Eötvös Loránd University

Faculty of Informatics

Dept. of Artificial Intelligence.

Enhancing Citation Reference Parsing: Integrating Formatted Information for Improved Accuracy

Supervisor:
Simonyi András
Lecturer

Author:
MHD Yasser Haddad
Computer Science MSc

Budapest, 2025

Declaration

I hereby declare that the thesis titled “Enhancing Citation Reference Parsing: Integrating Formatted Information for Improved Accuracy” and the research presented within it are entirely my own original work. I further confirm the following:

- The research was conducted entirely or primarily during my candidature for a master’s degree at Eötvös Loránd University.
- Any published work by others that I have consulted has been clearly cited.
- All direct quotations from other sources are properly acknowledged.
- All major sources of assistance have been duly credited.
- This thesis has not been submitted previously for any other degree or professional qualification.

Signed: MHD Yasser Haddad
Date: 21st April, 2025

Thank you all.

Contents

Chapter 1

Introduction

In any academic research, citations are vital because they help us to credit previous work, illustrate how the ideas provided in the work relate to others, and help whoever reads the work to explore similar ideas and research. In the majority of publications, this task is usually set as a reference string that contains structured segments at the end of the paper (or any type of work). These strings are filled with important metadata about the work, like the author names, article title, publication year, journal names, and more. Being able to interpret them is an important task that would help with building citation networks, tracking the impact of research, and keeping a clean digital library that we can search through it [1, 2].

But parsing reference strings isn't an easy task, while us humans can take a look at a reference and understand its structure in a fast way (identifying the authors, title, and pages fields), machines struggle. One of the reasons is that there are multiple varieties of citation styles–APA, MLA, IEEE, and more. Even sometimes, authors can use the same style in an inconsistent way. Other reasons that could throw off a machine from identifying a reference are typos, abbreviations, missing punctuation, or format switching mid-paper [1, 2]. Named entities like author names or journal titles constantly change, and new citation styles continue to appear, which makes dictionary-based or rule-based approaches weak and error-prone [2].

To solve this problem, researchers have turned to machine learning and treated the reference parsing problem as a **sequence labeling problem**. Which means that we can treat each word or token as part of a sequence, and we try to label it with the field it belongs to– like "Author", "Title", or "Year". Earlier methods relied on models like **Hidden Markov Models (HMMs)** and especially **Conditional**

Random Fields (CRFs), which capture local dependencies in sequences and have been used a lot in the natural language processing field for structured predictions [3, 4].

One of the most well-known tools built using these ideas is **ParsCit** (Peng and McCallum, 2004; [1]), a system that uses both CRFs and heuristic post-processing. Others, like CiteSeerX [5], and Mendeley, have adopted similar methods to power large-scale academic search and indexing [2]. These systems allowed us to see how a supervised learning models could generalize across a range of citation formats, if they were given good training data. Most recently, deep learning methods have entered the picture. **BiLSTM + CRF** models can combine both the sequential context modeling of BiLSTMs and the prediction power of CRFs. These types of models have shown strong results in capturing long-distance dependencies often found in complex reference strings. And in a newer architecture, **Transformer-based** models that use self-attention have made them powerful tools for sequence tasks, even though they are data-hungry models [6, 7, 8, 9].

Several studies have compared these models directly. One, described by [6], evaluated different approaches like CRFs, LSTMs, rules, and templates under the same data conditions. They found that CRFs remained solid performers, especially when training data was limited, but LSTM + CRF models were superior in accuracy. Another study [7] compared real and synthetic training data for this task and found that synthetic data can be useful, if generated carefully. Despite these advances, the problem hasn't been solved. Reference styles are still not very well structured and hard to annotate, and some models can handle different edge cases better than others. There's still room to explore new approaches, especially ones that can work well with limited or adapt to new citation styles.

This is where our work comes in; we reviewed the reference parsing problem with a new perspective and, instead of solely relying on deep learning architectures, we take inspiration from **AnyStyle**, an open-source reference parser that uses a set of hand-engineered features to guide the model. These hand features show us that a well-designed feature space can go a long way in improving the performance, especially when paired with an appropriate model [10]. We explore how these kinds of features can enhance the performance of sequence models for reference parsing. Specifically, we compare traditional CRFs and BiLSTM + CRF architecture under a shared framework, but we incorporate a set of hand-engineered features inspired by

AnyStyle with different types of embeddings. Our experiments use the **Giant** corpus [11] - The 1-Billion Annotated Synthetic Bibliographic-Reference-String Dataset for Deep Citation Parsing - for training and evaluating the model, ensuring that all models are tested under the same circumstances.

By analyzing performance across these models, we aim not only to evaluate their effectiveness but also to identify the most impactful combination of techniques. Our results show that using a CRF model with high-quality contextual embeddings—specifically, the Linq-Embed-Mistral BERT-style encoder—provides an optimal balance between accuracy and generalization. This demonstrates that a well-chosen embedding model can outperform more complex pipelines relying on handcrafted features. Through this study, we offer practical guidance for building accurate and scalable reference parsers, contributing to the development of robust citation indexing systems.

Chapter 2

Related Work

The problem with bibliographic reference string parsing has been a challenge in the field of information extraction for use inside digital libraries and indexing systems. And over the last couple of years, researchers have explored different approaches to segment and label reference strings more accurately, from handcrafted rule-based systems and feature-engineered machine learning models to modern neural architectures. These various approaches aim to convert normal citation strings into structured metadata fields such as author, title, journal, year, and volume, which are very important for digital indexing, citation networks, and scholarly search systems.

Reference string example:

```
Ritchie, E. and Powell, Elmer Ellsworth. (1907). Spinoza
and Religion. Philosophical Review 16 (3), p. 339-340,
[online] Available from: http://dx.doi.org/10.2307/2177340
```

Annotated reference string example:

```
<author>
    Ritchie E. and Powell, Elmer Ellsworth
</author>
<punc>
    (
</punc>
<year>
    1907
</year>
<punc>
    )
</punc>
<title>
    Spinoza and Religion.
</title>
<container-title>
    The Philosophical Review
</container-title>
<volume>
    16
</volume>
<punc>
    (
</punc>
<issue>
    3
</issue>
<punc>
    )
</punc>
<punc>
,
</punc>
<page>
    p. 339
</page>
<other>
    [online] Available from:
</other>
<URL>
    http://dx.doi.org/10.2307/2177340
</URL>
```

2.1 Early Methods and the Rise of Machine Learning

The first tries to automate the reference parsing task, relied mainly on regular expressions and template-matching techniques. These methods worked well within limited domains or under specific citation styles but quickly became weak and unreliable in the face of noise, new variations of reference styles started showing up, and multilingual citations. The introduction of probabilistic models marked a significant turning point. Notably, Hidden Markov Models (HMMs) were among the earliest probabilistic sequence models applied to this task [3]. However, the use of Conditional Random Fields (CRFs) surpassed HMMs because they offered a discriminative framework more suited for structured prediction tasks such as reference segmentation and reference parsing. CRFs were introduced as a general and effective replacement that could capture long-range dependencies and handle random, overlapping features without needing strong independent assumptions [4]. CRFs were the most widely used solutions in most of the early systems, including the well-cited ParsCite system, which combined a CRF model with a collection of heuristics to deal with pre- and post-processing [1]. ParsCit, in particular, was implemented as a modular, open-source tool and used 23 hand-engineered features that are token identity, position, punctuation, orthography, and external dictionaries [1]. It gave a strong performance on benchmark datasets such as CORA [12], and set a standard for other reference parsing systems. CiteSeerX and Mendeley also used similar techniques with dictionary lookups and CRF-based models [1, 5].

