# Key Sentence Extraction From CRISPR-Cas9 Articles Using Sentence Transformers

Master's thesis in Computer science and engineering

Brage Stranden Lae & Sandra Henningsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

# Key Sentence Extraction From CRISPR-Cas9 Articles Using Sentence Transformers

Brage Stranden Lae & Sandra Henningsson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Key Sentence Extraction From CRISPR-Cas9 Articles Using Sentence Transformers

Brage Stranden Lae & Sandra Henningsson

Supervisor: Mehrdad Farahani, Department of Computer Science and Engineering
Supervisor: Rasool Saghaleyni, Department of Life Sciences
Examiner: Richard Johansson, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Key Sentence Extraction From CRISPR-Cas9 Articles Using Sentence Transformers

Brage Stranden Lae & Sandra Henningsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

## Abstract

The annotation of *CRISPR*-related articles and extraction of key content has traditionally relied on manual efforts. Manual annotation is error-prone and time-consuming. This thesis presents an alternative approach using transfer learning and pre-trained models based on the Transformer architecture. Specifically, Sentence Transformer models are fine-tuned using a *CRISPR*-related dataset. The dataset contains articles and key sentences, enabling automatic extraction of keyphrases. The study explores various modifications to the models and data to enhance performance for this task.

The results demonstrate the effectiveness of fine-tuning Sentence Transformer models for keyphrase extraction, achieving an Average R-precision of 90.4 %. Future research could focus on alternative approaches or further automation to identify entities and relations within key sentences. Key sentence extraction is complex due to the varying definitions of key content, content location, and specific use cases. However, the potential benefits of time savings and improved workflow efficiency make this approach highly valuable.

Keywords: NLP, Transformers, CRISPR, semantic search, keyphrase extraction.

# Acknowledgements

We want to extend our gratitude to three people that supervised and played a significant role in the completion of this project.

First, we would like to express our appreciation for Mehrdad Farahani, who provided us with invaluable technical advice and creative ideas whenever we encountered difficulties throughout the project. Mehrdad's expertise in Natural Language Processing and positive attitude significantly contributed to this project.

We would also like to extend our gratitude to Rasool Saghaleyni, whose insight and explanations of biomedical concepts helped us understand the fundamentals of the texts we have studied. Rasool's enthusiasm and interest in the project were truly inspiring, and without his aid in the annotation process, this project would not have been possible.

Lastly, we thank Richard Johansson for guiding us through the entire project and providing, in several iterations, excellent feedback on the report and milestones throughout the thesis work.

Brage Stranden Lae & Sandra Henningsson, Gothenburg, 2023-06-13

# Contents

# List of Figures

# List of Tables

# Acronyms

**ANN** Artificial Neural Network. 11

**ARP** Average R-Precision. 28, 41, 45, 50, 55–59

**BERN2** Advanced Neural Biomedical Entity Recognition and Normalization. 30

**BERT** Bidirectional Encoder Representations from Transformers. 16, 18–22, 43, 44, 57

**Cas9** CRISPR-associated protein 9. 1

**CBOW** Continuous Bag of Words. 10

**CI** Confidence Interval. 45

**CNN** Convolutional Neural Network. 12, 26

**CRISPR** Clustered Regularly-Interspaced Short Palindromic Repeats. 1, 31, 33, 44, 53, 57

**FFNN** Feed Forward Neural Network. 11, 12, 14

**GloVe** Global Vectors for Word Representations. 10

**IR** Information Retrieval. 6, 27

**KD** Knockdown. 5, 32

**KI** Knockin. 5, 32

**KO** Knockout. 5, 31, 32, 48

**LSTM** Long Short-Term Memory. 12, 26

**MLM** Masked Language Model. 18, 21

**MSE** Mean Squared Error. 24, 28, 29, 40–42, 45, 50, 55, 57, 58

**MultiNLI** Multi-Genre Natural Language Inference. 35

**NEN** Named Entity Normalization. 30

**NER** Named Entity Recognition. 20, 21, 25, 30, 53

**NLP** Natural Language Processing. 2, 6, 9, 12, 14, 16, 25, 32, 39

**NLTK** Natural Language ToolKit. 33

**NSP** Next Sentence Prediction. 18, 19, 21

**OOV** out-of-vocabulary. 8

**PCC** Pearson Correlation Coefficient. 28, 41, 42, 45, 50, 55–58

**QA** Question Answering. 6, 20, 21, 25

**RD** Relation Detection. 20, 21, 25, 53

**RNN** Recurrent Neural Network. 12–14, 26

**SBERT** Sentence-BERT. 22, 24, 25, 36–38

**SNLI** Semantic Natural Language Inference. 23, 35

**STS** Semantic Textual Similarity. 22

**T-SNE** T-Distributed Stochastic Neighbor Embedding. 29, 30, 42, 47, 57

**WWM** Whole-Word Masking. 21

# 1 Introduction

Genetic engineering has long been a subject of fascination and controversy. With the development of Clustered Regularly-Interspaced Short Palindromic Repeats (CRISPR) technology, the ability to manipulate the genetic makeup of organisms has become more precise and efficient than ever before. CRISPR enables scientists to edit DNA sequences with unprecedented accuracy, opening up new possibilities for medical research, biotechnology, and agriculture.

CRISPR consists of two major parts: a guide RNA to match a target gene and Cas9, which stands for CRISPR-associated protein 9. Cas9 is an enzyme that can precisely cut DNA and therefore allows modification to the genome [1]. Recent advances in biological research, particularly the development of CRISPR-Cas9 technology, have revolutionized genetic engineering. CRISPR-Cas9 enables swift and precise manipulation of DNA sequences, making it possible to study the function of specific genes more efficiently [2]. This technology has significantly impacted medical research, biotechnology, and agriculture [3]. For instance, it has been used to modify wheat to reduce its gluten content genetically, making it more suitable for individuals with coeliac disease [4].

Despite its promise, CRISPR-Cas9 research still faces challenges that must be addressed for the technology to be used effectively. One issue is the time-consuming and resource-intensive task of establishing cell lines [5]. A cell line is a culture of cells originating from a primary cell culture [6]. There is also no established database containing information about available edited cell lines, resulting in redundant and expensive work [7]. Additionally, generating cell lines is variable, challenging reproducibility and leading to conflicting results [8]. The problem of reproducibility and the problem of redundant work could be solved by introducing a database containing available edited cell lines.

AddCell[1] is a site that aims to provide an overview of the current status of CRISPR edited cell lines. However, it relies heavily on manual annotation by domain experts, which is costly and time-consuming. Furthermore, to keep up with the pace of publications in research related to CRISPR technology, it would require a massive team of manual annotators to maintain and keep the database up to date. Hence, the primary focus of this thesis is to explore and develop methods to automate and aid in this undertaking.

[1]addcell.org

## 1.1 Aim

The current workflow of annotators relies on a primitive word frequency search algorithm, which fetches articles and proposed key sentences from a database of biomedical research articles. From these propositions, the annotators decide whether the proposed key sentences and the entire article are relevant. However, the search algorithm is described as inadequate for discovering correct articles and detecting the articles' key content.

Therefore, we address the problem of extracting key sentences from biomedical research articles. The ability to automatically identify the most critical sentences in such articles can significantly aid in summarization, information retrieval, and other downstream tasks. To achieve this aim, we propose to use Sentence Transformers, a recently developed method based on the Transformer architecture. This method has shown promising results in other natural language processing tasks [9]. We believe it has the potential to improve key sentence extraction as well.

We investigate the impact of various factors on the performance of key sentence extraction. Furthermore, we explore the dataset's quality as a training and evaluation benchmark for the task. Additionally, we investigate different methods for key phrase identification and the influence of fine-tuning from diverse pre-trained models.

The main questions we aim to answer are:

- How effective are Sentence Transformers for keyphrase extraction in CRISPR-Cas9 articles?
- Which factors, both in the training pipeline and dataset, influence the performance of the developed model?
- How could this model be used in a deployment scenario?

## 1.2 Limitations

Transformers have shown promising results across a broad range of Natural Language Processing (NLP) applications. In addition, Sentence Transformers have presented state-of-the-art results in several information retrieval tasks. However, several limitations to our work need to be acknowledged.

Firstly, our method relies solely on Sentence Transformers and will not explore other methods for information retrieval, such as statistical methods, named entity retrieval, or relation detection. Furthermore, while Sentence Transformers have been suitable for other NLP tasks, other methods might be more ideal for the specific study of key sentence extraction in CRISPR-Cas9 articles.

Secondly, our approach assumes that critical information in a biomedical research article can be distilled into a small number of key sentences. Input from domain experts and our dataset studies show this is true in most cases. However, there may be instances where the critical information is split between several sentences. Our approach may not be as practical for identifying important information in these cases.

Finally, our approach is limited by the quality and size of the dataset. While domain experts have manually annotated the dataset, it has some flaws due to the nature of its curation, which will be discussed in Subsection 5.1.1. Additionally, unknown biases might exist in the dataset, resulting in a model that doesn't generalize well. We have only considered articles regarding CRISPR-Cas9 research in our evaluation; hence, the performance of using the model for retrieving new texts is unknown. However, using the model for retrieving articles and searching within them is possible upon deployment.

## 1.3 Outline

This chapter has introduced the problem, the project's aim, and the limitations.

- In Chapter 2 the theory is introduced. This concerns definitions and explanations of the information extraction field and an overview of relevant machine learning models and techniques.
- In Chapter 3, methodology and implementation details are introduced.
- In Chapter 4 results are presented.
- In Chapter 5, the results are discussed, along with a brief conclusion.

# 2
# Theory

This chapter briefly explains biomedical terms regarding the CRISPR-Cas9 technology and a more thorough account of the technical background. First, a general introduction to natural language processing is covered in the technical background before narrowing it down to machine learning and Transformer models.

## 2.1 Biomedical Background

CRISPR-Cas9 technology contains some terminology that is important to have a fundamental knowledge of to understand the task better.

Organisms, cells, and genes are some rudimentary concepts in CRISPR-Cas9 research. Organisms consist of several cells, most of which contain the complete set of DNA for that organism. Organisms are single living entities, ranging from tiny bacteria to massive whales. They are defined by their ability to carry out the essential functions of life, such as metabolism, reproduction, growth, and response to stimuli. Whether a plant, animal or any other living entity, each organism is unique and designed to thrive and survive in its environment. A cell is a fundamental unit of life and the smallest building block of organisms. DNA, the primary molecule of heredity, is at the core of living cells. DNA acts as a blueprint that directs the structure and function of each cell and contains genetic information passed down from one generation to the next during reproduction. Genes, which are segments of DNA, play a crucial role in controlling physical development, behavior, and other traits of individual plants or animals. They are inherited from parents and are responsible for passing on characteristics that make us who we are [10].

When performing gene editing using CRISPR-Cas9, the three primary operations are Knockout, Knockin or Knockdown (KO, KI or KD, respectively). KO is the act of replacing or disrupting parts of a gene to deactivate the function of the gene. By performing KO, researchers can explore the effects of losing a gene in an organism. KI replaces a mutated DNA sequence in a gene with the endogenous (original) gene. KI allows analysis of how the knocked-in gene affects an organism. A third, more recent technique is the KD operation. KD is similar to KO because it disrupts the gene's function. However, KD uses a slightly different technique than KO and only temporarily erases the role of a gene [11].

## 2.2 Natural Language Processing

The amount of natural language text in the world is constantly increasing. The increased pace at which texts are released, particularly in the scientific domain, makes it difficult for researchers to stay updated. Natural Language Processing (NLP) is a collection of computational techniques to represent and analyze human language. To automatically analyze texts, a deep understanding of natural language is required. NLP can be broken down into several fields, where one broad distinction is between text mining and text generation. Text mining is the extraction of information, whereas text generation is the generation of texts [12]. Examples of NLP include online Information Retrieval (IR), aggregation and Question Answering (QA) [12]. Online IR refers to finding and obtaining relevant information from the vast amount of digital resources and databases available on the internet. As online content expands exponentially, efficiently finding and retrieving specific information has become increasingly vital across diverse domains such as research, education, business, and daily activities [13]. Data aggregation is the process of summarizing a large data pool for high-level analysis. The primary objective of QA is to create systems that can automatically deliver precise answers to questions asked by humans in their native language (see Section 2.9) [12].

Online IR, aggregation, and QA have mainly been based on algorithms that rely on the textual representation of web pages. These algorithms are good at fetching information, splitting text into parts, spellchecking, and word-level analysis. Nevertheless, they are unsuccessful in studying at the sentence or paragraph level. When it comes to interpreting sentences and extracting meaningful information, the capabilities of these models are limited [12]. The limitation of these approaches is that they can only process based on information they see in the text and don't have any underlying knowledge or background information. Humans don't have these kinds of limitations since every word in a text activates semantically related concepts, sensory experiences, and relevant episodes. This makes it possible for humans to complete complex NLP tasks such as word sense disambiguation (to know how the context affects the meaning of a word), textual entailment (to logically determine whether one sentence can be deduced or inferred from another) and semantic role labeling (to assign labels to words or phrases in a sentence that indicates their semantic roles within the sentence, such as agent, goal, or result), quickly and effortlessly. New techniques, such as those described in this Master's thesis, attempt to bridge this cognitive gap. This is done by emulating the processes recognized as part of the human brain and used for NLP by humans [12].

### 2.2.1 Challenges of NLP

It is a difficult task to develop a model that understands natural language. The ambiguity of words and sentences is significant. The word *"fly"* can, for example, mean both the insect and the verb *"to fly"*. The context can also determine the meaning of a sentence. Additionally, syntax helps to decide how to combine words into larger meanings. In the sentence *"I saw the Golden Gate bridge flying into San Francisco"*, the syntax makes it clear that the person was flying into San Francisco, not that the bridge was flying. An internal representation needs to be built, and the information needs to be used appropriately. Ambiguity is present in sentences as well. In the sentences *"Jack went to the store. He found the milk in aisle three. He paid for it and left."*. The word *"it"* could refer to the store, the milk, the aisle, or three. The internal representation is the most important part of determining the meaning of *"it"*. It leads to the following questions: what is the internal representation, how could it be used for these ambiguities not to occur, and how can a machine understand this the same way a human does [12]?

### 2.2.2 Tokenization and Segmentation

In natural language, it is essential to define what a sentence or word consists of. To define these units is difficult since many languages and writing systems exist. In addition, the natural language contains inherent ambiguities, which complicate the task. Text segmentation is the task of dividing a text into linguistically meaningful units. Individual characters form the smallest segmentation in a language's written system. Then come words, consisting of one or more characters, and sentences consisting of one or more words [14].

Tokenization is a process that breaks up sequences of characters in a text and is done by locating word boundaries, representing the end of one word and the start of another. In computational linguistics, these found words are referred to as tokens. In languages where word boundaries are not marked in the writing system, tokenization can be referred to as word segmentation [14]. The difference between segmentation and tokenization is that segmentation involves splitting the input text. In contrast, tokenization specifically focuses on assigning labels to words or phrases based on predefined criteria. These predefined criteria could, for example, be markings of word boundaries. Tokenization is a form of segmentation, but it uses semantic criteria or token dictionaries to assign token IDs for downstream processing [14].

Sentence segmentation can be defined as a process that determines more extended units, which consist of one or more words. In sentence segmentation, sentence boundaries must be defined between words in different sentences. Most languages have punctuation as a marker at the end of a sentence. Sentence segmentation is often called sentence boundary detection or sentence boundary recognition. An example of a simple sentence segmentation rule is using the period "." to identify the end of a sentence. However, a period is often used in acronyms, which illustrates why segmentation is a challenging task, as simple syntactic rules are rarely sufficient [14].

The primary tool for processing textual data is a tokenizer. A tokenizer can keep track of all the tokens and give them unique token IDs. In tokenization, a trade-off is made between tokenizing into tokens with semantic meaning while having an appropriate length vocabulary. The tokenizer can use word-based tokenization, character-based tokenization, and subword-based tokenization [15]. Nayak et al. (2020) list some examples of these types of tokenization. For example, consider the sentence:

*The girl loves playing with her toys*

**Word-based tokenization:**

*[The, girl, loves, playing, with, her, toys]*

**Character-based tokenization:**

[T, h, e, g, i, r, l, l, o, v, e, s, p, l, a, y, i, n, g, w, i, t, h, h, e, r, t, o, y, s]

**Subword-based tokenization:**

*[The, girl, love, s, play, ing, with, her, toy, s]*

*Word-based tokenization* requires an extensive vocabulary since every word has its token. Another drawback is that similar words will have different representations since they have different tokens. Hence, the words "play" and "playing" will get different tokens despite having similar semantics [16]. Besides, *word-based tokenization* could lead to issues with out-of-vocabulary (OOV) words. OOV words are words that have not been added to the vocabulary and are therefore unknown.

*Character-based tokenization*, on the other hand, does not require an extensive vocabulary. However, single characters do not carry a lot of information. As a result, the model has to consider several tokens to interpret the meaning of a word. In addition, the model has to handle more significant inputs since one word consists of more characters [17].

*Subword-based tokenization* can be viewed as a compromise between character-based tokenization and word-based tokenization. Thus, it can be considered a trade-off between tokens carrying semantic meaning and an appropriate length vocabulary. It is a common tokenization strategy in state-of-the-art models [18]. Typical sequences of characters are left as they are, but longer and uncommon words are split into subwords. These subwords can be concatenated into whole words at a later stage. An advantage of subword-based tokenization is that the vocabulary does not have to be as large as word-based tokenization. Additionally, subword tokenization allows for larger tokens, which carry more information than the character-based counterpart. Prefixes and suffixes can be learned, and similar words with different endings can be considered similar. For instance, the word "genes" might be tokenized as ["gene", "##s"], where the two hashtags indicate that "s" is a suffix. The downside is that the subword tokenization might split words in non-intuitive ways depending on how the tokenizer is trained [19].

Byte-Pair and WordPiece are the most common algorithms following the subword tokenization paradigm. The Byte-Pair algorithm first finds every unique word in a corpus before creating a vocabulary of these words and their respective word frequencies. Then a base vocabulary consisting of every character in the unique words is created. From the base vocabulary, temporary tokens are made in each iteration that consists of neighboring token pairs. The most frequent temporary token, which the previous count of words determines, is added to the vocabulary. This process is then repeated, adding one token per iteration until the specified size of the vocabulary is reached. Byte-Pair ensures that the most common words are represented in the vocabulary as one token. Meanwhile, less common words are divided into two or more subword tokens [20]. On the other hand, WordPiece starts by initializing a base vocabulary consisting of every character in the training data. The pair selection is based on whether the pair maximizes the likelihood of the training data when added to the vocabulary. The idea is that maximization of the likelihood of the pair, whose probability is higher than all the other pairs, results in the best training data [19]. WordPiece is explained further in Section 2.6.

### 2.2.3 Semantic Analysis

Semantic analysis is used to determine the meaning of a text. The meaning of a text, rather than the correctness of it, is essential because a language processing system could, for example, be told to do something in response to a text, like moving a robot arm or retrieving data [12]. Therefore, capturing the semantic meaning of a text plays a critical role in text-mining research. Traditional methods rely on a bag-of-words approach, which means the text is represented as a bag, or a multiset, of its words. However, these models may not accurately model the semantics due to the ambiguity of natural language [21]. For example, the sentences below are used to describe the same finding.

*"It has recently been shown that Craf is essential for Kras G12-induced NSCLC [21]"*

*"It has recently become evident that Craf is essential for the onset of Kras-driven non-small cell lung cancer [21]"*

The bag-of-words models do not capture the similar semantic meaning of these sentences. Another more promising approach is to represent the semantic meaning in the form of embeddings.

#### 2.2.3.1 Embeddings

A common strategy to help in the semantic analysis is to embed the text, where embedding is a translation from natural text to a numerical vector. A prevalent approach is first to tokenize the text, as described in Subsection 2.2.2, and then embed each word into word-embeddings. In embedding-based approaches, the semantics is represented as high-dimensional vectors. The embeddings are usually learned from large-text corpora, and words with similar semantic meanings are expected to have embeddings closer to each other in the vector space. Embeddings have shown promising results and are increasingly important in text mining research. Methods to create embeddings often involve machine learning or statistics [21]. The archetype when explaining embeddings is the example of the words woman, man, queen, and king. Given a numerical representation of these four words, the goal is to have representations such that $king - man + woman = queen$. This example is visualized in Figure 2.1. Embedding natural language is a powerful technique because it allows mathematical computation on texts. For instance, computing similarity and semantics becomes easier when performed on embeddings.

