlarge*** compute resource of AWS through AWS Sagemaker service which consisted of NVIDIA A10G Tensor Core GPUs with the GPU memory being 24GB. Graphical Processing Unit (GPU) is a processing unit specially designed to enhance graphics rendering and deep learning inferencing. It took about half an hour for our model to train with a single epoch and we have trained 6 such models.

3.2 System Architecture

The system architecture of the project was developed using a layered architecture approach, with separate layers for presentation, business logic, and data access. The layered architecture was chosen to improve the maintainability and flexibility of the system, as it allows for changes to be made to one layer without affecting the others.

The presentation layer consists of the user interface components, including web pages, forms, and controls. This layer was implemented using a Model-View-Controller (MVC) design pattern, which separates the user interface logic from the business logic. The MVC pattern also allows for the use of reusable components and simplifies testing.

The business logic layer contains the application logic, including the rules and procedures that govern the system's behaviour. This layer was implemented using a monolith approach, which makes use of a single executable file. The monolith approach provides us with the ease of using components inside a single application without having to manage different services and communication between the services.

To support the development of the system architecture, a number of tools and frameworks were used, including React, Django, Sentence Transformers, and the Milvus database hosted on the Zilliz cloud. The architecture was developed through an iterative approach, with regular feedback from mentors and supervisors.

To develop the system architecture for the project, we followed a structured approach that included the following methods.

3.2.1 Development Methodology

We used the incremental development approach to achieve the project. The Incremental Software Development Life Cycle (SDLC) model is a methodology that breaks down the software development process into smaller, more manageable pieces or increments. Each increment is developed and tested separately before being integrated with the previous increments to form the final product.

The project was broken down into its own modular parts based on the requirement analysis which could be separately implemented on its own. This approach is included in the project in the following ways.

Iteration 1
: The most critical feature of our application included using a sentence transformer model for a generation of vector embeddings. The pre-trained models from hugging face were used in this phase for generating the embeddings and basic testing of similarity using cosine similarity. This helped to further make plans for data collection and choose the best model architecture.

Iteration 2
: In the second iteration, the searching mechanism of questions based on embedding generated from the model was developed. The search used inverted indexing generated by FAISS and the search modules provided by FAISS for performing a search on the questions included in the runtime environment.

Iteration 3
: After gaining information about the working of sentence transformers, we used a pre-trained BERT-based model i.e. DistilBERT-base-uncased and trained it on AllNLI, MRPC, PAWS and STS-B datasets to form sentence transformers. This iteration still used FAISS for searching on the runtime environment only. During this phase, the front end for query insertion and API call requests was developed.

Iteration 4
: In the fourth iteration, the sentence transformer model trained during the

previous iteration was fined tuned on our primary dataset and embeddings were generated. Milvus database hosted on Zilliz Cloud was used to store the vector embeddings. FAISS was also replaced by Milvus's Knowhere vector engine which integrates FAISS, HNSW and Annoy for performing the vector search. This provided the team with a robust, and scalable database and search solutions independent of the Python runtime.

Iteration 5

In this iteration, the frontend along with the backend and database was fully integrated to perform semantic query searching using only the frontend provided to the user. Searching based on the subject filter was also introduced in this iteration.

Iteration 6

Here, we introduced the functionality to add the questions from the UI and also generated the report calculating the similarity score between the documents.

By using an incremental approach, we were able to break down the project into smaller, more manageable chunks This approach allowed us to focus on developing the most critical features first and then on that foundation to add more functionality. Throughout the process, we were able to gather feedback and make adjustments to the model as well as the website based on the need and preferences. In the end, the result was a functional product that met the needs of our requirements.

3.2.2 Software Development Tools and Frameworks

Software development tools and frameworks are an important part of the system architecture for any software development project. They provide a foundation for building, testing, and deploying software, as well as tools for managing the development process itself. Here are the software development tools and frameworks that we used in this project:

Integrated Development Environment (IDE)

VSCode was mostly used for performing the development of the frontend and backend systems. Occasionally, Jupyter Notebook provided by AWS Sagemaker was used for training the model.

Version Control System (VCS)

Git and GitHub were used to collaborate on code changes and track changes to the code over time. For the versioning of the sentence transformer model, Huggingface Hub was used which also provided an inference API for testing purposes.

Build and Automation Tools
: For building the frontend application, GitHub Actions was used in conjunction with Vite. The builds were triggered on every pull request and merge. Similarly, automatic dependency check tools such as dependabot provided by GitHub were also integrated.

Front-end and Back-end Frameworks
: Front-end and back-end frameworks provide a foundation for building the user interface (front-end) and the server-side logic (back-end) of a software application. React and Django were used in the front end and back end respectively.

By including these software development tools and frameworks in the system architecture, we streamlined the development process, improved collaboration, and ensured that the software functions as intended.

3.2.3 Project Management

It is an essential component of system architecture, as it involves planning, organizing, and coordinating resources and activities to ensure that a project is completed on time, within budget, and with the desired outcome.

Our project includes the following steps of project management:

Planning and goal-setting
: We approached this step using the top-down approach by short-listing all the tools and resources needed to accomplish the project. We divided the project into various milestones and sprints based on the listings which were achieved with every iteration. During this phase, architectural design work was also performed that provided a general overview of the working of the overall system and how each part communicates with the other.

Resource allocation
: During the first two iterations, all the time and resources were dedicated to developing an MVP that illustrated the whole application on a very small scale. During this phase, frequent coordination was made among the team members to ensure the resources are available when needed.

Communication
: Daily standups and regular meetups were organized between team members

and the supervisor to ensure that everyone has access to the information they need to complete their tasks effectively. Regular progress reports were provided to the supervisor and all the documentation regarding the project materials was maintained in a central repository.

Monitoring and control

The monitoring of the project was done using project management tools like Jira and plans was also adjusted as needed to ensure that the project stays on track. Various metrics were used against goals, and objectives to make adjustments and improve the project outcomes.