2.2 Limitations of Feature-Engineered Systems

Even though they were successful initially, CRF-based systems have some limitations. First of all, their reliance on hand-engineered features made them very dependent on the domain they're trained on. Features like name dictionaries or publisher lists don't generalize well between languages or domains. Second, while CRFs can model local context very easily, they can't handle more complex, long-range dependencies that could appear in many citation styles. Also, most CRF-based models have been trained on homogeneous corpora (typically English-language, computer science citations), which made them limited to use on real-world, multilingual corpora [2]. Studies such as CERMINE [13] and BibPro [14] further continued to improve finer feature sets and templates to raise accuracy, yet the limitations remained the same: scalability and generalization. The systems also typically ignored the growing availability of large-scale, unlabeled citation data and instead relied on limited, curated gold-standard datasets.

2.3 Transition to Deep Learning Models

To avoid the mentioned limitations, researchers began exploring neural network models that could potentially learn feature representations from annotated raw text without intermediate manual feature engineering. The neural ParsCit system [2] is one of the first models that tried to apply deep learning to solve the reference parsing problem. The researchers on that model used a BiLSTM + CRF architecture, in which both character and word embeddings are used as input to the bidirectional LSTM network, and the output was then decoded using the CRF layer. This combined approach managed to use the power of both deep and structured models: the BiLSTM managed to capture contextual information in both directions, and the CRF made sure that label sequences were consistent. Neural ParsCit showed better performance than traditional CRF-only models, especially on multilingual and out-of-domain references. More importantly, the researchers trained the model using large unlabeled citation corpora and showed that representation learning could help models generalize better to unfamiliar styles and vocabularies. The study also discussed how dictionary-based features used in a system like ParsCit are still vulnerable. It avoided rigid lexicons, and its neural model helped with robustness while handling named entities, low-frequency words, or unusual punctuation. This was a step forward in making parsing systems that would address the diversity of real-world academic citations.

2.4 Emergence of Transformer-Based Citation Parsers

As deep learning developed, transformer models began showing a huge dominance over recurrent architectures in sequence modeling tasks. This eventually translated into a new generation of citation parsing programs, such as TransParsCit [15], that used transformers openly for reference segmentation. TransParsCit introduces a Transformer-CRF hybrid that learned on the GIANT dataset. The size of the dataset provided generalizability across an enormous range of formatting styles, a common cause of challenge in citation parsing.

Instead of counting on recurrent layers or handcrafted features like BiLSTMs, TransParsCit utilizes a transformer encoder to provide contextual token representations. These are passed through a CRF decoding layer to maintain label consistency and inter-token dependency. Contrasting with earlier work such as Neural ParsCit, which depended on word-based and character-based embeddings, this architecture avoids feature engineering entirely and is parallelizable by design in transformers. The model was evaluated on the CORA benchmark test set, and it achieved an F1-score of more than 84% on principal fields upon training on 219,000 reference strings. With increased training data size, the performance consistently increased, indicating that transformer-based models are highly benefited by large, diverse training corpora.

This paper demonstrates the strength of transformer-based models even when there are no handcrafted linguistic features and indicates the strength of synthetic training data at scale.

2.5 Innovations with Contrastive and Prompt Learning

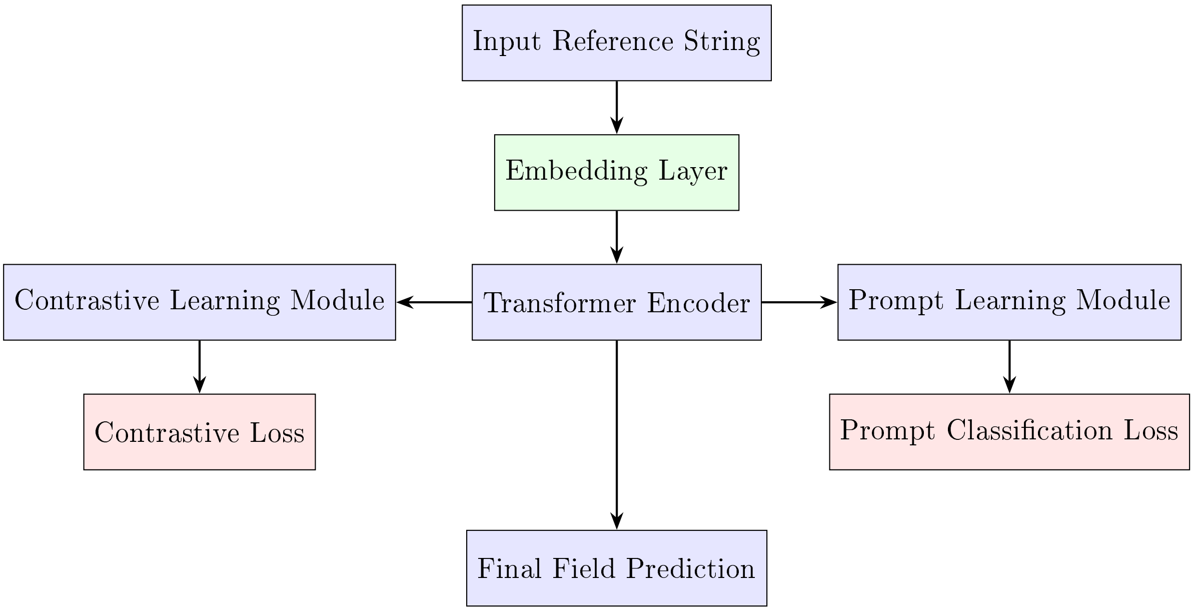
Recent state-of-the-art parsing bibliographic references have also turned towards state-of-the-art representation learning techniques to show better performance in low-resource and multilingual settings. One such notable contribution is Yin and Wang's [16] `CONT_Prompt_ParsRef` model, in which they introduce a robust combination of prompt-based learning and contrastive learning. Contrastive learning in this context helps the model discriminate between different kinds of metadata by training more discriminative token embeddings. In plain language, the model tries to group similar tokens (e.g., all tokens of type "Author") closer to one another in the embedding space, and push tokens of distinct types (e.g., "Author" and "Title") apart from one another. Clustering in this way enables the model to produce sharper decision boundaries, which makes it easier to predict correct labels — especially when dealing with overlapping or ambiguous field types. For example, the model might confuse "IEEE" as part of a journal name or even as part of a subset of a conference name. Using contrastive training, "IEEE" would be closer to other journal names in embedding space and farther away from out-of-domain entities like author names or page numbers. This improves classification and reduces errors caused by similar structures within domains.