Representing words in a numerical format while retaining their contextual meaning is crucial for performing NLP tasks. In this regard, Google introduced two

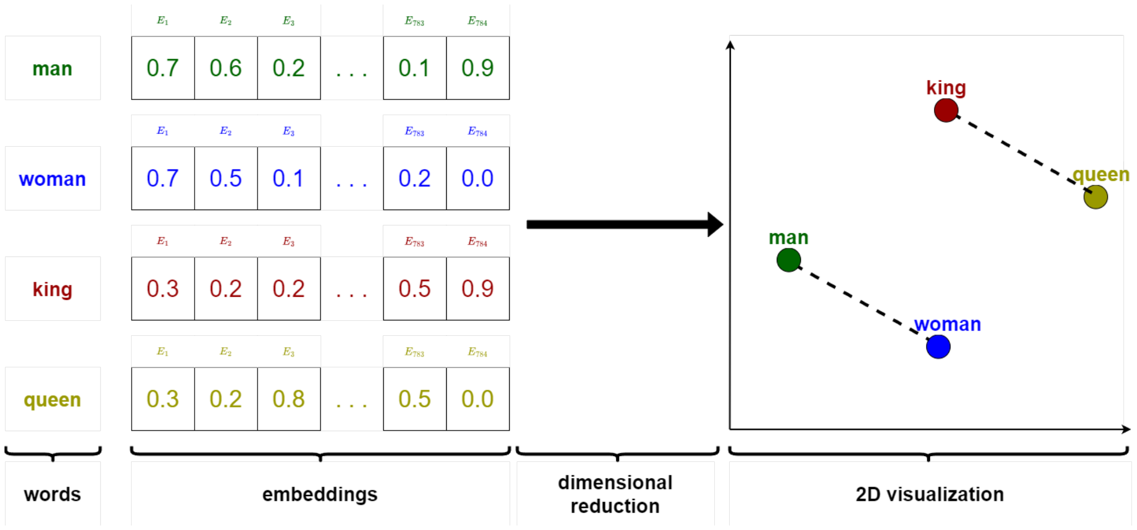


Figure 2.1: By translating natural language words into vectors, they can be visualized in a 2D space after dimensional reduction.

architectures that compute continuous vector representations of words from massive data sets, known as Continuous Bag of Words (CBOW) and Continuous Skip-Gram Model [22]. CBOW is trained on 6 billion words from Google News. The model does not depend on the order of the words, which is why it is referred to as a ”Bag of Words” approach [22]. The *Continuous Skip-Gram Model*, on the other hand, uses a log-linear classifier. As a result, this architecture produces more effective semantic and syntactic relationships between words [22].

Stanford University introduced a model for word embeddings known as GloVe, short for Global Vectors for Word Representations. This model has demonstrated superior performance compared to other models in tasks related to word similarity [23].

#### 2.2.3.2 Distance Functions

Words with similar semantic meanings are expected to have embeddings close to each other in the vector space. Meanwhile, words with different meanings are expected to have embeddings further apart [21]. Consequently, a distance metric for vector comparison is needed when interpreting semantic similarity between words or sentences. Some standard distance functions for calculating the semantic similarity are *Euclidean distance*, *Manhattan distance*, *Dot-product*, and *Cosine distance*.

The Euclidean distance between two n-dimensional vectors $X = (x_1, x_2 ... x_n)$ and $Y = (y_1, y_2 ... y_n)$ is defined as

$$Euclidean\ Distance = |XY|^2 = (x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2$$

, by [24].

The Manhattan distance calculates the distance between coordinates in a grid-like path. It calculates the distance between a pair of vectors by summing the absolute distance between the components of the two vectors [25]. Consider the n-dimensional vectors $X = (x_1, x_2 ... x_n)$ and $Y = (y_1, y_2 ... y_n)$. The Manhattan distance can be calculated [25] as:

$$Manhattan\ Distance = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n|.$$

There is often a trade-off between accuracy and speed when choosing between the Manhattan distance and Euclidean distance. It is hard to say when the Manhattan distance will be more accurate, but it is faster since there is no need to square the differences. Therefore, the Manhattan distance is better to use as the data dimension increases [25].

The dot-product distance (also known as the scalar product) takes two or more vectors and multiplies them element-wise, which results in a single scalar value. The dot product will be positive if the vectors are in the same direction. Conversely, different directions will result in a negative dot product [26].

Finally, the cosine distance is defined as $1 - cosine\_similarity$. Hence, there is an inverse relationship between the cosine distance and the cosine similarity. An increased distance results in decreased similarity and the other way around [27]. The cosine distance between the two points (X and Y) is defined as [27]:

$$Cosine\ Distance = 1 - \cos(\theta) = 1 - \frac{X \cdot Y}{||X||\ ||Y||}.$$

The distance between the sentence vectors is determined to calculate the semantic similarity between sentences. Cosine similarity considers the angle between the vectors in the vector space but not the weight or magnitude of the vectors compared to Euclidean distance. Cosine similarity is a measure that can be used when the vectors magnitude is unimportant. A typical example is working with text data [27].

## 2.3 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computing system inspired by biological neural networks. ANNs are designed to solve problems regarding, for example, pattern recognition, prediction, optimization, memory, and control [28]. There are multiple types and designs of ANNs, and the essential ones to this thesis are described in this section.

### 2.3.1 Feed Forward Neural Networks

ANNs can be viewed as weighted directed graphs with artificial neurons as nodes in the graphs, and there are directed edges with weights that are the connections between neuron inputs and outputs. Feed Forward Neural Networks (FFNNs) are a group of ANNs defined as simple graphs, i.e., they contain no loops, and the information flows forward in the network. The simplest form of FFNN is a single perceptron. In this case, there is only an input and output layer [28]. The perceptron is a linear classifier, and the following formula is used to compute the output:

$$y = \sigma(\sum_{i=0}^{n} w_i x_i + b),$$

where $w = weight$, $x = input$, $b = bias$ and $\sigma$ is a non-linear activation function [28].

In supervised learning, the training data consists of input-output pairs. The learning algorithm in this approach adjusts the model parameters, such as weights and biases, iteratively. The goal is to enable the model to map inputs to corresponding outputs accurately. The predicted output is compared against the expected value, and the difference is measured using a loss function. During training, the loss is decreased to minimize the difference between the predicted and expected output. For the perceptron algorithm, the learning process is straightforward. Weights are reduced when the predicted output exceeds the desired output. Otherwise, weights are increased [28]. Perceptrons utilize the gradient descent method to learn the weights. Gradient descent is an optimization technique that progressively updates the parameters of an objective function towards its minimum by moving in the opposite direction of the function's derivative [29].

In the multi-layer perceptron, neurons are organized into layers with unidirectional connections between the neurons. There are at least three layers, an input layer, hidden layer(s), and an output layer [28].

FFNNs are static, meaning they produce a set of output values, not a sequence of values from input. Furthermore, they are memoryless because their response to an input is independent of a previous state in the network. This makes FFNNs different from recurrent neural networks that are dynamic systems [28].

### 2.3.2 Recurrent Neural Networks

Recurrent Neural Network (RNN), particularly Long Short-Term Memory (LSTM) models [30] and Gated RNNs [31], formed the base architecture of the state-of-the art for sequential modelling problems until 2017 [32]–[34]. In comparison to the FFNN, the graph of RNNs contains loops or cycles [28].

RNNs has to be computed sequentially, where the input of each computation depends on the previous output. Thus, efficient parallelization is not possible. This forms a computational bottleneck of the RNN architecture for sequential learning problems. Furthermore, the RNN architecture suffers from the vanishing gradient problem, where the further away a sequential dependency lies, the harder it is for the model to capture this relationship. In other words, in the same way that a Convolutional Neural Network (CNN) favors spatial proximity, RNNs favors temporal proximity. The temporal proximity property of RNNs is often a disadvantage in NLP, as the model will forget the contextual dependency of words that are far apart from each other. LSTMs partly solves this problem, but not quite.

## 2.4 Sequence-to-Sequence Models

Sequence-to-sequence models are designed to transform one sequence to another, where a sequence of input bits is transformed into an output of output bits [35]. Typically, these models are composed of two major components: an encoder and a decoder, which used to consist of RNNs [36]. The encoder reads the input sentence, which is a sequence of vectors $\boldsymbol{x} = (x_1, ..., x_{T_x})$ into a vector $c$ [36]. The most common approach is to use the model in such a way that:

$$h_t = f(x_t, h_{t-1})$$

and

$$c = q(\{h_1, ..., h_{T_x}\}),$$

where $h_t \in \mathbb{R}^n$ is the hidden state at time $t$, $c$ is a vector that the encoder has generated from the sequence of hidden states, and $f$ and $q$ are some non-linear functions [36].

The decoder, on the other hand, is trained to predict the next word $y_{t'}$ by having access to the context vector $c$ and the previously predicted words $\{y_1, ..., y_{t'-1}\}$. The decoder defines the probability of the translation $\mathbf{y}$. This is done by decomposing the joint probability into ordered conditionals [36]:

$$p(\boldsymbol{y}) = \prod_{t=1}^{T_y} p(y_t | \{y_1, ..., y_{t-1}\}, c),$$

where $\boldsymbol{y} = (y_1, ..., y_{T_Y})$ [36]. With an RNN, the conditional probabilities can be defined as:

$$p(y_t | \{y_1, ..., y_{t-1}\}, c) = g(y_{t-1}, s_t, c),$$

where g is a non-linear, possibly multi-layered function. It outputs the probability $y_t$, where $s_t$ is the hidden state of the RNN [36].

### 2.4.1 Attention-Based Models

An issue with the encoder-decoder approach is that the neural networks must compress all necessary information of a source sentence into a fixed-size vector. Consequently, it is difficult for the model to handle longer sentences [36]. Cho et al. (2014) showed that the performance of an encoder-decoder model rapidly decreases as the length of the input sentences increases [37].

An extension to address this issue was introduced by Bahdanau (2014), which learns to align and translate jointly [36]. The extension mimics cognitive attention by deciding parts of the input sequence. In the proposed model, each generated word in a translation is accompanied by a (soft-) search process. This process identifies a set of positions in a source sentence that contains the most relevant information. The context is then used to predict a target word based on the associated context vectors of these positions and the previous target words generated in the sequence. This approach differs from the encoder-decoder technique in not trying to encode an entire input sentence into a fixed-size vector. Instead, the input sentence is encoded into a sequence of vectors. Then selectively chooses a subset of these vectors while decoding the translation. Hence, enabling the neural translation model to handle long sentences better and avoid compression of all information from a source sentence into a single fixed-size vector [36].

The approach of jointly learning to align and translate significantly improves translation performance over the encoder-decoder technique. The improvements are particularly noticeable with longer sentences but can be observed for sentences of any length [36].

## 2.5 Transformers

Transformers were presented in 2017 [34], and swiftly became a popular architecture for building NLP models [18], [38], [39]. Unlike RNN-based architectures, the Transformer represents the input as a set rather than a sequence, allowing for efficient data parallelization. To maintain the temporal property of the data, the input data needs to be positionally embedded. Furthermore, self-attention is introduced, which is an attention mechanism that relates different positions of one sequence to each other, such that a representation of the entire sequence can be computed. The central building blocks of the Transformer are the encoder and decoder.

### 2.5.1 Architecture

One challenge in sequential learning is to be able to process and return variable-length inputs and outputs. To solve this problem, the encoder-decoder architecture is common [32], [33]. This general architecture is also used by the Transformer, with six layers of encoder-decoders, where the encoder output in layer $i$ and $i-1$ is the input of the decoder in layer $i$. Each block of encoder and decoder contains two and three sub-layers, respectively. Each sub-layer has a residual layer and normalization such that the output of a sub-layer is $LayerNorm(x + SubLayer(x))$. $LayerNorm$ is the normalization function of the sub-layer, $x$ is the input of the sub-layer, and $SubLayer$ is the function of the sub-layer, for instance, a fully connected FFNN. A visualization of the architecture of one layer in the Transformer is shown in Figure 2.2, with emphasis on the encoder.

Because the input of the Transformer is modeled as a set, positional embeddings are added to the input before entering the encoder, which is done to maintain the temporal information of the input data. An example of input embeddings of natural language is given in Figure 2.3 in Section 2.6. The encoder block consists of two sub-layers, one multi-head-attention system, and one FFNN. The encoder processes the input sequence $\boldsymbol{x} = (x_1, ..., x_n)$, outputting a fixed-length vector representation $\boldsymbol{z} = (z_1, ..., z_n)$ of the input. The decoder takes this vector representation as input and generates the output sequence one element at a time, using the encoder's representation and the previously generated output elements as context. In addition to the two sub-layers of the encoder, the decoder has a masked multi-head attention layer, which enables the mapping of relevant tokens to the encoder for translation [40]. The final operation of the decoder is a linear layer and a softmax layer, which ultimately produces output probabilities.

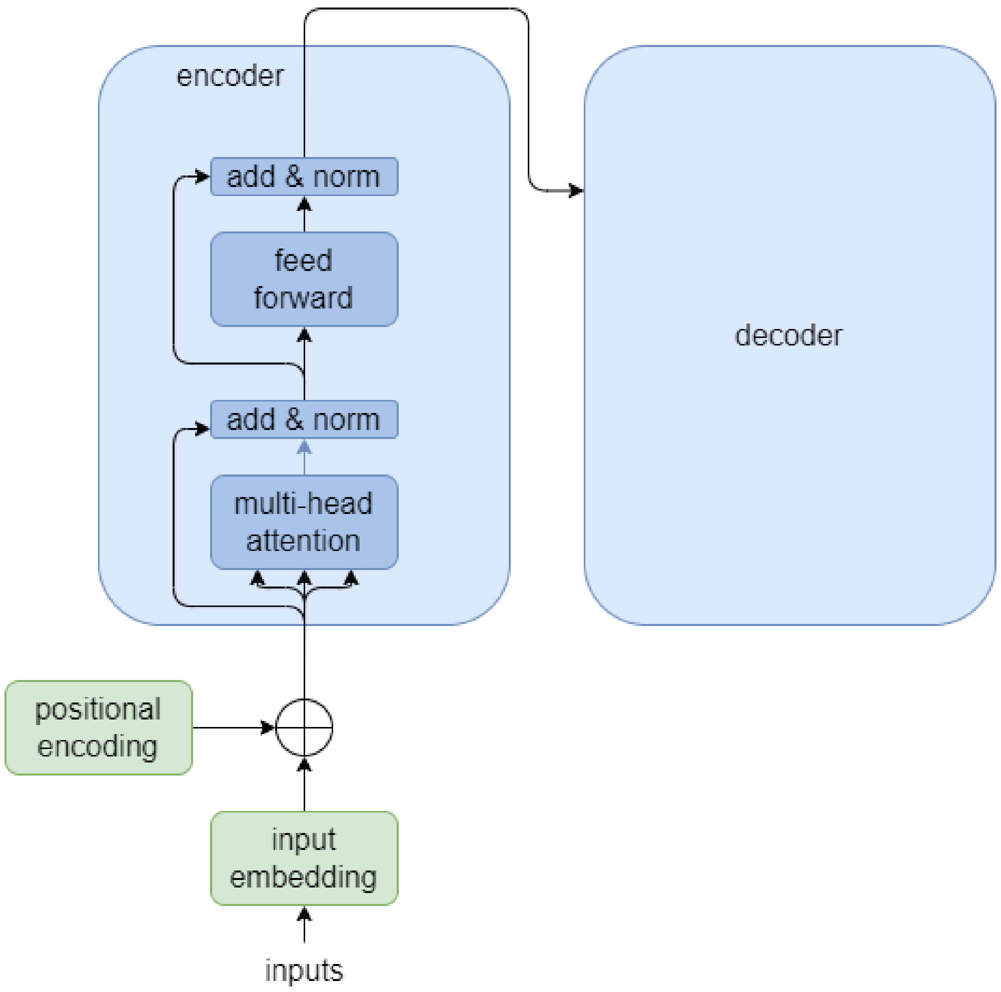


Figure 2.2: One layer of the encoder-decoder architecture of the Transformer, with emphasis on the encoder block.

## 2.5.2 Attention

In natural language, the context of words often plays a significant role in deducing the semantics of a text. The self-attention layer is a critical component of the Transformer architecture, which enables the look-up of remaining input words at different positions to determine the relevance of the currently processed word. This is done for all words, making a superior encoding possible and gaining a contextual understanding of all the input words. The Transformer model contains multi-headed self-attention mechanisms, which means that for each sequence, separate heads calculate separate attention vectors in parallel. Finally, a weighted sum of these vectors produces the final attention vector. By implementing several self-attention heads, the capacity of the model to emphasize context within a sequence improves, compared to only having one attention head [40].

Using self-attention instead of traditional recurrent approaches provides several advantages. Firstly, the computational complexity is significantly lower, as each layer provides faster computation, and parallelization is possible to a further extent between layers. Furthermore, dependencies between sequences with a long temporal distance are better learned by the self-attention mechanism than by traditional approaches [34]. Nevertheless, full self-attention is still computationally expensive, as each token needs to attend to all other tokens in a sequence. Hence, the complexity increases quadratically with the sequence length [41]. In other words, if the length of an input sentence is doubled, the computational complexity is quadrupled.

## 2.6 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained NLP model, developed by Google and presented in 2018 [18]. Pre-training is a technique in machine learning for training a model on general concepts without training for a specific task. Since the introduction of BERT, it has been an instrumental part in the development of NLP applications, and in just over a year since its publication, over 150 research articles have been published investigating and refining the model [42]. As the model's name implies, the model is based on the Transformer architecture. Bidirectionality is another important concept of BERT. In essence, bidirectionality entails that given a word, its meaning can be inferred from the context of the text before and after the word. It is strongly emphasized in the paper that bidirection is crucial to model complex linguistic structures accurately.

The architecture of BERT follows closely that of the Transformers' encoder block, as seen in Figure 2.2. Because Transformers take input as sets instead of sequences, additional annotation is needed to form the input in an NLP context. The annotation includes position embeddings, segment embeddings, and token embeddings. The position embeddings tell what order in the text a token is, i.e., they tell the model what order the words in a sentence have. Similarly, segment embeddings tell in what order different segments appear. Segments can be thought of as sentences, and they are separated by the [SEP] token. Finally, token embeddings are the actual content of the text, which has been divided into reasonable chunks, along with different labels such as [SEP], [CLS], etc. This separation of tokens is referred to as tokenization, and often a word and a token refer to the same thing. The input representation of the sentence "My dog is cute. He likes playing." is shown in Figure 2.3. The [CLS] tag marks the beginning of the sequence, replaces dots with [SEP], and otherwise, words are mostly kept as they are. For segment embeddings, all words in the first sentence are marked with an *A*, and all words in the second sentence are marked with a *B*. Finally, position embeddings mark the position of words in the sentence. The first word, “my”, is marked with 1, the second word, “dog”, is marked with 2, etc. The word “playing” is split into two, “play” and “##ing”, as the WordPiece embedding model does not have the word “playing” in its vocabulary. A double hashtag (“##”) denotes a word’s suffix.

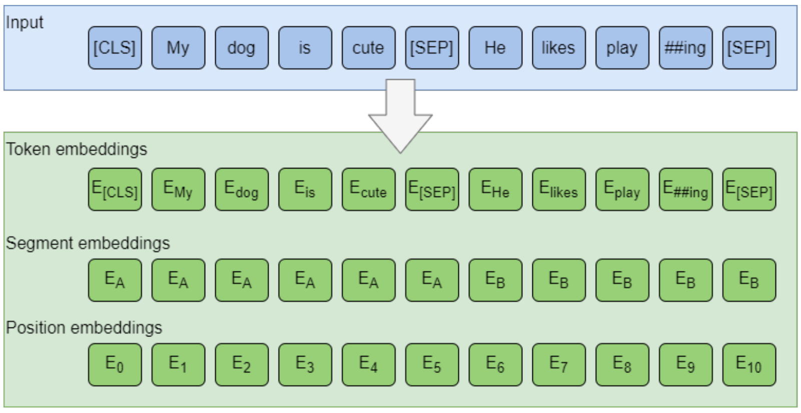

Input
[CLS]
My
dog
is
cute
[SEP]
He
likes
play
##ing
[SEP]
Token embeddings
$E_{[CLS]}$ $E_{My}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{He}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
Segment embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
Position embeddings
$E_0$ $E_1$ $E_2$ $E_3$ $E_4$ $E_5$ $E_6$ $E_7$ $E_8$ $E_9$ $E_{10}$

Figure 2.3: An example of the input representation of BERT [18].

