Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis

Extracting and Segmenting High-Variance References from PDF Documents with BERT

Hasan Evci

Course of Study:	Informatik
Examiner:	Prof. Dr. Steffen Staab
Advisor:	Anastasiia Iurshina, M.Sc.
Commenced:	June 14, 2021
Completed:	December 21, 2021

Abstract

The extraction and segmentation of references from scientific articles is a core task of modern digital libraries. Once references are extracted and segmented, the bibliographic information can be made publicly available and linked, enabling efficient literature study. However, references often vary in their structure and content. This makes the extraction and segmentation of references a challenging but valuable task.
The purpose of this thesis is to investigate whether Bidirectional Encoder Representations from Transformers (BERT) is suitable for the extraction and segmentation of bibliographic references. Therefore, we follow a deep learning approach for the extraction and segmentation of references from PDF documents. We use a neural network architecture based on BERT, a deep language representation model that has significantly increased performance on many natural language processing tasks. Over the BERT output, we put a linear-chain Conditional Random Field. We experiment with different BERT models and input formats and also examine two approaches for reference extraction and segmentation. The experiments are evaluated on a challenging dataset that contains both English and German social science publications with highly varying references.
Our results show that the best performing BERT models were pre-trained on similar data to the data that we used for the fine-tuning of the BERT models on the task of reference extraction and reference segmentation. Moreover, our findings show that long, context-based input sequences yield the best results. The extraction model identifies and extracts references with an average F1-score of 81.9%. References are segmented with an average F1-score of 93.6%. We show that our models compare well to one other previously published work. Our conclusion is that BERT is a suitable choice for reference extraction and reference segmentation.

Kurzfassung

Die Extraktion und Segmentierung von Referenzen aus wissenschaftlichen Artikeln ist eine Kernaufgabe moderner digitaler Bibliotheken. Sobald Referenzen extrahiert und segmentiert sind, können bibliografische Informationen öffentlich zugänglich gemacht und verlinkt werden. Dies ermöglicht ein effizientes Literaturstudium. Allerdings unterscheiden sich Referenzen oft in ihrer Struktur und ihrem Inhalt. Dies macht die Extraktion und Segmentierung von Referenzen zu einer anspruchsvollen, aber wertvollen Aufgabe.

In dieser Arbeit soll untersucht werden, ob sich Bidirectional Encoder Representations from Transformers (BERT) für die Extraktion und Segmentierung von bibliographischen Referenzen eignet. Dazu wird ein Deep-Learning Ansatz für die Extraktion und Segmentierung von Referenzen aus PDF-Dokumenten verfolgt. Es wird eine neuronale Netzwerkarchitektur verwendet, die auf BERT basiert. BERT ist ein tiefes Sprachrepräsentationsmodell, das die Leistung bei vielen Aufgaben zur Verarbeitung natürlicher Sprache deutlich erhöht hat. Über die Ausgabe von BERT wird eine Linear-Chain Conditional Random Field gelegt. Es werden Experimente mit verschiedenen BERT-Modellen und Eingabeformaten durchgeführt und es werden zwei Ansätze zur Referenzextraktion und -segmentierung untersucht. Die Experimente werden anhand eines anspruchsvollen Datensatzes ausgewertet, der sowohl englische als auch deutsche sozialwissenschaftliche Publikationen mit stark variierenden Referenzen enthält.

Unsere Ergebnisse zeigen, dass die leistungsstärksten BERT-Modelle auf ähnlichen Daten vortrainiert wurden wie die Daten, die wir für die Feinabstimmung der BERT-Modelle für die Aufgabe der Referenzextraktion und -segmentierung verwendet haben. Zudem zeigen unsere Ergebnisse, dass lange, kontextbasierte Eingabesequenzen die besten Ergebnisse liefern. Das Extraktionsmodell identifiziert und extrahiert Referenzen mit einem durchschnittlichen F1-Score von 81,9%. Referenzen werden mit einem durchschnittlichen F1-Score von 93,6% segmentiert. Es wird gezeigt, dass die ausgewählten Modelle gut mit einer anderen zuvor veröffentlichten Arbeit vergleichbar sind. Die Schlussfolgerung ist, dass BERT eine geeignete Wahl für die Extraktion und Segmentierung von Referenzen ist.

Contents

List of Figures

List of Tables

Acronyms

BERT Bidirectional Encoder Representations from Transformers. 2

BIO Beginning-Inside-Outside. 29

CD Complete Dataset. 27

CMV Contextual Majority Voting. 23

CRF Conditional Random Field. 4

CSV Comma-separated Values. 29

FN False Negatives. 15

FP False Positives. 15

HMM Hidden Markov Model. 5

LSTM Long Short-Term Memory. 5

M Million. 11

MEMM Maximum Entropy Markov Model. 13

MLM Masked Language Model. 12

NER Named Entity Recognition. 13

NLP Natural Language Processing. 2

NSP Next Sentence Prediction. 12

OOV Out-of-Vocabulary. 7

PDF Portable Document Format. 5

PES Proposed Dataset in English language. 28

PGS Proposed Dataset in German language. 28

SSOAR Social Science Open Access Repository. 27

SVM Support Vector Machine. 5

TN True Negatives. 15

TP True Positives. 15

URL Uniform Resource Locator. 30

XML Extensible Markup Language. 61

1 Introduction

Authors of scientific papers acknowledge scholarly works that contributed to their research field by citing them. Through the common practice of citation, a network of scientific papers emerges, consisting of the latest scientific achievements as well as findings from older publications. This helps to study scientific literature as one can discover new publications of a research field over time. Today's digital libraries support this process of studying by providing intelligent search tools that, for example, list the referenced works of documents or propose similar documents. The further maintenance of such helpful services is only guaranteed when digital libraries can easily identify metadata such as the author, title, publication year, and references of documents. It is unrealistic that a task like reference extraction is continued by human labor as the number of publications of scientific papers per year is high and keeps increasing[1].
Furthermore, the metadata of documents are not always easily extractable. Metadata information can lack good quality, sometimes be even missing, but more importantly, it is inconsistent in its structure and content as citation practice can differ significantly from one community to another. For example, at the end of scientific papers, there is often a section that brings together all referenced works in the form of a reference list or bibliography. However, in disciplines like German social sciences or humanities, references can be located in footnotes, endnotes, or sections other than the reference section. Such highly varying references references make the automatic extraction of references more difficult. Figure 1.1 illustrates the similarities and differences of high-variance references that occur in different locations of a document. In the figure, the text denoted by (a) the circle is a reference in a footnote, (b) the triangle is "non-reference" text in a footnote, and (c) the square is a reference in a reference section.
Due to the aforementioned reasons, the automatic extraction and segmentation of references is a challenging but valuable task. Many different approaches have been used to tackle the task [BAS19; CGK08; Lop09; PKK18; TSF+15].

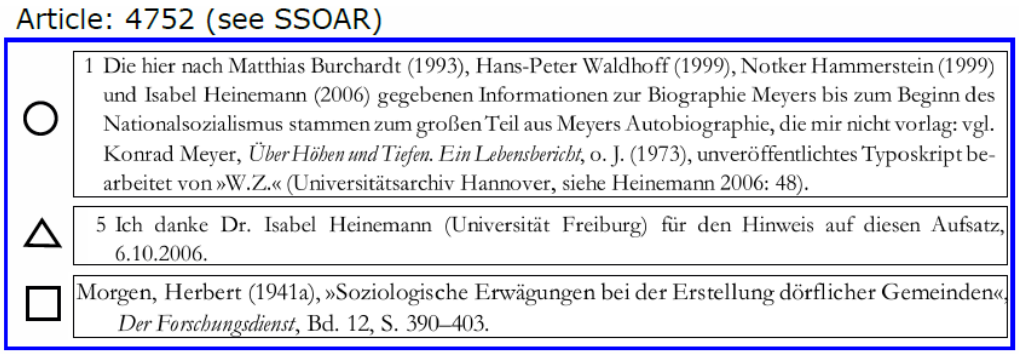

Figure 1.1: Example from an article showing highly varying references [BAS19]

[1] https://arxiv.org/help/stats/2018_by_area

We want to solve the task of reference extraction and segmentation with a deep learning approach using Bidirectional Encoder Representations from Transformers (BERT). BERT is a deep language representation model that learns embeddings for token representations in a neural-based way [DCLT19]. It has significantly boosted performance on many Natural Language Processing (NLP) tasks.
We hypothesize that BERT is a suitable choice for the task of reference extraction and segmentation. There are several reasons for choosing BERT over other models. One decisive reason is that BERT represents word embeddings based on contextual information. For example, a word in a sentence can have different representations in BERT, depending on its position in the sentence. Through contextual word embeddings, the words of a sentence can be represented very efficiently. This improves the performance of models on downstream tasks. Another reason is that BERT is already pre-trained on a large amount of data. For instance, the pre-trained model $BERT_{BASE}$ was trained on BooksCorpus [ZKZ+15] and English Wikipedia data, adding up to roughly 3,300 million words [DCLT19]. Consequently, BERT can be fine-tuned with little data and still achieves good results on downstream tasks. As there is little data available in the scope of this thesis, this is considered to be a major advantage.

1.1 Problem Definition

Before we define the task of reference extraction and reference segmentation formally, it is important to understand the difference between the terms "citation" and "reference" as they are often used interchangeably.
"Citation" refers to parts of a writing where an outside information source is acknowledged whereas a "reference" is the full bibliographic information of the corresponding citation. The citation style defines how references are structured and what content they include. This thesis is about extracting and segmenting references. The term "high-variance references" refers to references that strongly vary in their content, length, and location within a document [BAS19].
Both reference extraction and reference segmentation are formulated as sequence labeling tasks where each token in a given input sequence is assigned a class or label.

1.1.1 Reference Extraction

Reference extraction is the process of **identifying** and **extracting individual reference strings**.
We formally define the task of reference extraction as follows [CGK08]: Given a text T, it is broken down into a sequence of tokens $\{t_1, t_2, ..., t_n\}$. Then, each of these tokens is to be classified with one of the classes $C = \{$*reference, no-reference*$\}$. Tokens should be assigned the class *reference* when they are part of a reference string. Correspondingly, tokens should be assigned the class *no-reference* when they belong to normal text and are not part of a reference string. For the classification of the tokens, information from previous classifications can be used.

1.1.2 Reference Segmentation

Reference segmentation describes the segmentation of a reference string. In other words, a reference string is to be **broken down into bibliographic components** such as 'author', 'title', 'publisher', etc. For instance, given the reference string in Figure 1.2, the result of reference segmentation would yield the reference string broken into the colored bibliographic components.
Formally, reference segmentation can be defined as follows [CGK08]: Given a reference string R, it is broken down into a sequence of tokens $\{r_1, r_2, ..., r_n\}$. Then, each of these tokens is to be classified with the correct class from a pre-defined set of classes $C = \{c_1, c_2, ..., c_m\}$. For the classification of the tokens, it is allowed to use the information of the given reference string R, as well as information from previously made classifications.

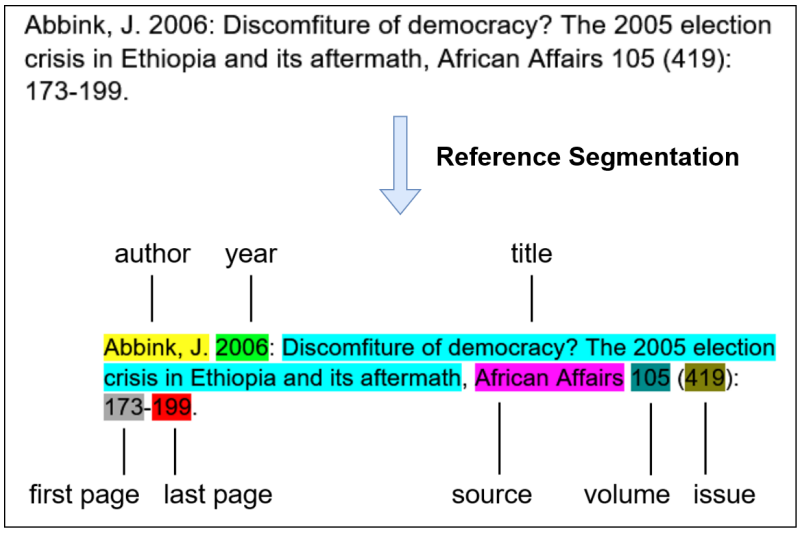

Figure 1.2: Visualization of reference segmentation

1.2 Research Questions

The main research question this thesis aims to investigate and answer is the following:

> *Is BERT a suitable choice for the task of reference extraction and reference segmentation?*

To answer this research question, we examine further sub research questions:

- *Which BERT model and input format achieve the best performance on the task of reference extraction and reference segmentation?*
- *What is the best approach for training a given BERT-based model on the task of reference extraction and reference segmentation?*
- *Can BERT improve the obtained results of EXparser [BAS19] ?*

1.3 Contributions

This bachelor thesis aims to extract and segment references following a deep learning approach using BERT. Furthermore, this thesis is intended to enhance the work done by Boukhers et al. [BAS19] in the scope of EXCITE project [HGKM19] where the EXparser was developed for reference extraction and segmentation. For this, we introduce neural-based embeddings (with BERT) instead of a manual feature engineering approach. The goal is to obtain comparable or better results than the EXParser [BAS19].
The main contributions of this bachelor thesis are as follows:

- Implementation of a neural network architecture based on BERT with a linear-chain Conditional Random Field (CRF) as the output layer for the task of reference extraction and segmentation.
- Conduct of experiments to find out the most suitable BERT model, input format, and the most principled approach for reference extraction and segmentation. Evaluation is done on a dataset that contains both German and English language articles with high-variance references.
- Comparison between the performance of the implemented system and the performance of the EXParser.

1.4 Thesis Outline

The thesis consists of eight chapters. After the introduction, Chapter 2 presents the related prior work in the field of reference extraction and reference segmentation. Then, Chapter 3 introduces the fundamental topics of this thesis. Subsequently, Chapter 4 presents our proposed method for reference extraction and segmentation. In particular, the model architecture used for reference extraction and segmentation is described. Furthermore, the choice of BERT models and input formats are discussed and two different approaches for reference extraction and segmentation are described. Chapter 5 shows the used datasets for the training and evaluation of our models. Then, Chapter 6 presents the results we achieved in different experiments. Additionally, this chapter compares our proposed models to a previously published work: the EXparser [BAS19]. In succession to the results, Chapter 7 interprets the results and reflects on the limitations of this thesis. Moreover, the possible future work in this research area is pointed out. Finally, Chapter 8 concludes the thesis by summarizing the preceding chapters.

2 Related Work

Several works on the extraction and segmentation of reference information from documents are reported in the literature. Generally, the methods for reference extraction and segmentation can be divided into two main categories: *rule-based* or *machine learning* approaches. Because we pursue a machine learning approach with the use of BERT, only prior related works that follow machine learning approaches are described in the following.

Seymore et al. [SMR99] use a Hidden Markov Model (HMM) [RJ86], a statistical machine learning technique, to extract relevant metadata information (including reference components) from the headers of computer science papers. Therefore, they search for information from the beginning of the paper until the first section occurs or the first page ends. Hetzner [Het08] also uses a HMM for reference segmentation and achieves high classification values on many reference components. Furthermore, Support Vector Machines (SVMs) [CV95] and Conditional Random Fields (CRFs) [LMP01] are often employed for the task of reference extraction and segmentation. Zou et al. [ZLT10] implement a SVM classifier to find references in medical articles. Here, text and geometric features are used from HTML medical articles. To segment references, Zou et al. [ZLT10] use both CRF and SVM. They compare the two approaches on their performance on 500 articles from medical journals and conclude that they are equally good.
Cermine [TSF+15] takes a scientific document in Portable Document Format (PDF) as input and outputs both metadata information of the document as well as its bibliography. To produce the outputs, a geometric hierarchical structure of the PDF is constructed beforehand where text content along with geometric features of the displayed text is stored. For the extraction of references, Cermine uses the machine learning algorithm of K-means clustering [Mac+67]. With this algorithm, two groups are formed: the first lines of references and the remaining lines. Clustering is conducted with the use of layout information from the PDFs. To parse the references, CRF are employed.
To extract reference strings, Lopez [Lop09] and Körner et al. [KGM+17] follow the approach of using CRFs instead of SVM or K-means clustering.
The former state-of-the-art reference string extraction and segmentation tool ParsCit [CGK08] uses a heuristic model to extract references from plain text and a CRF model to label the tokens of reference strings. The heuristic model begins searching for labeled reference sections, considering strings such as “References” or “Bibliography”. One of the used heuristics specifies that when such a reference section is found too early in the text, subsequent matches are considered instead. By making use of a whole set of heuristics, references are extracted. Before extracted references are passed to CRF, regular expressions and heuristics are used to find out where individual reference strings start and end.
In Neural-ParsCit [PKK18], a deep-learning approach is applied to parse references. Their model is based on a Long Short-Term Memory (LSTM) [HS97] neural network architecture with a linear CRF layer over the LSTM output. Through the usage of LSTM, long-range dependencies in reference strings are captured by the model and utilized for segmentation. They experiment with word embeddings and character-based word embeddings to represent words in their model. Their

results show a significant performance increase in reference segmentation compared to ParsCit [CGK08].
In the scope of the Excite project [HGKM19], several tools were developed to process references from PDFs. One of the implemented tools is the so-called EXparser [BAS19]. After text from PDF documents is extracted by Cermine [TSF+15], EXparser extracts references from the text and segments them into reference components. For the EXparser, Boukhers et al. [BAS19] use Random Forest Model to extract references and Conditional Random Field to segment references. Both models use a set of features that are extracted using each text line and token. The considered features can be categorized into format-based (e.g. existence of year format), lexical-based (e.g. existence of the keyword *Vol.*), semantic-based (e.g. existence of first or last name), and shape-based features (e.g. ratio of digits in a line/token) [BAS19]. The Random Forest model classifies each line into "non-, first, intermediate, or last reference line" [BAS19, p.5]. Chosen reference string candidates are segmented with CRF into constituent fields such as 'author', 'title', 'year', etc. For the training of the models, text from PDF documents of social science publications is extracted, manually annotated and used. EXparser achieves good classification scores for both reference extraction and segmentation model as the models outperform former state-of-the-art systems such as Cermine [TSF+15] or ParsCit [CGK08].