Besides this, the prompt learning component guides the model during training by introducing example-based templates to the input. The prompts provide the model with a structured context — e.g., examples of journal titles or author names — and serve as a soft supervision signal. Instead of learning from the data blindly, the model uses these prompts to "understand" what each label category typically looks like. During training, the goal is to move the representation of a token in the input sequence close to the representations of its corresponding prompt examples.

The combination of prompt learning and contrastive learning makes the system especially robust on difficult, varied citation styles and low-resource scenarios. In their experiments on a bilingual dataset (English and Chinese), the authors showed that their full model outperformed several strong baselines — BiLSTM + CRF, BERT + CRF, and CNN — with 96.39% F1 on English references. Their ablation study showed that both components were significant, with contrastive learning having a slightly greater impact on performance. This paper is a compelling demonstration that representation learning methods like contrastive clustering and template-based prompting can have a significant impact on reference parsing — not only in terms of accuracy, but also robustness and flexibility.

2.6 Comparative Evaluations and Modern Architectures

Following up on Neural ParsCit's [2] observations, recent work has focused on the task of comparatively assessing modern NLP architectures for reference parsing. In particular, Cuéllar Hidalgo et al. (2024) [17] conducted an empirical comparison of three well-known architectures: CRF, BiLSTM + CRF, and Transformer + CRF. Their study used the gigantic GIANT corpus [11] (comprising over 900 million an-



Schematic diagram showing input embedding,
shared encoder, and parallel contrastive/prompt learning paths

Figure 2.1: Overview of the CONT_Prompt_ParsRef architecture combining contrastive and prompt learning.

notated references across 1500+ citation styles) for training and the well-established CORA corpus [12] for testing. Their findings once more validated the superiority of BiLSTM + CRF over pure CRF models in identifying complex syntactic structures in reference strings. What was more interesting, Transformers-based models showed competitive performance but did poorly in cases with limited labeled data or noisy labels, which indicated that even with their theoretical advantages, Transformers still need a lot of careful tuning or more training data to outperform BiLSTMs in this use case. Perhaps one of their most significant contribution was to standardize preprocessing and consistent testing for all models. This way, it helped us to know that performance differences were because of the architectural differences between the models and not the inconsistencies in the data. These experiments also pointed out the importance of embedding techniques, using Byte-Pair Encoding with character-level features, which enhanced the performance of all models in handling token-level irregularities, such as hyphenated names or Roman numeral volume numbers.

2.7 Other Tools and Alternative Approaches

In addition to traditional and neural sequence models, several other systems have explored new directions. The ParsRec approach [18] suggested a meta-learning system that proposes the best parser (out of a set of candidate systems) for a specific reference string, according to metadata and structural information. This is part of a growing tendency towards ensemble and hybrid solutions that take advantage of the strengths of different tools. In addition, software like GROBID [19] and AnyStyle [10] have become very popular due to their simplicity and deployment in digital library pipelines. AnyStyle is especially valuable because it employs a hybrid approach with the utilization of hand-crafted features that serve as input into a CRF model, and simple training on proprietary datasets. Its adaptability and low-resource suitability make it suitable for libraries that have specialized citation formats or non-English content.

2.8 Summary and Open Challenges

In short, the field of bibliographic reference string parsing has progressed from rule-based and HMM to CRFs and, more recently, to neural models such as BiLSTM and transformer models. Each has addressed problems of the previous generation, from lack of generalization to dependence on hand-engineered features. But there remain issues, most systems get trained and tested on English-language datasets mostly, and noisy or OCR-extracted references are still a concern. While neural models generalize more, they are data hungry and may not work well in low-resource environments. Finally, deployment in real workflows (citation indexing, digital repositories) required robust tools that compromise between accuracy, speed, and flexibility. In our work, we attempt to further explore this balance by re-examining the value of hand-engineered features, like in models such as AnyStyle, and putting their cooperation with modern neural frameworks to the test. We hope that by integrating structured, interpretable features into data-driven learning, we can build models that are both efficient and practical in a wide range of bibliographic scenarios.

Chapter 3

Methodology

3.1 System Architecture

The goal of this system is to convert unstructured reference strings into structured bibliographic metadata by the process of assigning each token with its corresponding semantic field (e.g., author, title, publisher, year). This task is framed as a sequence labeling problem, and the model learns to create field-level labels for a string of tokens to produce labeled output that can be consumed by digital libraries, citation indexes, or metadata extraction programs. The system architecture is composed of modular stages: input ingestion and preprocessing, hand-engineered feature extraction, contextual embedding generation, feature concatenation, and token-level sequence modeling with supervised learning models. The stages are designed to be extendable and flexible in order to facilitate exploration with various types of embeddings, feature sets, and model architectures.

3.1.1 Input and Dataset Constraints

The system uses the GIANT dataset [11], which is one of the largest annotated reference string corpora available in this field anywhere. The complete GIANT corpus is made up of more than 900 million citations covering thousands of scientific styles and topics. However, due to computational constraints, it was not possible to train models on the entire corpus. A random sample of 5 million citation strings was chosen to ensure high diversity among citation styles, fields, and languages. Effective experimentation was conducted at this scale while preserving the dataset's challenging variability. All the strings in the dataset are first annotated in an XML-based hierarchical format that provides field-level segmentation, but it's not directly compatible with the standard sequence labeling formats. The XML annotation has both the main labels mentioned previously and additional metadata or formatting tags that are not relevant to the core parsing task. As a processing step, the extra labels were removed, and the remaining of the string turned into a flat reference.

Annotated reference string before parsing example:

```
<author>
    <family>Ritchie</family>,
    <given>E.</given> and
    <family>Powell</family>,
    <given>Elmer Ellsworth</given>
</author> (
<issued>
    <year>1907</year>
</issued>)
<title>Spinoza and Religion.</title>
<container-title>The Philosophical Review</container-title>,
<volume>16</volume>(
<issue>3</issue>), p.
<page>339</page>. [online] Available from:
<URL>http://dx.doi.org/10.2307/2177340</URL>
```

Annotated reference string after parsing example:

```
<author>
    Ritchie E. and Powell, Elmer Ellsworth
</author>
<punc>
    (
</punc>
<year>
    1907
</year>
<punc>
    )
    </punc>
    <title>
        Spinoza and Religion.
    </title>
    <container-title>
        The Philosophical Review
    </container-title>
    <volume>
        16
    </volume>
    <punc>
        (
    </punc>
    <issue>
        3
    </issue>
    <punc>
        )
    </punc>
    <punc>
    ,
    </punc>
    <page>
        p. 339
    </page>
    <other>
        [online] Available from:
    </other>
    <URL>
        http://dx.doi.org/10.2307/2177340
    </URL>
```

And then turned into a token-level BIO tagging scheme. In this scheme, every token is labeled as the beginning (B-) or inside (I-) of some field, or as outside (O) if it does not belong to an identified segment. This conversion was performed in order to get the data into a format that can be used with off-the-shelf sequence labeling models. Some of the last tags are B-Author, I-Title, B-Journal, and B-Year. By using BIO encoding, the models are able to learn fixed field boundaries in citation strings, even across highly varying formatting styles.