BERT uses WordPiece embeddings with a vocabulary of roughly 30,000 tokens [21]. WordPiece embeddings were initially developed by Google for speech recognition tasks in Asian languages with complex writing systems, a large inventory of characters and homonyms, and no or very few spaces between words. These issues make it necessary to segment the text, which can result in numerous out-of-vocabulary words in the model. To address the challenge of out-of-vocabulary words, WordPiece representation was proposed, which learns word units from extensive data without encountering out-of-vocabulary words [43]. In the WordPiece approach, the first token is represented by a special token, [CLS], and the final hidden state corresponding to this token is utilized as the aggregate sequence representation for classification tasks. To process pairs of sentences as a single sequence, another special token separates the sentences, [SEP], and an embedding is learned for every token that indicates whether it belongs to sentence A or B. Finally, the input representation for each token is constructed by summing the corresponding token-, segment-, and position-embeddings [43]. An embedding can represent a word, a sequence of words, or a sentence. Some ways to create sentence embeddings are to take the average of the word embeddings, by using the output of the [CLS] token, or by using different pooling functions. Pooling functions are filters through which the information is processed. It could, for example, be to take the mean or max of all the output vectors [44].

BERT is developed through two steps: pre-training and fine-tuning. The pre-training is done in a self-supervised fashion on a vast text dataset, whereas the fine-tuning is done to specialize the model for specific downstream tasks. The reason for splitting the training into two steps is closely related to the concept of transfer learning and domain adaptation. When there is not much data available in the target domain, and there exists data in a source domain that can be generalized to the target domain, there is a need for high-performing models that are trained on easily obtained data from a general source domain. This is referred to as transfer learning [45]. Hence, one reason why transfer learning is favorable is that it might yield higher performance. Another reason relates to computational efficiency. The assumption in NLP is that general semantic understanding is needed for most tasks. Hence, a lot of computational time is saved by pre-training a general model once on a large corpus and then fine-tuning it for downstream tasks.

Pre-training is done by two separate training operations done simultaneously. These are Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM and NSP are designed to train in a self-supervised fashion, so a large corpus can easily be used for pre-training. MLM is used for learning word semantics, and NSP is for learning sentence semantics. Both tasks' principles include hiding or masking words or sentences and having the model predict the hidden word or sentence. In Listing 2.1, some examples of different tasks, given the text: "The man went to the store. He bought a gallon of milk.", are shown. This experiment was conducted on a BERT model that was only pre-trained and not fine-tuned for any downstream task. The token [CLS] indicates the start of a text, [SEP] is a separator token between sentences, and [MASK] displays a hidden word that the model will predict. Firstly, MLM is demonstrated, where the model predicts what word is hidden by the [MASK] token:

Listing 2.1: Masked Language Model (MLM) example

```
1)
[CLS] the man went to the [MASK] [SEP] he bought a gallon of milk.
    prediction = store

2)
[CLS] the man went to the store [SEP] [MASK] bought a gallon of milk.
    prediction = he

3)
[CLS] the man went to the [MASK].
    prediction = door.
```

The first MLM example in Listing 2.1 illustrates how BERT predicts a word given the right context of the text. Because from the second sentence, "he bought a gallon of milk", the model infers that it was a store the man went to, which is the correct word. The second example shows the contrary, where BERT needs to figure out from the previous text who it was that went to the store. It correctly predicts "he", given that "the man went to the store". In the third example, removing the second sentence removes information, and BERT cannot deduce from the input where the man is going, so it guesses "door".

In the first NSP example in Listing 2.2, BERT predicts that it is natural that the second sentence follows from the first, indicated by the label "isNext". In the second sentence, "he bought a gallon of milk" is replaced with "penguins are flightless birds". Here, BERT outputs the label "notNext", meaning it no longer believes the second sentence follows from the first.

Listing 2.2: Next Sentence Prediction (NSP) example

```
1)
[CLS] the man went to the [MASK] [SEP] he bought a gallon of milk.
          Label = isNext.

2)
[CLS] the man went to the [MASK] [SEP] penguins are flightless birds.
          Label = notNext.
```

In order to pre-train BERT, the paper used the two tasks demonstrated above on a text corpus containing 3.3 billion words. The corpora consist of general-purpose language from English Wikipedia[1] articles and BooksCorpus [46]. 15% of words were replaced by the [MASK] token, which was then used for training the model with the MLM task. Given two sentences in the NSP part, the second sentence would be replaced by a random sentence with a probability of 50%.

Pre-training BERT is a process that requires immense computational power and time. Nevertheless, this is a process that only needs to be done once. Furthermore, when BERT has been pre-trained, it has the foundation to be fine-tuned for a wide range of downstream tasks. Fine-tuning is not at all as computationally expensive, and it is described in the paper as being easy to do due to the self-attention mechanism of the Transformer architecture, which allows BERT to model many downstream tasks. BERT uses a fine-tuning approach for transfer learning, meaning that after pre-training, the entire parameter space of the model is fine-tuned for the specific task. Fine-tuning should take a few hours for most tasks when training on a GPU, where the base of fine-tuning is the pre-trained model.

After pre-training BERT, it can be fine-tuned for several downstream tasks. One main advantage of separating the training into two steps, pre-training, and fine-tuning, is that the pre-training will train the model at general skills on a large corpus, such that it has a good foundation for all downstream tasks. Fine-tuning is typically done on a smaller corpus, where a domain shift is often desired. Figure 2.4 shows the BERT architecture for one such downstream task, sentence pair classification. Here, an additional output layer is integrated, so few parameters must be learned from scratch. In the figure, the bottom layer is a visualization of the input to BERT, including special tokens [CLS] and [SEP], as well as the tokens. The letter E represents the input embeddings, and the contextual representation is represented by T.

## 2.7 Biomedical Domain Models

By using domain-specific texts for training a language model for the biomedical domain, it has been shown that the performance of the models increases when used

[1]https://www.wikipedia.org

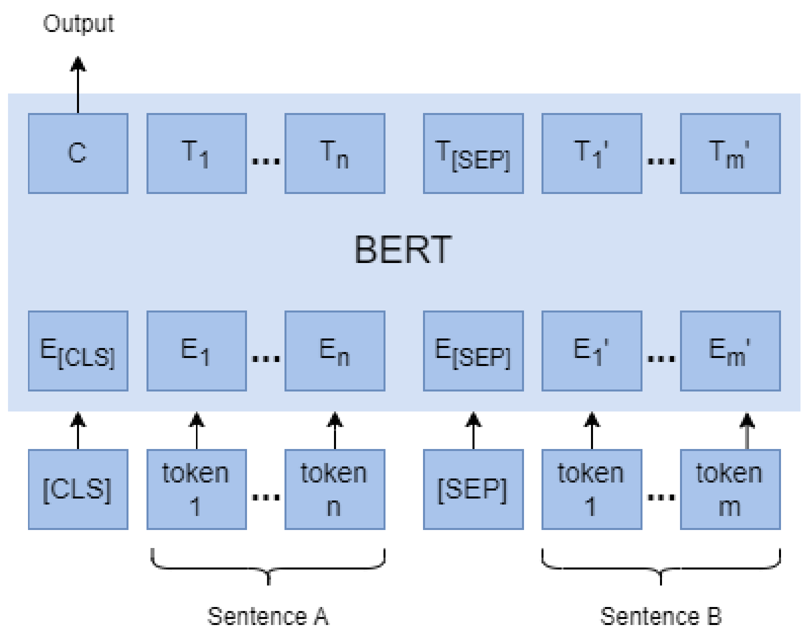


Figure 2.4: BERT architecture for sequence classification tasks.

for inference on biomedical texts [38], [47]–[49]. This section presents the general-purpose biomedical domain models, BioBERT and PubMedBERT.

### 2.7.1 BioBERT

BioBERT was developed by Lee et al. in 2019 as a biomedical domain-specific version of BERT [38]. BioBERT was developed by starting from the initial weights of BERT before it was pre-trained further from a corpus of texts from PubMed[2]. The new pre-training iteration was deemed necessary as biomedical texts have a different distribution than the general language texts BERT was trained on, in addition to additional vocabulary. After pre-training, BioBERT was fine-tuned on three downstream tasks for evaluation, NER, QA and Relation Detection (RD). NER is the identification of named entities, RD is the task of finding relations between such entities, and QA is the task of answering natural language questions, given related text to the query (see Section 2.9). An advantage of using the BERT architecture for modeling downstream tasks is that the fine-tuning approach requires minimal architectural adjustments. In the NER task, experiments were completed on datasets containing four entity classes (diseases, drugs and chemicals, genes and proteins, and species). In RD, the relations studied were gene-disease and protein-chemical relations. BioBERT outperformed BERT and presented state-of-the-art results for many BioRD and BioNER tasks.

[2]pubmed.ncbi.nlm.nih.gov

### 2.7.2 SciBERT

SciBERT is a pre-trained language model based on the BERT architecture [48]. SciBERT is a language model that specializes in the scientific domain, and it is pre-trained on a corpus of scientific texts. In addition to a corpus difference between BERT and SciBERT, a new scientific vocabulary is built using the WordPiece tokenization method. The resulting vocabulary of SciBERT, having the same size as the BERT vocabulary, only shares 58% of the same tokens, illustrating a substantial domain shift between the corpora, as the most frequent words are often different. The use of an in-domain vocabulary separates the methods of SciBERT from BioBERT, and ablation studies in SciBERT show that in-domain vocabulary improves performance. SciBERT was trained on a corpus that is a combination of papers from the computer science domain and the biomedical domain.

### 2.7.3 PubMedBERT

PubMedBERT was published in 2021 by Gu et al. [47]. PubMedBERT attempts to solve the same problem as BioBERT, which is to create a general pre-trained language model for the biomedical domain. However, it challenges the assumption that transfer learning is a sound approach to developing biomedical language models. Transfer learning in this context is to start training from the weights of BERT, which has been trained on a general domain corpora, towards the specific biomedical textual domain. BioBERT is an example of a model trained in such a fashion. Transfer learning is generally applicable if there is not much data available in the target domain and that the domain shift from the general domain to the target domain is not too large. It is argued in the PubMedBERT paper that in terms of pre-training, vast corpora of high-quality biomedical texts are available from PubMed. Therefore, the target domain has a lot of training data available. Furthermore, the domain shift between the general corpora that models such as BERT has been trained on and the PubMed corpus is perhaps too large. Gu et al. write: ”In fact, the majority of general domain text is substantively different from biomedical text, raising the prospect of negative transfer that actually hinders the target performance”. Finally, it is emphasized that the vocabulary of PubMedBERT contains words from PubMed, whereas the vocabulary of BioBERT only consists of the original BERT vocabulary.

PubMedBERT was developed using similar pre-training configurations as BERT, with MLM and NSP as the main objectives. It is pointed out that the effectiveness of NSP has been questioned [50]; however, NSP is included for a fair comparison with other models. An important distinction with the setup of MLM with BERT is that Whole-Word Masking (WWM) is used. WWM asserts that a token masked for MLM consists of an entire word, not a subword. This is the standard approach, as the language model will be encouraged to capture more semantic context. After pre-training, PubMedBERT was fine-tuned for downstream tasks and evaluated on a set of biomedical tasks such as NER, RD, and QA. The results were compared to BERT, RoBERTa [50], and BioBERT, among others. The outcome showed that PubMedBERT outperformed the other models on most tasks. Ablation studies display the positive effects pre-training from scratch had on the model performance.

It is also emphasized that because of pre-training from scratch, the vocabulary of PubMedBERT is better suited for the biomedical domain than models pre-trained on general domain data.

## 2.8 Sentence Transformer Models

This section presents Sentence-BERT (SBERT), a model with a novel siamese architecture for sentence semantics. Furthermore, S-PubMedBERT is presented, which is fine-tuned from PubMedBERT using the siamese framework.

### 2.8.1 Sentence-BERT

BERT and RoBERTa have demonstrated excellent performance on sentence-pair regression tasks, particularly on tasks that require measuring Semantic Textual Similarity (STS) [18], [50]. However, the primary challenge with these models is that they require a pair of sentences as input, which results in significant computational overhead. As a result, the computation time required to find the most similar sentence pair among a set of 10,000 sentences can be prohibitively long, typically around 65 hours. This computational overhead is because BERT has no good inherent semantic representation of individual sentences. The input of BERT for sentence-pair tasks consists of two sentences separated by the [SEP] token, which can be seen in Figure 2.4. Thus, each sentence pair needs to be computed and compared individually. For 10,000 sentences, that is $\sum_{n=1}^{9999} n = n \cdot (n-1)/2 = 49,995,000$ unique sentence pairs to be computed. This limitation renders BERT and RoBERTa unsuitable for tasks that require large-scale clustering [44].

SBERT [44] is a modified version of BERT that incorporates siamese and triplet networks, which drastically reduces the computation time required for semantic similarity tasks while maintaining the high accuracy of BERT. A siamese neural network, also called a twin neural network, consists of two identical sets of neural networks that work together in learning the representation of two input vectors [51]. The two networks cooperate to produce one output vector from the two input vectors, which can be interpreted as the semantic similarity between the input vectors. A triplet network is similar to a siamese network but consists of three input sentences: an anchor $a$, a positive sentence $p$, and a negative sentence $n$. The triplet objective is to minimize the distance between $a$ and $p$ and maximize the distance between $a$ and $n$ [52]. The siamese and triplet networks structure enables SBERT to perform large-scale semantic similarity comparisons, clustering, and information retrieval through semantic search. Therefore, SBERT is a more suitable approach for semantic textual similarity tasks compared to BERT [44].

To fine-tune the pre-trained models, different datasets require different structures. Therefore, both siamese and triplet networks are used to create semantically meaningful embeddings from each sentence. There are three objective functions: the classification and the regression objective functions for siamese networks and the triplet objective function. Figure 2.5 shows the SBERT architecture for siamese networks. In the figure, two sentences are fed into the siamese network structure.

Pooling is an operation that uses different techniques, such as taking the mean of the output vectors, to aggregate several word embeddings into a semantically meaningful sentence embedding. The sentence embeddings are represented as **u** and **v**, which are fed into the objective function.

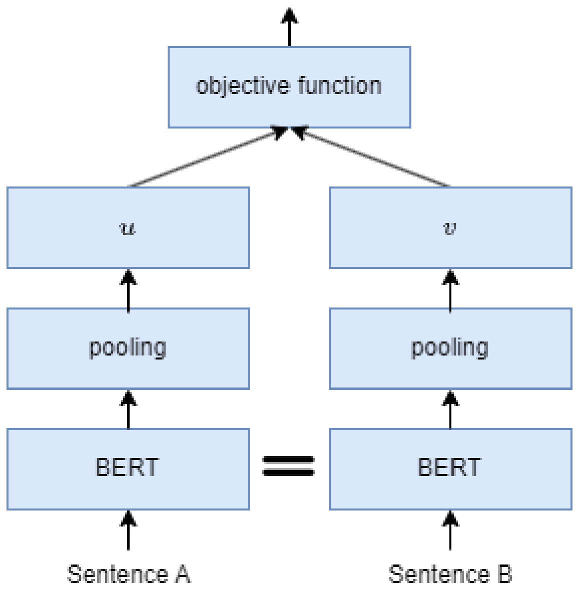


Figure 2.5: The siamese architecture of SBERT, with either classification or regression objective function. This example assumes a pre-trained BERT model to be fine-tuned. The two BERT models have identical weights.

The classification objective function is used for fine-tuning on the Semantic Natural Language Inference (SNLI)[3] dataset. The SNLI dataset contains a set of premise-hypothesis text pairs, with a label indicating whether the hypothesis is neutral, an entailment, or a contradiction to the premise. The classification objective function ($o$) is defined as:

$$o = \sigma(W_t \cdot (\mathbf{u}, \mathbf{v}, |\mathbf{u} - \mathbf{v}|))$$

, where $\sigma$ represents the softmax function, which is:

$$\sigma(z)_i = \frac{\exp^{z_i}}{\sum_{j=1}^{K} \exp^{z_i}}.$$

Hence, the softmax function output vector has the same shape as the input vector, and the sum of the elements of the output vector is equal to 1, and each element of the vector is between 0 and 1. The three vectors that form the softmax function parameters are the sentence embeddings **u** and **v**, and the absolute value of the element-wise difference between them. These three vectors are also multiplied with $W_t$, the trainable weight. Cross-entropy loss is used for optimizing the network.

[3]https://nlp.stanford.edu/projects/snli/

The regression objective function calculates the cosine similarity between **u** and **v**, which outputs a float between -1 and 1. Mean Squared Error (MSE) loss is used for evaluation, where MSE is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where $\hat{y}$ is the predicted cosine similarity and $y$ is the true label similarity. MSE is explained in more detail in Subsection 2.10.4.

The triplet objective function minimizes the distance between the anchor $a$ and the positive $p$ while increasing the distance between $a$ and $n$. The function is

$$max(d(s_a - s_p) - d(s_a - s_n) + \epsilon, 0),$$

where $s_x$ is the sentence embedding of the sentence. In SBERT, experiments are conducted with $\epsilon = 1$, and the distance metric $d$ is the Euclidean distance.

SBERT improved previous sentence embedding methods, as shown by evaluation on a range of sentence semantics tasks. Furthermore, SBERT does so computationally efficiently, with an architecture that can be applied to a range of tasks. Observations and ablation studies showed that using RoBERTa rather than BERT suggested no significant difference. Furthermore, among the pooling layer strategies for mapping a set of word embeddings into a single sentence embedding, the best-performing approach was using the mean pooling strategy. The mean pooling strategy is to compute the mean of all output vectors. This strategy was compared to max pooling and using the special CLS token. Finally, the negative Manhattan distance and the negative Euclidean distance function were studied as similarity measures, with similar results.

### 2.8.2 S-PubMedBert

S-PubMedBERT [9] is a Sentence Transformer model, fine-tuned from PubMed-BERT on the MS-MARCO[4] [53] dataset using the Sentence Transformer framework. MS-MARCO is a collection of datasets on deep learning in search. The training approach follows the bi-encoder architecture, with MSE loss (see Subsection 2.10.4), which seeks to reduce the distance between a query sentence and a positive sample while increasing the distance between the query and a negative sample. The MS-MARCO corpus consists of a 1,000,000-question dataset with questions and human answers, a natural language generation dataset, a passage ranking dataset, a keyphrase extraction dataset, a crawling dataset, and a conversational search dataset. S-PubMedBERT maps sentences and paragraphs to a 768-dimensional dense vector space, and the model can be used for clustering or semantic search. S-PubMedBERT can be used for information retrieval in the medical/health text-domain. One of the key findings in the S-PubMedBERT research was that fine-tuning from biomedical

[4]https://microsoft.github.io/msmarco/

domain models, rather than the generic domain SBERT, yields better performance on biomedical sentence similarity tasks [9].

## 2.9 Text Mining and Extraction Systems

The goal of text mining is to extract relevant information from textual data [54]. Hence, text mining can be viewed as the subset of NLP that concentrates on information extraction. A significant application of text mining in the biomedical domain is in instances of information overload [55], [56], which is when the amount of data exceeds what humans can absorb. Text mining is a broad term and encompasses several sub-tasks, for instance, retrieval of relevant documents; also referred to as text classification, keyphrase extraction, NER, RD and QA. Text classification is the task of finding relevant documents in a set of documents, often related to a query. Keyphrase extraction is similar to text classification, but rather than searching between documents, it is a search within documents for the main topic(s).

NER is the localization and classification of named entities such as a person, a place, or an organization in natural language. An example of NER is given in Listing 2.3. The example is a quote from an article by Yoshida et al. [57]. In the example, the output is an annotated version of the input, and the entities are categorized into predefined classes: genes, diseases, and organisms. The output shows the identification and categorization of a gene named Fbxl8, the diseases lymphoma and tumor, and the human organism.

Listing 2.3: NER Example

[Fbxl8]$_{gene}$ in [lymphomas]$_{disease}$ from [human]$_{organism}$
patients implicating [Fbxl8 ]$_{gene}$ functions as a
[tumor]$_{disease}$ suppressor.

NER forms the foundation of information extraction and RD [58]. The RD task is to find relations between entities located by NER, given the natural language input. In Listing 2.3, it is stated that “Fbxl8 functions as a tumor suppressor”, and a relation found by RD could be ”suppresses” between the entities Fbxl8 and tumor. RD has traditionally been implemented pipelined, where named entities are extracted first before relations are derived between these entities. However, recent research shows that using a shared network for these two tasks may be beneficial [59].

Given a question and a corpus of texts, a QA system produces an answer to the question. For instance, given the text from Listing 2.3 as a corpus and the question ”What gene functions as a tumor suppressor?”, it would be expected of the model to output ”Fbxl8”.

### 2.9.1 Keyphrase Extraction

Keyphrase extraction is the act of extracting the phrases of a document that concerns the central ideas or main topics of the article [60]. Keyphrases are typically defined as essential and meaningful terms that capture the essence of the document and can be used to summarize its main topics or themes. An efficient keyphrase extraction model allows for searching for topics in a vast amount of articles and searching for the main subject within a single article.