3 Fundamentals

This chapter gives an overview of the fundamental topics of this thesis. Two concepts of NLP, the Transformer, BERT, CRF, and classification performance metrics are reviewed.

3.1 Relevant Concepts of Natural Language Processing

Natural language processing is a sub-field of computer science dealing with the question of how computers can be used to understand the human language. The goal of NLP is to transform the human language into a formal representation that makes it easy for computers to understand and learn the human language. Well-known NLP tasks are Part-Of-Speech Tagging, Named Entity Recognition (NER) and Language Models.
In the following, we review the concepts of NLP that are relevant for the understanding of this thesis. Therefore, Section 3.1.1 explains tokenization and sentence segmentation, and Section 3.1.2 discusses neural-based word embeddings.

3.1.1 Tokenization and Sentence Segmentation

The field of NLP deals with text data. Without pre-processing, the text data cannot simply be used as input for machine learning models. One of the most important steps in pre-processing the text data is the so-called *tokenization*. It can be regarded as the initial phase in any kind of natural language text preparation [WK92]. The goal of tokenization is to split a text into basic meaningful units called *tokens*. There are different notions of what such tokens can be. The most popular levels of tokenization include tokenization on the word, sub-word, and character level. At each of these levels, the set of unique tokens extracted from the text corpus is referred to as the *vocabulary*.
In word tokenization, the text is split into words based on a particular delimiter. The space between two words is frequently used as the delimiter. For instance, the Word2Vec technique [MCCD13] uses this kind of tokenization.
In sub-word tokenization, the text is split into sub-words. This means that words are further divided here. As an example we consider the words “car” and “cars”. “car” is not split but “cars” is split into the sub-words “car” and “s”. This allows a model to capture the similar semantic meanings of the two words because the root word “car” is identified in both words. Byte Pair Encoding [Gag94] and WordPiece [WSC+16] use sub-word tokenization. Both word and sub-word tokenization suffer from the Out-of-Vocabulary (OOV) problem, where tokens that do not belong to the learned vocabulary are classified as ‘*unknown*’. This leads to a loss of information since a model will not learn anything about the unknown tokens.
In character tokenization, the text is split into characters. The size of the vocabulary is limited here since it consists of a unique set of characters. Hence, this tokenization method eliminates the OOV problem. However, characters usually do not carry the information that a word does. Thus, this

method suffers from a different information loss.
Apart from tokenizing text into minimal units like words or characters, it is also possible to tokenize text into sentences. *Sentence segmentation* or sentence tokenization describes the process of identifying sentences in text data and separating them from each other. Thus, here tokens represent individual sentences. In this context, a sentence can refer to a linguistic sentence but can also be defined as an arbitrary sequence of tokens.

3.1.2 Neural-based Word Embeddings

Word embeddings is a technique where words are converted into a numerical representation such as vectors. In the last decade, word embeddings have established themselves as a core element of many NLP systems. They are essential for learning algorithms like neural networks since an efficient text representation helps such models to understand the meaning of words and the relationship between words better. Word embedding algorithms are based on the assumption that words occurring in similar contexts also have similar meanings [Har54]. Good word embeddings should (a) capture both the syntactic as well as the semantic meaning of a word and (b) model the meaning of a word in different contexts (polysemy) [PNI+18]. For instance, they help improve performance by identifying and grouping similar words. In the following, we discuss word representations computed using neural networks.
Neural-based word embeddings are learned through offline training on large unlabeled text corpora. Interestingly, learned word vectors can represent actual relationships in the real world. For instance, by using simple vector arithmetic, the following is true: "vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector" [MSC+13, p.1]. Popular techniques where word embeddings are learned are Word2Vec [MCCD13] and GloVe [PSM14], which stands for "Global Vectors". Both techniques map words into a vector space where the distance between words is related to their semantic similarity. Word2Vec is a two-layer neural network. There are two types of Word2Vec networks where one is based on Continuous Bag of Words and the other on the Skip-gram model [MSC+13]. As input, Word2Vec takes a large corpus of words. From this, it produces a vector space where every word is represented through a vector. GloVe can be regarded as an extension to Word2Vec that also learns word embeddings efficiently. Unlike Word2Vec, GloVe combines the so-called global matrix factorization methods and local context window methods [PSM14].
One downside of both Word2Vec and GloVe is that their learned word embeddings are context-independent. This means that after training on a corpus, words with multiple meanings are represented as a single independent vector, no matter in which context of a sentence they appear. For example, if there are the following two sentences: "Let us stick to this plan. The dog brings back the stick.". Then, the word "stick" would have different meanings in the two sentences but would be represented by the same word embedding in both sentences when using Word2Vec or GloVe.
In more recent works, contextually-meaningful embeddings such as ELMo (Embeddings from Language Models) [PNI+18] and BERT [DCLT19] have been developed. ELMo uses a LSTM [HS97] neural network architecture and BERT uses a neural network architecture based on the Transformer [VSP+17]. They both learn *contextualized word-embeddings*, where words are represented by different embeddings depending on the semantics they have in the context of a sentence. In general, this improves performance notably on downstream tasks. As both ELMo and BERT are designed using whole sentences as context, Word2Vec or Glove still might use better word representations in tasks where the word contexts are not easily available.

3.2 Transformer

In 2017, a new model architecture called Transformer [VSP+17] was introduced. It solely relies on the so-called *self-attention* mechanism. The idea behind self-attention is to help the model shift its attention to relevant parts of an input sequence. Furthermore, the self-attention mechanism enables the Transformer to capture the relationships between words of an input sequence. Transformers take a sequence of tokens as input and generate an output token sequence. Therefore, Transformers are well-suited for sequence-to-sequence tasks such as translation tasks.
The major difference to former sequence transduction models is that the Transformers input and output representations are computed without using recurrent neural networks or convolutional neural networks. It only uses the attention mechanism. This comes with additional advantages. By only using self-attention, the total computational complexity is reduced and significantly more operations can be executed in parallel. In addition, long-range dependencies in the input sequence are learned efficiently [VSP+17].
This section introduces the key components of the Transformer. In particular, Sections 3.2.3 and 3.2.4 explain the attention mechanism of the Transformer.

3.2.1 Encoder-Decoder Architecture

Most neural transduction models, including Transformers, are based on an encoder-decoder architecture [VSP+17]. An encoder processes a variable-length input sequence $(x_1, ..., x_n)$ (source sentence) and compresses the information into a fixed-length vector representation $z = (z_1, ..., z_n)$ [CMBB14]. Then, z is used as input for the decoder. The decoder produces a variable-length output sequence $(y_1, ..., y_n)$ of symbols which is essentially the target sentence. At each step, previously computed symbols are used as additional input for the computation of the next symbol [VSP+17].

3.2.2 Model Architecture

The general model architecture of the Transformer [VSP+17] can be seen in Figure 3.1. The encoding component consists of six identical encoders and the decoding component consists of six identical decoders. Each encoder is broken down into two sub-layers, where the first layer is a multi-head self-attention mechanism and the second layer employs a fully connected feed-forward neural network. Likewise, each decoder is composed of the same sub-layers as the encoder, with an additional sub-layer. This additional sub-layer is located first in the decoding component and performs masked multi-head attention over the output of the encoder component. Furthermore, each layer in the decoder stack also has access to the output of the encoders. Sub-layers in both encoders and decoders use residual connection along with layer normalization.
Before an input sequence of tokens is fed into a Transformer, the tokens are converted to real-valued vectors through learned embeddings and combined with the positional information of the tokens. For the positional encodings, Vaswani et al. [VSP+17] use sine and cosine functions of different frequencies. Each input and output vector has a size of 512. All vectors of an input sequence flow through the encoder block **in parallel**. The output of the final decoder goes through a linear transformation and then through a softmax function. The softmax operation normalizes the output scores such that they can be interpreted as probabilities. These probabilities are used to predict the next token of the output sequence.

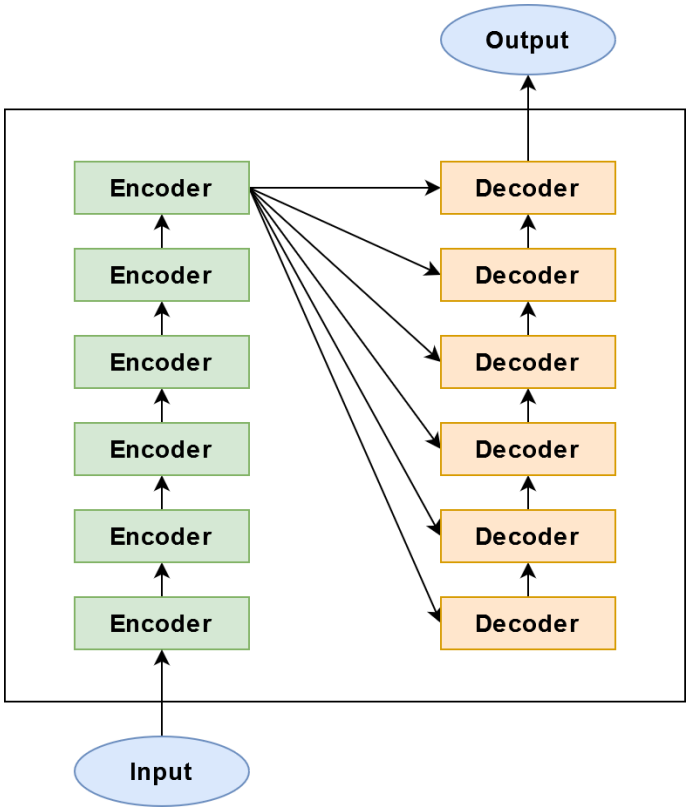

Figure 3.1: The general model architecture of the Transformer [VSP+17]

3.2.3 Self-Attention

Self-attention is a special type of the *attention* mechanism [BCB15]. By relating different words of an input sequence, the self-attention mechanism calculates a contextual representation for each word in the input sequence.
We show the benefits of the self-attention functionality by providing an example. Assume a translation task is to be conducted where the following sentence is to be translated: "John stayed at home because he felt sick.". To translate a sentence it is essential to know how words in the sentence depend on each other. For humans it is easy to understand that the word "he" in the sentence refers to "John", but this is not that easy for computers to recognize. Self-attention allows a model to capture such a dependency between words of a sentence. The mechanism looks for other positions in the input sequence to find indications that can help lead to a better encoding for this word.
Formally, for every term in the input sequence, a query vector q, a key vector k, and a value vector v are created. The vectors are computed by multiplying an initial embedding of the term with three learned weight matrices. All query vectors, key vectors, and value vectors are put into the matrices Q, K, and V, respectively. Then, an attention function computes the following output $Attention(Q,K,V)$ [VSP+17]:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where d_k is the scaling factor.

3.2.4 Multi-Head Attention

The self-attention layer is then further refined by adding the "multi-head" attention mechanism. The idea of multi-head attention is to apply the attention function on the queries, keys and values several times (h times) in parallel instead of only once. The outputs of each attention head are then concatenated and linearly transformed with a weight matrix [VSP+17]:

$$\begin{aligned} MultiHead\text{-}Attention(Q,K,V) &= Concat(head_1,...,head_h)W^O \\ \text{with} \quad head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

where W^O, W_i^Q, W_i^K and W_i^V are all parameter matrices to be learned. Vaswani et al. [VSP+17] state that this multi-head attention mechanism allows the Transformer to "jointly attend to information from different representation subspaces at different positions."[VSP+17, p.5].

3.3 BERT

BERT [DCLT19], which stands for Bidirectional Encoder Representations from Transformers, is a deep contextual language representation model that was introduced by Google in 2018. It has substantially increased performance on many NLP tasks. The distinctive feature of BERT is the way how different contexts of words are captured and used for the representation of individual words. Through the training on large unlabeled text corpora, BERT learns bidirectional text representations. Unlike previous language models, it learns these text representations "by jointly conditioning on both left and right context in all its layers" [DCLT19, p.1]. The pre-trained BERT can be fine-tuned (with little data) to create competitive models on many downstream tasks.
This section introduces BERT and gives an overview of its implementation. Section 3.3.1 presents the model architecture of BERT. Then, Section 3.3.2 describes BERTs input representations and Section 3.3.3 describes the pre-training procedure. Finally, Section 3.3.4 discusses different approaches for applying BERT to downstream tasks.

3.3.1 Model Architecture

BERTs model architecture is based on the deep learning architecture of Transformers [VSP+17] that has been introduced in the previous section. It is composed of multiple Transformer encoder layers that use the multi-head attention mechanism [DCLT19; VSP+17]. There are two versions that BERT comes with:

- **BERT$_{\text{BASE}}$**: L=12, H=768, A=12, Total Parameters: 110 Million (M) and
- **BERT$_{\text{LARGE}}$**: L=24, H=1024, A=16, Total Parameters: 340M.

In the above notation, L refers to the number of Transformer blocks within the model, H represents the hidden size, and A refers to the number of used self-attention heads.

3.3.2 Input Representations

BERT takes a sequence of words as input which is limited to a **maximum sequence length of 512**. These words are converted to a numerical input representation before being passed to the model. This input representation is created through the sum of three different embeddings [DCLT19]: *token*, *segment*, and *position* embeddings, as shown in Figure 3.2.

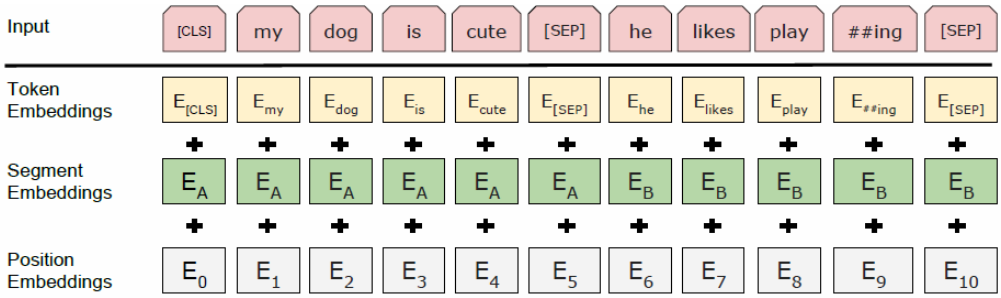

Figure 3.2: BERTs input representation [DCLT19]

Token embeddings are learned through the pre-trained WordPiece model [WSC+16], which generates a vocabulary of fixed-size. Each token in the vocabulary has a unique ID. If a word is tokenized under the WordPiece model, then it is checked first whether the word itself is already in the vocabulary. If not, then it is broken down into the longest *WordPieces* (sub-tokens) that are in the vocabulary. Furthermore, BERT uses special tokens such as *[CLS]* and *[SEP]* [DCLT19]. *[CLS]* is always the first token of an input sequence and *[SEP]* is used to separate sentences within the input sequence from each other. Ultimately, the token embedding is obtained by converting Wordpiece tokens and special tokens of the input sequence to their corresponding IDs.
To distinguish two sentences of the input from each other, BERT also learns segment embeddings for each sentence separately.
Lastly, BERT uses positional embeddings to store positional information of the distinct tokens of an input sequence.

3.3.3 Pre-training of BERT

BERT is pre-trained on two unsupervised tasks, namely Masked Language Model (MLM) and Next Sentence Prediction (NSP). Devlin et al. [DCLT19] show in their ablation studies, pre-training on both these tasks improves the performance of the model significantly on many NLP tasks.
Traditional language models are unidirectional, meaning that the text in such models is only processed in one direction. This limits the information that is captured by the model to this one processing direction. Through bidirectional processing of data which indicates that text is processed from left to right as well as from right to left, BERT overcomes the shortcomings of previous unidirectional language models.
BERT's bidirectionality is trained on the MLM task. In the MLM task, 15% of all WordPiece tokens in the input sequence are masked randomly using a special $[MASK]$-token. Then, the task of the model is to predict the original tokens that were replaced by the masked tokens.
In the task of NSP, the model receives pairs of two sentences and the model's task is to predict whether the second sentence of such a pair is the subsequent sentence to the first sentence in the

original corpus or not. In 50% of the cases, the second sentence is chosen to be the next sentence. In the other 50% of the cases, the second sentence is chosen randomly from the corpus. Training on this task, BERT learns the relationship between sentences which is essential for downstream tasks such as question-answering.

3.3.4 Application of BERT to Downstream Tasks

There exist two different approaches for applying the pre-trained BERT model to downstream tasks: *fine-tuning* and *feature-based* [DCLT19]. In the following, we briefly discuss both strategies.
In the fine-tuning approach, the weights of a pre-trained BERT model are used as initial weights and are optimized during the training of a downstream task. For fine-tuning, only the task-specific inputs and outputs are needed. When models follow the fine-tuning approach, then an additional task-specific output layer is needed on top of the BERT model. This additional layer converts BERTs representations to the desired output. Moreover, fine-tuning is regarded to be computationally inexpensive as it takes much less time compared to pre-training.
In the feature-based approach, fixed features of word representations are extracted from the pre-trained model and are used as inputs to other task-specific architectures (e.g. LSTM, CRF). Here, the weights of the pre-trained BERT model are kept frozen. This approach can have advantages over the fine-tuning approach as some tasks cannot "be easily represented by a Transformer encoder architecture, and therefore require a task-specific model architecture" [DCLT19, p.9].