3.1.2 Preprocessing and Tokenization

Raw reference strings can be different in the way they're formatted, their punctuation, and the language, which makes preprocessing an important step. The tokenization method used in the system is whitespace-sensitive, punctuation-sensitive, and character pattern-sensitive. This makes sure that tokens like initials, abbreviations, hyphenated names, volume/issue numbers, and DOIs are appropriately separated and paired with the correct label. All the special characters and punctuation are kept unless otherwise filtered because they can have essential roles in delimiting fields. For instance, the presence of parentheses for a year, or quotation marks for a title, often serves as an implicit indicator that is useful to handcrafted features as well as learned representations.

3.1.3 Feature Extraction

One of the innovations of this system lies in its use of hand-engineered features inspired by the AnyStyle [10] reference parsing framework. AnyStyle has been proven to perform well with lightweight CRF models that are guided by feature sets that are carefully crafted by hand. In this project, the same strategy is followed, and features that are domain-aware and structurally informative are derived. The feature space includes:

Orthographic features: capitalization, digits, punctuation types.

Lexical cues: dictionary matches on publication names, author names, or publication types that are common.

Positional features: token position from the start/end of the string, section, position.

Contextual features: features from the preceding and following tokens.

These are generated for each token within the reference string and encoded into a sparse vector format. They are primarily used for adding domain-specific patterns into the learning process, especially useful for rare citation styles and unusual formatting.

3.1.4 Embedding Generation and Feature Fusion

In addition to hand-engineered features, the system uses contextual embeddings to acquire deeper semantic and syntactic relationships. Two embedding methods are explored:

1. Byte-Pair Encoding (BPE) [20] embeddings: Subword embeddings that are robust to out-of-vocabulary, low-frequency, or unknown words. BPE is found to be beneficial in bibliography information, where there are many domain- or author-related words with low frequencies.

2. BERT [21] embeddings: Pre-trained contextual embeddings based on the Transformer architecture. BERT captures long-range dependencies and subtle meaning from the entire sentence context. Its success in numerous NLP applications encourages its application in this reference parsing pipeline.

These embeddings can be used individually or combined with the hand-engineered features to produce a joint token representation. This combination enables the system to take advantage of both data-driven learning and domain-guided structure, making use of both approaches.

3.1.5 Sequence Labeling and Output

The final step of the system is sequence labeling, in which the token representations are input into one of the supervised models:

- Conditional Random Fields (CRF) [4]: A strong baseline for structured prediction, especially with hand-engineered features.

- BiLSTM + CRF: A deep learning model that captures bidirectional context and combines it with a structured output layer.

These models are trained using annotated reference strings to learn the transition patterns between labels and to predict the correct field label for each token. The output is a sequence in which each token is marked up with its label.

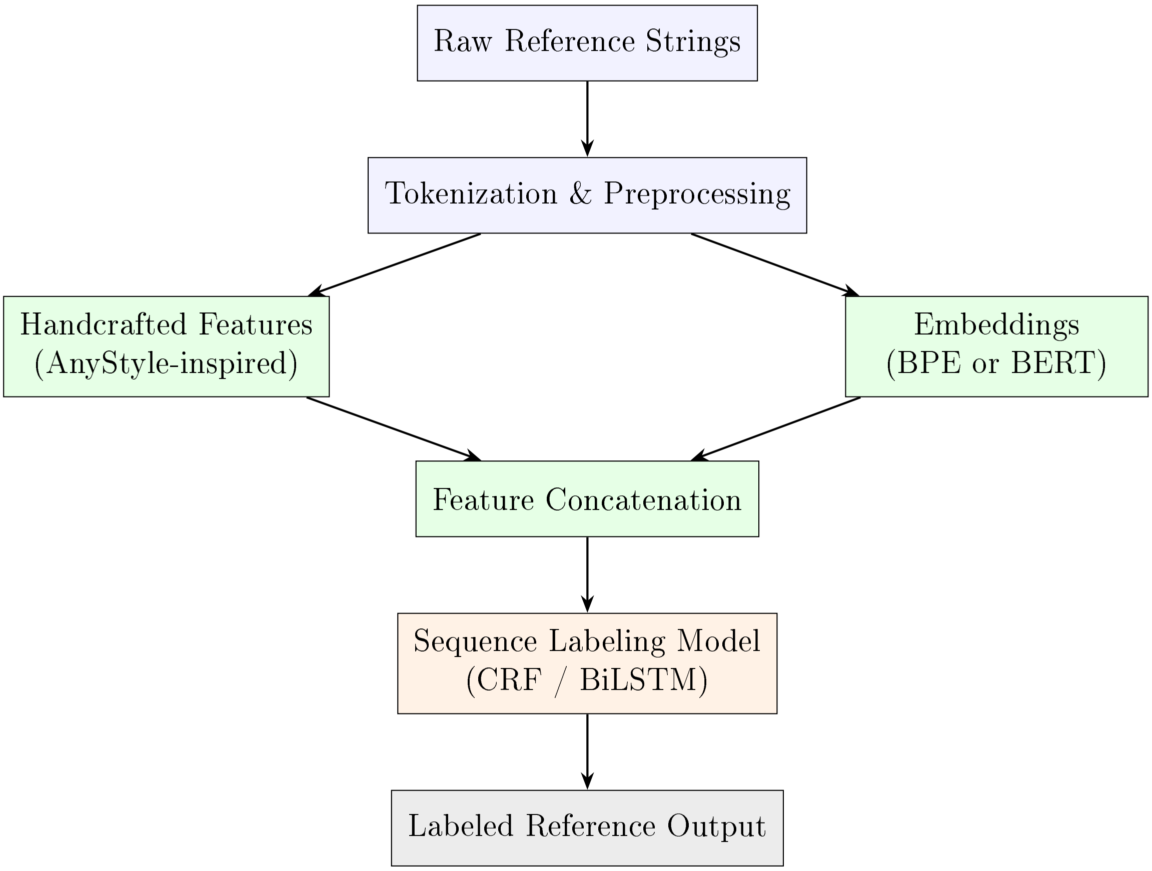

Raw Reference Strings
Tokenization & Preprocessing
Handcrafted Features (AnyStyle-inspired)
Embeddings (BPE or BERT)
Feature Concatenation
Sequence Labeling Model (CRF / BiLSTM)
Labeled Reference Output

Figure 3.1: System architecture for reference string parsing.