The keyphrase extraction process typically includes several steps: text preprocessing, candidate keyphrase generation, keyphrase ranking, and evaluation. First, text preprocessing is necessary to clean the raw text and transform it into a structured format where it can more easily be analyzed. Next, candidate keyphrases are found using some heuristic rule, where these rules are selected to avoid incorrect sentences and narrow down the search space for the ranking method [60]. Finally, in keyphrase ranking, an algorithm is developed to rank the candidate keyphrases such that the most likely keyphrases are ranked higher than the candidates that are unlikely keyphrases.

There exist several methods for keyphrase extraction. These can be categorized into supervised and unsupervised approaches. Unsupervised methods are popular for being domain-independent and not requiring annotated training data. On the other hand, supervised methods typically achieve higher performance than unsupervised methods when annotated data is available [61].

The keyphrase extraction problem can be viewed as a binary classification problem: whether a candidate sentence is a keyphrase or not. Traditional supervised methods include Naïve Bayes and Support Vector Machines [60], [61]. Although deep learning architectures such as CNN, RNN and LSTM improved upon the traditional approaches, the current most effective model architecture for keyphrase extraction is pre-trained Sentence Transformers [62].

Regarding challenges of keyphrase extraction, some corpora factors that affect the difficulty of keyphrase extraction are document length, structural consistency, topic change, and topic correlation [60]. The document length affects the difficulty, as with an increase in document length, there is an increase in candidate keyphrases. Structural consistency also influences the difficulty. If all documents in the set of all documents studied have a keyphrase in one particular section, all other sections can be disregarded. Hence, the search space is narrowed. Regarding scientific papers, keyphrases will typically appear in the abstract, introduction, or conclusion. Topic changes are related to the structural consistency of a text; assuming scientific texts are structured with an abstract and introduction at first and a conclusion at the end, the keyphrases will often appear at the beginning and end of the text. Finally, if there is a topic correlation, it is assumed that the keyphrases of a text are related. This typically holds in scientific articles but not in informal texts.

Finally, another challenge of keyphrase extraction concerns evaluation errors. Evaluation error is the term for errors that occur when a system fails to recognize semantically equivalent phrases [60]. Evaluation errors appear when a model correctly identifies a keyphrase, but upon testing the output, the system fails to recognize that the proposed keyphrase and the gold keyphrase are semantically equivalent. Hence, these errors result from a flawed evaluation, not a flawed model. Possible solutions to evaluation errors are to carry out manual evaluation or to identify all semantically equivalent phrases and use these to build an automatic evaluation system [60].

## 2.10 Metrics

IR evaluation is a complex task that requires specific metrics. The scores used for evaluation are often calculated between a candidate (generated translation or prediction) and one or multiple references (correct translations made from, for example, a translator). These metrics are based on a supervised approach.

### 2.10.1 Accuracy, Precision, Recall, F1-score

Consider binary classification. Given training data in the form $\{\langle x_1, y_1\rangle, ..., \langle x_n, y_n\rangle\}$ where $x_i$ is the vector of predicted labels and $y_i$ is a vector of true labels [63].

| | Actual Positive | Actual Negative |
|---|---|---|
| Predicted Positive | $TP$ | $FP$ |
| Predicted Negative | $FN$ | $TN$ |

Table 2.1: General structure of a confusion matrix.

The counts of $TP, TN, FP$ and $FN$ are presented in a confusion matrix as in Table 2.1 [63]. The accuracy is defined as [64]:

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP},$$

which is the fraction of all correctly predicted out of all predictions [64]. The precision, on the other hand, is defined as [63]:

$$Precision = \frac{TP}{TP + FP}.$$

Hence, precision is the fraction of all predicted positives that are true positives [63]. Recall is defined as [63]:

$$Recall = \frac{TP}{TP + FN}.$$

Recall is the fraction of the actual positives that are predicted as being positive [63]. The F1-score is the harmonic mean of precision and recall and is defined as [63]:

$$F1 = \frac{2}{(1/recall + 1/precision)} = \frac{2TP}{2TP + FP + FN}.$$

### 2.10.2 Average Precision and R-precision

An issue when evaluating information retrieval systems is that the number of correct outputs might vary between documents. Hence, a static prediction of the same number of outputs for each article might not reflect the performance of the retrieval system to a satisfactory degree. A proposed solution is to use the R-precision metric. As can be deduced from its name, R-precision is similar to precision in that it evaluates the true positives divided by the total number of positive predictions. $R$ is a dynamic variable set based on the number of gold-labeled true positives for the evaluated article, i.e., a cut-off for the calculation. Let $r$ denote the correct predictions among the cut-off $R$. Then, the R-precision is defined as $r/R$. For instance, for an article with five annotated key sentences, such that $R = 5$, and a retrieval system that correctly ranks two key sentences among the top 5 ($r = 2$), the R-precision is $2/5 = 0.40$. Accordingly, if a system ranks all keyphrases above the non-keyphrases, it will achieve a perfect R-precision score [60].

Average R-Precision (ARP) is defined as the arithmetic mean of the R-precision over $n$ queries [65]:

$$ARP = \frac{1}{n}\sum_{i=1}^{n} RP_i \ ,$$

where $RP_i$ is the R-precision of article i. As a concrete example, if $n = 2$, $R_1 = 3$ and $r_1 = 2$ for one article, and $R_2 = 6$ and $r_2 = 3$ for the other article, then the ARP calculation is:

$$\frac{\frac{2}{3} + \frac{3}{6}}{2} = \frac{\frac{7}{6}}{2} \approx 0.583 \ .$$

ARP and R-precision are two of the most cited metrics of retrieval performance [66].

### 2.10.3 Pearson Correlation

The Pearson Correlation Coefficient measures the correlation between two vectors, $a$ and $b$, and outputs a value between $-1$ and 1. An output of 0 implies no correlation, and an exact output of $-1$ or $+1$ implies a perfect linear relationship. Values close to $-1$ entail a negative correlation, i.e., as $a$ increases, $b$ decreases. Furthermore, values close to $+1$ imply a positive correlation; as $a$ increases, so does $b$. Accordingly, it is a common method for evaluating semantic textual similarity tasks. Given two zero-mean random variables $a$ and $b$ [67]. The PCC is defined as:

$$p(a,b) = \frac{E(ab)}{\sigma_a \sigma_b},$$

where $E(ab)$ is the cross-correlation between $a$ and $b$ and $\sigma_a^2 = E(a^2)$ and $\sigma_b^2 = E(b^2)$ are the variances of $a$ and $b$ respectively [67].

### 2.10.4 Mean Squared Error

MSE measures the dissimilarity between vector pairs. Given two vectors, $\mathbf{x} = \{x_i | i = 1, 2, ..., N\}$ and $\mathbf{y} = \{y_i | i = 1, 2, ..., N\}$, where $N$ is the number of samples and $x_i$ and $y_i$ are values in the $i$-th sample in $x$ and $y$ [68], MSE is defined as:

$$\mathbf{MSE}\ (\mathbf{x},\ \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2.$$

The error is the difference between the actual values and the predicted values [68]. This score is often used to measure how much vectors differ, and the goal is to minimize the MSE [68]. In addition, in the context of machine learning, it is often utilized as a loss function for regression problems.

### 2.10.5 T-SNE

T-Distributed Stochastic Neighbor Embedding (T-SNE) is a method for dimensionality reduction, which transforms a high-dimensional dataset $X = \{x_1, x_2, ..., x_n\}$ into a two or three-dimensional representation $Y = \{y_1, y_2, ..., y_n\}$, which can be visualized in a scatterplot. The reduced representation $Y$ can be described as a map and the individual data points' low-dimensional representations $y_i$ as map points. The objective of dimensionality reduction is to retain as much meaningful structure from the original high-dimensional data as possible in the lower-dimensional map [69].

T-SNE is a technique used to explore high-dimensional data. T-SNE is able to visualize the similarity of data [69]. It can create two-dimensional maps with hundreds or thousands of dimensions [70].

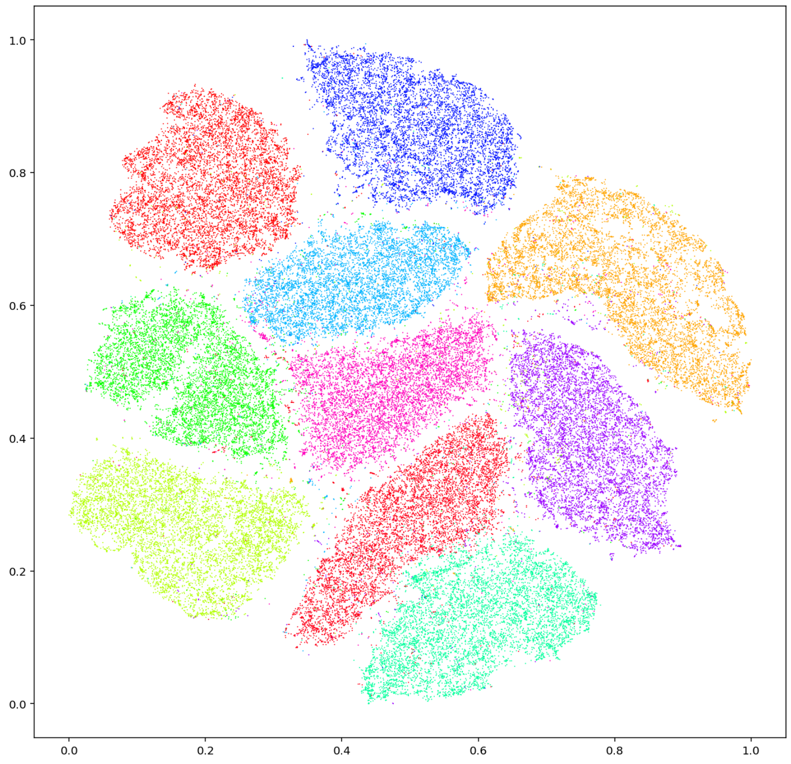


Figure 2.6: A T-SNE representation of a model trained to classify images of the numbers 0 to 9. Reprinted from [71].

The goal of T-SNE is to take points in high-dimensional space and find a representation of these points in a lower-dimensional space. The algorithm behind this is non-linear, and it adapts to the underlying data, which makes it transform the data in different ways depending on the region of the data. This can be a source of confusion. Besides, the technique doesn't always produce similar outputs in successive runs. Additionally, different hyper-parameters can produce different results, which makes T-SNE plots challenging to interpret. Nevertheless, the T-SNE plots can be used in simple cases to get an idea of what is going on [70].

## 2.11 Related Work

Advanced Neural Biomedical Entity Recognition and Normalization (BERN2) was presented in 2022 [72]. BERN2 is a NER model developed for the biomedical domain, with logic to recognize nine different entity classes (gene/protein, disease, drug/chemical, species, mutation, cell line, cell type, DNA, and RNA). The NER operation uses Bio-LM as a pre-trained model and fine-tunes for the specific downstream tasks [73]. After performing NER, BERN2 uses decision rules to correctly label those entities that overlap to handle the polysemy problem, and finally, a hybrid approach to Named Entity Normalization (NEN) is applied to normalize the entities and thus handle the synonym problem. The polysemy problem is the issue of deciding the contextual meaning of a word written similarly. For instance, apple might refer to the fruit or the company Apple. The synonym problem is, in some sense, the opposite of the polysemy problem, where one entity might have several syntactic representations [74].

Due to their self-attention mechanism, where each token attends to all other tokens in a sequence, computing long sequences with Transformer-based models is inefficient [41]. Transformer-based models scale quadratically with the input sequence length. Longformers present a technique for attention that can replace the self-attention mechanism of Transformers, which makes the longformer scale linearly with sequence length. This is done by introducing a local window of attention in combination with global attention. The longformer approach outperforms Transformer-based approaches on long-document tasks [41].

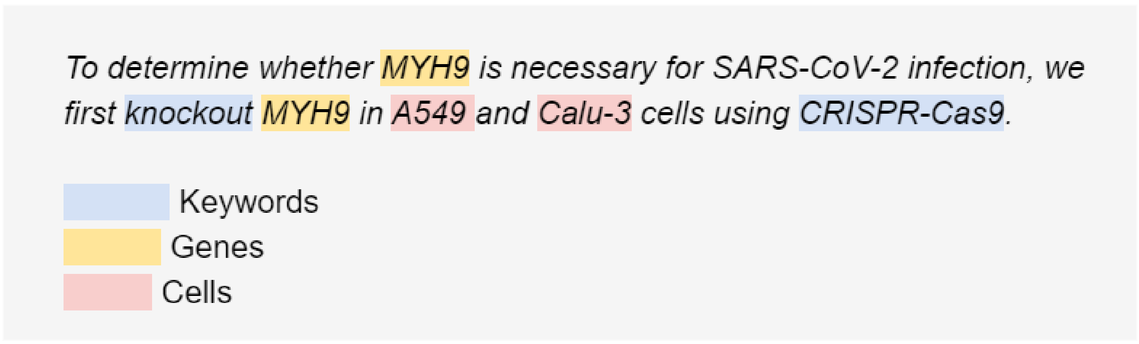
Augmented SBERT (AugSBERT) is a technique for improving bi-encoders for pairwise sentence scoring tasks [75]. AugSBERT combines bi-encoders' speed with the performance advantage by utilizing the full self-attention of cross-encoders. First, the cross-encoder is used to label input pairs, i.e., augment the training data, which the bi-encoder uses in a later stage. The first dataset, which forms the input to the cross-encoder, consists of a gold-labeled dataset and a set of unlabeled data. Next, the cross-encoder is fine-tuned on the gold dataset before it labels the unlabeled data to produce a silver dataset. Finally, the computationally efficient bi-encoder is fine-tuned on the silver and gold dataset [75].

# 3

# Methods

Firstly, the initial dataset is presented, conveying a basic understanding of the structure and overall relationships of the content of the dataset. Secondly, the curation of a new dataset is explained, which was made to contain features aligned with the fine-tuning approach. Thirdly, the development of the key sentence extraction model is explained, with hardware setup, model architecture and hyperparameters, and the experimental setup. Finally, the evaluation approach of the model is clarified and motivated.

Before diving into the method, the definition of a key sentence in the context of this project is presented. The hypothesis, as proposed by the annotators of the dataset, is that most articles contain one or multiple sentences that summarize the main content of the article. This content can be divided into three main categories, a set of genes, a set of cells, and a set of keywords or key phrases. These keywords are CRISPR- and KO-related terms such as *"CRISPR", "knockout", "edit", "KO", "mutat", "remove", "edit", "disrupt" and "silence"*. An example of a key sentence is shown in Figure 3.1.



*To determine whether MYH9 is necessary for SARS-CoV-2 infection, we first knockout MYH9 in A549 and Calu-3 cells using CRISPR-Cas9.*

🟦 Keywords
🟨 Genes
🟥 Cells

Figure 3.1: Example of a key sentence. MYH9 is a gene, A549 and Calu-3 are cell lines, and CRISPR-Cas9 and knockout are key terms.

## 3.1 Initial Dataset

The dataset consists of 2,774 manually annotated articles. The most important content of the dataset is article IDs and the full text of each article, relevant gene IDs, and cell IDs, along with associated names and synonyms for these genes and cells. The genes can be either KO, KI, or KD genes, depending on the research conducted in the article. The difference between KO, KI, and KD was explained in Section 2.1. As can be seen in Table 3.1, 97% of the articles only mention KO genes. As a result, the distribution of the various gene types within the dataset is disproportionate, leading to a scenario where only an insightful examination of articles containing KO genes could be carried out.

| Genes | Articles |
|---|---|
| Knockout | 2,699 |
| Knockin | 74 |
| Knockdown | 1 |

Table 3.1: The number of articles that mention at least one of the different gene types.

The gene-cell relationships form the foundation of our problem. They are most often a one-to-one relationship for each article. However, they do appear in one-to-many, many-to-one, and many-to-many relationships. Table 3.2 shows the number of articles by their relationships between KO genes and cells. All annotations are made on a document level, i.e., no sentence or word within the articles has been annotated. As a consequence, the relevant knockout cell/gene IDs and cell/gene names are for the entire article.

| Genes per article | Cells per article | Number of articles | Ratio |
|---|---|---|---|
| one | one | 1,898 | 70.3 % |
| one | many | 309 | 11.4 % |
| many | one | 412 | 15.3 % |
| many | many | 80 | 3 % |
| sum | | 2,699 | 100 % |

Table 3.2: The number of different relationship configurations between genes and cells for each article.

The fact that each article might have a different number of genes and/or cells associated with it introduces a challenge when creating a NLP model. The challenge is that the model needs to be able to produce variable length output, given an entity extraction approach. For some articles, the desired output is a simple gene-cell pair. For others, the desired output might be two different genes and two different cells. As a successful extraction needs to extract *all* genes and cells from an article, the implementation of an entity extraction approach is complicated with variable length outputs. Therefore, the extraction of key sentences, where a key sentence contains all relevant information, was the chosen approach. By finding the key sentences of an article, identification of the relevant entities becomes an easier task. To extract key sentences, the chosen approach was to fine-tune a Sentence Transformer model, which required the curation of a new dataset.

## 3.2 Dataset Curation

The curation of a new dataset with annotated key sentences was necessary to fine-tune a model for the task of key sentence extraction in a supervised fashion. The new dataset was created from the initial dataset, where the initial dataset consists of 2,699 articles, each with annotated genes, cells, and the full text of each article. The dataset curation consisted of three main steps. The first step concerned sentence tokenization and proposing key sentences. Secondly, these proposed key sentences were sent to domain experts for manual annotation. Lastly, some final data cleaning was necessary to use the data for fine-tuning and evaluation. Figure 3.2 displays an overview of the steps in creating the new dataset from the initial dataset.

The first step of this process consisted of generating a set of candidate key sentences from a Sentence Transformer model. These candidate key sentences were used as hints for domain experts doing manual annotation of the data. Firstly, text cleanup and tokenization needed to be performed. Some of the articles contained HTML tags and LaTeX code. The texts were cleaned using the Python library Beautiful Soup 4 [76] and regular expressions. After cleaning the texts, they were tokenized into sentences using the Natural Language ToolKit (NLTK)[77]. NLTK is a Python package for natural language processing. Each text was tokenized using the code:

```
gen = nltk.PunktSentenceTokenizer().span_tokenize(text)
spans = [s[1]-1 for s in gen].
```

The method *span_tokenize()* returns a generator that is used to find *spans*, which are indices at which each sentence starts and ends. These indices were later used to split the sentences, such that each article was formatted as a list of sentences. Sentence tokenization experiments were also conducted with the NLP framework spaCy [78]; however, without observed significant improvement over NLTK. Therefore, NLTK was chosen as it is more computationally efficient than the implementation of spaCy.