3.4 Conditional Random Fields

Lafferty et al. [LMP01] introduced Conditional Random Fields (CRFs), a sequence modeling framework to build probabilistic models for the labeling or segmentation of sequential data. Thus, CRFs can be regarded as a classification algorithm. In general, CRFs are suitable for tasks where contextual information or the states of neighbor labels are required for the current prediction. Therefore, it finds its use in many different applications such as Named Entity Recognition (NER) or Part-Of-Speech Tagging.
CRFs overcome the disadvantages of former models like Hidden Markov Models (HMMs) [RJ86] and Maximum Entropy Markov Models (MEMMs) [MFP00]. HMMs make very strict independence assumptions on the sequential data. As CRF is a conditional model, this is not necessary and therefore CRFs relax strong independence assumptions of the observed data [LMP01]. The downside of MEMMs is that they have the so-called *label bias problem*. This problem describes that MEMMs tend to create a bias toward states that have few outgoing transitions [LMP01]. Consequently, such states do not use the information of the observation to its full extent. CRFs solve the *label bias problem* by calculating the joint probability of the whole sequence of labels given the observation sequence [LMP01]. However, the limitation of CRF is that its training is computationally more expensive due to the "slow convergence of the training algorithm" [LMP01, p.10] compared to HMMs or MEMMs.
Formally, CRFs are introduced by defining two random variables X and Y [LMP01]. The random variable X describes sequence data to be labeled. X is always given or observed and can thus be regarded as "input data". Y is a random variable over label sequences. Given a sequence, it can be seen as "output data". For instance, applied to the field of NLP, X might range over the words in a

sentence while Y might range over the labels of the words such as person, location, organization. Here, it should be noted that CRFs follow the Markov property [LMP01]. With both X and Y, we can now state what is modeled with CRFs. We do this on the basis of the most popular form of CRFs which are linear-chain CRFs. A simple visualization of its linear sequence structure is shown in Figure 3.3. CRFs compute the **conditional probability P(Y|X)** of a possible output $Y = (y_1, y_2, ..., y_n)$ given the observation $X = (x_1, x_2, ..., x_n)$.

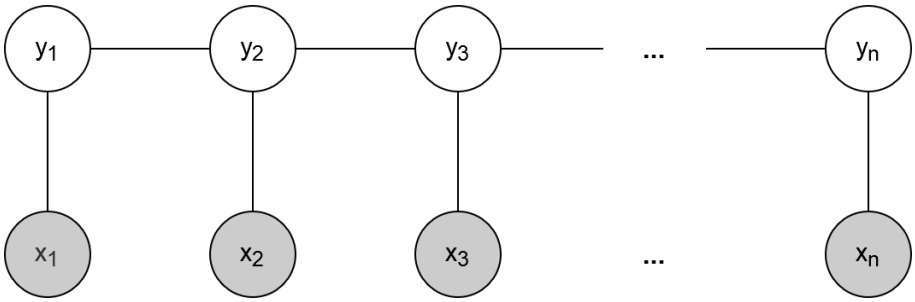

Figure 3.3: Graphical structure of linear-chain CRFs [LMP01]

For computing the conditional probability $P(Y|X)$ in a linear-chain CRF there is a general formula [SM12]:

$$P_\lambda(Y|X) = \frac{1}{Z_\lambda(X)} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i(y_{j-1}, y_j, X, j)\right),$$

where

- λ_i are weights to be trained. Each weight is assigned to a feature function f_i.
- $f_i(y_{j-1}, y_j, X, j)$ is a feature function that takes the states y_{j-1}, y_j, the observation X, and the position j as input. The output of the feature function is a real-valued number where the numbers are often just zero or one.
- $Z_\lambda(X)$ is the normalization given by

$$Z_\lambda(X) = \sum_{Y \in \mathcal{Y}} exp\left(\sum_{j=1}^{n}\sum_{i=1}^{m}\lambda_i f_i(y_{j-1}, y_j, X, j)\right),$$

  where $\mathcal{Y}$ denotes the set of all possible outputs Y. Its values range from zero to one.

To estimate the lambda-weights λ_i, the maximum-likelihood estimation can be applied [LMP01]. When the (optimal) parameters λ_i are estimated, conditional probabilities can be computed and this means it is possible to do inference. Given the observation X, the most likely labeling sequence Y^* can be predicted by the equation

$$Y^* = \arg\max_{Y \in \mathcal{Y}} P(Y|X) \quad .$$

For the computation of Y^*, the Viterbi algorithm can be used [SM12].

3.5 Classification Performance Metrics

There are various metrics that can be used to evaluate the quality of a model on a classification task. This section discusses classification performance metrics relevant to this thesis.
We introduce the different metrics in the context of a binary classification problem. With no loss of generality, the labels are + (positive) and - (negative) in such a setting. Then, a classification of a model belongs to one of the following four combinations of true class and predicted class: True Positives (TP) (correct positive predictions), True Negatives (TN) (correct negative predictions), False Positives (FP) (incorrect positive predictions) and False Negatives (FN) (incorrect negative prediction). The possible predictions can be summarized in a four-cell contingency table (also known as confusion matrix) as shown in Figure 3.4. In the depicted table, the green cells represents correct predictions and the red cells represents incorrect predictions. From this contingency table, many of the common classification metrics can be derived:

		True Class	
		Positive	Negative
Predicted Class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Figure 3.4: Contingency table for a binary classification problem

Precision: The Precision (Pr) denotes the proportion of positive predictions that are correctly true positives as in Equation 3.1.

$$Pr = \frac{TP}{TP + FP} \tag{3.1}$$

Recall: The Recall (Re) denotes the proportion of real positives that are correctly predicted as true positives as in Equation 3.2.

$$Re = \frac{TP}{TP + FN} \tag{3.2}$$

F1-score: The F1-score (F_1) is also called F-measure and is calculated by taking the harmonic mean of precision and recall as can be seen in Equation 3.3.

$$F_1 = \frac{2 \cdot Pr \cdot Re}{Pr + Re} \tag{3.3}$$

The value of the F1-score is between zero and one inclusive. F1-scores near one indicate a high classification performance while values near zero indicate low classification performance.

In what follows, we explain how to compute the presented measures for multiple classes.

Computing TP, TN, FP, FN for a multi-class classification problem is done with the same approach as for a binary classification problem. The multi-class classification problem is just evaluated as if it was a binary classification problem for every class individually. The class for which the measures are to be computed is considered as the positive class while all other classes are considered to be the negative class. We define how the performance measures can be computed for each class.
Suppose we have n classes, where each class is denoted by C_i and the class space is defined by $\{C_1, C_2, ..., C_n\}$. Then, after classification, Pr_i, Re_i, and F_{1_i} denote the precision, recall, and F1-score of class C_i, respectively. They can be calculated as shown in Equation 3.4.

$$Pr_i = \frac{TP_i}{TP_i + FP_i} \quad \text{and} \quad Re_i = \frac{TP_i}{TP_i + FN_i} \tag{3.4}$$

$$F_{1_i} = 2\frac{Pr_i \cdot Re_i}{Pr_i + Re_i} \tag{3.5}$$

In the above equations, TP_i, FP_i, FN_i refer to true positives, false positives, and false negatives of the classification results for class C_i.

We present two options to measure the overall precision and recall for the whole class space, namely *micro-averaging* and *macro-averaging*. In Equations 3.6 and 3.7 it is shown how they are calculated for overall precision and recall. Again, the averaged F1-score can be calculated by taking the harmonic mean of precision and recall of the respective averaging method.

1. Micro-Average:

$$\hat{Pr}^{\mu} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FP_i)} \quad \text{and} \quad \hat{Re}^{\mu} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)} \tag{3.6}$$

2. Macro-Average:

$$\hat{Pr}^{M} = \frac{\sum_{i=1}^{n} Pr_i}{n} \quad \text{and} \quad \hat{Re}^{M} = \frac{\sum_{i=1}^{n} Re_i}{n} \tag{3.7}$$

One can observe that micro-averaging gives equal importance to each classification instance (e.g. documents or images) and macro-averaging gives equal importance to each class. This means, that micro-averaging is preferable if there exists a class imbalance in the used dataset, i.e. one class occurs significantly more often than another class. Conversely, it is reasonable to calculate the macro-average when the classes in the used dataset occur with almost equal frequency. Macro- and micro-averaging produce the same results, if the data used is perfectly balanced.
Moreover, micro-averaged precision is **equal** to micro-averaged recall when including **all** classes for their computation. This is the case because when including all classes in a multi-class setting, all false instances are counted for the computation of micro-averaged precision and recall and it holds true that

$$\sum_{i=1}^{n} FP_i = \sum_{i=1}^{n} FN_i \quad .$$

Consequently, micro-averaged F1-score would also be equal to micro-averaged precision and micro-averaged recall.

4 Proposed Method for Reference Extraction and Segmentation

There are different approaches to fine-tune BERT models for reference extraction and segmentation. In the scope of this thesis, we train[1] models that are based on a BERT-CRF model architecture. The model architecture is explained in Section 4.1. Furthermore, Sections 4.2 and 4.3 discuss the choice of suitable BERT models and the choice of input formats for all models. We experiment with two different approaches for reference extraction and segmentation. Section 4.4 explains the standard two-models approach where one model is trained to extract references and the second model is trained to segment references. Then, Section 4.5 explains how we use a one-model approach for the task of reference extraction and segmentation.

4.1 Model Architecture

The model architecture is based on a BERT-CRF architecture as can be seen in Figure 4.1. It is composed of a BERT model with a linear classifier on top followed by a linear-chain CRF. Similar model architectures have also been used in other works ([HCWC19; LTWX20; ML19; SNA19]). For the following formal definition of our model architecture, we orientate ourselves at the explanations of Lample et al. [LBS+16] and Souza et al. [SNA19].
Given an input sequence $X = (x_1, x_2, ..., x_n)$ of n tokens, BERT generates an output $(h_1, h_2, ..., h_n)$ with hidden dimension H for each output vector h_i. The classification layer maps each of these hidden vectors to the tag space, $\mathbb{R}^H \rightarrow \mathbb{R}^T$, where T is the number of used tags (dimension of the tag space). Consequently, the produced output scores P by the linear layer have the dimension $\mathbb{R}^{n \times T}$. They are passed to the CRF layer. The parameters of the CRF layer are represented by a matrix of tag transitions $A \in \mathbb{R}^{(T+2)\times(T+2)}$. The dimension of the matrix is $(T+2) \times (T+2)$ and not $T \times T$ because in CRFs two additional states are needed, namely the start and end of the sequence. Otherwise, y_0 and y_{n+1} would not be defined for the later score computations. In the matrix A, $A_{i,j}$ denotes the score of the transition from the *i-th* tag to the *j-th* tag. Furthermore, $P_{i,j}$ is the score of the *j-th* tag of the *i-th* word in a sequence. Then, the score for a sequence of predictions $Y = (y_1, y_2, ..., y_n)$ is defined as:

$$score(X, Y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

[1] Note that we do not train any BERT model from scratch (no pre-training) in the scope of this thesis. Thus, "training" with respect to BERT always implies fine-tuning of already pre-trained BERT models.

With the softmax function, one obtains the normalized probability of Y given X:

$$P(Y|X) = \frac{exp\ (score(X,Y))}{\sum_{Y' \in \mathcal{Y}_X} exp\ (score(X,Y'))}$$

, where $\mathcal{Y}_X$ is the set of all possible tag sequences, given the input X.
In the training phase of the model, the log-probability of the correct tag sequence is maximized:

$$log(P(Y|X)) = score(X,Y) - log\left(\sum_{Y' \in \mathcal{Y}_X} exp(score(X,Y'))\right)$$

Finally, we can make predictions by choosing the output sequence Y^* that achieves the maximum score out of all other output sequences:

$$Y^* = \arg\max_{Y \in \mathcal{Y}_X} score(Y|X)$$

The Viterbi algorithm can be used for the computation of Y^*.
Losses and predictions are computed for all WordPiece tokens. However, for every token only the prediction of its first sub-token is taken into account and predictions of further sub-tokens are ignored.

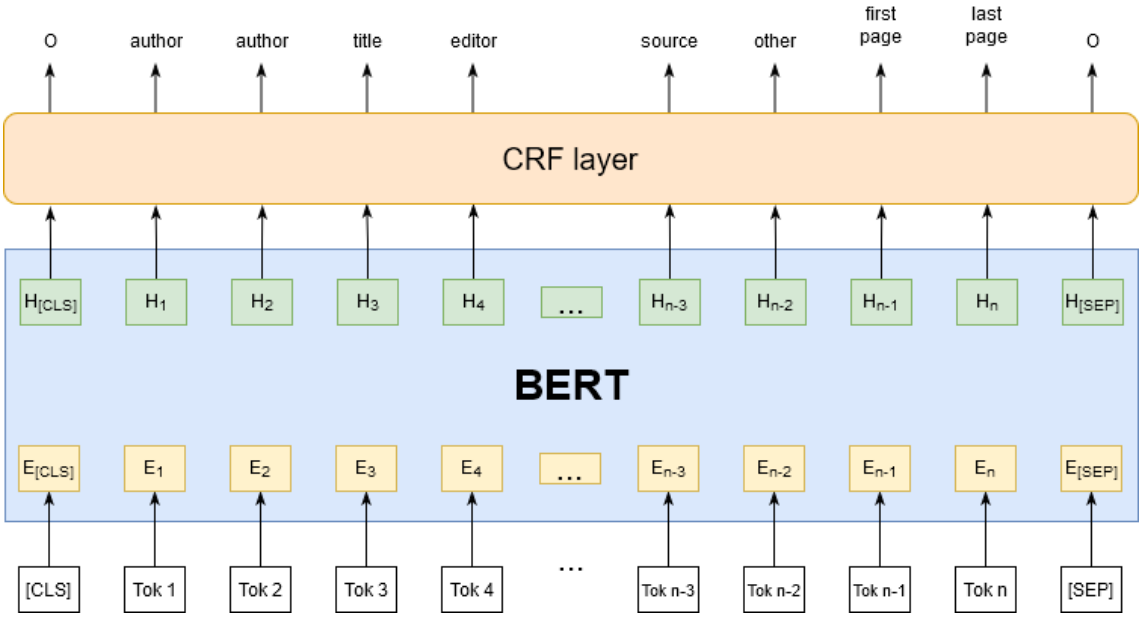

Figure 4.1: The model architecture of BERT-CRF [DCLT19]

4.2 Choice of BERT Model

The choice of the BERT model is crucial. This is because the BERT model returns contextual word representations. The more efficient words of a text can be represented, the better predictions can be made on them.
As different pre-trained BERT models can be used within the model architecture, we list our considered BERT models with their important characteristics:

- **$\text{BERT}_{\text{BASE}}$ cased [DCLT19]**: L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased English text from Wikipedia and BooksCorpus [ZKZ+15].
- **$\text{BERT}_{\text{BASE}}$ multilingual cased[2] [DCLT19]**: L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased text including the top 104 languages. For the text, the largest Wikipedia articles were considered.
- **$\text{BERT}_{\text{BASE}}$ german cased[3]**: L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased German text by Deepset.ai[4].
- **SciBERT cased** [BLC19]: L=12, H=768, A=12, Total Parameters: 110M. This model was trained on cased scientific data from Semantic Scholar[5].

In the above provided list, L is the number of layers, H is the hidden size, and A is the number of attention heads.
Every considered BERT model was pre-trained with data that is similar to our data used (later described in Chapter 5) in some properties. For example, some BERT models were trained on a language that matches the language in our text data. The BERT model *SciBERT cased* was trained with the same type of text data as our used text data (text from scientific articles).
Moreover, all considered pre-trained BERT models have the '*cased*' property. Each of these models also have an analog '*uncased*' version. The '*uncased*' property of a BERT model indicates that during pre-training of the BERT model,

- cased letters were always lowercased and
- accent markers on words were always removed

before the WordPiece tokenization step. The upper two pre-processing steps were not performed for the training of cased BERT models. This means, choosing cased BERT models ensures that a word in the same context of a sentence is represented differently depending on whether it is in lower case or capitalized.
Devlin et al. [DCLT19] used a cased WordPiece model for NER tasks. As both reference extraction and reference segmentation can be reformulated as NER tasks, it is reasonable to use cased BERT models here, too. Using cased BERT model over uncased BERT models for token classification tasks is also in line with prior work [HP19; SNA19]. We state reasons for preferring cased BERT models over uncased BERT models for our task:

- In references, the first word of the title is always capitalized and depending on the reference style used, other words might be capitalized as well. When certain words are always written in lower case and there is a sentence where they are not, then the words are more likely to be inside a reference. We argue that a cased BERT model is able to capture such information.

[2] https://github.com/google-research/bert
[3] https://www.deepset.ai/german-bert
[4] https://www.deepset.ai/
[5] https://www.semanticscholar.org/

- The German language uses accent markers which can appear in the German data that is used for the training of our models. Therefore, it is advantageous to use cased models. For instance, the German word “schön” means “beautiful”. When using an uncased BERT model, it will be converted to “schon” which means “already” in the English language. Thus, a cased BERT model is necessary to preserve the semantics of a German sentence.

To find out the most suitable BERT model for the training on our dataset, we will conduct an experiment including all the considered BERT models.