3.2.4 Performance Optimization

Software development must include performance optimization since it makes sure the application is quick to respond, effective, and able to manage a lot of users and data. The methods utilised to improve the software's performance are described in this portion of the study, including profiling, caching, network optimization, and hardware optimization. By improving the performance of the software, the application achieve its performance goals and provide users with a better user experience by performing better, which will improve adoption and satisfaction.

The following are the measures that we implemented in the performance optimization section of our project:

Performance goals

The performance of the system must be on par with that of a real-time system. The response time of the system is around 3-4 seconds after performing optimization. The database utilization metrics include CPU usage of 6%, storage usage of 0.06 GB, QPS of 0.07 on average as per our usage and query latency of 500ms on average.

Performance Profiling

The profiling technique used for measuring network latency is a waterfall diagram in Chrome Developer Tools. The major performance bottleneck is idle server latency of nearly 3.5s. This is due to the deployment of the database in the ***us-west (Oregon)*** and also the need to generate embeddings by the transformer model per query request. Similarly, the performance of the UI was measured using Lighthouse and PageSpeed Insights.

Optimization Strategies

The optimization on the backend was done using a singleton pattern to load the sentence transformer only once and use the same instantiated model

during all the query requests. The model is thus stored in the memory during runtime.
The system also used connection pooling for making requests to the database so that the request doesn't have to make connections with the database for each and every request. Similarly, the optimization on the front end was done using a compressed build version and using gzip compression on the browser.

Network Optimization
: The front end was deployed on the CDN provided by Vercel to serve static assets. Also, the data in transit is provided with minimal response so that it can travel across the network in minimum time.

By implementing these optimization strategies in the front end, back end and database of the software application, we drastically reduced the response time and increased the performance of the system.

3.2.5 Deployment and maintenance

Deployment and maintenance of software are critical phases in the software development lifecycle. Deployment involves moving the software from the development environment to a production environment, while maintenance involves managing the software after it has been deployed to ensure that it continues to function properly and meet the needs of users. Here is an overview of the deployment and maintenance of software:

Deployment
: The deployment of the sentence transformer model is done in the HuggingFace hub, database in Zilliz Cloud, and front end in Vercel.

Maintenance
: The maintenance of the code is done using code analysis and dependency alert tools provided by GitHub Action.

3.2.6 System workflow

The user is presented with an interface where they can enter questions. The user then enters the query (question) for which they want to calculate the similarity score and see the similar questions.

The front end then sends the API request containing the question to the backend service. The backend service loads the custom-trained sentence transformer model

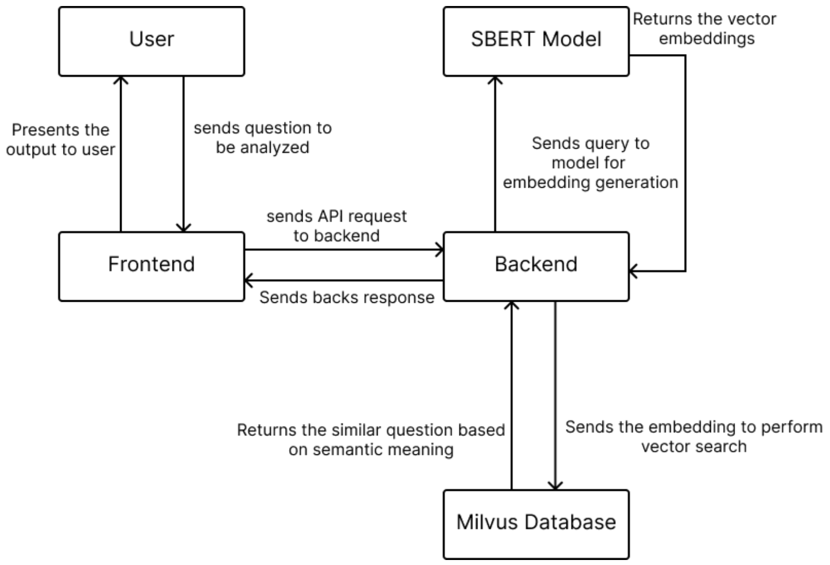

User
SBERT Model
Returns the vector embeddings
Presents the output to user
sends question to be analyzed
Sends query to model for embedding generation
sends API request to backend
Frontend
Backend
Sends backs response
Returns the similar question based on semantic meaning
Sends the embedding to perform vector search
Milvus Database

Figure 5: System Workflow Diagram

which generates the embeddings of the query. The vector embeddings are sent over the network to the Milvus Database hosted on Zilliz Cloud. The database receives the embeddings which are in turn passed to the Knowhere vector engine which is included in the Milvus Database. The Knowhere then returns a set of candidate vectors that are similar to the query, and Milvus then retrieves the corresponding vectors from its vector database.

The response however received in this stage does not contain the questions from the database, rather it only contains the identifier and distance metrics. Again, the backend makes the query request to the Milvus database for the questions and corresponding embeddings based on the id returned in the previous response. The backend then calculates the cosine similarity of the received embeddings against the embeddings generated from the query questions and prepares the corresponding response to be consumed by the frontend application.

The frontend application then receives the JSON response from the backend which is then presented in an easy-to-understand format.