To generate proposed key sentences for manual annotation, a query sentence with CRISPR-related terms was embedded by a Sentence Transformer model. The embedding was compared with each sentence in the dataset before returning the top five sentences for each article. The idea was that the key sentences should be among these sentences, and it would be more effective for manual annotators to go through these suggested sentences than the whole article. The query sentence used was

**"[gene name] knockout from [cell name]"**,

where *[gene name]* and *[cell name]* were the gene name(s) and cell name(s) mentioned as relevant knockout gene(s) and cell(s) in the specific article. For instance,

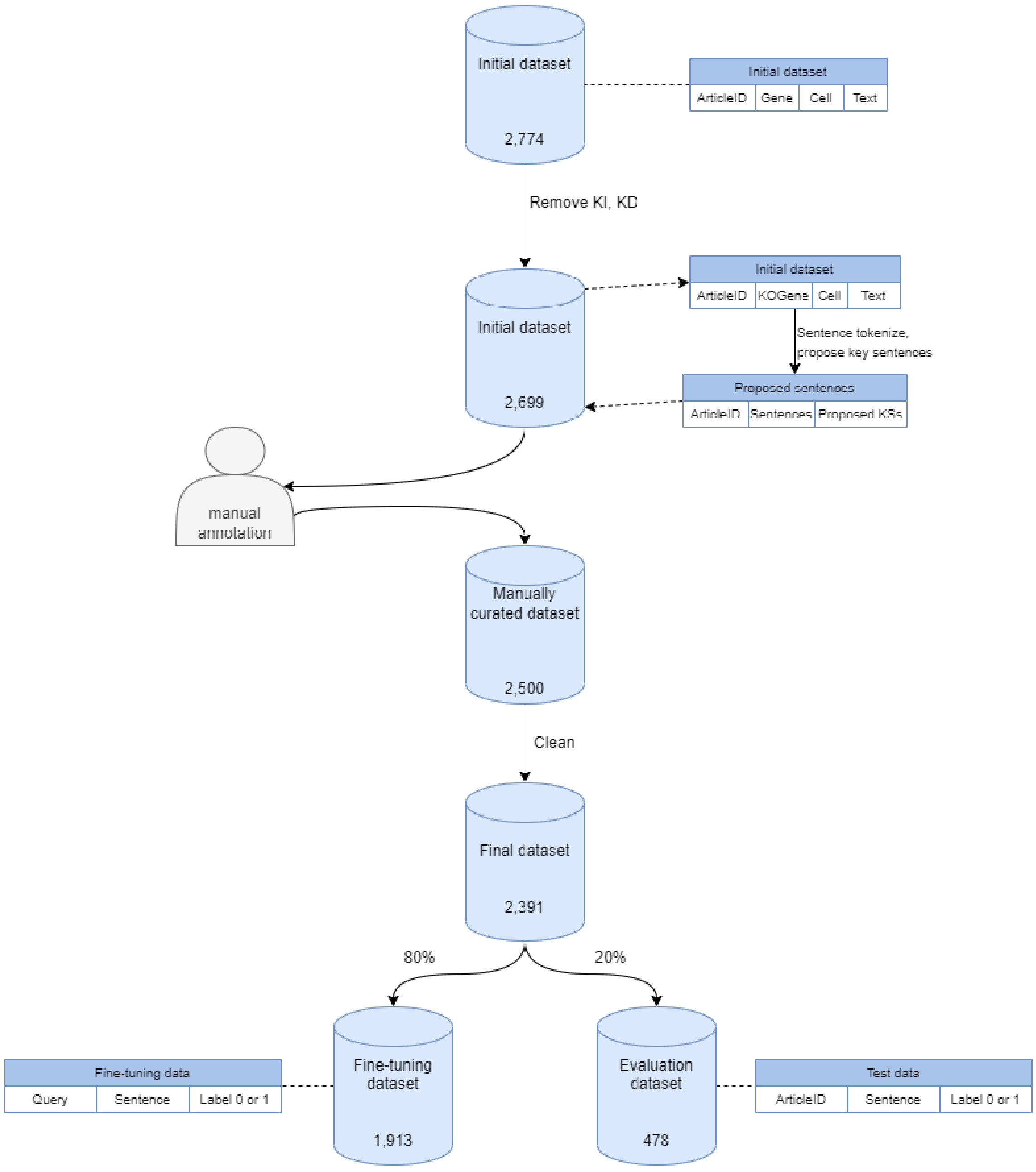


Figure 3.2: Dataset curation for key sentence extraction. The numbers within each cylinder indicate the total number of articles in the datasets.

the query sentence from the example key sentence in Figure 3.1 would be "MYH9 knockout from A549 Calu-3". The embedding was done by BioBERT-NLI [1], a Sentence Transformer model fine-tuned on the SNLI [79] and Multi-Genre Natural Language Inference (MultiNLI)[80] datasets from BioBERT. The top five sentences of each article were selected by returning the sentences with the highest cosine similarity score in relation to the query sentence. In addition to BioBERT-NLI, proposed key sentence extraction from a general domain Sentence Transformer model, all-MiniLM-L6-v2[2], and S-PubMedBERT (see Subsection 2.8.2), were also conducted. Observations suggested that propositions from S-PubMedBERT and BioBERT-NLI were the most accurate. As no factual evaluation was available at this stage, and no clear difference between BioBERT-NLI and S-PubMedBERT were observed, the proposed sentences by BioBERT-NLI were used. Furthermore, a simple rule-based algorithm doing a term frequency search over keywords, cell names, and gene names was also developed. However, observation indicated worse performance than the Transformer-based approaches.

In addition to the top five sentences of each article, the cosine similarity scores and the section at which each sentence was found were sent to annotators to speed up the annotation process. These proposed key sentences were used by the annotators to select the correct key sentence(s) for each article. Therefore, only a subset of the sentences of each article was considered upon annotation, and not all true key sentences were labeled as such. Throughout the annotation process, some articles were discarded due to inadequacies in the texts, either because the articles were too short or because the text cleaning was insufficient. Accordingly, 2,500 articles were annotated.

Upon completion of manual annotation, some more cleaning was necessary, as the formatting of the texts was slightly altered. The set of key sentences in each article should be a subset of the set of sentences in the entire article. However, upon investigation of the dataset, some key sentences could not be found in the set of sentences. By stripping spaces in the comparison, it was possible to extract 2,391 articles in which all key sentences were found. For the remaining articles, no programmatic approach was found to uncover the key sentences, so they were dropped.

The final dataset contained 699,277 sentences, of which 4,494 were key sentences. In other words, there were 6.43‰ key sentences per sentence and an average of 1.88 key sentences per article. There were between 1-5 key sentences per article, as seen in Table 3.3. There were an average of roughly 292 sentences per article, but the number ranged from 37 to 786 sentences per article. The data was split into a fine-tuning dataset and an evaluation dataset, containing 1,913 and 478 articles, respectively.

### 3.2.1 Preprocessing

In order to train, validate and test the model on the manually annotated data, some formatting of the dataset was required. This data consists of sentence pairs with

[1]https://huggingface.co/gsarti/biobert-nli
[2]sentence-transformers/all-MiniLM-L6-v2

| Number of key sentences | Number of articles |
|---|---|
| 1 | 918 |
| 2 | 1,002 |
| 3-5 | 417 |

Table 3.3: Number of articles with a specific number of key sentences.

a related label that indicates whether they belong in the same class (label 1) or if they are in contrasting classes (label 0). One sentence in the sentence-pair is always a pre-defined query, referred to as the query sentence. By introducing the query sentence, the goal was to have an anchor, similar to the approach in the triplet objective function of SBERT (see Subsection 2.8.1), where key sentences would be closer to the query sentence than non-key sentences. The query sentence could be represented as a natural language sentence or by introducing a fresh token to the model vocabulary. By using natural language sentences that are semantically similar to the general structure of key sentences, the assumption was to provide the model with a head start in training, as the model would early succeed in differentiating between key sentences and non-key sentences. Another reason for this approach is that it simplifies the evaluation of regression models because all sentences can be evaluated against the same query sentence.

A training input to the model has the following format if the sentence is an annotated key sentence label is 1, and the query sentence is "Gene knockout from cell", which is a sentence that is semantically close to the key sentence.

{"Gene knockout from cell", "To determine whether MYH9 is necessary for SARS-CoV-2 infection, we first knockout MYH9 in A549 and Calu-3 cells using CRISPR-Cas9.", label: 1}.

Otherwise, if the sentence is a non-key sentence, the training input has the following format, with label = 0

{"Gene knockout from cell", "The regulation of protein synthesis is essential for maintaining cellular homeostasis, especially during stress responses, and its dysregulation could underlie the development of human diseases.", label: 0}.

The studied query sentences can be categorized into two kinds, the introduction of a fresh token and natural text sentences that are constructed to resemble the content of key sentences. The fresh token introduction is performed by adding a new word, written "[QUERY]", to the vocabulary of the model and either initializing the weights to a random vector or initializing the weights to zeros. The intention of the fresh token is that the initial weights of the query sentence are less biased toward semantics that the model might carry from its pre-training. However, this semantic understanding might be beneficial, which is why experiments with natural text queries are also investigated. The natural text queries, developed in cooperation with domain experts, are:

1. knock mutat silence disrupt crispr lack cell gene remove edit
2. Gene knockout from cell
3. Gene knockout from cell using CRISPR

*Sentence 1* is a list of frequent words found in key sentences by domain experts.

*Sentence 2 and 3* follows a structure closer to that of human language. It relies on the model to infer that the term "gene" should be a specific gene, that the "knockout" term might be a semantically close word in actual key sentences, and that the term "cell" should be a specific cell.

## 3.3 Fine-tuning

The performance of base models is sought to be enhanced through fine-tuning. The overall goal of fine-tuning is for the model to learn to distinguish between sentence embeddings such that a distance metric can be applied to differentiate between sentence classes. The fine-tuning objective is visualized in a 2-dimensional space in Figure 3.3. Each point in the figure represents a sentence embedding, which in reality, is a 784-dimensional vector. To effectively identify key sentences, a transfer learning approach was employed for the model training process. This decision was prompted by the limited size of the available data and the potential benefits of leveraging certain general aspects from other models. The most important studied model domains were general domain sentence semantics and understanding in the biomedical domain. The architecture of the training is presented in Figure 3.4, which closely resembles the SBERT regression objective function. The implementation of the training pipeline is based on the Python framework SentenceTransformers[3], which is based on SBERT [44] (see Subsection 2.8.1).

Differentiating between key sentences and non-key sentences could be solved as a binary classification task. However, annotators have expressed a desire to receive information about the model confidence in each prediction. Accordingly, the problem has been modeled as a regression task, where outputs are floats in a range between 0 and 1. Outputs close to 1 indicate high similarity between the input sentence pair, and outputs close to 0 indicate dissimilar input sentences. Despite outputting a floating point number, the problem is still a question of classification, whether a sentence is a key sentence or not. Classification can easily be performed after regression by comparing the regression scores and ranking the pairwise similarity between an anchor and the sentences of an article. The top $n$ sentences in an article can be labeled as key sentences. Otherwise, a cut-off threshold can be set, where scores above the threshold are labeled as key sentences. In most articles, a single key sentence contains most of the necessary information; if correct, it is sufficient to return one key sentence per article. However, it might be discovered at a later point that it is more beneficial to return several sentences for an article. Regardless, by treating the problem as a regression task, the model will be more flexible upon deployment, as it will be easy to change how many sentences to return.

[3]https://www.sbert.net/

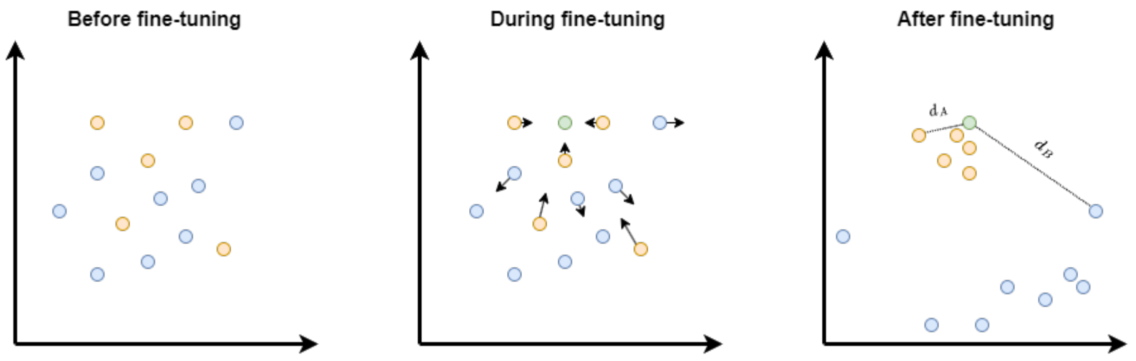


Figure 3.3: A 2-dimensional visualization of the fine-tuning objective. The goal is to cluster the orange class such that the distance to the anchor (green) is lower than the distance to the blue class ($d_A < d_B$).

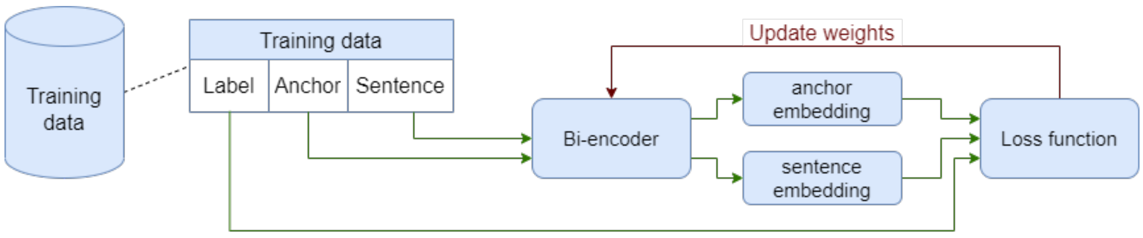


Figure 3.4: The fine-tuning architecture. The *Anchor* is the same for each fine-tuning experiment, and *Sentence* is the list of all sentences in the dataset.

The first step in fine-tuning is to split the fine-tuning dataset into a training- and validation set. The training set is used to update the weights of the Transformer model, whereas the validation set is used for the evaluation of model performance throughout the training. The training data consists of 75% of the fine-tuning data, and the validation data consists of 25%. Hence, including the test data, the total split is 60%, 20%, and 20% for train, validation, and test, respectively.

A word embedding model was downloaded as the initial checkpoint before applying a pooling layer to the word embedding model to form a Sentence Transformer. Some informal experiments were conducted on different pooling strategies, which used the special [CLS] token to represent the sentence, max-pooling, or mean-pooling. The implementation of max-pooling returns the maximum element in each dimension over all tokens, and the implementation of mean-pooling takes the element-wise mean of each token. These experiments showed, similarly to what was concluded by Reimers and Gurevych in the paper on SBERT [44], that mean-pooling slightly outperforms other approaches. Consequently, the formal approaches presented in Chapter 4 only use the mean-pooling strategy.

After sentence embeddings are calculated, both embeddings are fed into the loss function in order to optimize the weights of the network. One of the chosen loss functions was contrastive loss. Contrastive loss was introduced by Hadsell et al. (2006), which serves as an approach to differentiate distances between high-dimensional vectors of samples belonging to the same class, and samples from different classes [81]. The experiments conducted with the loss function consist of image classification tasks using a siamese network architecture. Despite originally being used for computer vision tasks, it has recently gained a reputation in NLP, with promising results in pairwise sentence scoring, regression, and ranking [82]. The loss function presented by Hadsell et al. is:

$$\mathcal{L}(W) = \sum_{i=1}^{P} (1-Y) L_S(D_W^i) + Y L_D(D_W^i).$$

$P$ is the number of training pairs, $Y$ is the label, which is either 0 or 1, and $D_W^i$ is the distance function between the input vectors. $L_S$ and $L_D$ are defined as the partial loss functions of similar and dissimilar vectors, respectively.

The implementation in this project defines $D_W$ as $D_W^i(u_i, v_i) = 1 - cos\_sim(u_i, v_i)$, i.e., the cosine distance between the embeddings of the query sentence and the sentence sample. Furthermore, $L_S(D_W) = 0.5(D_W)^2$ is the partial loss function for similar vectors and $L_D(D_W) = 0.5(max(0, m - D_W))^2$ for dissimilar vectors, where $m$ is the margin. Due to the margin, dissimilar vector pairs only contribute to the loss if they are sufficiently dissimilar. If $D_W > m$, then $L_D(D_W) = 0$; hence, $m$ can be thought of as a threshold, where if the distance between dissimilar vectors exceeds the threshold, loss is not applied. $m$ was set to 0.5 as the threshold in all experiments in this project. Figure 3.5 illustrates the effect of the margin parameter of the loss function. The anchor is pairwise compared to all other vectors. In this project, the anchor will correspond to the query sentence, dissimilar vectors are non-key sentences, and similar vectors are key sentences.

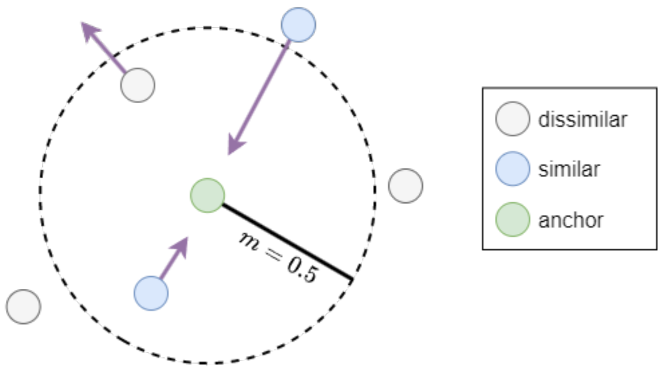


Figure 3.5: The effect of the *margin* on the contrastive loss function. The effect of the loss function is depicted as purple arrows. No loss is applied to dissimilar vector pairs outside the radius $m$.

A final remark on the contrastive loss function is that the implementation vectorizes the input of each batch. The computation is not done sequentially for each input pair. Furthermore, the loss is returned as the average of the losses in each sample for each batch, not the sum.

The cosine similarity loss function was also used in the experiments for comparison with contrastive loss. The principle is similar to contrastive loss, where similar sentences should be pulled closer to each other in the vector space, and dissimilar sentences be pushed further apart. Cosine similarity loss takes two sentences and a floating point label as input. In essence, the cosine similarity loss calculates the MSE between the cosine similarity of the input sentences and the label. As previously mentioned, in this project, one sentence is always the anchor, and the other is either a non-key sentence or a key sentence. The model computes the cosine similarity between the input sentences, and the loss function minimizes the following:

$$||label - cos\_sim(\mathbf{u}, \mathbf{v})||_2,$$

where the label can be between 1 and 0. $\mathbf{u}$ and $\mathbf{v}$ are the embeddings of the input sentences. The predicted cosine similarities are compared against the labels of floating point numbers, and the loss is minimized. For well-predicted data points, the losses are suppressed so that the loss focuses on data points that are harder to classify [83].

The main motivation for evaluation during training is to save the weights of the model at the time in training when the model has the best results on the validation task. By saving the model that performs best on the validation data, the expectation is that this model checkpoint also will perform best on the test data and further down the line upon deployment. Overfitting would result in a performance that is greater for the training data but worse for the test data. In other words, the model would be bad at generalizing to other data than the training data. Additionally, evaluating during training aids in model development. The *BinaryClassificationEvaluator* class of the SentenceTransformer framework is used for evaluation throughout training on the validation dataset. This evaluator requires labels to be either 0 or 1 and calculates accuracy, precision, recall, F1 score, and average precision. To evaluate the similarity, the model weights are frozen before they are used to embed sentence pairs. Then, the pairwise Manhattan distance, Euclidean distance, dot product, and cosine similarity are calculated between each sentence embedding pair. To decide whether a model checkpoint should be saved, the maximum average precision of the four distance metrics is compared to a potential previous maximum. If the previous best checkpoint has a best average precision of 0.7, and the new evaluation iteration returns 0.5, 0.6, 0.75, and 0.4 in average precision using Manhattan distance, Euclidean distance, cosine similarity, and dot product, respectively, the new model weights will be saved, as $max(0.5, 0.6, 0.75, 0.4) = 0.75 > 0.7$. As displayed in Figure A.1 in Appendix A, all distance metrics produce similar scores.

The fine-tuning process employed contrastive learning, which is geared towards efficient representation learning. The core idea behind contrastive learning is to bring together semantically similar sentences within the vector space while pushing apart dissimilar sentences [84].

## 3.4 Evaluation

An overview of the evaluation on the test data is displayed in Figure 3.6. Here, the query is only embedded once. The query is equal to the anchor on which the model was fine-tuned. All sentences of the test dataset are embedded before a pairwise comparison between the query and the sentence embeddings is performed. To compute MSE and PCC, the vector of all cosine similarities is compared to the vector of all labels in the test dataset. Computing ARP is a bit more complicated. Each article has $R$ labels that equal 1, i.e. each article has R annotated key sentences. Therefore, after sorting the cosine similarities in each article and returning the top $R$ similarities, the predictions are compared to the true key sentences such that $r/R$ can be computed.

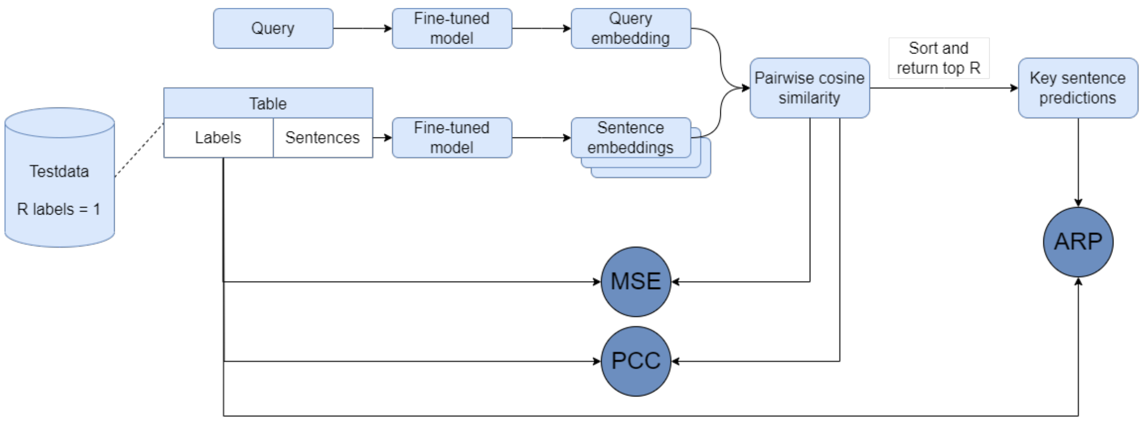


Figure 3.6: Evaluation of the fine-tuned model on the test dataset.