4.3 Choice of Input Format

Although the general model architecture is already defined and different pre-trained BERT models are considered, there is still another important decision to be made about the models. The efficient representation of tokens of input sequences depends on how input sequences that go into BERT are created. Therefore, the **input format** can have a decisive impact on the training and the resulting performance of the models. This section discusses three different input formats for our model. Therefore, it is important to explain how we decided to create input sequences and on what tokens of the input sequence classifications are made.
BERT takes a sequence of tokens as input. This token sequence is passed to a WordPiece model which may tokenize the tokens into further sub-tokens (WordPiece tokens or WordPieces) [WSC+16]. For example, the token “flightless” is broken down into further sub-tokens “flight” and “##less”. Based on WordPiece tokens, the numerical input representation that goes into BERT is computed. Then, BERT outputs a hidden state for every WordPiece token. This means that for every WordPiece token a prediction can be made. In our model, the CRF layer makes predictions on every WordPiece token. However, some predictions are filtered out as we will explain later. In the further course of this chapter, the words “token” and “WordPiece token” are used interchangeably.
For the input format to BERT, there are different options. Before passing an input sequence to BERT, arbitrary pre-processing steps can be made. For example, available text data can be passed line-wise, sentence-wise, as a randomly shuffled sequence of tokens, etc. Furthermore, the original text of the used data can be manipulated beforehand as this sometimes can come with advantages. Throughout this thesis, we use the terms “sentence” and “linguistic sentence” to refer to a sequence of tokens that form a normal sentence in a language. We chose to construct input sequences based on linguistic sentences due to the following reasons. In the pre-training of BERT [DCLT19], long contiguous sequences are used for the input. Furthermore, BERT was pre-trained on next sentence prediction tasks where sequences are always pairs of sentences. As this implies that BERT was pre-trained with many linguistic sentences and learned the relationship between them, it is meaningful to use linguistic sentences as basic units in the input sequences.
This means that we also split the text inside references into sentences. Although references are texts that do not follow structural and grammatical rules of linguistic sentences, it is still possible to identify positions in a reference, that can be assumed to mark the end of a sentence in a reference. It was also possible to process a whole reference as one linguistic sentence. We decided not to follow this approach because we believed that a model would perform poorly on new text data with this approach. When there are references in the new text data, it is not possible to process every reference as one linguistic sentence, since the new text data is unlabeled and we do not know where references start and end. But some kind of sentence segmentation of the text is required. When a model is always trained with whole references in the input sequences and is then applied to new text

data where input sequences may contain only parts of whole references, this model will probably perform poorly. This would not happen if the text data is always split into linguistic sentences using the same tokenizer, no matter if it is reference text or not.
We construct input sequences using one or more linguistic sentences. For the input sequence, linguistic sentences can be concatenated, overlapped, merged, etc. However, the input length to the model is limited by BERT. The sequence length of a BERT model allows at most 512 WordPiece tokens as input. This implies that when input sequences have more WordPiece tokens than the specified sequence length of the BERT model, the token sequence is truncated to the sequence length. If one linguistic sentence is tokenized by BERTs WordPiece model and is then already composed of more WordPiece tokens than the sequence length seq_len, then the tokens that come after the seq_len-*th* token do not go into computation and are consequently also not predicted. However, this case only occurs very rarely as linguistic sentences are usually not that long.
Before input sequences are passed to BERT, they are converted into a numerical representation. Therefore, they undergo necessary pre-processing steps. A detailed explanation of the different steps can be found in Appendix A.
We want to find out the most principled input format to our model as we seek to achieve high classification performance. Therefore, the upcoming sections discuss three different model input formats based on linguistic sentences.

4.3.1 *Single* Input Format

This input format has the simplest characteristics. Here, we choose to pass the text sentence-wise to the model. This means, from the available text data, sentences are extracted (e.g. by a pre-trained sentence segmentation model). Each sentence goes as input to the model separately. In the further course of this thesis, we refer to this input format as the *single* input format.
Passing text sentence-wise lets the BERT model receive no further context for the classification of the tokens of a sentence. It only has the information of the sentence itself.
However, the input format benefits from the fast training of the model. The reason for this is that the sequence length can be kept short and the shorter the sequence length, the faster the model trains.

4.3.2 *Maximum Context* Input Format

The drawback of the last input format can be overcome by providing the model with **more context around a sentence**, instead of solely creating inputs sentence by sentence. Therefore, the following input format implements this idea. The input format of one input instance is created as follows:
At first, sentences are extracted from the text data. Other than the previous model, sentences are then concatenated until the maximum sequence length (512 WordPiece tokens) is reached. The concatenation of sentences stops as soon as adding the next sentence would lead to a sequence length greater than the maximum sequence length. The sentence that would have led to a sequence length greater than the maximum sequence length is then used as the first sentence of the next input sequence. We refer to this input format as the *maximum context* input format.
Using this input format, each sentence that goes into the model has context around it. The model benefits from this input format, since it can utilize much more information for the prediction of tokens.
A possible problem with this input format is that sentences that are right-most or left-most only

have one-sided context. This can be problematic when, for example, the right-most sentence is the beginning of a reference string and all sentences before are plain text. In this case, the additional context for the sentence that is a reference can drag the model more towards the assumption that the right-most sentence is also just plain text and not part of a reference string.
Beyond that, training of the model lasts considerably longer as the maximum sequence length is used.

4.3.3 *CMV-based* Input Format

The third and most complex input format preserves the idea of having context for each sentence but also counteracts the stated problem of the previous input format (section 4.3.2). The idea here is to concatenate sentences again with the inclusion of **overlapping sentences**. This way, each sentence is passed to the model multiple times, depending on how and how many sentences are overlapped. We limit the number of sentences being concatenated to the number of three. In what follows, we describe the resulting input format formally.
Assume a total of $n \geq 3$ sentences are extracted from the text data and $\mathcal{S} = \{s_1, s_2, s_3, ..., s_n\}$ denotes the set of all extracted sentences where sentences are assigned a unique number i ordered by the order of their appearance in the text data. For an arbitrary sentence s_i, we define the subsequent sentence to be s_{i+1} for $i \in 1, ..., n-1$ and to be s_1 if $i = n$. Then our desired input format $\mathcal{I}$ is obtained by concatenating three consecutive sentences together and by overlapping two consecutive input samples. Overlapping is implemented using the following rule. Given an input sample, the next input sample is obtained by removing the first sentence of the current input sample and adding the sentence that would chronologically come after the last sentence of the current input sample. Consequently, $\mathcal{I}$ is defined as follows:

$$\begin{aligned}\mathcal{I} &= \{concat(s_i, s_{i+1}, s_{i+2}) \quad | \quad i \in \{1, ..., n-2\}\} \quad \cup \\ &\quad \{concat(s_{n-1}, s_n, s_1), concat(s_n, s_1, s_2)\} \\ &= \{concat(s_1, s_2, s_3), ..., \\ &\quad concat(s_{n-2}, s_{n-1}, s_n), concat(s_{n-1}, s_n, s_1), concat(s_n, s_1, s_2)\}\end{aligned}$$

As one can see, in two consecutive input samples of $\mathcal{I}$, the last two sentences of the first input sample and the first two sentences of the second input sample overlap.
Because one individual token may be classified up to three times here, determining the prediction for one individual token is more complex here. In a perfect scenario, every extracted sentence is passed to the model three times at three different positions with different contexts. From this follows that every token of a sentence is classified three times. The final prediction of a token is decided by taking the majority vote of the token's three predictions that are made at the three different positions with different contexts.
However, the input sequence length of our model is limited to the maximum length of 512 WordPiece tokens [DCLT19]. When the maximum sequence length is exceeded after WordPiece tokenization, WordPiece tokens are truncated from the left of the input sample until the input length fits the maximum sequence length. It is possible that two sentences (or even one) already reach the maximum sequence length. In such edge cases, taking a majority vote is not possible as there would be less than three predictions for one token. Therefore, we define the final classification of a token as follows:

- If there are three classifications for a token, the final classification of that token is computed by taking the majority vote of the three classifications (cf. Figure 4.2). If the three classifications are all different, then the classification of the token from the sentence that is located between two sentences in an input sample is the final classification. We argue that taking this classification is better than the other two classifications, because the classification was made on the token that is located in a sentence that has prior and subsequent context.
- If there are only two classifications for a token, then the classification of the token from the sentence that is located second in an input sample is the final classification. For the classification of a token, we argue that the prior context of a token is more informative than the following context to a token.
- If there is only one classification for a token, then this classification is also its final classification.

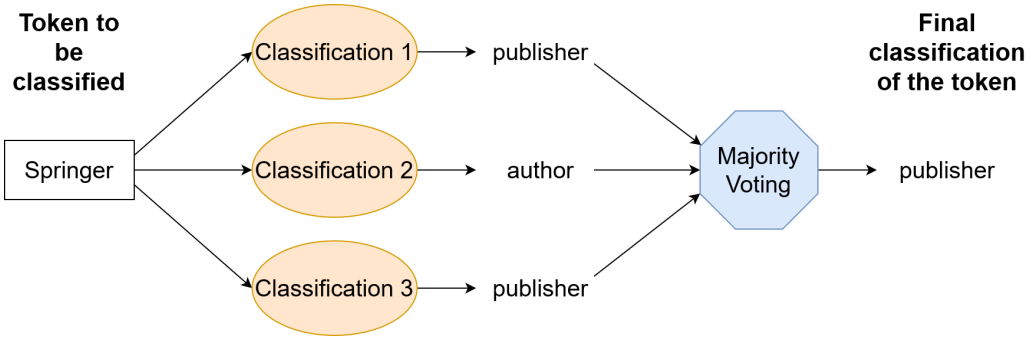

Figure 4.2: Idea of Contextual Majority Voting [LP20]

This described method is based on the idea of Contextual Majority Voting (CMV) that was proposed by Luoma and Pyysalo [LP20]. Hence, we refer to the just described input format as the *CMV*-based input format.
In this format, the model highly benefits from having almost always **right context and left context for each sentence**. Nearly in all cases, a sentence is located once at the beginning, once in the middle between the first and last sentence, and once at the end of an input sample. This means, nearly every sentence is located once between two surrounding sentences. This can be helpful to the model when, for example, the first and last sentence have indications of a reference string. Because then, it is more likely that the sentence located between them is also part of a reference string. That is one of the reasons why this method is expected to increase the model's performance on reference extraction and segmentation compared to other input formats.
The downside of this input format is that the amount of sentences (total data) that goes into the model is doubled. In other words, redundancy is added. Additionally, the computation of the prediction of a word is much more complex.

We briefly summarize the presented input formats with their different characteristics in Table 4.1. Later in the results chapter, the presented input formats will be compared against each other's performance when applied to the model.

	Single	Maximum context	CMV-based
Creation of an input sequence	single sentences	filling sentences up to the maximum sequence length of BERT	concatenating three consecutive sentences with the inclusion of overlapping sentences
Decision of the final prediction of a WordPiece token	as predicted by model (CRF layer)	as predicted by model (CRF layer)	applying Contextual Majority Voting

Table 4.1: Summary of the input formats

4.4 Standard Two-Models Approach

Our first approach to the problem of reference extraction and segmentation is to train two separate models: one model for reference extraction and one model for reference segmentation. Both models are based on the same model architecture presented in section 4.1 but are trained with different data as they pursue different tasks. The reference extraction model is trained with plain text and annotated references. This serves the purpose that the model learns the difference between normal text and text in references. Classifications are made on the tokens of a text. The reference extraction model is used to classify each token into either: non-, first, or intermediate reference token. The classified tokens are used afterward to compose reference strings. This is done by concatenating a classified first reference token with subsequent classified intermediate reference tokens until a non-reference token or a first reference token is reached.
To avoid the influence of former phases on the evaluation of the reference segmentation model, the references are assumed to be correctly extracted. Consequently, the segmentation model is trained with reference strings that are segmented into constituent reference components. Given a reference string, the model learns to divide a reference into its different parts. The segmentation model is used to classify each token of its input into components such as ‘author’, ‘title’, ‘year’, etc. The classified tokens are used afterward to compose reference components. To do this, tokens that are classified into the same component and appear in sequence in the input are concatenated. When multiple reference strings are passed to the segmentation model at once, it is assumed that the first and last token of every reference string is already known through the reference extraction model. Thus, reference strings do not need to be reconstructed after classification.
The standard two-models approach benefits from having two models that are trained for two different tasks. The reference extraction model can be optimized to extract references better whereas the reference segmentation model can be optimized to segment references better into their bibliographic components.
The disadvantage is that the training of two models generally takes more time.

4.5 One-Model Approach

The task of reference extraction and reference segmentation is often tackled by training two models. However, it is also possible to train only **one model that combines the tasks of both reference extraction and segmentation**. To achieve this, the model is trained with plain text and already segmented references. The idea of such a model is to directly classify tokens of references into reference components such as 'author', 'title', 'year', etc. When the model classifies tokens into bibliographic components, this indicates that these tokens are part of a reference string according to the model. When tokens are not part of a reference string, they are classified as non-reference. The classified tokens are used afterward to compose reference components. This is done by concatenating tokens that are classified into the same component and appear in sequence in the input.

This model has the advantage that the task of both reference extraction and segmentation is solved with the training of only one model. This often saves time and memory. Moreover, when this model is implemented in a real application, raw text data must only be passed to one model for immediate feedback on what parts of the text data belong to references and of what components these references are composed of.

The drawback of the described model is that it is harder to compose reference strings after classification. This is because we do not segment into 'first reference token' or similar. When a set of reference strings is passed to this model, the beginning and end of a reference are not clearly evident. For this reason, a set of heuristics need to be defined which makes this model less practical.

It is difficult to compare the reference extraction performance of this approach to the reference extraction performance of the previous approach (4.4). However, this approach can be compared to the segmentation model of the two-models approach as they share all classification components except for the additional non-reference component.

4.6 Implementation

The implementation for this thesis is written in the Python programming language (with Python version 3.9.7). The training of the models is implemented using the deep learning framework *PyTorch* [PGM+19] as well as HuggingFace's *Transformers* library [WDS+20]. The linear-chain CRF is implemented with the use of the Pytorch-CRF[6] module. For the sentence segmentation and tokenization of the data, the *spaCy*[7] module was used. In the *spaCy* module, there are pre-trained models that provide both the functionality of sentence segmentation and tokenization, but often for only one language. As a consequence, we used two of such *spaCy* models, one for the documents in the German language and one for the documents in the English language of our data used. To evaluate the performance of our models on different classification metrics (precision, recall, and F1-score), we use the Python module *Scikit-learn* [PVG+11] as well as the Python *seqeval* package[8]. Further implementation details of our method are in Appendix B.
The code of this thesis can be found at GitLab[9].

[6] https://github.com/kmkurn/pytorch-crf
[7] https://github.com/explosion/spaCy
[8] https://github.com/chakki-works/seqeval
[9] https://gitlab-ac.informatik.uni-stuttgart.de/iurshiaa/bert-references

5 Data

In this chapter, we describe the data used for training and evaluation of the models. Therefore, the different annotations of the data are described in Section 5.1 and in Section 5.2 it is explained how the data are prepared to be used for our models.

To train and evaluate the different models that were presented in Chapter 4, text data is needed. For the training of our models, we use the provided data from the *EXgoldstandard*[1]. The dataset contains altogether 350 articles in PDF format that were collected from Social Science Open Access Repository (SSOAR)[2]. Boukhers et al. [BAS19] extracted the lines of the articles using Cermine [TSF+15] and then annotated them for the task of reference extraction and reference segmentation. The dataset contains high-variance references which is the main reason why it can be regarded as a challenging dataset. In total, there are 9141 reference strings in the dataset. When the text data is segmented into sentences and split into tokens, then about 137,400 sentences and 3,478,000 tokens are counted. Depending on the language of the documents inside the dataset, they are divided into two categories: documents in German language and documents in English language. In the following, we describe these two categories in more detail.

- **German articles:**
  This category consists of 251 annotated articles in the German language. They are about social science and can be divided into three types of articles: 1) 219 articles have located their references in a specific section at the end of the document and 2) 12 papers have references in a section at the end of the document as well as in footnotes 3) the references in the remaining 20 articles are located somewhere in the document (e.g. footnotes) and not in a specific reference section. After our own investigation, we identified that one document appears in two of the three document categories. Thus, in total there are 250 annotated unique articles in the German language.
  The articles not only include academic literature, but also grey literature [BAS19]. This implies that untypical references appear more often.

- **English articles:**
  This second category consists of 100 annotated articles in the English language. Other than in the German articles, references are from different languages in the English articles. In all 100 articles, the references are located only in a specific reference section at the end of the document.

Different datasets are derived from this complete dataset for the training and evaluation of the different models. That is why we introduce different abbreviations for each dataset. In the further course of this thesis, we refer to the just described dataset with its 350 articles as the **Complete Dataset (CD)**.

[1] https://github.com/exciteproject/EXgoldstandard/tree/master/Goldstandard_EXparser
[2] https://www.gesis.org/ssoar/home

All our models (extraction models, segmentation models, combined extraction and segmentation model) are trained and evaluated on CD, except for the extraction and segmentation models that are trained for the comparison to EXparser [BAS19]. For the training and evaluation of those models, we use the Proposed Dataset in German language (PGS) [BAS19] and the Proposed Dataset in English language (PES) [BAS19], the same datasets that were used for training and evaluation of EXparser.