3.2.7 Use Case

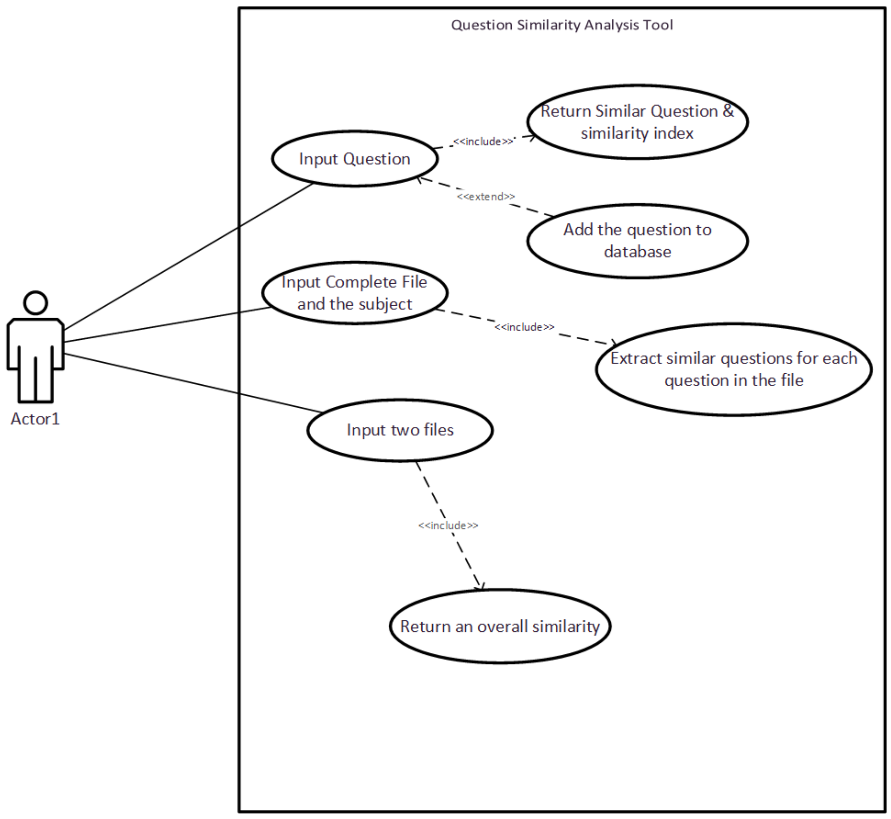

Question Similarity Analysis Tool
Input Question
<<include>>
Return Similar Question & similarity index
<<extend>>
Add the question to database
Input Complete File and the subject
<<include>>
Extract similar questions for each question in the file
Input two files
<<include>>
Return an overall similarity
Actor1

Figure 6: Use Case Diagram

3.3 Implementation Details

3.3.1 Data Collection

The data for this project was collected from a variety of sources. We used publicly available datasets hosted on Hugging Face such as AllNLI which provide a large and diverse dataset for training and evaluating Natural Language Inference (NLI) models. AllNLI is a collection of several existing NLI datasets, including the Multi-Genre Natural Language Inference (MNLI), the Stanford Natural Language Inference (SNLI), and the Corpus of Linguistic Acceptability (CoLA).

MRPC dataset which specializes in training and evaluating natural language processing models for paraphrase identification is also used which helps in determining whether two sentences have the same meaning.

PAWS dataset which also specializes in paraphrase identification is also used. However, this dataset is unique in that it focuses on a specific type of paraphrase that is generated by word scrambling.

STS dataset which specializes in the semantic similarity between two sentences is

3.2.8 Sequence Diagram

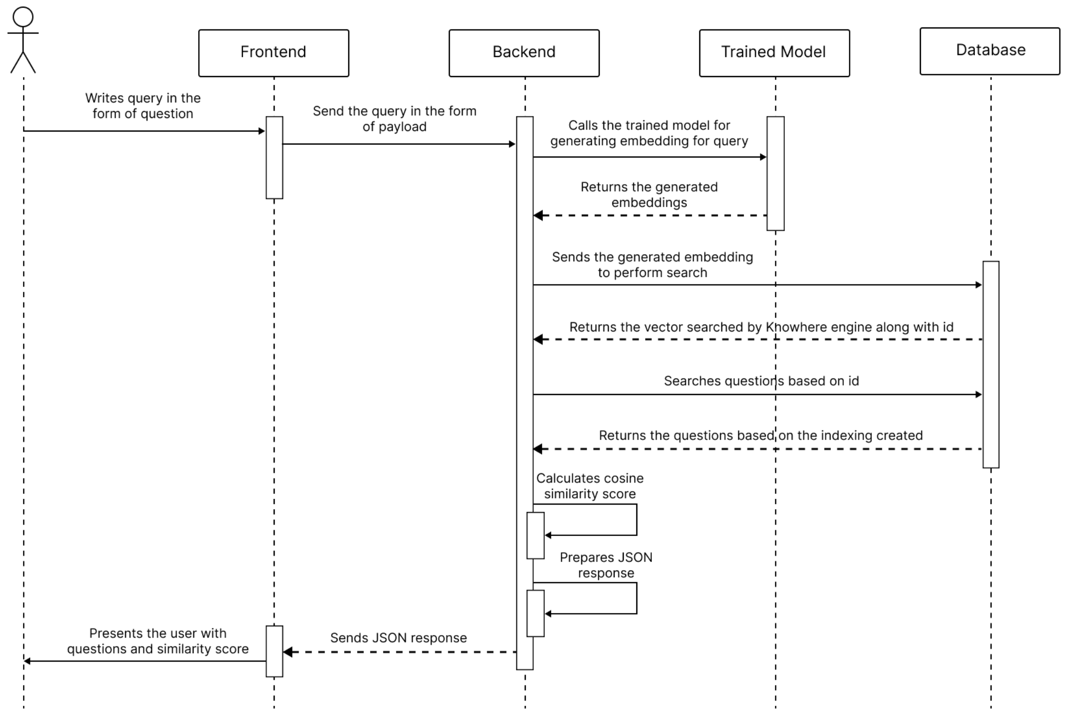

Frontend
Backend
Trained Model
Database
Writes query in the form of question
Send the query in the form of payload
Calls the trained model for generating embedding for query
Returns the generated embeddings
Sends the generated embedding to perform search
Returns the vector searched by Knowhere engine along with id
Searches questions based on id
Returns the questions based on the indexing created
Calculates cosine similarity score
Prepares JSON response
Presents the user with questions and similarity score
Sends JSON response

Figure 7: Sequence Diagram

used for training as well as evaluation of the model. The main use of the STS dataset in model training is to evaluate and improve the performance of NLP models in measuring the similarity between two pieces of text.

The primary dataset for our project is collected with the help of Google Docs and Google Drive API, collecting all the questions from the digital version of past question papers used at Pokhara University.