Information retrieval evaluation can be done in several ways. The correlation between the predicted similarities and the labeled similarities between sentences indicates how well the model follows the labels. For measuring how precise the predictions are in relation to the labels, MSE can also be used. However, ranking the sentences within each article and comparing the top $n$ sentences with the labeled key sentences provides more direct insight into the precision of the model. After all, the most important property of the model is its ability to reliably identify key sentences.

Considering there is a varying amount of labeled key sentences for each article, and the true number of key sentences in each article is unknown, the chosen evaluation for classification is the average R-precision metric, where the top $R$ sentences are compared to the gold-labeled key sentences. The motivation for this metric is that it is uncertain how many key sentence predictions are wanted, and a perfect ranking, with all key sentences ranked above all non-key sentences, would result in a perfect ARP score of exactly 1. A related metric is the Precision@K score, which defines a static cut-off $K$ for the entire system instead of the dynamic cut-off $R$. Given a low amount of key sentences for each article, most articles have one or two key sentences. The presumption is that ARP carries more information than Precision@K. Furthermore, by defining a static cut-off, for instance, $K = 1$, only the first sentence in each article will be evaluated, with the result that many key sentences will not be evaluated.

The evaluation can also be treated as a regression- and correlation assessment, mainly to provide insight into the model's confidence in its predictions. This is because the cosine similarity scores output by the model can be interpreted as an indication of how confidently the model separates between key sentences and non-key sentences. Besides, the cosine similarity metric is used by the annotators to decide whether the sentence is a key sentence or not. The evaluation of the regression- and correlation task is done by passing two vectors, the cosine similarity scores and the labels, into the MSE and PCC. The golden labels are still binary, either 0 for dissimilar or 1 for similar.

The definition of the query is done in two separate ways depending on if the model has been fine-tuned or not. For the fine-tuned models, the models have been trained against a query sentence, which represents an anchor. The model has been trained to interpret the semantic meaning of this query as similar to the meaning of a key sentence. Hence, this query is embedded by the model, and its similarity is computed by a distance metric against sentence embeddings in the test dataset, where high similarities are expected when comparing the query sentence with a key sentence, and low similarities are expected between the query sentence and non-key sentences. For the models that have not been fine-tuned, the query sentence comparison approach is not as appropriate since these models have not been trained to interpret the semantic meaning of this query similar to the meaning of a key sentence. Instead, the element-wise arithmetic mean of the key sentences in the training dataset is used, and this average key sentence embedding is pairwise compared to all sentences in the test dataset to produce cosine similarity scores. The fact that the average key sentence embedding works better than a query sentence can also be interpreted as an indication that fine-tuning is necessary, as the pre-trained model does not grasp the semantic relations that are desired.

The relation of key sentences versus non-key sentences in the dataset makes accuracy an insufficient metric for evaluation. As the dataset only contains 6.43‰ key sentences, a model that only predicts non-key sentences would obtain an accuracy of 99.36%. Furthermore, the ability of the model to predict non-key sentences is not very interesting. Consequently, accuracy as a metric does not provide valuable information about the task.

Models can be visualized using T-SNE to show whether the embeddings of key sentences are pushed away from the embeddings of other sentences. Throughout the process of dimension reduction, some information is lost, and T-SNE is valuable more as a tool for getting insight into the dynamics of a model rather than a precise metric. The location of a point in the T-SNE plot carries no information in itself, but the position of points in relation to each other roughly reflects the relations in the true vector space.

## 3.5 Experiments

The experiments are conducted to evaluate the performance of the models and the training process. As presented in Section 3.3, binary evaluation with contrastive loss and cosine similarity loss are generally used in the experiments. A batch size of 32 is used, and the model is evaluated on the validation data every 1,000 steps, or every 32,000 training samples, which results in a total of 14 evaluations when training on the full train dataset, which takes 13,069 steps. The AdamW [85] optimizer is used with a weight decay of 0.01, and a triangular scheduler is used for the learning rate, with a linear increase in the first 100 training steps, which equates to 3,200 training samples, up to a learning rate of $2 \cdot 10^{-5}$, before linearly decreasing throughout the rest of the training. All training was performed over a single epoch.

### 3.5.1 Setup

All model training was conducted on an NVIDIA A40, where fine-tuning the base-sized model with approximately 110 million parameters took approximately one hour on the full dataset, consisting of roughly 410,000 sentence pairs. Evaluation on 478 articles took roughly 3 minutes.

### 3.5.2 Query Sentence

The specific formulation of the query sentence, frequently referred to as the anchor, is studied in the first experiment. The question is whether the initial semantic understanding of the query sentence might affect the performance of the model, even after fine-tuning. Therefore, three different natural language sentences were formulated, along with the introduction of a fresh token. The intention of introducing a fresh token was to try to analyze how a random query sentence, with no predefined semantic meaning to the model, would affect the outcome. Two initializations of the fresh token were investigated, zeroing out the vector and initializing to a random vector. All experiments were conducted five times, and other than the query sentence, all parameters were equal. The training was done from a checkpoint of S-PubMedBERT.

### 3.5.3 Models

Five different base models were compared against each other to evaluate the vocabulary of the models and which base model had the most accurate performance for key sentence extraction. These models were also used as checkpoints, and their performance was compared after fine-tuning.

Different models have different vocabularies, hence, different word encodings, which eventually affect the sentence embeddings. One possible factor that might influence the performance of the models is the vocabulary of the models. The BERT-based models use WordPiece embeddings, and in “Domain-Specific Language Model Pre-training for Biomedical Natural Language Processing”, it is argued that an advantage of domain-specific pre-training is obtaining an in-domain vocabulary [47].

Additionally, it is shown in ablation studies that in-domain vocabularies significantly improve performance for several biomedical downstream tasks [47]. Hence, the vocabulary of relevant models is evaluated against the genes and cells in the training dataset.

The two main abilities that are desired from these checkpoints are proficiency in the biomedical domain and sentence semantics. Experiments with S-PubMedBERT[4] and PubMedBERT[5] are conducted to explore the value of pre-training on sentence semantics versus no specific pre-training on sentences. Furthermore BERT[6], PubMedBERT, BioBERT[7] and SciBERT[8] are compared to provide insight into the effect of biomedical understanding of the models on the key sentence task. All models are base sized and uncased, except BioBERT, as no uncased version was found.

### 3.5.4 Loss Functions

The margin parameter of the contrastive loss function was studied to analyze how adjusting this parameter affects the model's performance. As previously mentioned, all experiments were run five times. As the margin increased, the partial loss of dissimilar sentences became more similar to the cosine similarity loss, which is also used in one experiment. After running experiments with different loss functions, the cosine similarity output of the two best-performing models from the experiments, trained with a margin of 0.5 and cosine similarity loss, was explored. It should be noted that as the margin approaches 1, the contrastive loss becomes similar to the cosine similarity loss. The cosine similarity loss is defined as $\mathcal{L} = ||label - cos\_sim(\mathbf{u}, \mathbf{v})||_2$. If $m = 1$, the partial loss functions of contrastive loss, $L_S$, and $L_D$ (see Section 3.3), becomes equal, and the contrastive loss function is $\mathcal{L} = 0.5(cos\_sim(\mathbf{u}, \mathbf{v}))^2$. The motivation for studying the cosine similarity output was to provide insight into the cosine similarity metric and what precision and recall can be expected for different cosine similarity thresholds. Particularly, this study might be beneficial if the model is deployed in a document retrieval setting, where a possible approach is to input unseen texts to the model and return documents that contain sentences with a cosine similarity score above a certain threshold. In this setting, a trade-off between precision and recall needs to be made because one does not want to return too many documents that are not related to the CRISPR-domain while detecting as many documents as possible that are actually related to the CRISPR-domain.

[4]https://huggingface.co/pritamdeka/S-PubMedBert-MS-MARCO
[5]https://huggingface.co/microsoft/BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext
[6]https://huggingface.co/bert-base-uncased
[7]https://huggingface.co/dmis-lab/biobert-base-cased-v1.2
[8]https://huggingface.co/allenai/scibert_scivocab_uncased

# 4 Results

The first experiment presented in this chapter thoroughly evaluates the test dataset, presenting the predictions of a fine-tuned model on the test dataset compared to a manual annotation of the model predictions. A review of the model's confidence in relation to its correctness is also presented. In terms of fine-tuning, an investigation of the effects of using different query sentences and fine-tuning from various model checkpoints is given. The model checkpoints are compared before and after fine-tuning, and their vocabulary is evaluated on gene and cell names in the dataset.

The training is a stochastic process. Hence results are presented as an average of five runs, along with Confidence Interval (CI) with a confidence level of 95% to show the variance between different runs. Averaging several runs yields more conclusive results of the effect of varying training setups. A CI of 95% is the same as ± two standard deviations, where the standard deviation of a population is defined as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$

where $N$ is the population size, $x$ is the vector of samples, and $\bar{x}$ is the population mean.

## 4.1 KS-PubMedBERT

As there is a varying amount of annotated key sentences for each document, the evaluation metrics are ARP, along with MSE and PCC. The test dataset consists of a total of 141,693 sentences, where 886 are labeled as key sentences (positives). In terms of labeling, which was explained in Section 3.1, only a subset of the actual key sentences has been labeled as such. In other words, an unknown number of sentences in both the test and train datasets have not been labeled as key sentences that are semantically equivalent to a key sentence. The most studied model, which was developed using a list of general keywords ("knock mutat silence disrupt crispr lack cell gene remove edit") as a query from S-PubMedBERT, achieves an ARP of 0.81 on the test data. This model will be referred to as *KS-PubMedBERT*. However, upon a manual review by domain experts of the test dataset compared to the predicted key sentences of the model, the true performance of the model is higher than indicated by the evaluations on the test data.

The test dataset consists of 478 articles, each with a varying amount of key sentences. Denoting the number of key sentences article $i$ contains as $N_i$, the model is asked to predict $N_i$ key sentences for each article. In this setup, *KS-PubMedBERT* correctly predicts 709/886 key sentences, and 177 predictions are labeled as wrong. However, a manual examination of the 177 predictions labeled as being wrong concluded that 115 were semantically equivalent to a key sentence, 11 were inconclusive, and 51 were definitely false positives. In percentages, 65% of the predictions labeled as wrong are actually correct, 6% were inconclusive, and 29% were still wrong. Hence, by appending the 115 true predictions to the old test data and running a new evaluation, *KS-PubMedBERT* obtains an ARP of

$$\mathbf{90.4}\%$$

after the manual evaluation. Examples from the manual annotation are shown in Table 4.1.

| | | |
|---|---|---|
| Equivalent | annotation | a CRISPR/Cas9 gene editing technology was employed to generate COPA deficient THP-1 cell lines. |
| | prediction | Taking advantage of this proposed functional distinction, we deleted COPG1 and COPG2 isoforms individually in THP-1 WT cells using the CRISPR/Cas9 approach. |
| Inconclusive | annotation | To elucidate the role of USP3 in embryonic stem cells, we generated single-cell-derived USP3 knockout clones in a human embryonic carcinoma cell line (NCCIT). |
| | prediction | FOXD1 was knocked out in the 786-O cell line and known targets were analyzed. |
| Wrong | annotation | In this study, we performed a loss-of-function study of USP3 in ESCs utilizing the CRISPR/-Cas9 system. |
| | prediction | CRISPR/Cas9-mediated knockout cells were generated by the Synthego Corporation (Cambridge, MA, USA). |

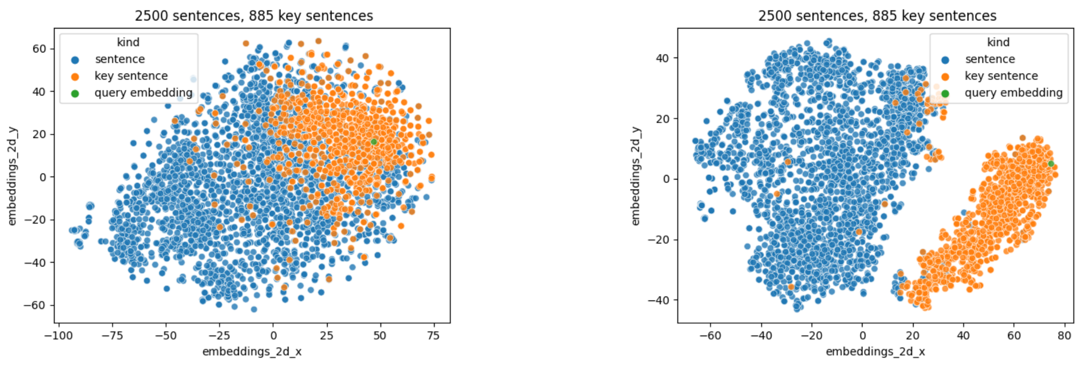
Table 4.1: Examples of predictions by *KS-PubMedBERT* compared to the true annotation.

Further analysis of the *KS-PubMedBERT* model confidence in predictions has been done by calculating the cosine similarity between the query string embedding and the predictions. Higher cosine similarities are interpreted as higher confidence in the prediction. Results of the confidence analysis are presented in Table 4.2, where the cosine similarity in different subsets of the test data are shown. The subsets are the true predictions, and the false predictions are further categorized into three subsets based on the outcome of the manual annotation. All similarities are presented as an average of all the sentences in the subset compared to the query. The average cosine similarity is highest for the true predictions and lowest for the predictions labeled as false by the test dataset and the manual annotation.

| Subset | Manual Annotation | Total Predictions | Avg Cosine Similarity |
|---|---|---|---|
| True | – | 709 | 0.883 |
| False | correct | 115 | 0.845 |
| | inconclusive | 11 | 0.705 |
| | wrong | 51 | 0.663 |

Table 4.2: The *KS-PubMedBERT* model average cosine similarity for different outcomes in the test data.

The sentences are embedded into a 784-dimensional space, i.e., each sentence is represented as a vector of length 784. By using the statistical T-SNE method, this can be reduced to a 2-dimensional space for visualization. This dimensional reduction will not keep all information. However, it might provide some insight into the semantic understanding of the model. In Figure 4.1, the plots compare the representation of sentences, key sentences, and the query sentence before and after fine-tuning. 2500 sentences are randomly sampled from the test set, along with all the key sentences from the test data.



(a) Before fine-tuning.

(b) After fine-tuning.

Figure 4.1: Semantic understanding of key sentence versus non-key sentence. The query embedding is the embedding of the string "knock mutat silence disrupt crispr lack cell gene remove edit".

## 4.2 Effect of Query Sentence

This section presents the results of altering the query sentence. The natural text queries, developed in cooperation with domain experts, are as follows:

1. knock mutat silence disrupt crispr lack cell gene remove edit
2. Gene knockout from cell
3. Gene knockout from cell using CRISPR

Apart from these natural text queries, two new queries are used. These queries are both utilizing a fresh token, "[QUERY]", which is added to the model vocabulary before training. The embedding of the token is initialized to zeros or a randomly generated vector.

Table 4.3 displays the results of fine-tuning S-PubMedBERT from different queries. It should be noted that the query labeled Generic 1 is the same training setup as used for training *KS-PubMedBERT*. All query experiments obtain similar results.

| Query | ARP | 95% CI | MSE | PCC |
|---|---|---|---|---|
| Generic 1 | 0.800 | ± 0.014 | 0.186 | 0.721 |
| Generic 2 | 0.797 | ± 0.013 | 0.167 | 0.704 |
| Generic 3 | 0.808 | ± 0.004 | 0.186 | 0.721 |
| Fresh 0 | 0.805 | ± 0.008 | 0.188 | 0.691 |
| Fresh 1 | 0.797 | ± 0.005 | 0.199 | 0.704 |

Table 4.3: Results from selecting different query sentences in the contrastive learning approach. Generic sentences are explained in the list above, Fresh 0 is a fresh token initialized to zeros, and Fresh 1 is a fresh token initialized to a randomly generated vector.

## 4.3 Vocabulary

The training dataset contains 1,913 articles. Each article contains an annotation of the related KO gene(s) and cell(s). Counting all these genes, cells, and their respective synonyms, a total of 11,618 terms are investigated. A few examples of tokenization using PubMedBERT and BERT, both uncased, are shown in Table 4.4. Here, 10 terms are randomly sampled from the total term set. In all 10 examples, BERT tokenizes the terms into at least as many tokens as PubMedBERT. A check-mark means that the vocabulary of the model contains the word, and the word was successfully tokenized.

A more comprehensive review of tokenization can be seen in Table 4.5, where PubMedBERT outperforms other models, having $906/11,618 = 7.8\%$ of terms in its vocabulary. The average length in the table displays the average number of tokens the terms are broken into by the models.

| Term | PubMedBERT | BERT |
|---|---|---|
| Kop | ['ko', '##p'] | ['ko', '##p'] |
| LC3 | ✓ | ['lc', '##3'] |
| USP30 | ['usp', '##30'] | ['us', '##p', '##30'] |
| PSD4 | ['psd', '##4'] | ['ps', '##d', '##4'] |
| p300 HAT | ['p300', 'hat'] | ['p', '##30', '##0', 'hat'] |
| MET-1 | ['met', '-', '1'] | ['met', '-', '1'] |
| HiPS201B7 | ['hips', '##201', '##b7'] | ['hips', '##20', '##1', '##b', '##7'] |
| KM12-L4A | ['km', '##12', '-', 'l4', '##a'] | ['km', '##12', '-', 'l', '##4', '##a'] |
| HuH7 | ✓ | ['huh', '##7'] |
| AD-38 | ['ad', '-', '38'] | ['ad', '-', '38'] |

Table 4.4: Review of the tokenization between BERT and PubMedBERT.

| | PubMedBERT | SCIBERT | BERT | BioBERT |
|---|---|---|---|---|
| Terms in vocab | 906 | 486 | 306 | 135 |
| Avg length | 2.80 | 3.18 | 3.58 | 4.04 |

Table 4.5: The number of gene and cell terms from the dataset in the vocabulary of some relevant pre-trained models.

## 4.4 Models

When fine-tuning the Transformer model for identifying key sentences, the training is conducted from a checkpoint that is a pre-trained model. Table 4.6 shows how the aforementioned models perform without any fine-tuning. As the models have no fine-tuning on specific query sentences, evaluation is done by calculating the cosine similarity between the sentences in the test dataset and the average key sentence embeddings from the training dataset. That is, the average key sentence embedding replaces the query sentence.

| Model | ARP | MSE | PCC |
|---|---|---|---|
| S-PubMedBERT | 0.480 | 0.843 | 0.180 |
| PubMedBERT | 0.550 | 0.956 | 0.086 |
| BERT | 0.349 | 0.864 | 0.078 |
| SciBERT | 0.566 | 0.730 | 0.099 |
| BioBERT | 0.377 | 0.863 | 0.067 |

Table 4.6: Base model performance without fine-tuning.

The comparison between fine-tuning from S-PubMedBERT, PubMedBERT, BERT, BioBERT, and SciBERT are shown in Table 4.7. In these experiments, all models are evaluated and fine-tuned using Generic Query 1, which is "knock mutat silence disrupt crispr lack cell gene remove edit".

| Model | ARP | 95% CI | MSE | PCC |
|---|---|---|---|---|
| S-PubMedBERT | 0.800 | ± 0.014 | 0.186 | 0.721 |
| PubMedBERT | 0.804 | ± 0.004 | 0.183 | 0.676 |
| BERT | 0.767 | ± 0.004 | 0.194 | 0.614 |
| SciBERT | 0.795 | ± 0.005 | 0.196 | 0.598 |
| BioBERT | 0.771 | ± 0.007 | 0.198 | 0.639 |

Table 4.7: Mean of 5 training iterations with S-PubMedBERT, PubMedBERT, BERT, SciBERT, and BioBERT as checkpoints.