5.1 Data Annotations

In CD, there are two different annotations of the documents, where one can be used for the training of extraction models and the other for the training of segmentation models.
In the first annotation format of the documents, the text that does not belong to references remains unchanged. Each reference is enclosed by a *ref*-tag. This annotation format is suitable for the training of the extraction model because the model sees both normal text and reference lines during training. Consequently, it learns to distinguish normal text from reference text. We denote the data in this annotation format by CD_e. In the following, we show an example of an annotated reference in CD_e:

> **<ref>**Abbink, J. 2006: Discomfiture of democracy? The 2005 election crisis in Ethiopia and its aftermath, African Affairs 105 (419): 173-199.**</ref>**

In the second annotation format, text that is not part of reference strings is removed. References are already extracted and segmented into reference components. Each reference component is enclosed by its corresponding *label*-tag. It is reasonable to use this annotation format for the training of a segmentation model as plain text is removed and the model only learns segmentation of references. We denote the data in this annotation format by CD_s. We identified inconsistencies in the data annotation of CD_s that we have manually corrected. Detailed information about the modified data files can be found in Appendix C. In CD_s, the example reference string from above looks like follows:

> **<author><surname>**Abbink**</surname>**,**<given-names>**J.**</given-names></author> <year>**2006**</year>**: **<title>**Discomfiture of democracy? The 2005 election crisis in Ethiopia and its aftermath**</title>**,**<source>** African Affairs**</source> <volume>**105**</volume>** (**<issue>**419**</issue>**): **<fpage>**173**</fpage>**-**<lpage>**199**</lpage>**.

A combination of CD_e and CD_s yields data that is useful for the training of the proposed combined extraction and segmentation model from the one-model approach. In this annotation format, plain text is present in the data and references are already segmented and not only enclosed by a *ref*-tag. The data in this annotation format is annotated by CD_c.

5.2 Data Preparation

To prepare our data for input sequences to our model, we tokenized CD_e, CD_s, and CD_c and assigned every token its corresponding label in the data. Tokens that were not annotated by any label are tagged with self-introduced tags which we will explain. All sub-tokens of a token (when tokenized by a WordPiece model) are assigned the same label as the original token in our data.
To structure the data and store additional information for every token, we considered the Comma-separated Values (CSV) as a suitable file format. In the CSV file, each line consists of four values: line number, sentence number of the token, the token itself, and the label of the token. In Figure 5.1, it is demonstrated how an annotated reference from CD_s is represented in the CSV file. Linguistic sentences are created by taking all tokens with the same sentence number in their order of appearance in the file.

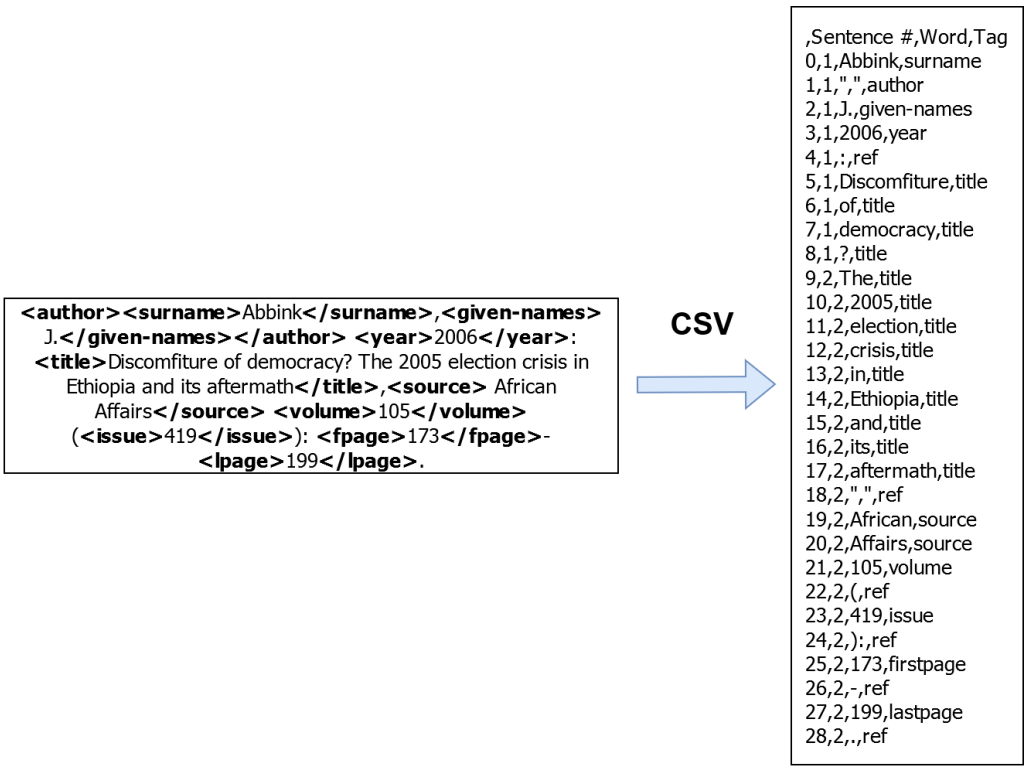

Figure 5.1: Visualization of how the annotated text data is converted to the CSV file format

In the following, we describe the labels used and tagging format for CD_e, CD_s, and CD_c, respectively. They are in line with our described method from Sections 4.4 and 4.5.

5.2.1 Data Preparation for Reference Extraction

For reference extraction we use CD_e. Here, we distinguish between two types of tokens: Tokens that belong to plain text (O-tag) and tokens that belong to references (ref-tag). The Beginning-Inside-Outside (BIO) tagging format is a suitable format for the efficient tagging of the tokens of a reference. When applying here, the set of labels consists of B-REF, I-REF, and O.

B-REF is used for the first token of a reference string, I-REF is used for intermediate tokens of a reference string, and O is used for all other tokens. Our used tagging method for reference extraction is in line with prior related work [KGM+17].

5.2.2 Data Preparation for Reference Segmentation

The dataset CD_s is used for reference segmentation. In this annotation format, we distinguish between tokens of different reference components, where the reference components are also the used labels. Tokens are assigned the label that they are enclosed by in the data. However, there are two edge cases:

- When a token is not enclosed by any tag, it is assigned the label 'ref'. The label 'ref' stands for "reference". We decided to assign this label to tokens that have no specific segmentation label, but are still part of a reference. This was often the case for punctuation characters, e.g. '. : ; - [ ] { }'.
- When a token is enclosed by an *author* tag, but inside of the enclosed author tags it is neither enclosed by *surname* tags nor *given-names* tags, then it gets the label '*author*'. This was the case for punctuation characters inside author tags, e.g. a comma that separates the surname from the given name of an author.

The following reference components are used: publisher, first-page, last-page, title, Uniform Resource Locator (URL), author, surname, given-name, volume, source, editor, identifier, year, other, ref.

5.2.3 Data Preparation for the Combination of Reference Extraction and Segmentation

Finally, CD_c is used for the one-model approach, where reference extraction and reference segmentation are combined in one model. Here, the same labels are used as in the previous section (5.2.2) with the additional *O*-tag. This is because the model is trained with plain text and already segmented references. Tokens of the plain text are assigned the *O*-tag.

In Table 5.1, we summarize all labels that are used for the training of the different models.

	Extraction model	Segmentation model	Combined extraction and segmentation model
Used labels	B-ref, I-ref, O	publisher, first-page, last-page, title, URL, author, surname, given-name, volume, source, editor, identifier, year, other, ref	publisher, first-page, last-page, title, URL, author, surname, given-name, volume, source, editor, identifier, year, other, ref, O

Table 5.1: Labels used for the training of the different models

6 Results

This chapter presents the results. First, Section 6.1 describes the conducted evaluation procedure for the results. Subsequently, the results of the carried out experiments are shown in Section 6.2. Section 6.3 compares our proposed models to EXparser [BAS19].

6.1 Evaluation Procedure

The evaluation procedure comprises the used evaluation metrics, the calculation of the results and the model selection during training time.

6.1.1 Evaluation Metrics

The performance of the Extraction and Segmentation models are evaluated based on the metrics of precision, recall, and F1-score. The averaged F1-score is mainly used to evaluate the performance. Models are mainly evaluated on the (micro-averaged) F1-score because it has been shown in the literature that the metrics precision and recall alone are not very effective for measuring classification performance [Seb02]. For this reason, the combination of precision and recall has been used to measure the performance of classification tasks. The most popular combination is to take the harmonic mean between precision and recall which is known as the F1-score.

6.1.2 Calculation of the Results

In what follows, we state how the results are calculated for both the reference extraction models and the reference segmentation models.
Given an input sequence, it is tokenized by the underlying WordPiece model of a selected BERT model. It is of high importance that each tokens prediction in the original input sequence is determined by taking into account only the prediction of its first sub-token. Thus, for the prediction of the word "flightless", only the prediction of its first sub-token "flight" is taken into account. The prediction of further sub-tokens of a word (here: "##less") is ignored. This is almost in line with Devlin *et al.'s* [DCLT19] approach when they fine-tuned BERT on NER tasks. The only difference is that Devlin et al. [DCLT19] do not pass sub-tokens that are not the first sub-token of a word as input to the classifier (early filter out). In our implementation, all sub-tokens are passed to the classifier but their predictions are filtered out later, based on what sub-tokens are first sub-tokens of words (late filter out).
Depending on the model, the results are calculated differently. In the following explanation, the term "token" always refers to the first sub-token of a word (WordPiece token) after the word has been tokenized by a WordPiece model:

- In the data for reference extraction, we used the BIO tagging format which is a common tagging scheme for the task of NER. For the evaluation of the reference extraction models, we treat reference strings as named entities. Such a named entity is composed of a first B-*ref* tag and subsequent I-*ref* tags. We evaluate on the **entity-level** which means that a reference string is only considered to be classified correctly by the model if and only if all its tokens are classified as *reference*. Therefore, the first token of a reference string needs to be predicted as B-*ref* and all subsequent tokens as I-*ref*.
  Furthermore, the tag *O* is not considered in the calculation of the results since it is not a meaningful tag. This is because (a) it annotates plain text and (b) if the model was to predict every token with the tag *O*, still high performance would be achieved for the tag *O* (because it is by far the most dominant tag in the annotated data).
- The segmentation models are evaluated on the **token-level** and not on the entity-level. When, for example, the 'title' component of a reference string consists of multiple tokens, then the individual classification of every 'title' token goes into the calculation of the metrics. This means we evaluate the segmentation models less strictly than the extraction models.
- The combined extraction and segmentation model is also evaluated on the **token-level**. Again, the *O*-tag is not considered in the calculation of the results because of the aforementioned reasons.

Lastly, the results presented are all average performance, using 10-fold cross-validation which we describe in the upcoming section.

6.1.3 Model Selection

We have a limited amount of data that we use in the scope of this thesis. Therefore, the approach of splitting the data into one training and one testing set and solely evaluating the model on its performance on the testing set is not reasonable. Due to that, we conduct a 10-fold cross-validation on all of our proposed models. Therefore, we split the data CD such that 80% are used for training, 10% for validation, and the remaining 10% for testing. The validation set is used for model selection and the results are reported on the test set. Model selection describes the process of choosing the model's "best state" during the training time based on its performance on the validation set. A model is applied to the validation set after every epoch where one epoch implies one pass of the entire training set that the model has completed. After applying the test set on the 10 different models, the results are averaged with different averaging methods such as micro-averaging or macro-averaging. The averaged results are the reported final results for each shown model.
For the comparison to EXparser's models, we perform 10-fold cross-validation using the same datasets (PGS and PES) and folds as Boukhers et al. [BAS19].

6.2 Experiments

This section presents the results of our conducted experiments. First, we compare the performance of the considered BERT models in a reference segmentation task. Then, for the next experiments, we take the BERT model with the best performance in the first experiment and examine the performance of different input formats applied to both reference extraction models and reference segmentation

models. Furthermore, the results of the one-model approach are presented. For the experiments, the extraction models, segmentation models, and the combined reference extraction and segmentation model were trained and evaluated on the datasets CD_e, CD_s, and CD_c, respectively.

6.2.1 BERT Model

Figure 6.1 compares the performance (macro- and micro-averaged F1-score) of segmentation models with different pre-trained BERT models but the same input format *single*. What stands out in the diagram is that the model with $BERT_{BASE}$ *multilingual cased* achieved the highest performance with a macro-averaged F1-score of 89.5% and a micro-averaged F1-score of 91,9%. However, there is no significant performance difference between $BERT_{BASE}$ *multilingual cased* and *SciBERT cased* or *BERT base cased*. Only the model that used $BERT_{BASE}$ *german cased* performs relatively poorly.

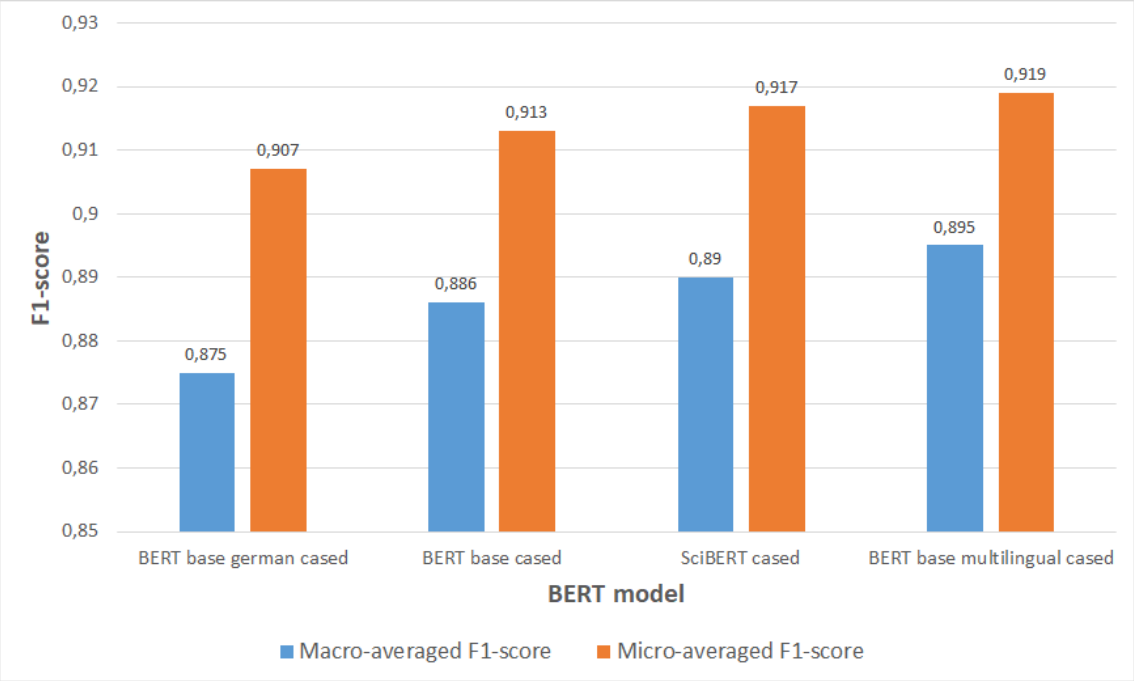

Figure 6.1: Results of reference segmentation models with different BERT models

6.2.2 Input Format

The following models were all trained with $BERT_{BASE}$ *multilingual cased* as this BERT model achieved the best results in the previous experiment.

We start by comparing the extraction models of the two-models approach. Here, the results indicate how well every model extracted reference strings. Table 6.1 presents the results of the different extraction models in terms of the three micro-averaged evaluation metrics precision, recall and F1-score. Here, precision measures the proportion of relevant reference strings among the

retrieved ones. Recall measures the proportion of retrieved reference strings among the total amount of reference strings in the dataset. We denote a model with its input format as *input format* model. As demonstrated, the *maximum context* model achieves the best performance on precision, while the *CMV-based* model achieves the best results for both recall and F1-score. However, there is no significant difference between the performance of the *maximum context* model and the *CMV-based* model. Compared to the results of both these models, the *single model* performs poorly.