3.3.2 Data Cleaning

However, the data retrieved using the API was very raw and contained irrelevant information such as marks, heading information, and noisy data that were gathered while parsing the JSON data received from API.

The data was cleaned using regex as well as organized in a suitable pattern manually. Still, the data contained a large number of mathematical questions which were not wanted in the training dataset. We manually removed those mathematical questions. Also, there were multiple instances where a single question was distributed in multiple rows. Those questions were also manually corrected. We also had to manually add the subject and year in which the question was asked in order to provide more information. After these steps, we had a proper dataset on our hands.

3.3.3 Data Preparation

The primary dataset collected was not suitable for training our sentence transformer as transformer had to be trained under supervised conditions. So, the suitable dataset was prepared by paraphrasing the collected questions using the ***text-davinci-003*** language model from OpenAI.

Question 1	Paraphrased Question 1
Paraphrased Question 1	Question 1

Table 1: Example of the Dataset used for training

Let us suppose a question "Question 1" and its paraphrased version "Paraphrased Question 1". Then we created the dataset for training as shown in the figure below. By this method, our 9000+ rows dataset was doubled for training purposes.

3.3.4 Modeling

We developed several models for domain adaptation using supervised as well as unsupervised learning approaches. We first used GPL for unsupervised learning due to a lack of labelled training data however due to larger training time and complexity and low correlation i.e. 0.68 we switched to the MNR Loss method for supervised learning.

3.3.5 Software Development

Model Development
The implementation of the models was done by using the Sentence Transformer library provided by UKP Lab. This library includes modules for performing the evaluation, calculating loss function measures, cosine similarity values as well as different correlations metrics.

The first step included using a pre-trained BERT model, ***distilbert-base-uncased*** which does not take casing into account and converting it into SBERT by adding a pooling layer at the output and then training it on the AllNLI dataset. The MNR Loss function was used for optimizing the model as it required only positive pairs in contrast to what is required when using other types of loss functions.

Further training was done in MRPC, PAWS, and STS training datasets to tune the model for paraphrase detection as well as similarity scoring. These datasets were specifically chosen for training the model as they specialized in the domain of paraphrasing as well as similarity detection.

The fine-tuning of the model in the engineering domain was performed by training the model on the custom-prepared primary dataset.

The training of all these models on different datasets was performed using AWS Sagemaker with a training batch size of 128, a maximum sequence length of 512, word embedding dimensions of 768, and a mean pooling method. The training was performed only with a single epoch as suggested by the paper on SBERT. The model architecture used looks as shown in Figure 8.

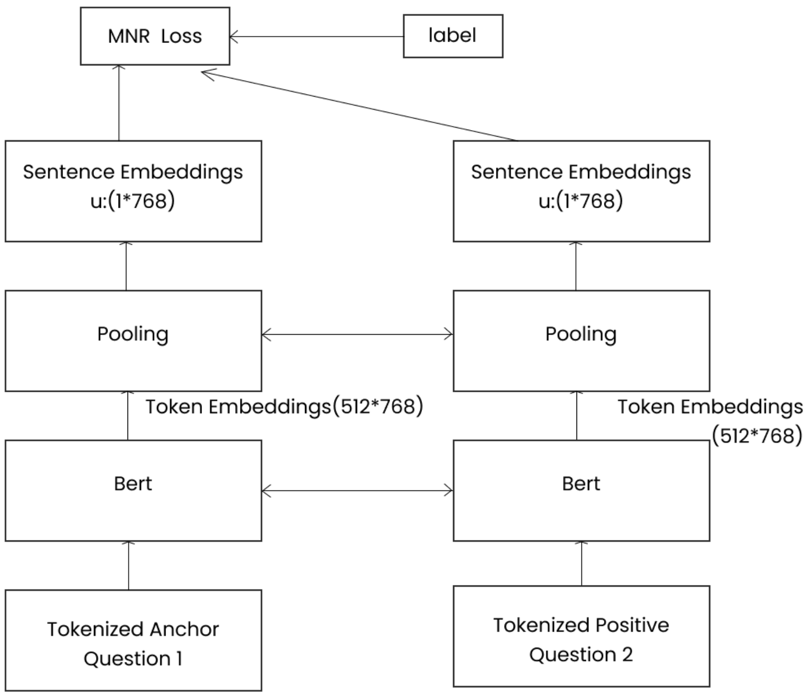

MNR Loss
label
Sentence Embeddings u:(1*768)
Sentence Embeddings u:(1*768)
Pooling
Pooling
Token Embeddings(512*768)
Token Embeddings (512*768)
Bert
Bert
Tokenized Anchor Question 1
Tokenized Positive Question 2

Figure 8: System Architecture Diagram

Frontend Development

The implementation of the front end included using React JS library for implementing the UI and all the core logic of the front end. The React library was used as the project required lots of data communication between various components as well as reactivity without needing to reload the page. It also defines the concept of reusable components which makes the development process quick.

The designs and prototypes of the UI were made in Figma. The environment of the frontend development setup was performed using Node.js, yarn and Vite with all the necessary dependencies. This included the proper setup of credentials for using the backend services and all the environment variables that were used in the deployment of the front end. Vite is used as it supports faster development, and faster production build times as it used ESBuild instead of Webpack for bundling, has high configurability and uses fewer resources.

The axios library was used for making all the HTTP requests to our backend service. It allows for working with various types of data that needs to be sent over the network. It also provides a consistent API for making HTTP requests across different platforms and browsers.

Backend Development
The implementation of the backend included using Django which is a Python web framework for creating the backend. The main purpose of using Django was to make ease in using the sentence transformer library used for generating the embeddings to perform the comparison. The Django library was used in building the backend server and creating APIs for communicating with our model, database and frontend side. The library used for interfacing with the database was also easily integrable with Django. It also provides for future enhancements regarding the segregation of every service in case of production deployment.

Database
Milvus hosted on Zilliz Cloud which in turn deploys the resources in the AWS platform is used as a database for this project. Milvus is an open-source vector database designed for handling large-scale machine-learning workloads. It is optimised for storing and searching high-dimensional vector data, such as embeddings generated by deep learning models. The main use of Milvus is to provide a highly scalable and efficient platform for building intelligent applications that require fast and accurate similarity searches.