3.2 Feature Engineering

Feature engineering is vital to the reference parsing system introduced in this work. The bibliographic reference segmentation is a prediction problem, one where the model must assign a field-level label (author, title, year, etc.) to each token in a sequence. In order to be able to do this efficiently, the model requires informative representations of each token, those representations should capture both the token's content and its contextual significance.

In the past few years, there has been a growing popularity in employing deep contextual embeddings like BERT [21] to embed text in natural language processing tasks. Even in tasks like citation parsing, where structure and formatting have strong semantic clues, traditional hand-engineered features can still have a significant value. They extract surface information that is often consistent across citation styles, like punctuation patterns, capitalization, or token order.

To gain a balance between generalization and interpretability, our system uses two types of features:

1. A set of **handcrafted features** inspired by the AnyStyle [10] citation parser.
2. **Learned embeddings** derived from either subword-level models (BPE) [20] or deep contextual models (BERT) [21].

In the following sections, we describe each feature group in detail, beginning with the handcrafted features.

3.2.1 Handcrafted Features

In this work, one of the design decisions was to complement neural embeddings with a robust set of hand-engineered features. These features provide interpretable, low-level cues that are very useful for detecting structural patterns in citation strings, patterns that are often overlooked ot inconsistently represented in pretrained language models. Our approach to feature engineering is influenced by AnyStyle, an open-source library for citation parsing developed by Sylvester Keil [10].

AnyStyle models citation parsing as a sequence labeling task and relies on a CRF model that has been trained on a wide range of citation styles. Unlike neural architectures, which can rely a lot on pretraining and embeddings, AnyStyle achieves high performance through a well-crafted set of features that encode orthographic, lexical, and contextual information. These include features such as token capitalization, punctuation patterns, and dictionary lookups, all these features are designed to enable the model to detect bibliography entities in different types of citations.

In our system, we take and extend this method by constructing a set of features to feed into the model, concatenated with subword and contextual embeddings. This mixed representation allows the model to take advantage of deep contextual understanding through embeddings and interpretable surface signals from the hand-engineered features. To facilitate compatibility, we re-implemented a subset of AnyStyle's feature classes, modifying or adding to some where necessary.

In the following sections, we'll describe each feature class, its role, and how it helps in the overall reference parsing task.

Affix Feature

The affix feature extracts fixed-length prefixes and suffixes of tokens. It is useful for recording common patterns in names, abbreviations, or technical terms that appear repeatedly in bibliography citations. For instance, author initials like "J." or "Ph.", or journal abbreviations like "IEEE" or "JMLR", have a specific pattern at the start or end of words. This feature works by taking a number of characters from the beginning (prefix) or end (suffix) of the token. In our case, two variations of this feature class were applied: one that takes the first two characters (prefix) from the token, and another one that takes the last two characters (suffix) from the token. These affix pieces are built up incrementally–so the prefix extractor for a token like "Journal" would extract "J" and "Jo", and the suffix extractor would extract "l" and "al".

Affix features make the model able to generalize to out-of-vocabulary tokens by picking up on subword patterns that could indicate specific field types. For example, journal names often have a shared suffix like "-ology" or "-ics", and author names often have a predictable pattern of abbreviations. While it's simple in structure, these features are helpful in field prediction, especially where token-level embeddings on their own can sometimes lack attention to detail.

Brackets Feature

The brackets feature captures data on whether a token is enclosed by or adjacent to typical bracket symbols, such as parentheses `()`, square brackets `[]`, or angle brackets `<>`. This is a useful feature because citation formats could put some metadata in brackets – e.g., publication years, volume numbers, or reference indices – to visually distinguish them from other fields. This feature adds a tag to the token, such as `parens`, `square-brackets`, `angle`, or more specific tags such as `opening-paren`, `closing-square-bracket`, etc., depending on the shape of the token. For example, the token (2003) would be tagged as `parens`, and a token starting with `[` would be tagged as `opening-square-bracket`. Tokens without any brackets are simply tagged as `none`. When the token does not belong to one of the typical patterns, it is tagged as `other`.

By flagging bracket types and positions, this feature enables the model to recognize common citation patterns, like parenthesized years or square-bracketed reference numbers, that are used across many citation styles. This kind of formatting hint is sometimes necessary to help get an accurate segmentation and recognize which field this token might belong to, especially in noisy or reference strings extracted using OCR.

Canonical Feature

The canonical feature provides a normalized representation of each token by removing accents and converting it to a standardized, lowercase form. The main goal is that tokens of the same structure or meaning are treated in the same way by the model, whether they appear different due to language, style, or formatting differences. In order to achieve this, the feature captures the first shape of the token (before any change) and performs some normalization operations. It initially performs Unicode normalization to break the characters down to their base form – i.e., separating a letter from its accent marks. It then removes all the accents and other diacritical marks, retaining only the base characters. Finally, it scrubs the string of any additional formatting noise and makes it lowercase.

For instance, the name `García` is shortened into `garcia`, and `MÜLLER` to `muller`. This makes it easier for the model to recognize that these are likely the same name or entity with minor stylistic or typographic differences. This capability proves useful, especially while tokenizing citations with names, titles, or journal names in multiple languages or styles. It minimizes inconsistency in the input and allows the model to generalize more effectively over the large variety of tokens found in citation strings.

Caps Feature

The caps feature tracks information about the capitalization pattern of each token. Capitalization tends to be an effective cue in reference strings — i.e., it can signal the occurrence of names, acronyms, or stylized title spacing. By observing and categorizing these patterns, the model has the ability to enhance on identify between different types of fields such as author names, journal names, or abbreviations.

This property works by looking at the original shape of the token and assigning one of a set of classes depending on whether and where there is one or more uppercase letters:

- **single:** A single uppercase letter (e.g., `J`), common in shortened names or initials.
- **initial:** A token that starts with an uppercase letter followed by a lowercase letter (e.g., `John`), typically used for names or title-cased words.
- **caps:** A word token that consists entirely of all capital letters (e.g., `IEEE`, `SCIENCE`), typically delimiting acronyms or publication titles.
- **lower:** A word token consisting only of lowercase letters (e.g., `and`, `volume`), typically less informative in a stand-alone.
- **other:** An extra bucket for anything token-wise not accommodated by the above patterns, e.g., words of mixed cases, numbers, or punctuation.

By assigning tokens to these capitalization classes, the model is more likely to correctly infer the possible function of a token within a citation, particularly when combined with other structural or contextual information.

Category Feature

The category feature determines Unicode character type of the first and last token characters to yield basic structural details. Through this analysis, the model achieves token classification based on characters, which provides information about first-letter identification and terminal punctuation, and symbol and number presence.

The feature retrieves the first and last tokens and then assigns them to the appropriate Unicode general categories through its mapping process. The Unicode general categories cover uppercase letters (`Lu`), lowercase letters (`Ll`), modifier letters (`Lm`), numbers (`N`), punctuation types (`P`, `Pc`, `Pd`, etc.), symbols (`S`), and unspecified other categories. Character data that cannot be categorized goes under the category of `none`.