	Precision	Recall	F1-score
single	0.598	0.773	0.674
maximum context	**0.800**	0.832	0.816
CMV	0.789	**0.851**	**0.819**

Table 6.1: Results of the reference extraction models with different input formats (boldly printed numbers indicate the best result in the respective column)

Next, we compare the segmentation models of the two-models approach. Table 6.2 demonstrates the results of the different segmentation models on all components of a reference (labels used) as well as their micro-averaged and macro-averaged metrics. The results point out how well every model segmented given reference strings into their constituent components. For one particular reference component C, precision measures the proportion of relevant reference components C among the retrieved ones. Recall measures the proportion of retrieved reference components C among the total amount of reference components C in the dataset. In the table, the score of the 'Author' component is computed by averaging the classification scores of the tags 'surname', 'given-name', and 'author'. Consequently, macro-average is calculated here by averaging the results of all reference components except for 'Author Surname' and 'Author Given-name' as they are already included in the calculation of the 'Author' component.
The results show that the segmentation model with the *CMV-based* input format achieves the best results overall. Nearly all reference components are best classified by the model the *CMV-based* input format and it has the highest macro- as well as micro-averaged scores. However, when comparing the different micro-averages of the models, the results of the *CMV-based* model is only ahead by 0.1 percentage points compared to the *maximum context* model. Even the segmentation model with the *single* input format does not perform poorly here as there is only a difference of 1.7 percentage points between it and the *CMV-based* model. In terms of macro-averaged F1-score, the *CMV-based* model is ahead by 0.5 percentage points than the *maximum context* model. All models perform very well on the segmentation of the components 'author', 'title', and 'URL' but perform relatively poorly on the segmentation of the components 'volume', 'issue', and 'other'. Interestingly, it can be observed that the *single* model classifies URLs best since its F1-score is the highest among all models on this tag. The *maximum context* model achieves the best classification performance on components such as 'title' or 'source' but only by a very small margin.
In summary, these results show that our best results on the dataset CD are attained through the BERT model $BERT_{BASE}$ *multilingual cased* and with the input format *CMV-based* for both reference extraction and reference segmentation. However, the results also indicate that applying the *maximum context* input format can be regarded as equally good.
In the following, we discuss the obtained results for the combined extraction and segmentation model in Table 6.3. The results are particularly interesting because only one model was trained to simultaneously extract and segment references into their components. What stands out in the table is that the 'title', 'author', and 'editor' components are classified very well while the 'volume',

	Precision			Recall			F1-score		
	single	maximum context	CMV	single	maximum context	CMV	single	maximum context	CMV
Publisher	**0.927**	0.916	0.925	0.925	**0.938**	0.936	0.926	0.926	**0.931**
First Page	**0.928**	0.918	0.921	0.831	0.821	**0.842**	0.877	0.867	**0.88**
Last Page	0.923	0.904	**0.93**	0.913	0.9	**0.922**	0.918	0.902	**0.926**
Title	0.934	**0.967**	0.964	0.932	**0.959**	**0.959**	0.933	**0.963**	0.961
URL	**0.961**	0.945	**0.961**	**0.991**	0.986	0.983	**0.976**	0.965	0.972
Author	0.957	0.959	**0.96**	**0.974**	0.973	0.972	0.965	**0.966**	**0.966**
Author Surname	0.945	0.946	**0.95**	0.943	**0.949**	0.946	0.944	**0.948**	**0.948**
Author Given-name	0.94	0.936	**0.943**	0.955	**0.959**	0.956	0.948	0.947	**0.949**
Volume	0.772	**0.812**	0.795	0.768	0.766	**0.785**	0.77	0.788	**0.79**
Source	0.826	**0.886**	0.882	0.796	**0.874**	0.87	0.811	**0.88**	0.876
Editor	0.907	0.916	**0.917**	0.936	0.942	**0.946**	0.922	0.929	**0.931**
Identifier	0.899	0.872	**0.905**	0.898	0.88	**0.91**	0.899	0.876	**0.907**
Year	0.921	**0.938**	0.922	0.904	0.905	**0.915**	0.913	**0.921**	0.918
Issue	0.726	0.717	**0.736**	0.79	**0.809**	0.799	0.757	0.76	**0.766**
Other	0.842	**0.856**	0.849	0.801	0.8	**0.805**	0.821	**0.827**	**0.827**
Ref	0.936	0.942	**0.945**	0.947	0.951	**0.952**	0.942	0.947	**0.949**
macro Average	0.897	0.902	**0.906**	0.894	0.901	**0.906**	0.895	0.901	**0.906**
micro Average	0.919	0.935	**0.936**	0.919	0.935	**0.936**	0.919	0.935	**0.936**

Table 6.2: Results of the reference segmentation models with different input formats (boldly printed numbers indicate the best result in the row per metric)

'issue', and 'other' components were rather predicted imprecise. This observation also applied to the obtained results of the segmentation models in the two-models approach (cf. Table 6.2). The model achieves a macro-averaged F1-score of 87% and a micro-averaged F1-score of nearly 91%, only about 3 percentage points lower than the same values in Table 6.2.
Overall, the results show that the segmentation models in the two-models approach segment references substantially better than the combined reference extraction and segmentation model. On the dataset CD_e, our best reference extraction model reaches a micro-averaged F1-score of 81,9%. And on the dataset CD_s, our best reference segmentation model reaches a micro-averaged F1-score of 93.6%.

	Precision	Recall	F1-score
Publisher	0.895	0.911	0.903
First Page	0.88	0.826	0.852
Last Page	0.903	0.912	0.907
Title	0.925	0.948	0.936
URL	0.885	0.92	0.902
Author	0.923	0.945	0.934
Author Surname	0.914	0.922	0.918
Author Given-name	0.924	0.951	0.937
Volume	0.748	0.788	0.767
Source	0.843	0.858	0.851
Editor	0.874	0.929	0.901
Identifier	0.861	0.906	0.883
Year	0.88	0.882	0.881
Issue	0.747	0.778	0.762
Other	0.819	0.781	0.8
Ref	0.901	0.925	0.913
macro Average	0.863	0.879	0.871
micro Average	0.896	0.916	0.906

Table 6.3: Results of the combined reference extraction and segmentation model

6.3 Comparison of proposed Models and EXparser

This section compares EXparsers models to our proposed reference extraction and reference segmentation models. Our proposed models are trained with BERT models whose pre-training data are similar to the PGS and PES datasets [BAS19] in some respects. PGS and PES are used for training and testing of our models. Furthermore, the *CMV-based* input format is applied to the proposed models since it achieved the best results in the conducted experiments (6.2).
We begin by comparing the extraction models first. Since the PGS dataset contains only German text data, we initially assumed that the BERT model $BERT_{BASE}$ *german cased* would fit best here as the model learned word embeddings through German texts. This was not the case. In our experiments, $BERT_{BASE}$ *multilingual cased* achieved better results. Consequently, in this

comparison, our proposed extraction model is composed of $BERT_{BASE}$ *multilingual cased* with the *CMV-based* input format.
Table 6.4 presents the micro-averaged results for both EXparsers extraction model and our proposed extraction model (Proposed) on PGS [BAS19]. Here, it should be noted that EXparser evaluated their extraction model on classified reference lines whereas we evaluated our extraction models on classified full reference strings. EXparser outperforms our model in terms of the recall score. It has a recall value of 84%, almost 5 percentage points more than our model. This means that EXparser correctly retrieved more references in total. However, our model performs substantially better in retrieving references that are relevant. With a precision value of 78%, we have an increase of more than 11 percentage points compared to EXparser. Overall, our model has a higher F1-score which is 78.8%. That is 4.88 percentage points more than EXparsers micro-averaged F1-score. However, as our model is not better in all metrics, it cannot be said that it performed better in reference extraction.

	Precision	Recall	F1-score
EXparser	0.67	**0.84**	0.74
Proposed	**0.783**	0.793	**0.788**

Table 6.4: Result of reference extraction on PGS using Proposed and EXparser [BAS19] (boldly printed values indicate the best result in the column)

We trained and evaluated two segmentation models, one model on the PGS dataset and the other one on the PES dataset.
For our proposed segmentation model on PGS, we have tried out the $BERT_{BASE}$ *german cased* model again but it performed 1-2 percentage points worse in terms of micro-averaged F1-score. Consequently, we do not include it here and used $BERT_{BASE}$ *multilingual cased*.
The results of our proposed segmentation model (P) and EXparsers segmentation model (EXp) on PGS are provided in Table 6.5. Just as it was done by Boukhers et al. [BAS19], we included three additional macro-averaged values that are calculated by averaging only certain reference components. Furthermore, our used 'ref'-tag was excluded from the calculation for a fair comparison. As can be seen from the table, EXparsers segmentation model has higher precision values for most of the components and the proposed segmentation model has higher recall values for most of the components. EXparser almost always classifies reference components such as 'publisher', 'first page', 'last page', or 'URL' correctly which is indicated by the components' very high precision values. Our proposed segmentation model has high classification performance on essential reference components such as 'author' or 'title'. When comparing the two models with respect to their F1-score on the different components, they can be regarded as equally good. This is illustrated by the values in the different macro-averaged F1-scores. In overall macro-averaged F1-score, our model achieves a higher value of 1.5 percentage points. EXparser did not provide micro-averaged results here. That is why it is left out in the table.

	Precision		Recall		F1-score	
	P	EXp	P	EXp	P	EXp
Publisher[1,2,3]	0.911	**0.964**	**0.899**	0.811	**0.905**	0.875
First Page[1,2,3]	0.882	**0.979**	0.857	**0.938**	0.869	**0.958**
Last Page[2]	0.859	**0.991**	0.935	**0.962**	0.895	**0.976**
Title[1,2,3]	**0.953**	0.894	0.931	**0.961**	**0.942**	0.925
URL[3]	0.937	**0.996**	**0.986**	0.8	**0.961**	0.881
Author[1,3]	**0.941**	0.926	**0.945**	0.793	**0.943**	0.854
Author Surname[2]	**0.958**	0.91	**0.968**	0.787	**0.963**	0.843
Author Given-name[2]	**0.93**	0.89	**0.91**	0.823	**0.92**	0.855
Volume[1,2,3]	0.667	**0.932**	**0.79**	0.78	0.723	**0.848**
Source[1,2,3]	0.837	**0.89**	**0.834**	0.749	0.836	**0.81**
Editor[3]	**0.923**	0.878	**0.956**	0.751	**0.939**	0.808
Identifier	**0.904**	0.902	**0.915**	0.706	**0.91**	0.754
Year[1,2,3]	0.844	**0.904**	0.867	**0.901**	0.856	**0.903**
Issue[2]	0.812	**0.964**	**0.789**	0.703	**0.8**	0.799
Other[3]	0.840	**0.848**	**0.767**	0.735	**0.801**	0.785
macro Average[1]	0.862	**0.927**	**0.875**	0.848	0.868	**0.882**
macro Average[2]	0.865	**0.927**	**0.878**	0.841	0.871	**0.879**
macro Average[3]	0.874	**0.921**	**0.883**	0.822	**0.878**	0.865
macro Average	0.87	**0.928**	**0.882**	0.815	**0.875**	0.86
micro Average	**0.912**	N/A	**0.904**	N/A	**0.908**	N/A

Table 6.5: Result of reference segmentation on PGS using Proposed and EXparser [BAS19] (boldly printed values indicate the best result in the row per metric)

For our proposed segmentation model on PES, the $BERT_{BASE}$ *cased* achieved substantially better results than $BERT_{BASE}$ *multilingual cased*. This is because PES contains solely English text data. As the BERT model *SciBERT cased* was trained with English scientific articles, we found it meaningful to apply it on PES, too. Hence, we included results of two proposed segmentation models with different BERT models, denoted by P_{BASE} and $P_{SciBERT}$.
Table 6.6 presents the results of our two proposed segmentation models P_{BASE} and $P_{SciBERT}$, and EXparsers segmentation model (EXp) on the PES dataset. Our used 'ref'-tag was excluded from the calculation. A comparison of the results reveals that both proposed segmentation models segment references significantly better than EXparsers segmentation model on the dataset PES. This can be seen particularly by the macro-averaged F1-scores. Our proposed segmentation models outperform EXparser by more than 5 percentage in terms of macro-averaged F1-score. In almost all cases $P_{SciBERT}$ has the highest values for precision, recall, and F1-score. However, there is no substantial performance difference between $P_{SciBERT}$ and P_{BASE}. What stands out in the table is that the 'page' reference components ('first page' and 'last page') are almost always classified correctly by $P_{SciBERT}$, as their precision values are very close to 100%. This also holds true for the labels 'title', 'URL', 'author', and 'year'. The components 'volume', 'issue', and 'other' that had relatively poor results in Table 6.5 have high scores here, too. The component 'identifier' in $P_{SciBERT}$ has the lowest F1-score with 85%.

In summary, we achieve comparable results to EXparser for reference extraction and segmentation. The only significant performance increase can be seen at our segmentation models on PES. Overall, we achieve a micro-averaged F1-score of 78.8% for reference extraction on PGS, 90.8% for reference segmentation on PGS, and 96% for reference segmentation on PES.

	Precision			Recall			F1-score		
	P_{BASE}	$P_{SciBERT}$	EXp	P_{BASE}	$P_{SciBERT}$	EXp	P_{BASE}	$P_{SciBERT}$	EXp
Publisher[1,2,3]	0.89	0.934	**0.959**	0.937	**0.946**	0.845	0.913	**0.94**	0.897
First Page[1,2,3]	0.983	0.99	**0.997**	0.981	**0.983**	0.98	0.982	0.986	**0.989**
Last Page[2]	0.984	0.99	**0.994**	0.98	**0.986**	0.984	0.982	0.988	**0.989**
Title[1,2,3]	**0.979**	**0.979**	0.932	0.974	**0.975**	0.973	**0.977**	**0.977**	0.951
URL[3]	0.971	**0.976**	0.965	**0.996**	0.994	0.764	0.983	**0.985**	0.809
Author[1,3]	0.967	**0.976**	0.971	0.975	**0.985**	0.91	0.971	**0.98**	0.938
Author Surname[2]	0.982	**0.986**	0.952	0.953	**0.977**	0.884	0.967	**0.982**	0.915
Author Given-name[2]	0.968	**0.98**	0.941	0.989	**0.991**	0.912	0.978	**0.986**	0.925
Volume[1,2,3]	0.948	0.952	**0.956**	0.948	**0.957**	0.937	0.948	**0.954**	0.925
Source[1,2,3]	0.925	0.925	**0.943**	0.917	**0.919**	0.835	0.92	**0.922**	0.884
Editor[3]	0.943	**0.954**	0.898	0.938	**0.945**	0.778	0.94	**0.95**	0.832
Identifier	0.938	0.91	**0.96**	**0.896**	0.805	0.701	**0.916**	0.854	0.733
Year[1,2,3]	**0.983**	**0.983**	0.944	**0.983**	0.981	0.933	**0.983**	0.982	0.939
Issue[2]	0.929	0.939	**0.958**	0.927	**0.934**	0.889	0.928	**0.937**	0.922
Other[3]	**0.864**	0.843	0.846	0.796	**0.82**	0.722	0.828	**0.831**	0.777
macro Average[1]	0.954	**0.963**	0.957	0.959	**0.964**	0.916	0.956	**0.963**	0.932
macro Average[2]	0.957	**0.966**	0.958	0.959	**0.965**	0.917	0.958	**0.965**	0.934
macro Average[3]	0.945	**0.951**	0.941	0.945	**0.951**	0.868	0.945	**0.951**	0.894
macro Average	0.946	**0.95**	0.92	**0.942**	0.941	0.865	0.944	**0.945**	0.891
micro Average	0.958	**0.96**	N/A	0.954	**0.959**	N/A	0.956	**0.96**	N/A

Table 6.6: Result of reference segmentation on PES using Proposed and EXparser [BAS19] (boldly printed values indicate the best result in the row per metric)

7 Discussion

In this chapter, the results are discussed in more detail. First, the obtained results in the experiments (Section 6.2) and in the comparison (Section 6.3) are interpreted by relating them to the sub research questions. After that, the main research question is answered. Finally, this section is concluded by describing the limitations of this thesis and by pointing out potential future work.

7.1 Interpretation of the Results

We begin the interpretation of the results by focusing on the first sub research question of this thesis:

> *Which BERT model and input format achieve the best performance on the task of reference extraction and reference segmentation?*

In the experiment of Section 6.2.1, we saw that the performance of the segmentation models varies on the same dataset, depending on what BERT model is used. Table 6.1 showed that the model with $BERT_{BASE}$ *multilingual cased* performed best on the dataset CD. The reason for this is that it learned word embeddings for both German and English texts in its pre-training procedure. Consequently, it achieved better results than the BERT models $BERT_{BASE}$ *german cased* or $BERT_{BASE}$ *cased* that only learned word embeddings for one of the two languages. From this follows that it was important to always choose $BERT_{BASE}$ *multilingual cased* in experiments where CD was used for evaluation. As the English articles of CD also contain references from languages other than English and German, it is advantageous to use $BERT_{BASE}$ *multilingual cased* here once again.
However, when models were trained on PES in Section 6.3, it was important to switch to a more suitable BERT model that was pre-trained with data that are similar to PES. For instance, $BERT_{BASE}$ *cased* is more suitable than $BERT_{BASE}$ *multilingual cased* when training on PES. This is due to the fact that PES only contains English text and $BERT_{BASE}$ *cased* was pre-trained only with large amounts of English text data whereas $BERT_{BASE}$ multilingual cased was trained with the same magnitude of data on 104 different languages. Consequently, $BERT_{BASE}$ *cased* represents English words better in a sentence and therefore also processes English text data better.
Furthermore, Table 6.6 shows that the BERT model *SciBERT cased* had achieved the best results on the PES dataset. The reason behind its remarkable results is that it was pre-trained solely on English scientific articles. Thus, during its pre-training, it has also seen and processed many reference strings. Since *SciBERT cased* was trained with data that matches both the language (English) and the domain-specific text corpus of our data (scientific articles), it was able to segment references most accurately with a micro-averaged F1-score of 96%.
Moreover, as each BERT model has a '*cased*' property or not, it was important to choose the

cased variants. In references, the capitalization of words plays an important role. Choosing cased BERT models ensures that contextualized word embeddings are learned for both the lower case and capitalized version of a word in the same context of a sentence. This helps the model to make more certain predictions because, for instance, there are much more words that are capitalized in references than in normal text. We listed further reasons for this in section 4.2.
Our observations and interpretations show that BERT models have a noticeable impact on the classification performance of references. We argue that the better a BERT model represents words of a reference in their context, the more accurate predictions can be made. Furthermore, our findings lead to the statement that the choice of the BERT model is highly dependent on what data is used for training and testing of the corresponding reference extraction or segmentation model. Depending on the data, a BERT model should be selected based on

- **the language(s)** it was trained on (e.g. English, German, etc.),
- **the domain-specific text corpus** it was trained on (e.g. Wikipedia articles, scientific articles, etc.),
- whether a **case-preserving** WordPiece Model is used or not.

The more criteria match between the BERT model's pre-training data and the BERT model's fine-tuning data, the better the BERT model captures both syntactic and semantic meanings of references.
For the task of reference extraction and segmentation, it leads to a significant performance increase when the used BERT model is pre-trained on large amounts of scientific articles that contain reference strings. This statement is supported by the high-performance results of the proposed segmentation model that used *SciBERT cased* (cf. Table 6.6). Furthermore, a BERT model with a case-preserving WordPiece model should be preferred for reference extraction and segmentation.