This project uses the Milvus database for storing sentence embedding generated by the fined-tuned sentence transformer model and also performing an accurate similarity search. Implementation details of performing data insertion and similarity search are handled by the ***pymilvus*** library. The search process used Knowhere as the underlying vector similarity search engine. Knowhere provides optimised indexing algorithms that are used to index and search large-scale vector databases. When a query is submitted to Milvus, it is processed by Knowhere using the specified indexing algorithm. Knowhere then returns a set of candidate vectors that are similar to the query, and Milvus then retrieves the corresponding vectors from its vector database.

4 Results and Analysis

4.1 Training Results

The first phase of our training included training ***distilbert-base-uncased*** on AllNLI dataset to convert it into SBERT for 1 epoch.

steps	cosine pearson	cosine spearman	euclidean pearson	euclidean spearman
220	0.8364516097	0.83932855	0.8163054757	0.8196672197
440	0.8451248998	0.850080908	0.8258113817	0.8294317432
660	0.8425308757	0.847485762	0.8279182604	0.8308162577
880	0.8469245836	0.8517480746	0.8293103285	0.8322019925
1100	0.8511393044	0.8549902573	0.8348620622	0.837028036
1320	0.8488461649	0.852874159	0.8321725367	0.834869006
1540	0.8511160436	0.8552112051	0.8331633293	0.8355860688
1760	0.8507647935	0.8542689411	0.8351606703	0.8371928955
1980	0.8511629525	0.8542907622	0.8349342985	0.8369407245
2200	0.8516398052	0.8550346605	0.8352028955	0.8372999223
-1	0.8516401066	0.8550360047	0.8352029808	0.8372999223

steps	manhattan pearson	manhattan spearman	dot pearson	dot spearman
220	0.815920996	0.8190446133	0.7501043149	0.7435983726
440	0.8258887841	0.8297153804	0.7718244485	0.7672620021
660	0.827903569	0.8307285761	0.7780585242	0.7745490036
880	0.8294952675	0.8325163247	0.7915537884	0.7882690753
1100	0.8348559833	0.8369807833	0.7935572455	0.7897020283
1320	0.8322332832	0.8349613708	0.7914952347	0.7875094787
1540	0.8331363587	0.8356607575	0.7951959209	0.7910028245
1760	0.835193458	0.8373388981	0.7984664532	0.7942523979
1980	0.8349066513	0.8370105652	0.8000965923	0.7953404225
2200	0.8351443516	0.837306086	0.7997876431	0.7951892189
-1	0.8351444426	0.8373055308	0.799788498	0.7951890427

Table 2: Evaluation metrics while training on AllNLI

Further training of the model on MRPC for 1 epoch resulted in different metrics as shown below

steps	cosine pearson	cosine spearman	euclidean pearson	euclidean spearman
440	0.8343577256	0.839353702	0.8164808715	0.8205871499
440	0.8363515928	0.8430247081	0.8102966348	0.8160169891
880	0.8425916223	0.8489752834	0.8199459757	0.8263628532
1320	0.8432426234	0.8503495868	0.8257562947	0.8313146627
1760	0.8470568724	0.8517104614	0.8276564905	0.8306492423
2200	0.8494367121	0.8543035397	0.8340444222	0.8363041608
2640	0.8472370418	0.8528592416	0.8303407719	0.8336411518
3080	0.8464874389	0.8526450208	0.8291406845	0.8321360529
3520	0.847422422	0.8526455831	0.8289576243	0.8323210691
3960	0.8483662702	0.8535172065	0.8292084499	0.8326882277
4400	0.8485433281	0.8535884876	0.8294968247	0.8328049555
-1	0.8485450498	0.8535941834	0.8294988829	0.8328111601

steps	manhattan pearson	manhattan spearman	dot pearson	dot spearman
440	0.816497681	0.8205425591	0.7419814576	0.7363160508
440	0.8101810775	0.8158083584	0.7330022373	0.7313526577
880	0.8203358746	0.8267426382	0.7630823272	0.7611710182
1320	0.8258374865	0.831420451	0.7771250706	0.773988027
1760	0.8276305395	0.8306525384	0.7847924916	0.7824372126
2200	0.8340054982	0.8365270204	0.8003641672	0.7964372614
2640	0.8302999444	0.8337021376	0.7921299057	0.7889456761
3080	0.8290396245	0.8320951721	0.7922173564	0.7898946145
3520	0.8287994804	0.8324727773	0.7935284262	0.7910307452
3960	0.8289307603	0.8322407914	0.7932115619	0.7904691807
4400	0.8292640683	0.8325136297	0.7953522708	0.7924759045
-1	0.8292661261	0.8325165888	0.7953531276	0.792478449

Table 3: Evaluation metrics while training on MRPC

Further training of the model on PAWS for 1 epoch resulted in higher correlation scores.

steps	cosine pearson	cosine spearman	euclidean pearson	euclidean spearman
34	0.8397202405	0.8040743712	0.8134474323	0.7990306822
68	0.8397202405	0.8040743712	0.8134474323	0.7990306822
102	0.8397202405	0.8040743712	0.8134474323	0.7990306822
136	0.8397202405	0.8040743712	0.8134474323	0.7990306822
170	0.8397202405	0.8040743712	0.8134474323	0.7990306822
204	0.8397202405	0.8040743712	0.8134474323	0.7990306822
238	0.8397202405	0.8040743712	0.8134474323	0.7990306822
272	0.8397202405	0.8040743712	0.8134474323	0.7990306822
306	0.8397202405	0.8040743712	0.8134474323	0.7990306822
340	0.8397202405	0.8040743712	0.8134474323	0.7990306822
-1	0.8397202405	0.8040743712	0.8134474323	0.7990306822
34	0.8676114699	0.8696745486	0.8554968638	0.8565594808
68	0.8676114699	0.8696745486	0.8554968638	0.8565594808
102	0.8676114699	0.8696745486	0.8554968638	0.8565594808
136	0.8676114699	0.8696745486	0.8554968638	0.8565594808
170	0.8676114699	0.8696745486	0.8554968638	0.8565594808
204	0.8676114699	0.8696745486	0.8554968638	0.8565594808
238	0.8676114699	0.8696745486	0.8554968638	0.8565594808
272	0.8676114699	0.8696745486	0.8554968638	0.8565594808
306	0.8676114699	0.8696745486	0.8554968638	0.8565594808
340	0.8676114699	0.8696745486	0.8554968638	0.8565594808
-1	0.8676114699	0.8696745486	0.8554968638	0.8565594808