## 4.5 Loss Function

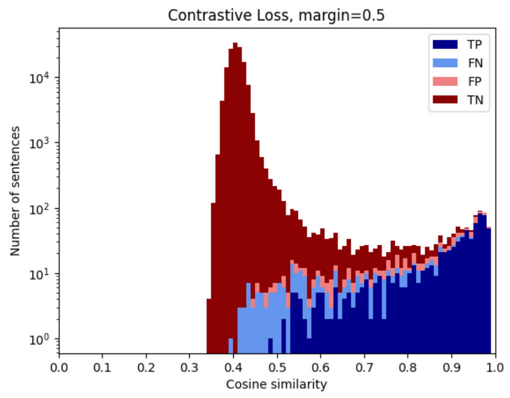
A higher *margin* hyperparameter of the contrastive loss function encourages the model to output lower scores for dissimilar sentences but does not affect the loss applied to similar sentences. In previous experiments, contrastive loss with a margin of 0.5 has been used. In Table 4.8, results of margins of 0.1, 0.3, 0.5, 0.7, and 0.9 are presented, along with the results of using cosine similarity as the loss function. ARP results are barely affected by the loss function. However, MSE and PCC are affected. The largest difference can be seen in MSE, where contrastive loss with margin 0.1 has 0.698 in MSE, whereas cosine similarity loss has 0.003 MSE. This is probably because of the data distribution with many more negative samples than positives. Hence, encouraging cosine similarity outputs close to 0 will yield low MSE. The PCC is also affected, but not to the same degree as MSE.

| Loss Function | Margin | ARP | 95% CI | MSE | PCC |
|---|---|---|---|---|---|
| Contrastive Loss | 0.1 | 0.792 | ± 0.012 | 0.698 | 0.433 |
| Contrastive Loss | 0.3 | 0.800 | ± 0.007 | 0.359 | 0.639 |
| Contrastive Loss | 0.5 | 0.800 | ± 0.014 | 0.186 | 0.721 |
| Contrastive Loss | 0.7 | 0.800 | ± 0.007 | 0.052 | 0.718 |
| Contrastive Loss | 0.9 | 0.803 | ± 0.007 | 0.005 | 0.740 |
| Cosine Similarity Loss | – | 0.802 | ± 0.007 | 0.003 | 0.764 |

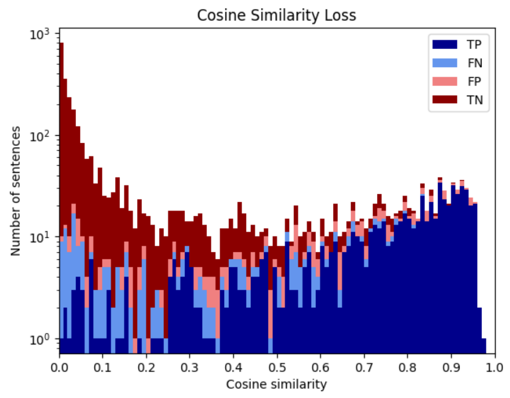
Table 4.8: Mean of 5 training iterations with different loss function configurations.

Figure 4.2 displays histograms of KS-PubMedBERT and the best-performing model trained with cosine similarity. Both had an ARP of 0.81 on the test dataset. However, as indicated by the different MSE results in Table 4.8, the output cosine similarities are contrasting. Most notably, the contrastive loss with a margin of 0.5 appears to not encourage the model to produce any similarity scores lower than roughly 0.35 in Figure 4.2a. Training with the cosine loss function seems to produce more evenly distributed cosine similarities, as seen in Figure 4.2b. Notice that the y-axis has a logarithmic scale.

Figure 4.3 also presents the difference between cosine similarities with the two different loss functions. Here, *positives* are the frequency of total positives found above a certain cosine similarity threshold with respect to the total number of positives in the test dataset. The *positives* indicates how many key sentences the



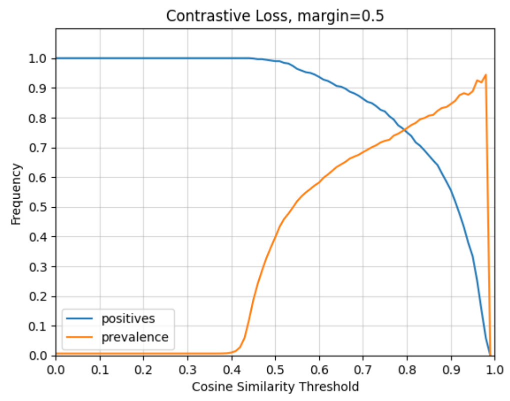
(a) Contrastive Loss



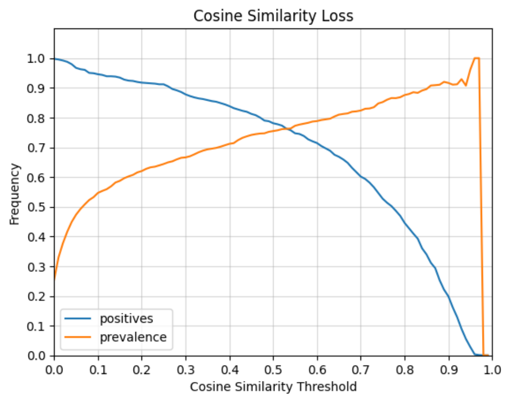
(b) Cosine Similarity Loss

Figure 4.2: Number of sentences in intervals of cosine similarity. Each bar represents an interval of 0.01 cosine similarity. The blue colors represent the positive labels (TP and FN), and the red represents the negative labels (FP and TN).

model retrieves in comparison to the total number of key sentences in the dataset, given a threshold. The *prevalence* metric indicates how many positives are found in relation to the total number of sentences found. Prevalence is calculated as $(TP + FN)/(TP + FN + FP + TN)$ and serves as an indication as to how certain one can be that a sentence above a certain threshold is, in fact, a key sentence. For instance, in Figure 4.3b at a cosine similarity threshold of 0.9, all sentences with a cosine similarity of above 0.9 are evaluated. Roughly 20% of positive sentences are found out of the total 886 positives the test dataset contains. The prevalence is about 90%, meaning that from all the sentences with a higher cosine similarity than 0.9, 90% of them are labeled as key sentences in the test dataset. Hence, both a high *prevalence* and *positives* score is desired. Both Figure 4.3b and Figure 4.3a seem to indicate comparable performance, where the model trained with contrastive loss function is truncated with respect to the cosine similarity in comparison to the model with cosine similarity loss.



(a) Contrastive Loss



(b) Cosine Similarity Loss

Figure 4.3: The prevalence and frequency of positives found above the cosine similarity threshold.

# 5
# Discussion and Conclusion

In this chapter, we discuss the results obtained, in addition to an account of possible future work. Finally, we conclude this project with a summary.

## 5.1 Discussion

The objectives of this project were to fine-tune a Sentence Transformer model for the task of keyphrase extraction and to explore different training setups that affect the performance of the models. In the subsequent sections, we will discuss the selected approaches, the datasets used, the evaluation process, and the outcomes.

### 5.1.1 Data and Evaluation

The initial CRISPR-related dataset was labeled on the document level and not the sentence level. For each article, the relevant cells and genes are annotated, but the annotation is per article and not for each sentence. The consequence was that an approach using NER and RD became more complicated to execute. While extraction of named entities and relations is possible to explore, this project chose to instead explore the approach of key sentence extraction. Nevertheless, NER and RD could be used to extract named entities and relations from the key sentences in a later stage.

As presented in Section 4.1, there was an issue with evaluation errors, where the model predicted key sentences that were not annotated as such. The evaluation error problem, which was introduced in Subsection 2.9.1, is quite prevalent. From the false positive predictions, there was an evaluation error of $65 - 71\%$, where the uncertainty comes from the 11 sentences that were annotated as inconclusive. Since the test dataset and the training dataset has been built in a similar fashion, there is reason to believe that a significant amount of actual key sentences are not labeled in the training dataset as well. The training dataset flaw might contribute to worsening the outcome. A possible solution is to manually annotate the entire dataset, but this would probably be too expensive. Another interesting outcome of the manual annotation was that from the 177 sentences, 11 were inconclusive. Hence, deciding whether a sentence is a key sentence is a hard task, even for experts, and is a matter of subjective interpretation. The evaluation errors also indicate one of many reasons why information retrieval systems are hard to evaluate accurately. Furthermore, analysis of the output cosine similarity scores indicates the usefulness of cosine similarity as an expression of the model confidence. Higher cosine similarity seems to increase the likelihood that the predicted sentence is, in fact, a key sentence. From the original 177 sentences that were evaluated as wrong by the evaluation system, 115 turned out to be correct, and the average cosine similarity of these sentences is 0.845. For the 51 sentences that were still wrong after manual annotation, the average cosine similarity is only 0.663.

The structural consistency of key sentences was not studied, and there might exist simple heuristics or a possibility to increase the performance by applying a hybrid approach with statistics. For instance, key sentences could appear more frequently in the abstract or the conclusion section than in other sections. If this is the case, encouraging key sentence predictions in these sections could be beneficial. Nonetheless, no obvious structural consistency was found.

### 5.1.2 Discussion of Methods

The discussion regarding the chosen method will cover four different subjects. Firstly, our approach is compared against an alternative approach without a query to retrieve relevant sentences. Secondly, the problem of retrieving relevant sentences is discussed. Thirdly, there is a discussion about the difficulty of defining key sentences. Lastly, scenarios where key content is found in multiple sentences, as well as the flexibility of the query approach, will be covered.

By introducing a query, also referred to as an anchor, the hope was to achieve faster convergence. The idea was that providing a more static point in the vector space, an anchor, which would repel or attract sentences, would facilitate the model during fine-tuning. An alternative approach would have been to train key sentences to be semantically equal and non-key sentences in relation to key sentences to be semantically unrelated. Then, the average key sentence embedding could be used to retrieve the relevant sentences. This approach was briefly and informally explored during the early stages of implementation, but the query approach seemingly provided higher performance. Therefore, this project only presents results with an approach with a query sentence. Another advantage of the query sentence is that it provides an easy method to evaluate by computing the similarity score between a query and the input sentence.

Without a query, the average key sentence embedding from the training dataset could be used to retrieve relevant sentences. In this case, the training task differs from the retrieval task. The model is not trained against the average key sentence embedding, but the average key sentence embedding is used to retrieve key sentences. This could be one argument for why a query approach resulted in higher performance. A possible development of the current model could be to use the average key sentence embedding as an anchor/query and to train with this embedding as an anchor instead of the embeddings of general sentences or randomized embeddings. On the other hand, it is possible that this would still not affect the results. The reason for this is that the query itself seems to have an insignificant effect on the result.

Another drawback of our approach, if it were to be used in other domains, is that it is possibly difficult to define key sentences in other domains. It is possible that the information differs a lot between key sentences. Sometimes general information could be found between the key sentences, but in some domains, there could be few or no common denominators. Then, it could be hard to annotate a dataset for key sentence extraction. On the other hand, any embedding can be used as a query, and it can be randomized without significantly affecting the result.

Lastly, in some cases, key content could be found in multiple sentences such that a single sentence is not enough to gain the full information. The assumption of a key sentence containing all relevant information might not always hold. In our approach, incomplete key sentences that are missing some key content can be predicted in the lack of a complete key sentence. The reason is that sentences closest to the query are predicted as key sentences. When using the query approach for sentence retrieval, the number of predicted sentences can be adjusted for the specific use case. If many sentences are needed, it is easy to predict more key sentences. In this way, the query and similarity approach has high flexibility, and the approach is adjustable for different scenarios. Furthermore, outputting cosine similarity scores provides information when no sentence in the text is semantically close to the query sentence. Finally, another related issue with the sentence approach is that some sentence tokenizer needs to be applied to the text before feeding the text to the model. A suboptimal sentence tokenizer might impair the performance of the key sentence retrieval system.

### 5.1.3 Result Analysis

The objective of this section is to discuss the research questions formulated in Section 1.1 in relation to the results obtained. These questions can be summarized as whether Sentence Transformers are effective for key sentence extraction, which factors influence the performance of the models, and how the models can be used in a deployment scenario. Firstly, the impact of formulating different queries on the performance of the model is covered. Secondly, there is a comparison between different base models and how their vocabulary influences performance. Thirdly, there is a discussion about evaluation errors and challenges in the task of key content extraction. Fourthly, the impact of the loss function is analyzed. The discussion of results is wrapped up with an observation on the usefulness of different metrics and a commentary on possible uses for the model upon deployment.

The results showed that it was an effective approach to fine-tuning a Sentence Transformer model for the task of key sentence extraction. The ARP of *KS-PubMedBERT* of 90.4% makes the model likely to predict a key sentence. Altering the query sentence did not seem to significantly alter the performance of the model. However, results suggest that selecting the correct base model before fine-tuning is necessary. Finally, the loss function did not alter the outcome in terms of ARP. However, it did affect the MSE and PCC scores.

By evaluating different queries, the goal was to investigate whether the initial semantics of the anchor mattered for the semantic understanding of key sentences after fine-tuning. All the queries, both those mentioning biomedical terms and fresh tokens, resulted in an ARP of around 80%. Hence, the query itself does not significantly affect the result. As the query embedding relates to the model's initial understanding of a key sentence, it is possible that a smaller training dataset would cause the query to have a larger impact on the result. However, no clear trend was found upon comparing the evaluation throughout the training of fresh tokens and natural language queries. An example of evaluation can be seen in Appendix A. One conclusion to draw from the fact that the starting point seems to be unimportant is that the training of the model is quite robust. Regardless of the initial state, the model often appears to learn the most important semantics necessary to classify key sentences.

The assumption when analyzing the vocabulary of different models was that having more gene- and cell terms in the vocabulary would benefit the models, as they might take advantage of a higher quality word vocabulary in understanding sentences. One weakness of this particular study is that it only considers genes and cells, which form a subset of all the words in the entire dataset. Over 11,000 unique gene- and cell names were found in the training dataset, which is quite large when taking into consideration that the entire vocabulary of the models is roughly 30,000 words. The best-suited model was PubMedBERT with 7.8% of the studied terms in-vocabulary. S-PubMedBERT has the same vocabulary as PubMedBERT. SciBERT, BioBERT, and BERT showed a significantly worse vocabulary than PubMedBERT. When comparing the models before fine-tuning, there seems to be some relation between the vocabulary and the performance on the sentence semantics task. SciBERT, the model with second best vocabulary, did best with an ARP of 0.566, followed by PubMedBERT and S-PubMedBERT with an ARP of 0.550 and 0.480, respectively. BioBERT and BERT, which had the worst vocabulary performance, also had the worst ARP. S-PubMedBERT did, however, outperform the other models as measured by the PCC. After fine-tuning, the differences between the performance of the models narrowed. However, BERT and BioBERT were roughly 3 percentage points worse than the other models in ARP.

Without fine-tuning, the base models performed with an ARP around 35-55 %, and after fine-tuning, the ARP was around 80%. Utilizing the average key sentence embedding to retrieve relevant sentences implies that all key sentences are semantically similar and differ from other sentences without fine-tuning. It is possible that the key sentences don't always differ much from other sentences. As seen in Figure 4.1 before fine-tuning, the average key sentence embedding could lead to many sentences being misclassified as key sentences and the other way around. Some sentences can mention similar terms as key sentences without being considered as key sentences. This can lead to the average key sentence embedding being closer to a non-key sentence than a key sentence in the vector space, which leads to a wrongly predicted sentence. A key sentence can also be misclassified as a non-key sentence if the embedding has a larger distance to the average key sentence embedding than a non-key sentence. To conclude, using base models and the average key sentence embedding leads to worse performance than fine-tuning and using a query to retrieve key sentences.

The T-SNE visualization showed the comparison between the pre-trained model S-PubMedBERT and the fine-tuned version *KS-PubMedBERT*. The plot reveals two quite distinctive clusters after fine-tuning, which represent key sentences and non-key sentences for the most part. This visualization suggests that training is successful in differentiating between key sentences and sentences. However, there are a few non-key sentences that seem to belong to the cluster of key sentences. A few key sentences belong to the cluster of non-key sentences. This could be an explanation for the score of the model. It is interesting that almost all sentences that belong to the ”wrong” cluster are at the edge of the cluster. The reason behind this could be that these sentences are similar to key sentences but don’t fulfill the criteria to be a key sentence or that they are key sentences but seem more similar to a non-key sentence. According to the analysis of evaluation errors, a substantial part of the issue is likely that many sentences that, in fact, are key sentences are not labeled as such.

The base models must understand the semantic meaning of the key sentences to realize how they differ from other sentences. In the CRISPR-related domain, it is possible that the models don’t fully understand the semantic meaning of the CRISPR-related texts without fine-tuning. This can be strengthened by S-PubMedBERT and PubMedBERT, which are biomedical-domain models, performing better than BERT, which is a general domain model. S-PubMedBERT and PubMedBERT, since trained on biomedical domain texts, should be able to more accurately represent the semantic meaning of the key sentences than BERT is able to do. The result confirms this, where the biomedical domain models have higher performance in the task than BERT. Comparing S-PubMedBERT and PubMedBERT, it seems like general semantic sentence training does not transfer to the particular task of identifying key sentences as defined in this project.

The results showed that *KS-PubMedBERT* had issues with evaluation errors, which were introduced in Subsection 2.9.1. For *KS-PubMedBERT*, there were 115 key sentences not annotated as such out of the total of 177 wrong key sentence predictions. Therefore, it seems reasonable that most training configurations could obtain ARP close to 0.9 after manual annotation. Besides, there is not a clear-cut distinction between key sentences and non-key sentences. For some key content, several sentences are needed to cover all the relevant content. Genes/cells, knockout, and CRISPR-related terms could be mentioned in two or more sentences. This scenario is not covered by our model since the model only searches for occurrences of these terms in one sentence. If no such sentence exists, the model can predict a sentence that mentions parts of the information, and such an incomplete sentence could therefore be predicted.

In the loss function analysis, the main objective was to evaluate how different margins affected the ARP. As expected, the MSE decreased as the margin increased, as this encouraged the model to output cosine similarity scores close to 0 more frequently. This reduces the MSE because the dataset is unbalanced with a high number of non-key sentences. As the margin increased, the PCC also improved. The results, however, show that the ARP was not affected by altering the loss function. Further studies were conducted to analyze the frequency of key sentences and non-key sentences in different ranges of cosine similarity. When comparing a model that has been trained with contrastive loss with a margin of 0.5 and a model trained with cosine similarity loss, there were no clear differences in their ability to differentiate between key sentences and non-key sentences. Therefore, it is a subjective judgment of which types of cosine similarity scores one wants to use. If the cosine similarity loss is interpreted as a probability that the queried sentence is a key sentence, it might be more intuitive with cosine similarity loss, as the output utilizes the entire range between 0 and 1. On the other hand, it might be illogical that many key sentences are predicted with a cosine similarity score close to 0. Nevertheless, in either case, the cosine similarities seem to indicate to a satisfactory degree the likelihood that the sentence is a key sentence. In other words, as the cosine similarity increases, so does the likelihood that the sentence is a key sentence.

The metrics presented throughout the results, ARP, MSE, and PCC, are used for different reasons. The motivation was that ARP would indicate how well the model performs on the actual task of key sentence extraction, whereas MSE provides an indication as to how well the cosine similarity output follows the labels. PCC is a covariance metric, which, similarly to MSE, measures how the cosine scores follow the labels. Following the analysis done in Section 4.5, MSE and PCC does not seem to be very useful, as a big part of their score is modeling how well the model predicts outputs close to 0 for non-key sentences. This property is irrelevant for key sentence extraction, as illustrated by the outcome of training with a contrastive loss with a margin of 0.1. Here, the ARP was $0.792 \pm 0.012$, hence, not substantially worse than other margins. However, the MSE and PCC scores were poor, with an MSE of 0.698 and PCC of 0.433.

In terms of deployment, the most natural application of the model, which has been studied in this thesis, is key sentence extraction given a research article related to the CRISPR-Cas9 domain. However, another possible application is to use the model for document retrieval. The simplest form of document retrieval could state that if an article contains a key sentence, it is likely that the entire article is related to the CRISPR domain. For instance, according to the analysis presented in Figure 4.3a, applying the model with a cut-off frequency of 0.8 in cosine similarity, roughly 75% of actual key sentences will be identified, and roughly 75% of the predicted sentences will be true positives. This example assumes that the model performs similarly on the test dataset as on queried documents.

### 5.1.4 Future Work

Our approach was to fine-tune pre-trained models using a siamese network architecture for key sentence extraction. A weakness in this approach was that the curated dataset did not have all key sentences labeled. Therefore, a possible future research area in improving the Sentence Transformer model is to investigate methods to improve upon the training given an incomplete dataset. One possible approach to achieve this is to take inspiration from the Augmented SBERT [75] method. Here, the human-annotated but incomplete golden dataset is passed to a cross-encoder, which is used to predict labels that form a silver dataset. Both the gold and silver datasets are used as training samples for a bi-encoder. There are some differences between this approach and the issues at hand in this project. For instance, it assumes that a well-performing cross-encoder already exists. Nevertheless, taking inspiration from this method seems promising, especially as the manual annotation presented in Section 4.1 indicates that the frequency of annotated key sentences in the training dataset is too low to correctly reflect reality.

To automate the annotation process further, NER could be used in the sentences extracted by our model to automatically detect genes and cells in the key sentences, which would reduce the amount of manual work further. Particularly, the BERN2 [72] model has shown promising results in biomedical NER tasks. Another approach for finding key sentences could be to apply BERN2 in parallel with a Sentence Transformer, such that both entities and sentence semantics are considered in the sentence extraction. Another unsolved problem is to study the differences between articles regarding KO, KI, and KD genes. As presented in Section 3.1, the current dataset is unbalanced with a majority of KO genes. Nevertheless, an interesting task is to identify whether an article is about KO, KI, or KD.