Next, we discuss what input format achieves the best performance. The obtained results indicate that the input format to a BERT model has a significant impact on the model's overall classification performance. Models with the *single* input format had relatively poor performance compared to the two other input formats. The reason behind this is that the *maximum context* and CMV-based input format both provide significantly more context around single sentences, allowing the model to utilize more information and make predictions with higher certainty. Extracting a reference solely using the information of its inner structure and content is harder than using further contextual information. Thus, references can be identified better when the information around the reference is used. The information around reference strings can provide many clues to the model. Based on this observation, for high classification performance, rather long input sequences should be preferred such that there is sufficient context around each token. This does not hold true always as, for instance, the reference component 'URL' was best classified by the segmentation model that used the *single* input format (cf. Table 6.2). To prevent far left and far right tokens to suffer from one-sided context, the idea of CMV can be used. That CMV improves performance can also be seen in Table 6.2 where the model with *CMV-based* input format achieves slightly better results than the model with *maximum context* input format.
When comparing the results of the different input formats on the task of reference extraction and reference segmentation (cf. Table 6.1 and 6.2), the results show that the input format has a larger impact on the performance of reference extraction models. The model with *single* input format had a significantly worse performance in extracting references than the models with *maximum context* or *CMV*-based input format. Its micro-averaged F1-score was almost 15 percentage points lower

than the other micro-averaged F1-scores. However, there was not such a substantial performance difference between *single* segmentation model and the other segmentation models. This means that a context-based input format is very important for the task of reference extraction but not that important for reference segmentation. Intuitively this makes sense because additional text around a reference can help a model to identify that reference better. For example, when there are references around a reference string to be classified, chances are high that this reference string is identified and extracted. However, additional text around a reference does not effectively help a model to segment that reference better. No matter what kind of references are located to the left and right of a reference string, the reference string is rather segmented with the information of its inner structure and content.
We conclude that input formats that generally define long, context-based input sequences significantly improve the performance of both reference extraction and reference segmentation models. We argue that such input formats lead to better contextual word representations. Furthermore, we observe that additional context in the input format is far more required for the extraction of references than for the segmentation of references.

In the following, the second sub research question is discussed:

> *What is the best approach for training a given BERT-based model on the task of reference extraction and reference segmentation?*

We have seen the results of two approaches on the task of reference extraction and segmentation: results of the standard two-models approach and results of the one-model approach. When comparing the segmentation performance between the two approaches (cf. Table 6.2 and Table 6.3), it is evident that the segmentation model from the two-models approach segments references better. It achieves a higher micro-averaged F1-score by 3 percentage points. This was expected as the segmentation model is only trained with reference strings and consequently only learns to divide a reference into its constituent components. We cannot effectively compare the reference extraction performances as the reference extraction performance of the combined model cannot be measured directly.
Another argument that speaks for the standard two-models approach is that it is more practical than the one-model approach. It has a more simple and concrete online process. Text can be passed to the extraction model to identify and extract references and then the extracted references can be passed to the segmentation model to segment them. For the combined reference extraction and segmentation model, the online process is more complex. To compose references from classification, either heuristics need to be defined or complex training labels are needed that mark the beginning token of a reference and annotate it with a reference component simultaneously.
The two aforementioned reasons lead us to the conclusion that the standard two-models approach is the better approach to extract and segment references. Although the one-model approach may bring advantages like a lower computational overhead, the results of the standard two-models approach are more promising. Furthermore, the two-models approach is more practical.

In what follows, the last sub research question is discussed:

> *Can BERT improve the obtained results of EXparser [BAS19]?*

When comparing our proposed models to EXparser [BAS19], it cannot be clearly said that our proposed BERT models improved the performance on reference extraction and segmentation. The reason for this is that our proposed models were not always better in all three metrics of precision, recall, and F1-score. This can be seen in the Tables 6.4, 6.5, and 6.6. Our proposed extraction model on PGS had a higher micro-averaged precision value and a higher micro-averaged F1-score by almost 5 percentage points. But EXparser has a higher recall score by nearly 5 percentage points. On the dataset PGS, our proposed segmentation model on PGS achieved a higher macro-averaged recall and a higher macro-averaged F1-score by 1.5 percentage points. However, EXparser achieved a higher macro-averaged precision by more than 5 percentage points. The only decisive performance increase could be seen on the dataset PES. There, our proposed segmentation model performed better than EXparser as its macro-averaged precision, recall, and F1-score were all higher. The macro-averaged F1-score was higher by more than 5 percentage points. Based on these results, we conclude that our proposed models achieved comparable results to EXparser. For a better comparison between our models and EXparser, more extensive experiments between the two models need to be carried out, for instance, by evaluating on more datasets.

Finally, the main research question is discussed and answered in the following:

> *Is BERT a suitable choice for the task of reference extraction and reference segmentation?*

Yes, BERT is a suitable choice for the task of reference extraction and reference segmentation. We justify this answer with the following arguments:
We have fine-tuned BERT-based models with little data and still achieved high reference extraction and segmentation performance. This indicates that little data is sufficient such that BERT models can learn word representations efficiently on the downstream task of reference extraction and segmentation by fine-tuning. Furthermore, this partly shows the great capability of neural-based embeddings for reference extraction and segmentation. With a suitable pre-trained BERT model, $BERT_{BASE}$ *multilingual cased*, and a meaningful input format, CMV-based, we were able to extract and segment references with high quality.
Although our dataset contains both English and German language articles with references that strongly vary in their content, length, and location, our extraction model extracted references with an average F1-score of 81,9%. This shows that BERT is capable of learning the differences and similarities of high-variance references and is therefore suitable for the task of reference extraction. Moreover, it is important to highlight that in our segmentation model, the essential reference components ‘author’, ‘title’, ‘publisher’, and ‘year’ belong to the reference components with the highest classification performance in all trained segmentation models. In addition, BERT was able to capture differences between the given-name and surname of authors as both these reference components have high classification performance. In contrast, the less relevant components ‘volume’, ‘source’, ‘issue’, and ‘other’ have a rather poor performance. This was expected since these components are (a) not always part of a reference and thus appear less frequent in training instances and (b) vary more in their structure and content than other components. This is especially true for the ‘other’ component. Overall, our segmentation model segmented references with an average F1-score of 93.6%. Because of this overall performance and the fact that the most essential components of a reference were classified with the highest averaged F1-scores, this strongly suggests

that BERT is suitable for reference segmentation.
Our statement is once again confirmed by the comparison to EXparser [BAS19]. EXparser managed to achieve better results than the previous state-of-the-art methods Cermine [TSF+15], Grobid [Lop09], and ParsCit [CGK08] on many datasets. With BERT being part of our model architecture, it was possible to achieve comparable results to EXparser in both reference extraction and segmentation. Thus, it can be argued that our proposed BERT models are suitable for both reference extraction and segmentation.

7.2 Limitations and Future Work

Although satisfactory results were achieved, the trained extraction and segmentation models are far from being perfect. There are some important limitations to consider when interpreting the results. Devlin et al. [DCLT19] have published two different model sizes of BERT: $BERT_{BASE}$ and $BERT_{LARGE}$. Whenever there was the choice between these two model sizes, $BERT_{BASE}$ was always selected. This is because training with $BERT_{LARGE}$ is significantly more computationally expensive. Devlin et al. [DCLT19] show that the $BERT_{LARGE}$ model performed substantially better at nearly all tasks. Considering that, we speculate that it would also improve the performance on the task of reference extraction and segmentation.
Another limitation to be considered when interpreting the results is the amount of data that the models were trained with. All models in this thesis were fine-tuned with data, consisting of 350 documents (roughly 3,478,000 tokens) or less. Thus, all models are trained with little data and are evaluated on little data. To set it in contrast, the published BERT models were trained with about 3,300 million words. Future research in this field could fine-tune BERT models with more data (with annotated references) and consequently evaluate on more data, finding out how well BERT generalizes on reference extraction and segmentation. Datasets such as the dblp dataset[1] that contain a large amount of annotated bibliographic records can be considered.

For future work, we discuss potential improvements to the constructed models in this thesis. Given a BERT model, we found out that using CMV enhances the model's quality as more context flows into the model for the classification of each token. Out of three possible classifications for one token, the majority vote of the three classifications was chosen to be the final classification for that token. To determine the final classification from three previous classifications of a token, the decision can be made by taking the classification probabilities into account. For example, out of three classifications, one possibility is to take the classification that was made with the highest probability.
As evident in the results, the BERT model SciBERT cased achieved the highest results in reference segmentation on PES as it was pre-trained on English scientific data. Unfortunately, there does not exist a similar model to SciBERT that was pre-trained on solely German scientific articles. Pre-training such a model would probably result in a similar performance on German data such as PGS.
Furthermore, the performance of BERT ensembles should be investigated. In our implementation, we always used the last checkpoint of pre-trained BERT models. However, the performance of BERT models can be examined using different checkpoints that were made earlier than the last

[1] https://dblp.uni-trier.de/xml/

checkpoint during the pre-training phase.
Moreover, BERT could be combined with a feature engineering approach as Boukhers et al. [BAS19] did. Including additional format information about a document such as the file format, font size, font type, etc. in the input to the model could improve performance significantly. The additional information in the input can give valuable hints whether or not a text line is part of a reference or not. For instance, in many scientific articles, the references in a reference section are written in a smaller font size or a different font type than the text in the body of the article. Including features in the input that capture this information would most likely lead to a high reference extraction performance. Furthermore, with the use of format-based features, obvious reference components such as 'year' or 'URL' could be segmented almost always correctly. For example, by using a feature that includes the information about the existence of a hyperlink format (e.g. '*www*', '*http*'), the reference component 'URL' could be segmented correctly in nearly all cases.

8 Conclusion

This thesis aimed to extract references from PDF documents and to segment them into their bibliographic elements with the use of BERT. It extends the work done by Boukhers et al. [BAS19]. To train the most principled reference extraction and segmentation models, different approaches, BERT models, and input formats were examined. Our findings showed that the standard two-models approach is more suitable for reference extraction and segmentation. Furthermore, we found out that the more context is provided around the words of an input sequence, the more precisely do the models extract and segment references. Also, the choice of the BERT model plays an important role. Choosing a BERT model whose pre-training data fits the fine-tuning data leads to better results since better word embeddings are used. Overall, our results confirm that BERT is a suitable choice to extract and segment references. This is because references were extracted from text data with a high classification performance and elementary reference components were categorized correctly with high precision. We achieved comparable results to EXparser, partly illustrating that neural-based embeddings are a strong alternative to manual feature engineering approaches.
As we pointed out earlier, there is still much potential to improve the obtained results in the research field of reference extraction and segmentation. We believe that BERT combined with a feature engineering approach would lead to a promising model, as additional valuable information would be preserved and could be utilized by the model.
Ultimately, extracting and segmenting references automatically is the main factor to build and maintain large bibliographic databases. It would be very interesting and practical to develop a BERT-based service that goes from the text extraction of PDF documents to the automatic identification and segmentation of references, finding its use in production. The insights gained from this Bachelor thesis could help build reference extraction and segmentation models with good performance. For example, such an application could find its usage in online servers like Arxiv[1] or Semantic Scholar[2] where documents are published daily and bibliographic information is an essential part of their databases.

[1] https://arxiv.org/
[2] https://www.semanticscholar.org/

Bibliography

[BAS19] Z. Boukhers, S. Ambhore, S. Staab. "An End-to-End Approach for Extracting and Segmenting High-Variance References from PDF Documents". In: *19th ACM/IEEE Joint Conference on Digital Libraries, JCDL 2019, Champaign, IL, USA, June 2-6, 2019*. Ed. by M. Bonn, D. Wu, J. S. Downie, A. Martaus. IEEE, 2019, pp. 186–195. DOI: 10.1109/JCDL.2019.00035. URL: https://doi.org/10.1109/JCDL.2019.00035 (cit. on pp. 1–4, 6, 27, 28, 31, 32, 36–38, 40, 43–47, 58).

[BCB15] D. Bahdanau, K. Cho, Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio, Y. LeCun. 2015. URL: http://arxiv.org/abs/1409.0473 (cit. on p. 10).

[BLC19] I. Beltagy, K. Lo, A. Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by K. Inui, J. Jiang, V. Ng, X. Wan. Association for Computational Linguistics, 2019, pp. 3613–3618. DOI: 10.18653/v1/D19-1371. URL: https://doi.org/10.18653/v1/D19-1371 (cit. on p. 19).

[CGK08] I. G. Councill, C. L. Giles, M. Kan. "ParsCit: an Open-source CRF Reference String Parsing Package". In: *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2008, 26 May - 1 June 2008, Marrakech, Morocco*. European Language Resources Association, 2008. URL: http://www.lrec-conf.org/proceedings/lrec2008/summaries/166.html (cit. on pp. 1–3, 5, 6, 45).

[CMBB14] K. Cho, B. van Merrienboer, D. Bahdanau, Y. Bengio. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*. Ed. by D. Wu, M. Carpuat, X. Carreras, E. M. Vecchi. Association for Computational Linguistics, 2014, pp. 103–111. DOI: 10.3115/v1/W14-4012. URL: https://aclanthology.org/W14-4012/ (cit. on p. 9).

[CV95] C. Cortes, V. Vapnik. "Support-Vector Networks". In: *Mach. Learn.* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018. URL: https://doi.org/10.1007/BF00994018 (cit. on p. 5).

[DCLT19] J. Devlin, M. Chang, K. Lee, K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, T. Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423 (cit. on pp. 2, 8, 11–13, 18–20, 22, 31, 45, 55, 56, 59).

[Gag94] P. Gage. "A new algorithm for data compression". In: *C Users Journal* 12.2 (1994), pp. 23–38. URL: https://www.derczynski.com/papers/archive/BPE_Gage.pdf (cit. on p. 7).

[Har54] Z. S. Harris. "Distributional Structure". In: WORD 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520 (cit. on p. 8).

[HCWC19] W. Huang, X. Cheng, T. Wang, W. Chu. "BERT-Based Multi-head Selection for Joint Entity-Relation Extraction". In: *Natural Language Processing and Chinese Computing - 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9-14, 2019, Proceedings, Part II*. Ed. by J. Tang, M. Kan, D. Zhao, S. Li, H. Zan. Vol. 11839. Lecture Notes in Computer Science. Springer, 2019, pp. 713–723. DOI: 10.1007/978-3-030-32236-6_65. URL: https://doi.org/10.1007/978-3-030-32236-6_65 (cit. on p. 17).

[Het08] E. Hetzner. "A simple method for citation metadata extraction using hidden markov models". In: *ACM/IEEE Joint Conference on Digital Libraries, JCDL 2008, Pittsburgh, PA, USA, June 16-20, 2008*. Ed. by R. L. Larsen, A. Paepcke, J. Borbinha, M. Naaman. ACM, 2008, pp. 280–284. DOI: 10.1145/1378889.1378937. URL: https://doi.org/10.1145/1378889.1378937 (cit. on p. 5).

[HGKM19] A. Hosseini, B. Ghavimi, D. Kern, P. Mayr. "EXCITE - A Toolchain to Extract, Match and Publish Open Literature References". In: *19th ACM/IEEE Joint Conference on Digital Libraries, JCDL 2019, Champaign, IL, USA, June 2-6, 2019*. Ed. by M. Bonn, D. Wu, J. S. Downie, A. Martaus. IEEE, 2019, pp. 432–433. DOI: 10.1109/JCDL.2019.00105. URL: https://doi.org/10.1109/JCDL.2019.00105 (cit. on pp. 4, 6).

[HP19] K. Hakala, S. Pyysalo. "Biomedical Named Entity Recognition with Multilingual BERT". In: *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks, BioNLP-OST@EMNLP-IJNCLP 2019, Hong Kong, China, November 4, 2019*. Ed. by J. Kim, C. Nédellec, R. Bossy, L. Deléger. Association for Computational Linguistics, 2019, pp. 56–61. DOI: 10.18653/v1/D19-5709. URL: https://doi.org/10.18653/v1/D19-5709 (cit. on p. 19).

[HS97] S. Hochreiter, J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on pp. 5, 8).

[HSK+12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580 (cit. on p. 59).

[KB15] D. P. Kingma, J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio, Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 59).

[KGM+17] M. Körner, B. Ghavimi, P. Mayr, H. Hartmann, S. Staab. "Evaluating Reference String Extraction Using Line-Based Conditional Random Fields: A Case Study with German Language Publications". In: *New Trends in Databases and Information Systems - ADBIS 2017 Short Papers and Workshops, AMSD, BigNovelTI, DAS, SW4CH, DC, Nicosia, Cyprus, September 24-27, 2017, Proceedings*. Ed. by M. Kirikova, K. Nørvåg, G. A. Papadopoulos, J. Gamper, R. Wrembel, J. Darmont, S. Rizzi. Vol. 767. Communications in Computer and Information Science. Springer, 2017, pp. 137–145. DOI: 10.1007/978-3-319-67162-8_15. URL: https://doi.org/10.1007/978-3-319-67162-8_15 (cit. on pp. 5, 30).

[LBS+16] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer. "Neural Architectures for Named Entity Recognition". In: *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*. Ed. by K. Knight, A. Nenkova, O. Rambow. The Association for Computational Linguistics, 2016, pp. 260–270. DOI: 10.18653/v1/n16-1030. URL: https://doi.org/10.18653/v1/n16-1030 (cit. on p. 17).

[LH19] I. Loshchilov, F. Hutter. "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7 (cit. on p. 59).

[LMP01] J. D. Lafferty, A. McCallum, F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*. Ed. by C. E. Brodley, A. P. Danyluk. Morgan Kaufmann, 2001, pp. 282–289. URL: http://www.aladdin.cs.cmu.edu/papers/pdfs/y2001/crf.pdf (cit. on pp. 5, 13, 14).