steps	manhattan pearson	manhattan spearman	dot pearson	dot spearman
34	0.8134910179	0.7985155715	0.814138864	0.7711614911
68	0.8134910179	0.7985155715	0.814138864	0.7711614911
102	0.8134910179	0.7985155715	0.814138864	0.7711614911
136	0.8134910179	0.7985155715	0.814138864	0.7711614911
170	0.8134910179	0.7985155715	0.814138864	0.7711614911
204	0.8134910179	0.7985155715	0.814138864	0.7711614911
238	0.8134910179	0.7985155715	0.814138864	0.7711614911
272	0.8134910179	0.7985155715	0.814138864	0.7711614911
306	0.8134910179	0.7985155715	0.814138864	0.7711614911
340	0.8134910179	0.7985155715	0.814138864	0.7711614911
-1	0.8134910179	0.7985155715	0.814138864	0.7711614911
34	0.8550622291	0.8563361568	0.8245345726	0.8216147965
68	0.8550622291	0.8563361568	0.8245345726	0.8216147965
102	0.8550622291	0.8563361568	0.8245345726	0.8216147965
136	0.8550622291	0.8563361568	0.8245345726	0.8216147965
170	0.8550622291	0.8563361568	0.8245345726	0.8216147965
204	0.8550622291	0.8563361568	0.8245345726	0.8216147965
238	0.8550622291	0.8563361568	0.8245345726	0.8216147965
272	0.8550622291	0.8563361568	0.8245345726	0.8216147965
306	0.8550622291	0.8563361568	0.8245345726	0.8216147965
340	0.8550622291	0.8563361568	0.8245345726	0.8216147965
-1	0.8550622291	0.8563361568	0.8245345726	0.8216147965

Table 4: Evaluation metrics while training on PAWS

The model was again further trained on the STS-B dataset which specialized in semantic similarity and the results can be seen below:

steps	cosine pearson	cosine spearman	euclidean pearson	euclidean spearman
-1	0.8839867553	0.8834848135	0.8630618932	0.86820227
-1	0.8841050468	0.8827960889	0.8631319044	0.8669466003
-1	0.8853258557	0.8838996234	0.8645618255	0.8683514241
-1	0.8852344061	0.8837954746	0.8652939742	0.8691081008

steps	manhattan pearson	manhattan spearman	dot pearson	dot spearman
-1	0.8627218612	0.8676591542	0.8444908252	0.8451809331
-1	0.8628397082	0.8665157922	0.8473994321	0.8469431763
-1	0.864324514	0.8678471401	0.8498861724	0.8493216049
-1	0.8650384421	0.8684945177	0.8484928546	0.8475987631

Table 5: Evaluation metrics while training on STS-B

After fine-tuning the model on our dataset, and again training for 1 epoch, the results obtained while performing the test on the STS-B test set are as follow:

steps	cosine pearson	cosine spearman	euclidean pearson	euclidean spearman
-1	0.8833105324	0.8824940887	0.8676245283	0.8701836815

steps	manhattan pearson	manhattan spearman	dot pearson	dot spearman
-1	0.8670872804	0.8695489242	0.8527779223	0.8504752382

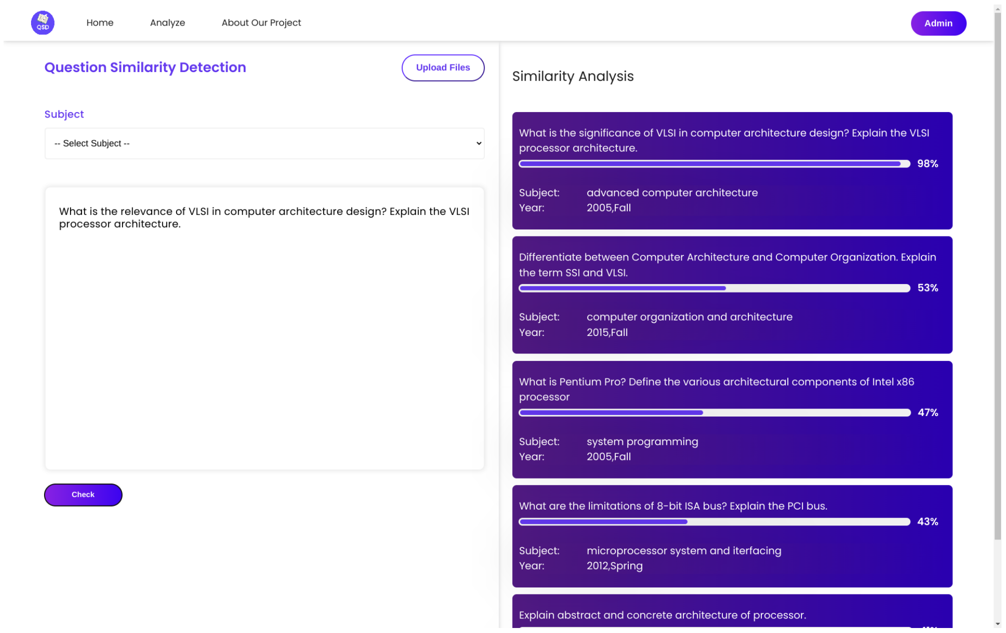
Table 6: Evaluation metrics while finetuning on our dataset

After all of the above fine-tuning approaches when the model was tested on the STS-B test dataset, the spearman correlation value was found to be 0.863 and in the test dataset of MRPC, the accuracy was found to be 88.7%.