For example:

- When applied to `Vol.` the Vol. feature would provide output as `Lu` for `V` and `P` for the period.
- The token 2023) provides both `N` as a number category and `Pe` for parenthesis.

The model can apply its knowledge to numerous token forms because this feature ignores token content. The reference string benefits from this feature if it contains structured formatting cues that indicate field boundaries or field types through specific brackets or numbers and characters.

Dictionary Feature

The dictionary feature tries to see if a token matches a known word in a collection of lists that includes names, places, publishers, and journal names. The lists are being read from a dictionary file, grouped into the lists mentioned above. It is useful for detecting entities that would show up in a frequent way in reference strings, such as publisher names, city names of publication, or common journal abbreviations. Each token is compared and checked to see if it exists in one of these lists:

- **name** - common first or last names in the author field
- **place** - cities or locations typically in publisher field
- **publisher** - publishing firm or organisation names
- **journal** - full journal titles or journal abbreviations

If a token exists in one of the lists, it is marked as `T` (true) for that category; otherwise, it'll be marked `F` (false). The output vector from this feature will have four components, each referencing one of the dictionary categories. For example, token `Springer` might return `[F, F, T, F]`, which indicates that there is a match in the publisher dictionary but not in other dictionaries. This feature will allow the model to create a connection between a token with a role for it, which could lead to an increase in accuracy in the field segmentation.

The dictionary lists are extracted from structured reference data and preprocessed for consistency. Although these features are not very reliable and couldn't be sufficient to provide correct labeling, they help with adding precision signals when used with contextual or structural features.

Keyword Feature

The keyword feature uses keyword patterns to assess token classification regarding previously known semantic categories. This detection method is intended to identify particular metadata signals within citation strings by analyzing both words and symbols.

The internal operation uses pre-determined groups of regular expressions that control category matching. The semantic roles (`editor`, `journal`, `date`, `volume`) exist as separate categories, and the system checks each token against different language versions of relevant keywords. Editor-related terminology in the metadata section contains multiple English designs, including `ed.`, `editors`, and `edited`, in addition to German (`herausgeber`) and Spanish or French equivalents (`compilador`). The grammar accepts character sequences from the symbolic series and the CJK series. This feature evaluates by testing every pattern one by one until it encounters a category expression that perfectly matches the current token. The feature operates without producing an output whenever no suitable matching pattern exists. Some example categories include:

- **editor:** tokens like `ed.`, `Hrsg.`
- **volume:** `vol.`, `no.`, `issue`, `heft`
- **date:** matches several months or season names such as `May`, `Fall`, and `Herbst`.
- **journal:** words like `Journal`, `Quarterly`, `Review`, or `Zeitschrift`
- **accessed:** metadata like `retrieved`, `accessed`, `abgerufen`.
- **locator:** `doi`, `url`
- **etal:** short forms like `et al.`, `others`

Through this feature, the model gains enhanced semantic cues, which improve its ability to detect tokens with functional significance above visual presentation form. The feature precisely detects citation components, especially for their publication placement (`in`) as well as their authorship (`author`, `editor`, `translator`) and reference access methods (`url`, `arxiv`, `pubmed`). Tokens that match the established keyword categories function as strong indicators for the field, but only some tokens have matching entries.

Locator Feature

The Locator function identifies persistent digital identifiers along with external resource pointers that function as tokens. The system contains the main groups of locators: DOIs, URLs, ISBNs, and PubMed IDs, along with additional academic citation locator types. Recognition of these tokens is vital because they most commonly occur at citation endings while holding semantic and structural differences from the rest of the fields. The feature uses regular expression patterns to detect persistent digital identifiers, which also include commonly used external resource pointers:

- The token detection system verifies terms which include `DOI`, `ISBN`, `URL`, `PMCID` and `PubMed`.
- A Digital Object Identifier stands as `10.` followed by a numeric prefix and a suffix which combines as `10.1000/xyz123`.
- The feature detects typical URI forms which start with `http://`, `https://`, or `ftp://` within web addresses.

The feature returns `'T'` (true) when any defined patterns match within the analyzed token, indicating a potential locator. Otherwise, it returns `'F'` (false). Example:

- `https://doi.org/10.1007/s00799-018-0242-1` → `T`
- `PMID: 12345678` → `T`
- `Springer` → `F`

Through this feature, the model detects tokens containing external source references and accesses information, thus enhancing its ability to accurately classify fields, particularly for digital and web-based references.

Number Feature

The number feature categorizes tokens based on whether and how there is numerical information. Numbers are essential in the middle of many bibliographic fields — e.g., years, volume numbers, ranges of pages, ISBNs, or identifiers — and understanding their format can help the model to decide the token's likely function in a citation. This feature applies a series of pattern-matching rules to translate all numeric tokens into a particular category. Some of the main categories include:

- **volume:** Tokens that have the appearance of a volume and issue format, such as `12(3)` or `5:7`.
- **isbn:** Strings that conform to the pattern of ISBN numbers, both starting with `978` and those starting with `979`.
- **year:** Four-digit years from a reasonable range of history, such as `1998` or `2023`.
- **quad, triple, double, single:** Tokens made up of 4, 3, 2, or 1 digits respectively. These simple forms typically represent years, page numbers, or brief identifiers.
- **all:** Tokens made entirely of digits, but not in one of the other specialized forms.
- **range:** Numerical ranges with hyphens, e.g., `123-145`, commonly page numbers.
- **idnum:** Alphanumeric identifiers in which a number is prefixed with letters or codes, e.g., `ABC-123` or `ISSN2049`.
- **ordinal:** Numerical and alphabetical combination tokens, e.g., `3rd`, `21st`, or `2a`, appearing infrequently in editions or titles.
- **numeric:** Tokens having at least one digit but none of the preceding patterns.
- **roman:** Roman numerals like `III`, `XIV`, or `iv`, appearing infrequently to number chapters, volumes, or appendices.

Otherwise, if the token does not belong to any known numeric pattern, it gets labeled as `none`.

By classifying these fine-grained numeric types, the model gains a deeper sense of the organization of the reference, having the ability to distinguish between a publication year, an ISBN, and a volume/issue number, even when all appear to be numbers. This helps to improve the precision of field classification, especially for citation styles that vary in the way and where numbers appear.

Position Feature

The position feature holds the relative location of a token within the reference string. In citation parsing, when a token's position can powerfully predict its role, its position tends to reveal a great deal about its function. Authors' names tend to be toward the front, say, whereas publication dates, URLs, or page numbers tend to appear closer to the end. This feature gives back one of the following values, depending on the token's position in the sequence:

- **only:** If the token is the only token in the sequence
- **first:** If the token is the first token in the sequence
- **last:** If the token is the last token in the sequence
- **A relative position value** (as an integer between 0 and 10) if the token is in the middle somewhere

The relative position is calculated as a coarse-grained proportion: the index of the token divided by the number of tokens and adjusted by an absolute level of precision (for example, 0 to 10). As an example, a token in the middle of a sequence of 20 tokens would receive the value 5.