[Lop09] P. Lopez. "GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction for Scholarship Publications". In: *Research and Advanced Technology for Digital Libraries, 13th European Conference, ECDL 2009, Corfu, Greece, September 27 - October 2, 2009. Proceedings*. Ed. by M. Agosti, J. Borbinha, S. Kapidakis, C. Papatheodorou, G. Tsakonas. Vol. 5714. Lecture Notes in Computer Science. Springer, 2009, pp. 473–474. DOI: 10.1007/978-3-642-04346-8_62. URL: https://doi.org/10.1007/978-3-642-04346-8_62 (cit. on pp. 1, 5, 45).

[LP20] J. Luoma, S. Pyysalo. "Exploring Cross-sentence Contexts for Named Entity Recognition with BERT". In: *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*. Ed. by D. Scott, N. Bel, C. Zong. International Committee on Computational Linguistics, 2020, pp. 904–914. DOI: 10.18653/v1/2020.coling-main.78. URL: https://doi.org/10.18653/v1/2020.coling-main.78 (cit. on p. 23).

[LTWX20] M. Liu, Z. Tu, Z. Wang, X. Xu. "LTP: A New Active Learning Strategy for Bert-CRF Based Named Entity Recognition". In: *CoRR* abs/2001.02524 (2020). arXiv: 2001.02524. URL: http://arxiv.org/abs/2001.02524 (cit. on p. 17).

[Mac+67] J. MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297. URL: http://www.cs.cmu.edu/~bhiksha/courses/mlsp.fall2010/class14/macqueen.pdf (cit. on p. 5).

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Y. Bengio, Y. LeCun. 2013. URL: http://arxiv.org/abs/1301.3781 (cit. on pp. 7, 8).

[MFP00] A. McCallum, D. Freitag, F. C. N. Pereira. "Maximum Entropy Markov Models for Information Extraction and Segmentation". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. Ed. by P. Langley. Morgan Kaufmann, 2000, pp. 591–598. URL: http://www.ai.mit.edu/courses/6.891-nlp/READINGS/maxent.pdf (cit. on p. 13).

[ML19] J. Mao, W. Liu. "Hadoken: a BERT-CRF Model for Medical Document Anonymization". In: *Proceedings of the Iberian Languages Evaluation Forum co-located with 35th Conference of the Spanish Society for Natural Language Processing, IberLEF@SEPLN 2019, Bilbao, Spain, September 24th, 2019*. Ed. by M. Á. G. Cumbreras, J. Gonzalo, E. M. Cámara, R. Martínez-Unanue, P. Rosso, J. Carrillo-de-Albornoz, S. Montalvo, L. Chiruzzo, S. Collovini, Y. Gutiérrez, S. M. J. Zafra, M. Krallinger, M. Montes-y-Gómez, R. Ortega-Bueno, A. Rosá. Vol. 2421. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 720–726. URL: http://ceur-ws.org/Vol-2421/MEDDOCAN_paper_11.pdf (cit. on p. 17).

[MSC+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by C. J. C. Burges, L. Bottou, Z. Ghahramani, K. Q. Weinberger. 2013, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf (cit. on p. 8).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 26).

[PKK18] A. Prasad, M. Kaur, M.-Y. Kan. "Neural ParsCit: a deep learning-based reference string parser". In: *International Journal on Digital Libraries* 19.4 (Nov. 2018), pp. 323–337. ISSN: 1432-1300. DOI: 10.1007/s00799-018-0242-1. URL: https://link.springer.com/article/10.1007/s00799-018-0242-1 (cit. on pp. 1, 5).

[PNI+18] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. by M. A. Walker, H. Ji, A. Stent. Association for Computational Linguistics, 2018, pp. 2227–2237. DOI: 10.18653/v1/n18-1202. URL: https://doi.org/10.18653/v1/n18-1202 (cit. on p. 8).

[PSM14] J. Pennington, R. Socher, C. D. Manning. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by A. Moschitti, B. Pang, W. Daelemans. ACL, 2014, pp. 1532–1543. DOI: 10.3115/v1/d14-1162. URL: https://doi.org/10.3115/v1/d14-1162 (cit. on p. 8).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf (cit. on p. 26).

[RJ86] L. Rabiner, B. Juang. "An introduction to hidden Markov models". In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16. DOI: 10.1109/MASSP.1986.1165342. URL: https://doi.org/10.1109/MASSP.1986.1165342 (cit. on pp. 5, 13).

[Seb02] F. Sebastiani. "Machine learning in automated text categorization". In: *ACM Comput. Surv.* 34.1 (2002), pp. 1–47. DOI: 10.1145/505282.505283. URL: https://doi.org/10.1145/505282.505283 (cit. on p. 31).

[SM12] C. Sutton, A. McCallum. "An Introduction to Conditional Random Fields". In: *Found. Trends Mach. Learn.* 4.4 (2012), pp. 267–373. DOI: 10.1561/2200000013. URL: https://doi.org/10.1561/2200000013 (cit. on p. 14).

[SMR99] K. Seymore, A. Mccallum, R. Rosenfeld. "Learning Hidden Markov Model Structure for Information Extraction". In: *Proceedings of AAAI '99 Workshop on Machine Learning for Information Extraction*. July 1999, pp. 37–42. URL: https://www.aaai.org/Papers/Workshops/1999/WS-99-11/WS99-11-007.pdf (cit. on p. 5).

[SNA19] F. Souza, R. F. Nogueira, R. de Alencar Lotufo. "Portuguese Named Entity Recognition using BERT-CRF". In: *CoRR* abs/1909.10649 (2019). URL: http://arxiv.org/abs/1909.10649 (cit. on pp. 17, 19).

[TSF+15] D. Tkaczyk, P. Szostek, M. Fedoryszak, P. J. Dendek, L. Bolikowski. "CERMINE: automatic extraction of structured metadata from scientific literature". In: *Int. J. Document Anal. Recognit.* 18.4 (2015), pp. 317–335. DOI: 10.1007/s10032-015-0249-8. URL: https://doi.org/10.1007/s10032-015-0249-8 (cit. on pp. 1, 5, 6, 27, 45).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett. 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (cit. on pp. 8–11).

[WDS+20] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*. Ed. by Q. Liu, D. Schlangen. Association for Computational Linguistics, 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://doi.org/10.18653/v1/2020.emnlp-demos.6 (cit. on p. 26).

[WK92] J. J. Webster, C. Kit. "Tokenization As The Initial Phase In NLP". In: *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*. 1992, pp. 1106–1110. URL: https://aclanthology.org/C92-4173/ (cit. on p. 7).

[WSC+16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144 (cit. on pp. 7, 12, 20, 55).

[ZKZ+15] Y. Zhu, R. Kiros, R. S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, S. Fidler. "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.11. URL: https://doi.org/10.1109/ICCV.2015.11 (cit. on pp. 2, 19).

[ZLT10] J. Zou, D. X. Le, G. R. Thoma. "Locating and parsing bibliographic references in HTML medical articles". In: *Int. J. Document Anal. Recognit.* 13.2 (2010), pp. 107–119. DOI: 10.1007/s10032-009-0105-9. URL: https://doi.org/10.1007/s10032-009-0105-9 (cit. on p. 5).

All links were last followed on December 13, 2021.

A Pre-processing Steps for Input Sequences in BERT

Within BERT, input sequences for classification tasks require some pre-processing steps.
First, the input sequence needs to be tokenized. To split an input sequence into tokens, BERT uses a WordPiece model [DCLT19; WSC+16]. Words that can not be broken down into the tokens of the WordPiece vocabulary are represented by a special *[UNK]* tag. In addition to the WordPiece tokenization, the extra tokens *[CLS]* and *[SEP]* are added to the start and end of the tokenized sequence, respectively. From this follows that the first and last token of the input sequence are always reserved for the two special tokens. If there is more than one sentence in the input sequence, the *[SEP]* token can optionally be inserted between them to separate the sentences from each other. After adding these special tokens, it is necessary to pad the input sequence with special *[PAD]* tokens such that the total length of the token sequence is equal to the specified sequence length of the BERT model. The last step is to convert each WordPiece token to its corresponding unique ID. For instance, the special tokens *[UNK]*, *[CLS]*, *[SEP]* and *[PAD]* have the unique IDs 100, 101, 102, and 0, respectively. In our implementation, classifications of the special tokens are not considered for the evaluation of a model's classification performance.
In Figure A.1, we provide an example where the mentioned pre-processing steps are applied to an input sequence. For the example, the tokenizer of the pre-trained $BERT_{BASE}$ *cased* model was used.

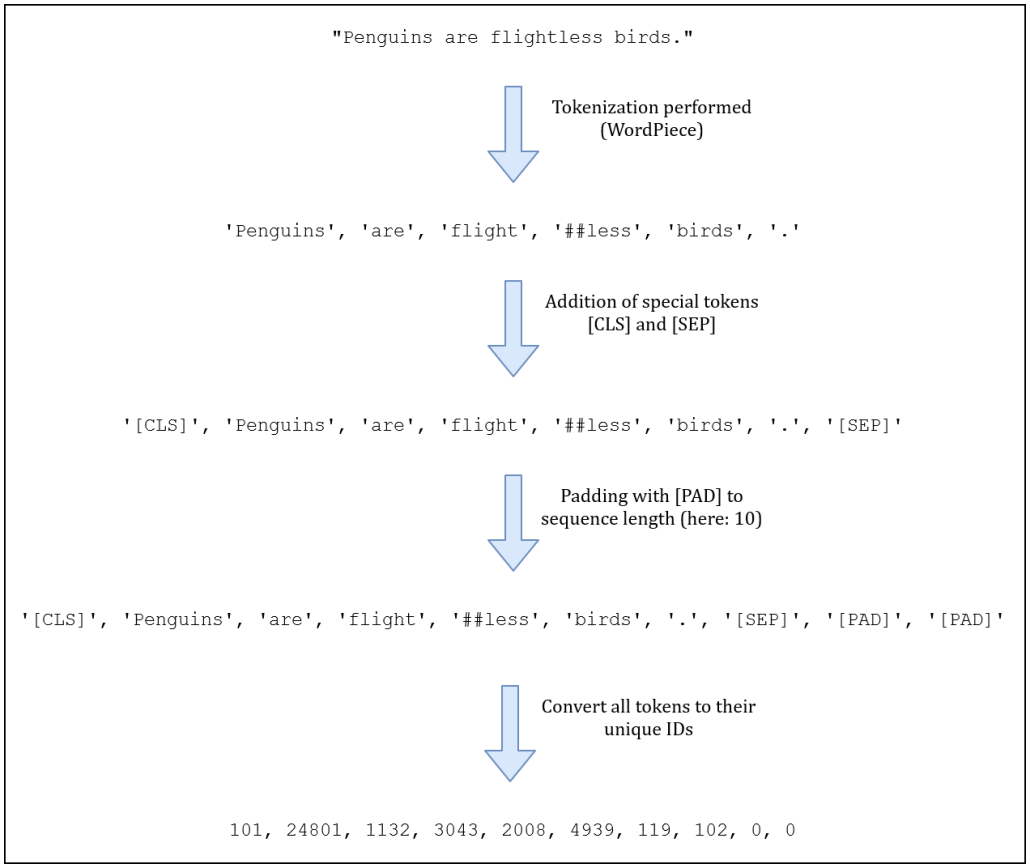

Figure A.1: Pre-processing steps for input sequences in BERT applied to an example sentence [DCLT19]

B Implementation Details

B.1 Hardware

All models were trained on one machine with an Intel Core i7-4770K CPU and one NVIDIA GeForce RTX 3080 Ti GPU. CUDA[1] was set up and run for GPU acceleration. To make our results as reproducible as possible, we limited the number of sources with non-deterministic behavior by using the same random seed of 999 for the training of all models.

B.2 Schedule

We fine-tuned all proposed models for 1 to 5 epochs. Training on more epochs than 5 led to overfitting of the model as the training loss decreased further while the validation loss increased at the same time. For all model approaches, we keep the same hyperparameters **except** for the parameters of batch size and BERTs sequence length. This is because depending on how high the sequence length is set, the batch size is limited up to a certain number. As pointed out in the detailed description of BERTs official repository[2], this memory issue occurs when using GPUS with 12-16 GB of RAM. In the description, they provided a table where the maximum batch size is specified, given a sequence length. As the GPU in this implementation has 12 GB of RAM, batch size was restricted, given a certain sequence length. For instance, for the maximum sequence length of 512 tokens, the maximum batch size was 8. With this configuration, the training of one model takes very long compared to other configurations. The time available did not allow to train every model with this configuration. Still, we chose meaningful values for the sequence length depending on the model to be trained.
Table B.1 shows the different configurations of sequence length and batch size depending on what model was trained. The models with the *single* input format were trained with a sequence length of 64 as the sentences in our datasets rarely contain more than 64 WordPiece tokens when being tokenized. Since we defined models with the *maximum context* input format to have the maximum sequence length for maximum context, these models were trained with a sequence length of 512. For the models with CMV-based input format, we set the sequence length to 128, higher than for the *single* input format models because we concatenate three sentences here.
In the one-model approach, we trained one model with CMV-based input format with a sequence length of 256 and a batch size of 16.
When we trained models for the comparison to the EXparser's extraction and segmentation model,

[1] https://developer.nvidia.com/cuda-toolkit
[2] https://github.com/google-research/bert

we trained both extraction and segmentation model with the maximum sequence length of 512, in order to achieve high performance.

	Model type, BERT model(s), input format	Sequence Length	Batch Size
Two-models approach	Extraction model, multilingual, single	64	64
	Extraction model, multilingual, maximum context	512	8
	Extraction model, multilingual, CMV-based	128	32
	Segmentation model, base / german / multilingual / SciBERT, single	64	64
	Segmentation model, multilingual, maximum context	512	8
	Segmentation model, multilingual, CMV-based	128	32
One-model approach	Combined Extraction model and Segmentation model, multilingual, CMV-based	256	16
Comparison to EXparser [BAS19]	Proposed extraction model on PGS, multilingual, CMV-based	512	8
	Proposed segmentation model on PGS, multilingual, CMV-based	512	8
	Proposed segmentation model on PES, base / SciBERT, CMV-based	512	8

Table B.1: Overview of the trained models with their respective sequence lengths and batch sizes

B.3 Optimizer

We use the AdamW optimizer [LH19], a modified version of the original Adam optimizer [KB15]. AdamW improves Adam's implementation of weight decay, leading to less overfitting and better generalization of a model. For AdamW, the following parameters were used:

- Constant learning rate of $lrate = 5 \cdot 10^{-5}$ as recommended by Devlin et al. [DCLT19].
- Adam's betas parameters with $\beta_1 = 0.9$, $\beta_2 = 0.999$.
- Adam's epsilon parameter that is used for numerical stability with $\epsilon = 10^{-8}$.
- L_2 weight decay (new parameter in AdamW) of $wdecay = 0.01$

Over the course of training, we did not apply a warm-up schedule to the given learning rate.

B.4 Regularization

In our implementation, we use dropout which is a regularization technique for deep neural networks to counteract overfitting [HSK+12]. For all layers of BERT, we use a dropout probability of 0.1 [DCLT19]. Between BERT and the linear layer, we also regularize through dropout training with a probability of 0.1.

C Modified Data Files

We identified inconsistencies in the data annotation of CD_s. Most of the inconsistencies occur in connection with the author tag. For example, in the file 16563.xml[1], there is a line where tags are overlapping:

<author><surname>Weber <author><given-names>Max
</surname></author></given-names></author>

Such lines made parsing the file more challenging because usually the overlapping of tags is not allowed in Extensible Markup Language (XML) or XML-similar documents. For this reason, we manually corrected such inconsistencies in the files. Another more rare inconsistency that was present in the data was that opening tags were not matching their closing tags.
In the following, we firstly provide a list of German documents with the lines where inconsistencies were identified and manually corrected (".xml" is omitted):

- 11721 (line 30), 16563 (line 14), 18508 (line 9), 20786 (line 30), 21690 (line 10), 21699 (line 45), 22006 (line 17), 24743 (line 39), 28254 (line 3), 32211 (line 34), 36025 (line 2, 9), 36493 (line 1, 3), 37466 (line 1, 2, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15), 38850 (line 8), 39204 (line 6), 39323, 39454 (line 10, 30), 41507 (line 21), 42302 (line 9), 44475 (line 5), 44553 (line 12), 46910 (line 9, 17), 47961 (line 17), 48259 (line 16), 5675 (line 2), 6041 (line 6), 6926 (line 4).

Next, we provide a list of English documents with the lines where inconsistencies were identified and manually corrected (".xml" is omitted):

- 12279 (line 4, 9, 12, 13, 15, 16), 12325 (line 2), 19468 (line 1), 19733 (line 1), 20291 (line 1, 12, 23, 24), 20324 (line 1), 20369 (line 8), 22355 (line 5, 46), 24572 (line 1, 3, 4), 24626 (line 1), 25465 (line 1), 25468 (line 1, 10), 26178 (line 10, 14), 26735 (line 1), 45661 (line 13), 51256 (line 5), 53780 (line 7), 53995 (line 8), 5473 (line 6), 55362 (line 5), 55386 (line 4).

[1] https://github.com/exciteproject/EXgoldstandard/blob/master/Goldstandard_EXparser/1-German_papers/1-German_papers(with_reference_section_at_end_of_paper)/5-References_segmented_by_EXRefSegmentation/16563.xml

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar *stimmt mit allen eingereichten Druck-Exemplaren überein.*

Datum und Unterschrift: 21.12.2021 A. Evci

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: 21.12.2021 A. Evci