4.2 Analysis

From the data obtained above, we can conclude that converting the BERT into SBERT and continuous fine-tuning on datasets like ALLNLI, MRPC, PAWS, STS-B, and then the custom dataset showed an increase in performance. This is because fine-tuning on these datasets allowed the model to learn general language patterns and relationships that are relevant to a wide range of natural language processing tasks. These datasets cover a broad range of tasks, including sentence pair classification, paraphrase detection, and semantic similarity, allowing BERT to learn a wide range of linguistic features and improve its performance on these tasks. So at the last step of fine-tuning on our custom dataset, the model is able to effectively capture the feature in the semantic similarity of the duplicate questions. The correlation and distance metrics obtained at the last step are greater than the previous steps which showed that the model is able to learn the specific features in our custom dataset.

4.3 Web Application Output



Question Similarity Detection

Subject

-- Select Subject --

What is the relevance of VLSI in computer architecture design? Explain the VLSI processor architecture.

Similarity Analysis

What is the significance of VLSI in computer architecture design? Explain the VLSI processor architecture.
98%
Subject: advanced computer architecture
Year: 2005,Fall

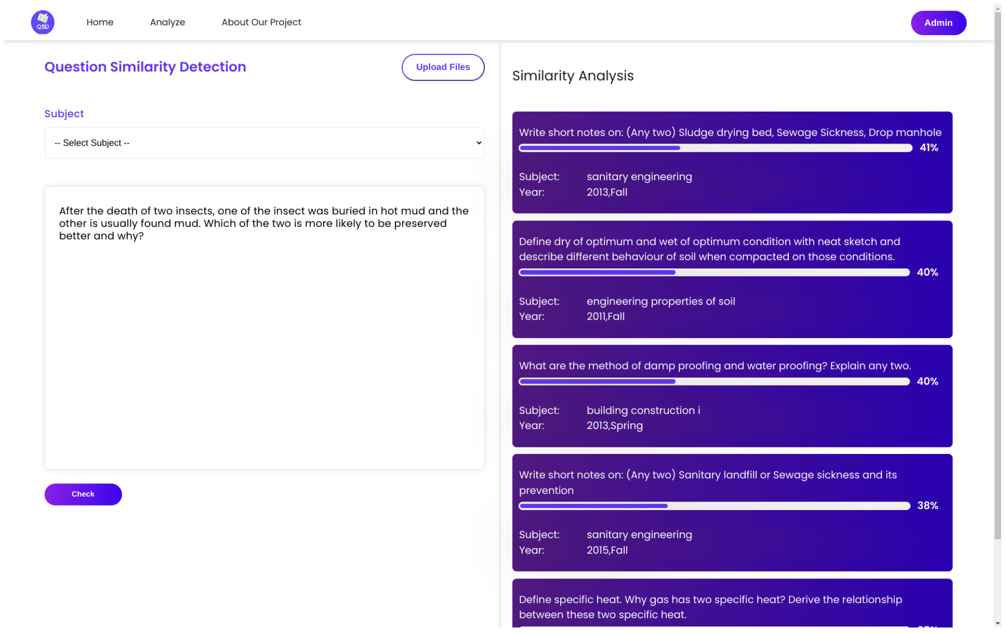
Differentiate between Computer Architecture and Computer Organization. Explain the term SSI and VLSI.
53%
Subject: computer organization and architecture
Year: 2015,Fall

What is Pentium Pro? Define the various architectural components of Intel x86 processor
47%
Subject: system programming
Year: 2005,Fall

What are the limitations of 8-bit ISA bus? Explain the PCI bus.
43%
Subject: microprocessor system and iterfacing
Year: 2012,Spring

Explain abstract and concrete architecture of processor.

Figure 9: Response having the same question



Question Similarity Detection

Subject

-- Select Subject --

After the death of two insects, one of the insect was buried in hot mud and the other is usually found mud. Which of the two is more likely to be preserved better and why?

Similarity Analysis

Write short notes on: (Any two) Sludge drying bed, Sewage Sickness, Drop manhole
41%
Subject: sanitary engineering
Year: 2013,Fall

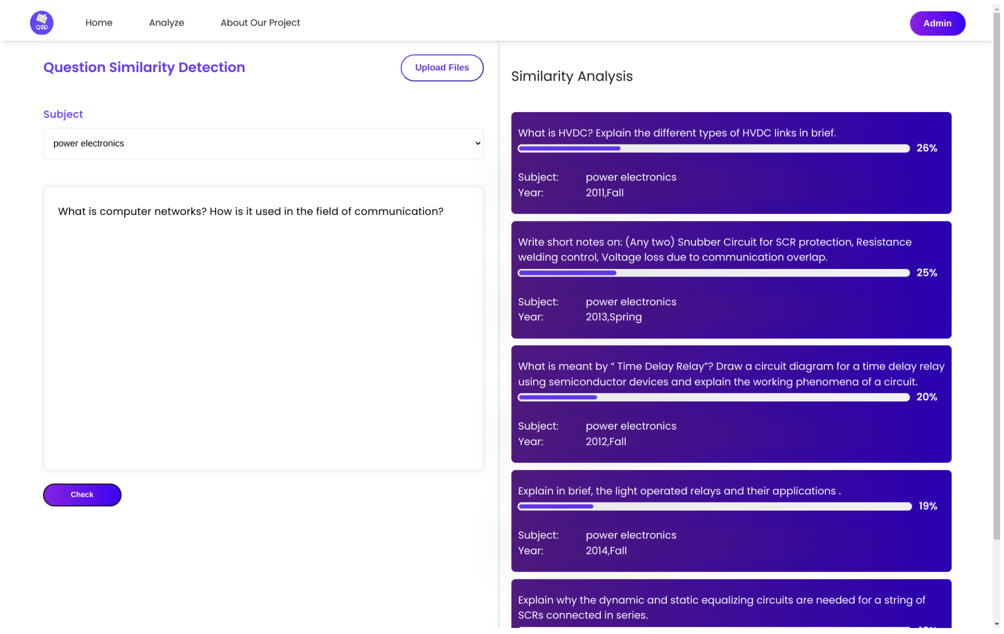
Define dry of optimum and wet of optimum condition with neat sketch and describe different behaviour of soil when compacted on those conditions.
40%
Subject: engineering properties of soil
Year: 2011,Fall

What are the method of damp proofing and water proofing? Explain any two.
40%
Subject: building construction i
Year: 2013,Spring

Write short notes on: (Any two) Sanitary landfill or Sewage sickness and its prevention
38%
Subject: sanitary engineering
Year: 2015,Fall

Define specific heat. Why gas has two specific heat? Derive the relationship between these two specific heat.