By exposing the model to this positional information, the feature helps the model learn to recognize in which portions of a reference string one will most likely discover specific types of information. This can be extremely helpful in free-form or otherwise variably styled references, where even the formatting cannot give sufficient indication for field separation.

Punctuation Feature

The punctuation feature recognizes the presence and type of punctuation in a token. Punctuation will generally play a structural role in citation strings, separating fields or designating formatting conventions. For example, colons will separate titles and subtitles, periods will designate abbreviations, and hyphens will denote ranges like page numbers or dates. The feature looks at each token and labels it based on the punctuation that it contains:

- **none:** The token does not have any punctuation.
- **colon:** The token contains a colon (`:`), which is used for title or subtitle delineation.
- **hyphen:** The token contains a hyphen or dash, which can indicate a range (i.e., `123-145`) or be part of a compound word.
- **period:** The token contains a period (`.`), which can indicate abbreviations or sentence finality.
- **amp:** The token has an ampersand (`&`), often used to join author names (e.g., `Smith & Johnson`).

- **other:** The token has punctuation not in one of the above classes.

By capturing these distinctions, the punctuation feature provides useful cues to token boundaries, field separators, and potential abbreviations — all of which are useful to the model in annotating different parts of a citation.

Terminal Feature

Terminal property identifies in which way a token is terminating, namely if it is terminating in punctuation, brackets, or quotes. In citations of bibliography, the punctuation a token is terminating in often signals the end of a field or the separation between units of meaning, e.g., the end of an author name, title, or publication date. This feature looks at the trailing characters of the token and categorizes it as one of four based on the quality of the ending punctuation it possesses:

- **strong:** The token contains a strong punctuation mark such as a period (`.`), closing parenthesis (`)`), or square bracket (`]`), possibly followed by a quotation mark. These tend to mark the end of a field or sentence.
- **moderate:** The token is suffixed with a quotation or colon. It may be preceded by a lighter punctuation character at times. These may mark a transition, like the start of a subtitle or inline citation.
- **weak:** The token is preceded by a lighter punctuation character like a comma, semicolon, hyphen, or exclamation mark. These mark continuation but can still segment parts of a field.
- **none:** The token ends without any meaningful punctuation.

For example:

- `2023).` → `strong`
- `"Chapter 2:` → `moderate`
- `Vol. 5,` → `weak`
- `Science` → `none`

By maintaining these distinctions, the terminal feature helps the model to learn where fields most likely begin or end, especially in citation styles where such patterns are not absolutely dictated by format but instead are a function of punctuation. It plays a silent but important role in improving the accuracy of token labelling across citation formats.

Summary of Handcrafted Features

The new hand-crafted features employed in this work are inspired by how reference parsing is addressed in AnyStyle [10], but are reworked and extended to more effectively meet the needs of modern neural models. They encode a dense array of linguistic, structural, and semantic cues — including orthographic features, token position, punctuation, character types, and semantic dictionaries. Each of these provides a different perspective on the reference string, and together they form a dense representation that allows accurate field labeling independent of citation style. Later, we evaluate the contribution of these features in isolation as well as in combination with embedding-based methods.

Table 3.1: Summary of handcrafted features used in the citation parsing model

Feature	**Description**
`AffixFeature`	Extracts prefixes and suffixes from each token to detect morphological patterns and common abbreviations.
`BracketsFeature`	Identifies tokens enclosed in or adjacent to brackets like `()`, `[]`, or `<>`, often used for years or references.
`CanonicalFeature`	Produces a normalized lowercase version of the token without accents or formatting noise.
`CapsFeature`	Classifies tokens based on capitalization, e.g., all-caps, initials, or lowercase.
`CategoryFeature`	Returns the Unicode category of the first and last character, helping to detect punctuation, digits, or letters.
`DictionaryFeature`	Checks if a token exists in domain-specific dictionaries: names, publishers, journals, or places.
`KeywordFeature`	Matches tokens against keyword patterns to detect roles like editor, translator, journal, or date terms.
`LocatorFeature`	Detects persistent identifiers like DOIs, URLs, PubMed IDs, and ISBNs.
`NumberFeature`	Classifies numeric tokens (e.g., years, volumes, page ranges, Roman numerals) based on their structure.
`PositionFeature`	Encodes the position of a token within the reference string (first, last, middle, etc.).
`PunctuationFeature`	Identifies punctuation within a token (colon, hyphen, ampersand, etc.).
`TerminalFeature`	Examines how a token ends to infer punctuation strength and field boundaries.

3.2.2 Embedding-Based Features

Embedding-based features provide dense, learned token representations through embedding semantic and contextual details. Contrary to handcrafted features that rely on surface patterns and professional knowledge, embeddings are learned in large bodies of text and may involve subtle language use, structural dependencies, and sense. In this paper, we experiment with two types of embeddings: Byte-Pair Encoding (BPE) embeddings and BERT-based contextual embeddings. These embeddings are used alone or blended with hand-crafted features to enhance the model's ability to label tokens appropriately in reference strings.

Byte-Pair Encoding (BPE) Embeddings

Byte-Pair Encoding (BPE) embeddings offer a tokenization-free, light, and multilingual way of subword representation. We incorporate in this paper pre-trained BPE embeddings released by the BPEmb project, which offers subword-level vector representations for 275 languages, including low-resource languages [20]. BPE is a data-driven compression algorithm that continuously merges the most frequent adjacent symbol pairs in a sequence. For example, in English, the most frequent pair, such as `t` and `h` might be merged into `th`, then further pairs such as `th` and `e` into `the`, depending on frequency. The number of merge operations the unit receives will determine its granularity as a subword unit, with fewer being more finely grained at the character level and more being units that are more like whole words [20].

BPEmb takes advantage of this idea to split untokenized, raw Wikipedia text into many languages and then learns embeddings across subword units using the GloVe algorithm [20, 22]. The method has the following significant advantages:

- **No whole-word tokenization required,** especially suitable for languages without word boundaries (for example, Chinese, Japanese).
- **Multilinguality,** with embeddings trained over a wide variety of languages, ranging from high-, medium-, and low-resource languages.
- **Compact size,** outperforming other models like FastText in certain languages while using significantly less memory (e.g., 11MB for BPEmb vs. 6GB for FastText) [20, 23].

On tasks of evaluation, such as fine-grained entity typing, BPEmb outperformed or matched both FastText and character-based models in several languages, including

English, Chinese, and Tibetan [20]. This makes BPE embeddings particularly valuable in low-resource settings or scenarios involving efficient memory usage. The embeddings used here were selected based on their performance–dimensionality trade-offs and were fused as features in addition to handcrafted features. This allowed the model to leverage both learned semantic representations and clear, human-designed signals.