One assumption that was formulated at the beginning of this project was that each article contains at least one sentence that contains all the key content of a CRISPR-related article. Each such sentence has been referred to as a key sentence, but this assumption might not hold for all texts. Additionally, there might exist better ways to extract key content than by looking at individual sentences. The self-attention mechanism of the Transformer, which scales quadratically, makes it unsuited to process large sequences of texts. This was an important factor as to why it was decided to split the text into sentences. A possible solution to the key sentence assumption is to investigate the application of a Longformer [41] architecture to this task. By doing so, entire articles could be processed at a time with a reasonable performance.

Finally, another possible area to explore is to further analyze patterns in the text structure, i.e. if it is sufficient to only look at the abstract, introduction, or conclusion to find key sentences. If that is the case, the problem of key content extraction could be made simpler, and it would save time and work to not have to go through the full articles.

## 5.2 Conclusion

We developed a computationally efficient Transformers-based model for key sentence extraction of CRISPR-Cas9 research articles with siamese network architecture. By formulating the task as a regression problem and utilizing a query sentence as an anchor in the training, the model outputs useful cosine similarity scores, in addition to the ability to easily be used for classification.

We show, with a thorough analysis of the model, that it improves upon fine-tuning. Average R-Precision of **90.4%** is achieved on the key sentence extraction task.

The extraction of key content poses a multifaceted challenge, often necessitating diverse methods tailored to specific use cases. However, the investment of time and effort in developing a model for automated key content extraction is justified by the significant benefits it offers in terms of saving time and facilitation of work.

# Bibliography

[1] M. Redman, A. King, C. Watson, and D. King, "What is crispr/cas9?" *Archives of Disease in Childhood - Education and Practice*, vol. 101, no. 4, pp. 213–215, 2016, ISSN: 1743-0585. DOI: 10.1136/archdischild-2016-310459. eprint: https://ep.bmj.com/content/101/4/213.full.pdf. [Online]. Available: https://ep.bmj.com/content/101/4/213.

[2] P. D. Hsu, E. S. Lander, and F. Zhang, "Development and applications of crispr-cas9 for genome engineering," *Cell*, vol. 157, no. 6, pp. 1262–1278, 2014, ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2014.05.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867414006047.

[3] M. Adli, "The crispr tool kit for genome editing and beyond," *Nature Communications*, vol. 9, no. 1, p. 1911, May 2018, ISSN: 2041-1723. DOI: 10.1038/s41467-018-04252-2. [Online]. Available: https://doi.org/10.1038/s41467-018-04252-2.

[4] S. Sánchez-León, J. Gil-Humanes, C. V. Ozuna, *et al.*, "Low-gluten, nontransgenic wheat engineered with crispr/cas9," *Plant biotechnology journal*, vol. 16, no. 4, pp. 902–910, 2018. DOI: https://doi.org/10.1111/pbi.12837.

[5] C. A. Lino, J. C. Harper, J. P. Carney, and J. A. Timlin, "Delivering crispr: A review of the challenges and approaches," *Drug Delivery*, vol. 25, no. 1, pp. 1234–1257, 2018, PMID: 29801422. DOI: 10.1080/10717544.2018.1474964. eprint: https://doi.org/10.1080/10717544.2018.1474964. [Online]. Available: https://doi.org/10.1080/10717544.2018.1474964.

[6] S. Sarntivijai, Y. Lin, Z. Xiang, *et al.*, "Clo: The cell line ontology," *Journal of biomedical semantics*, vol. 5, no. 1, pp. 1–10, 2014.

[7] N. Perera, M. Dehmer, and F. Emmert-Streib, "Named entity recognition and relation detection for biomedical information extraction," *Frontiers in Cell and Developmental Biology*, vol. 8, 2020, ISSN: 2296-634X. DOI: 10.3389/fcell.2020.00673. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fcell.2020.00673.

[8] C. B. Gurumurthy, A. R. O'Brien, R. M. Quadros, *et al.*, "Reproducibility of crispr-cas9 methods for generation of conditional mouse alleles: A multicenter evaluation," *Genome Biology*, vol. 20, no. 1, p. 171, Aug. 2019, ISSN: 1474-760X. DOI: 10.1186/s13059-019-1776-2. [Online]. Available: https://doi.org/10.1186/s13059-019-1776-2.

[9] P. DEKA, A. JUREK-LOUGHREY, and P. DEEPAK, “Improved methods to aid unsupervised evidence-based fact checking for online health news,” *Journal of Data Intelligence*, vol. 3, no. 4, pp. 474–504, 2022.

[10] E. Szathmáry, F. Jordán, and C. Pál, “Can genes explain biological complexity?” *Science*, vol. 292, no. 5520, pp. 1315–1316, 2001. DOI: `10.1126/science.1060852`. eprint: `https://www.science.org/doi/pdf/10.1126/science.1060852`. [Online]. Available: `https://www.science.org/doi/abs/10.1126/science.1060852`.

[11] J. P. Manis, “Knock out, knock in, knock down genetically manipulated mice and the nobel prize,” *New England Journal of Medicine*, vol. 357, no. 24, pp. 2426–2429, 2007, PMID: 18077807. DOI: `10.1056/NEJMp0707712`. eprint: `https://doi.org/10.1056/NEJMp0707712`. [Online]. Available: `https://doi.org/10.1056/NEJMp0707712`.

[12] K. R. Chowdhary, “Natural language processing,” in *Fundamentals of Artificial Intelligence*. New Delhi: Springer India, 2020, pp. 603–649, ISBN: 978-81-322-3972-7. DOI: `10.1007/978-81-322-3972-7_19`. [Online]. Available: `https://doi.org/10.1007/978-81-322-3972-7_19`.

[13] V. Suseela, “The use of online information search/retrieval services in university of hyderabad,” Nov. 2014.

[14] D. D. Palmer, “Tokenisation and sentence,” *Handbook of natural language processing*, p. 11, 2000.

[15] A. Nayak, H. Timmapathini, K. Ponnalagu, and V. Gopalan Venkoparao, “Domain adaptation challenges of BERT in tokenization and sub-word representations of out-of-vocabulary words,” in *Proceedings of the First Workshop on Insights from Negative Results in NLP*, Online: Association for Computational Linguistics, Nov. 2020, pp. 1–5. DOI: `10.18653/v1/2020.insights-1.1`. [Online]. Available: `https://aclanthology.org/2020.insights-1.1`.

[16] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, *Transformer-xl: Attentive language models beyond a fixed-length context*, 2019. arXiv: `1901.02860 [cs.LG]`.

[17] A. Radford, R. Jozefowicz, and I. Sutskever, “Learning to generate reviews and discovering sentiment,” *arXiv preprint arXiv:1704.01444*, 2017.

[18] 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.

[19] M. Schuster and K. Nakajima, “Japanese and korean voice search,” in *International Conference on Acoustics, Speech and Signal Processing*, 2012, pp. 5149–5152.

[20] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: `10.18653/v1/P16-1162`. [Online]. Available: `https://aclanthology.org/P16-1162`.

[21] Q. Chen, Y. Peng, and Z. Lu, "Biosentvec: Creating sentence embeddings for biomedical texts," in *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, 2019, pp. 1–5. DOI: `10.1109/ICHI.2019.8904728`.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[23] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[24] A. S. Thakur and N. Sahayam, "Speech recognition using euclidean distance," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 3, pp. 587–590, 2013.

[25] D. Sinwar and R. Kaushik, "Study of euclidean and manhattan distance metrics using simple k-means clustering," *Int. J. Res. Appl. Sci. Eng. Technol*, vol. 2, no. 5, pp. 270–274, 2014.

[26] T. Ogita, S. M. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1955–1988, 2005. DOI: `10.1137/030601818`. eprint: `https://doi.org/10.1137/030601818`. [Online]. Available: `https://doi.org/10.1137/030601818`.

[27] H. Liao and Z. Xu, "Approaches to manage hesitant fuzzy linguistic information based on the cosine distance and similarity measures for hfltss and their application in qualitative decision making," *Expert Systems with Applications*, vol. 42, no. 12, pp. 5328–5336, 2015.

[28] A. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996. DOI: `10.1109/2.485891`.

[29] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. arXiv: `1609.04747`. [Online]. Available: `http://arxiv.org/abs/1609.04747`.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: `10.1162/neco.1997.9.8.1735`.

[31] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. arXiv: `1412.3555`. [Online]. Available: `http://arxiv.org/abs/1412.3555`.

[32] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: `1406.1078`. [Online]. Available: `http://arxiv.org/abs/1406.1078`.

[33] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf`.

[34] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Avail-

able: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[35] G. Neubig, "Neural machine translation and sequence-to-sequence models: A tutorial," *CoRR*, vol. abs/1703.01619, 2017. arXiv: 1703.01619. [Online]. Available: http://arxiv.org/abs/1703.01619.

[36] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[37] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[38] J. Lee, W. Yoon, S. Kim, *et al.*, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, Sep. 2019, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz682. eprint: https://academic.oup.com/bioinformatics/article-pdf/36/4/1234/32527770/btz682.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz682.

[39] A. Radford, K. Narasimhan, T. Salimans, and i. Sutskever, "Improving language understanding by generative pre-training," *Technical Report, OpenAI*, 2018.

[40] S. Singh and A. Mahmood, "The nlp cookbook: Modern recipes for transformer based deep learning architectures," *IEEE Access*, vol. 9, pp. 68 675–68 702, 2021. DOI: 10.1109/ACCESS.2021.3077350.

[41] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[42] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in BERTology: What we know about how BERT works," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 842–866, 2020. DOI: 10.1162/tacl_a_00349. [Online]. Available: https://aclanthology.org/2020.tacl-1.54.

[43] Y. Wu, M. Schuster, Z. Chen, *et al.*, *Google's neural machine translation system: Bridging the gap between human and machine translation*, 2016. arXiv: 1609.08144 [cs.CL].

[44] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *CoRR*, vol. abs/1908.10084, 2019. arXiv: 1908.10084. [Online]. Available: http://arxiv.org/abs/1908.10084.

[45] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, May 2016.

[46] Y. Zhu, R. Kiros, R. Zemel, *et al.*, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[47] Y. Gu, R. Tinn, H. Cheng, *et al.*, "Domain-specific language model pretraining for biomedical natural language processing," *ACM Trans. Comput. Healthcare*, vol. 3, no. 1, Oct. 2021, ISSN: 2691-1957. DOI: 10.1145/3458754. [Online]. Available: https://doi.org/10.1145/3458754.

[48] I. Beltagy, K. Lo, and A. Cohan, *Scibert: A pretrained language model for scientific text*, 2019. arXiv: 1903.10676 [cs.CL].

[49] Y. Peng, S. Yan, and Z. Lu, "Transfer learning in biomedical natural language processing: An evaluation of BERT and elmo on ten benchmarking datasets," *CoRR*, vol. abs/1906.05474, 2019. arXiv: `1906.05474`. [Online]. Available: `http://arxiv.org/abs/1906.05474`.

[50] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. arXiv: `1907.11692`. [Online]. Available: `http://arxiv.org/abs/1907.11692`.

[51] D. Chicco, "Siamese neural networks: An overview," in *Artificial Neural Networks*, H. Cartwright, Ed. New York, NY: Springer US, 2021, pp. 73–94, ISBN: 978-1-0716-0826-5. DOI: `10.1007/978-1-0716-0826-5_3`. [Online]. Available: `https://doi.org/10.1007/978-1-0716-0826-5_3`.

[52] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[53] T. Nguyen, M. Rosenberg, X. Song, *et al.*, "Ms marco: A human generated machine reading comprehension dataset," *choice*, vol. 2640, p. 660, 2016.

[54] M. Peji Bach,. Krsti, S. Seljan, and L. Turulja, "Text mining for big data analysis in financial sector: A literature review," *Sustainability*, vol. 11, no. 5, p. 1277, Feb. 2019, ISSN: 2071-1050. DOI: `10.3390/su11051277`. [Online]. Available: `http://dx.doi.org/10.3390/su11051277`.

[55] L. L. Wang and K. Lo, "Text mining approaches for dealing with the rapidly expanding literature on COVID-19," *Briefings in Bioinformatics*, vol. 22, no. 2, pp. 781–799, Dec. 2020, ISSN: 1477-4054. DOI: `10.1093/bib/bbaa296`. eprint: `https://academic.oup.com/bib/article-pdf/22/2/781/36654452/bbaa296.pdf`. [Online]. Available: `https://doi.org/10.1093/bib/bbaa296`.

[56] A. M. Cohen and W. R. Hersh, "A survey of current work in biomedical text mining," *Briefings in Bioinformatics*, vol. 6, no. 1, pp. 57–71, Mar. 2005, ISSN: 1467-5463. DOI: `10.1093/bib/6.1.57`. eprint: `https://academic.oup.com/bib/article-pdf/6/1/57/814809/57.pdf`. [Online]. Available: `https://doi.org/10.1093/bib/6.1.57`.

[57] A. Yoshida, J. Choi, H. R. Jin, *et al.*, "Fbxl8 suppresses lymphoma growth and hematopoietic transformation through degradation of cyclin D3," *Oncogene*, vol. 40, no. 2, pp. 292–306, Jan. 2021.

[58] V. Yadav and S. Bethard, "A survey on recent advances in named entity recognition from deep learning models," *CoRR*, vol. abs/1910.11470, 2019.

[59] Z. Li and L. Fu, "A relation-aware span-level transformer network for joint entity and relation extraction," in *2022 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2022, pp. 1–8.

[60] K. S. Hasan and V. Ng, "Automatic keyphrase extraction: A survey of the state of the art," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 1262–1273.

[61] E. Papagiannopoulou and G. Tsoumakas, "A review of keyphrase extraction," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 2, e1339, 2020. DOI: `https://doi.org/10.1002/widm.1339`. eprint: `https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1339`. [Online]. Avail-

able: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1339.

[62] R. Devika, S. Vairavasundaram, C. S. J. Mahenthar, V. Varadarajan, and K. Kotecha, “A deep learning model based on bert and sentence transformer for semantic keyphrase extraction on big social data,” *IEEE Access*, vol. 9, pp. 165 252–165 261, 2021. DOI: 10.1109/ACCESS.2021.3133651.

[63] Z. C. Lipton, C. Elkan, and B. Narayanaswamy, *Thresholding classifiers to maximize f1 score*, 2014. arXiv: 1402.1892 [stat.ML].

[64] B. Gambino, “Reflections on accuracy,” *Journal of Gambling Studies*, vol. 22, no. 4, pp. 393–404, 2006.

[65] S. M. Beitzel, E. C. Jensen, and O. Frieder, “Average r-precision,” in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 195–195, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_491. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_491.

[66] J. A. Aslam, E. Yilmaz, and V. Pavlu, “A geometric interpretation of r-precision and its correlation with average precision,” in *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’05, Salvador, Brazil: Association for Computing Machinery, 2005, pp. 573–574, ISBN: 1595930345. DOI: 10.1145/1076034.1076134. [Online]. Available: https://doi.org/10.1145/1076034.1076134.

[67] I. Cohen, Y. Huang, J. Chen, *et al.*, “Pearson correlation coefficient,” *Noise reduction in speech processing*, pp. 1–4, 2009.

[68] Z. Wang and A. C. Bovik, “Mean squared error: Love it or leave it? a new look at signal fidelity measures,” *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009. DOI: 10.1109/MSP.2008.930649.

[69] L. van der Maaten and G. Hinton, “Viualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov. 2008.

[70] M. Wattenberg, F. Viégas, and I. Johnson, “How to use t-sne effectively,” *Distill*, vol. 1, no. 10, e2, 2016.

[71] W. Commons, *File:t-sne embedding of mnist.png — wikimedia commons, the free media repository*, [Online; accessed 24-May-2023], 2022. [Online]. Available: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:T-SNE_Embedding_of_MNIST.png&oldid=660516828%7D.

[72] M. Sung, M. Jeong, Y. Choi, D. Kim, J. Lee, and J. Kang, “BERN2: an advanced neural biomedical named entity recognition and normalization tool,” *Bioinformatics*, vol. 38, no. 20, pp. 4837–4839, Sep. 2022, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac598. eprint: https://academic.oup.com/bioinformatics/article-pdf/38/20/4837/46535173/btac598.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btac598.

[73] P. Lewis, M. Ott, J. Du, and V. Stoyanov, “Pretrained language models for biomedical and clinical tasks: Understanding and extending the state-of-the-art,” pp. 146–157, Nov. 2020. DOI: 10.18653/v1/2020.clinicalnlp-1.17. [Online]. Available: https://aclanthology.org/2020.clinicalnlp-1.17.

[74] S.-S. Lee and H.-S. Yong, "Component based approach to handle synonym and polysemy in folksonomy," in *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, 2007, pp. 200–205. DOI: `10.1109/CIT.2007.9`.

[75] N. Thakur, N. Reimers, J. Daxenberger, and I. Gurevych, "Augmented SBERT: data augmentation method for improving bi-encoders for pairwise sentence scoring tasks," *CoRR*, vol. abs/2010.08240, 2020. arXiv: `2010.08240`. [Online]. Available: `https://arxiv.org/abs/2010.08240`.

[76] L. Richardson, *Beautiful soup documentation*, 2007.

[77] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[78] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," To appear, 2017.

[79] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," *CoRR*, vol. abs/1508.05326, 2015. arXiv: `1508.05326`. [Online]. Available: `http://arxiv.org/abs/1508.05326`.

[80] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: `http://aclweb.org/anthology/N18-1101`.

[81] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1735–1742. DOI: `10.1109/CVPR.2006.100`.

[82] A. Chernyavskiy, D. Ilvovsky, P. Kalinin, and P. Nakov, "Batch-softmax contrastive loss for pairwise sentence scoring tasks," *CoRR*, vol. abs/2110.15725, 2021. arXiv: `2110.15725`. [Online]. Available: `https://arxiv.org/abs/2110.15725`.

[83] C. Yang, F. Rottensteiner, and C. Heipke, "Investigations on skip-connections with an additional cosine similarity loss for land cover classification," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. V-3-2020, pp. 339–346, Aug. 2020. DOI: `10.5194/isprs-annals-V-3-2020-339-2020`.

[84] M. T. R. Laskar, J. X. Huang, and E. Hoque, "Contextualized embeddings based transformer encoder for sentence similarity modeling in answer selection task," English, in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, Marseille, France: European Language Resources Association, May 2020, pp. 5505–5514, ISBN: 979-10-95546-34-4. [Online]. Available: `https://aclanthology.org/2020.lrec-1.676`.

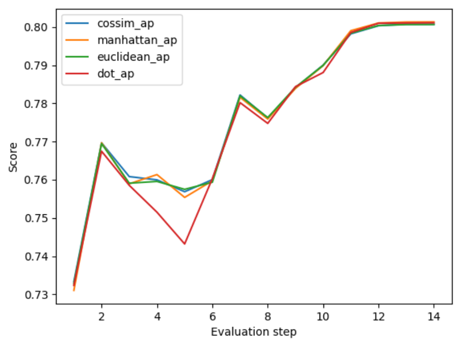
[85] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. arXiv: `1711.05101`. [Online]. Available: `http://arxiv.org/abs/1711.05101`.

# A

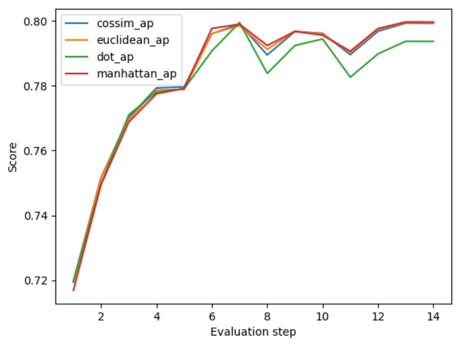
# Appendix 1

## A.1 Evaluation Plots

This section presents a brief review of the evaluation during training. The experiment is explained in Section 4.2. A total of 25 models were trained in this experiment and Figure A.1 displays the outcome of evaluation throughout training on two of these models. Each evaluation step in the figures represents 32,000 training samples. These figures represent the main findings in evaluation, which is that most models have an AP of around 72% in the first evaluation step, and 80% after the final evaluation step. This seemingly holds regardless of whether the model has been trained with a fresh token or an embedding of a natural language sentence. These figures also display another general finding upon evaluation, that the choice of similarity function matters little for the outcome.



(a) Fresh randomly initialized query vector.



(b) Natural language query "Gene knockout from cell", embedded.

Figure A.1: Evaluation on validation data throughout training as explained in Section 4.2.