Figure 10: Response with unrelated question



Question Similarity Detection

Subject

power electronics

What is computer networks? How is it used in the field of communication?

Similarity Analysis

What is HVDC? Explain the different types of HVDC links in brief. 26%

Subject: power electronics
Year: 2011,Fall

Write short notes on: (Any two) Snubber Circuit for SCR protection, Resistance welding control, Voltage loss due to communication overlap. 25%

Subject: power electronics
Year: 2013,Spring

What is meant by " Time Delay Relay"? Draw a circuit diagram for a time delay relay using semiconductor devices and explain the working phenomena of a circuit. 20%

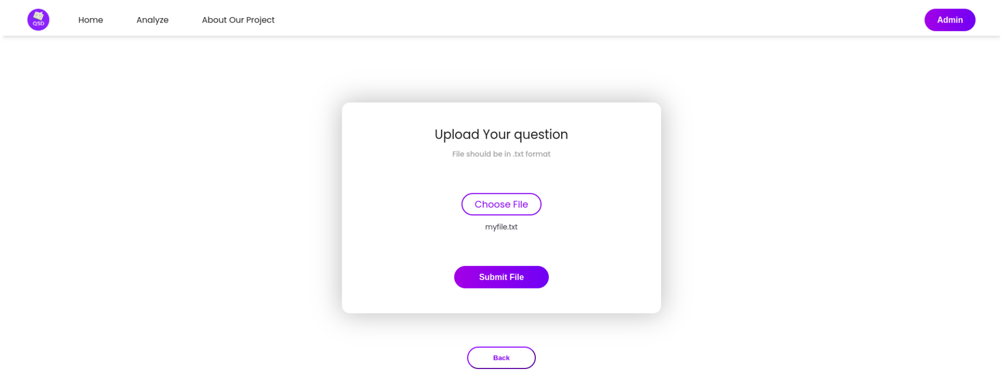
Subject: power electronics
Year: 2012,Fall

Explain in brief, the light operated relays and their applications . 19%

Subject: power electronics
Year: 2014,Fall

Explain why the dynamic and static equalizing circuits are needed for a string of SCRs connected in series.

Figure 11: Response for question with subject filtering



Upload Your question

File should be in .txt format

myfile.txt

Figure 12: File upload page

5 Conclusion

In conclusion, this project “Question Similarity Detection and Analysis” based on the SBERT language model is going to be helpful to the question setters in the university who need to be aware of past question repetition and need proper question preparation mechanism. Since SBERT has the ability to generate high-quality sentence embeddings, which can capture the meaning of a sentence in a more accurate and nuanced way than traditional bag-of-words or word-level embeddings, this project has the ability to understand the semantic meaning of the question and takes that meaning into account while performing a similarity check. We have successfully adapted our own engineering domain questions in our NLP model that helps to precisely detect the frequency of input questions as well as their similarities and is able to show the result in numbers for the analysis.

Through the implementation of the SBERT model, it was found that we were more accurately able to find the similarity between the sentence and better represent the semantics of the sentence than what was possible by just using BERT

Overall, this project Question Similarity Detection and Analysis has great potential for implementation in the question-setting task. The features added in this project are going to help the university in maintaining the authenticity and uniqueness of the question papers for the board exams.

6 Future Enhancement

After successfully developing a system to check the similarity of a question with the corpus of past questions, a few actions are desirable. These actions will allow this system to be implemented for use by the concerned authorities. The desired future enhancement tasks are as follows:

1. Currently, the database is composed of questions from Pokhara University. So, if we are to use this system in IOE, Tribhuvan University, the database needs to be populated with the questions from IOE by creating a new collection and following the same schema as before.

2. The functionality for administrative access with specific roles can be developed.

3. The system is limited to normal textual questions which can further be extended to accommodate the questions from the Mathematical domain. Further, the system can also be trained to recognize image patterns.

4. For now, the system is limited to the questions of the engineering domain. So, we could adapt the model for various other education domains and use it accordingly.

References

[1] Jafar A Alzubi, Rachna Jain, Anubhav Singh, Pritee Parwekar, and Meenu Gupta. Cobert: Covid-19 question answering system using bert. *Arabian journal for science and engineering*, page 1—11, June 2021.

[2] Y. Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. volume 3, pages 932–938, 01 2000.

[3] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch, 2011.

[4] P. Dangeti. *Statistics for Machine Learning*. Packt Publishing, 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[6] Lynette Gao. Sentence embeddings for quora question similarity. 2021.

[7] Z.S. Harris. *Mathematical Structures of Languages*. Interscience tracts in pure and applied mathematics. Wiley, 1968.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[9] Christopher Olah. Understanding lstm networks, 2015.

[10] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[11] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[12] Wataru Sakata, Tomohide Shibata, Ribeka Tanaka, and Sadao Kurohashi. Faq retrieval using query-question similarity and bert-based query-answer relevance. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1113–1116, New York, NY, USA, 2019. Association for Computing Machinery.

[13] Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou, Euripides G.M. Petrakis, and Evangelos E. Milios. Semantic similarity methods in wordnet and their application to information retrieval on the web. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, WIDM '05, page 10–16, New York, NY, USA, 2005. Association for Computing Machinery.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[15] Jiapeng Wang and Yihong Dong. Measurement of text similarity: A survey. *Information*, 11(9):421, Aug 2020.