BERT Embeddings

BERT (Bidirectional Encoder Representations from Transformers) is a text encoder model created by Devlin et al. [21] to obtain superior performance on downstream NLP tasks by providing deep, bidirectional contextual embeddings. Unlike earlier models such as GPT [24] or ELMo [25], BERT is pre-trained to embed tokens based on both left and right context together at all layers. BERT uses a multi-layer Transformer encoder model, expanding on Vaswani et al. [26]. There are two main versions of the model:

- **BERT$_{\mathbf{BASE}}$:** 12 layers, 768 hidden units, 12 attention heads, 110 million total parameters.
- **BERT$_{\mathbf{LARGE}}$:** 24 layers, 1024 hidden units, 16 attention heads, 340 million parameters.

To pre-train its embeddings, BERT relies on two significant unsupervised objectives:

- **Masked Language Modeling (MLM):** 15% of the tokens in each input sequence are randomly masked during training, and the model is trained to predict the original tokens based on the entire bidirectional context. This trains the model to capture deeper language patterns and dependencies.
- **Next Sentence Prediction (NSP):** The model is trained to predict if sentence B follows sentence A, in order to enable it to learn sentence-level relations crucial for tasks like question answering and textual entailment.

BERT's input representation is constructed by combining two types of embeddings:

- **Token embeddings** based on WordPiece tokenization [27].
- **Positional embeddings** that convey token order.

These parts are summed up to form the final embedding of each token. All inputs begin with a special [CLS] classification token. There are two major ways to use BERT:

- **Fine-tuning:** The entire model is fine-tuned on a specific downstream task by adding a small output layer on top.
- **Feature extraction:** BERT is used to generate contextual token embeddings, which are then used as input to another model, e.g., a sequence tagger.

Benefits of BERT Embeddings:

- They are deeply bidirectional, representing full context around each token.
- They are pre-trained on large corpora, including the Books Corpus and English Wikipedia, so they are robust and generally applicable.
- They achieve state-of-the-art performance on a range of tasks, including sentence classification, named entity recognition, and question answering.

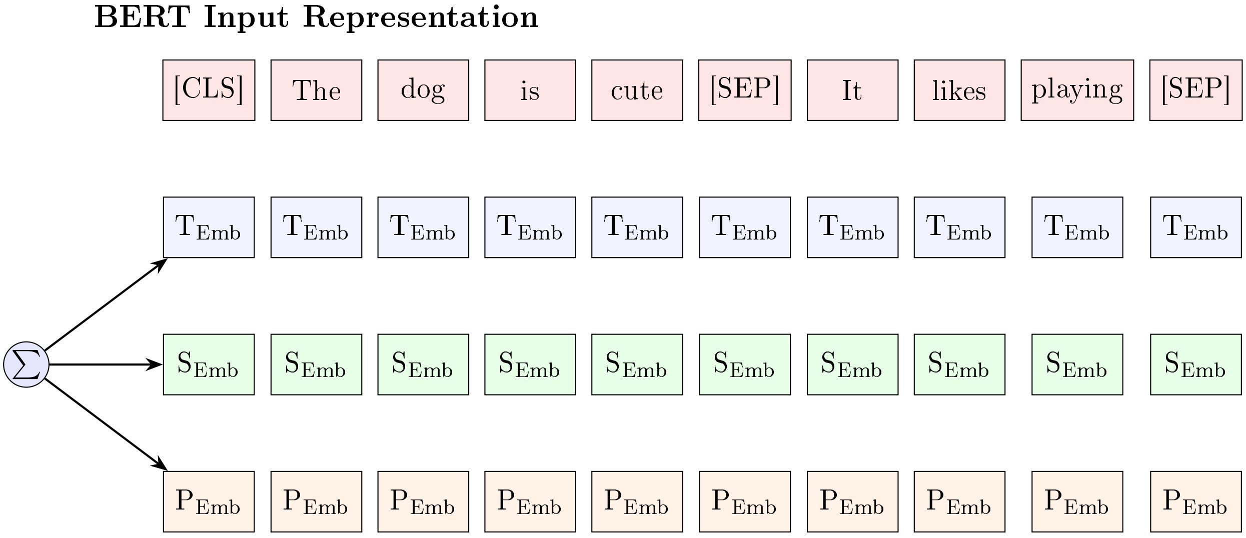


Figure 3.2: The BERT input representation: each token is represented as the sum of its token, and positional embeddings. Inspired by the design shown in [21].

Linq-Embed-Mistral: A Modern BERT-style Encoder

While the original BERT model has been an essential architecture for natural language processing, it has been the target of numerous improvements in terms of training efficiency, multilingual generalization, and embedding quality. As explained in the Hugging Face article about new variants of BERT, newer models such as MiniLMv2, E5, and Mistral-based encoders (ModernBERT) have comparatively much better performance on embedding tasks, particularly for retrieval, classification, and sentence similarity use cases.

In this work, rather than using the baseline BERT model [21], we decided to use a modern BERT-style encoder from the Hugging Face leaderboard, specifically the

Linq-Embed-Mistral model [28]. The encoder is a fusion of the Mistral transformer backbone with simplified embedding tuning, offering dense high-quality representations well adapted to token-level tasks like reference parsing. The model was one of the top-performing multilingual text encoders and offered strong out-of-the-box performance in zero-shot or low-shot settings — a valuable property for parsing reference styles not directly seen at training time. The Linq-Embed-Mistral model was used in the same application as BPE or standard BERT embeddings, generating subword-level context embeddings for each token in a reference string.

DistilBERT: Lightweight Multilingual Embeddings

In addition to modern transformer architectures, we also experimented with using lighter and faster versions of substitutes to make token embeddings. DistilBERT [29] is a light version of the original BERT base model [21], which is trained to retain most of BERT's performance with 40% less model size, 60% increased inference speed, while preserving nearly 97% of its language knowledge capabilities.

To conduct our experiments, we employed specifically the multilingual cased model of DistilBERT, by Hugging Face [30]. This is trained on the same data as mBERT (Multilingual BERT) and performs well over 100 languages. The multilingual cased DistilBERT provided compact, efficient representation that proved of immense value to handle citation data in many languages, preserving casing information in the process — essential to reference parsing since proper names, acronyms, and titles can significantly depend on capitalisation. In this paper, the DistilBERT embeddings were generated at the token level similar to Linq-Embed-Mistral or BPE, allowing us to contrast directly the impact of model architecture size as well as representational quality on the reference parsing task.

3.2.3 Feature Integration

A key challenge in this work is integrating information from different levels of granularity: **subword-based embeddings** (e.g., BERT, BPEmb) and **word-level handcrafted features** (e.g., affix, capitalization, punctuation). This integration is handled differently depending on the model architecture.

Neural Models (e.g., BiLSTM + CRF)

In models that consume dense vector representations, each input token is represented by combining learned contextual embeddings with structured handcrafted features:

- **Subword Embeddings:** Tokens are passed through pretrained models such as BERT or BPEmb. These models tokenize words into subword units internally (e.g., "information" may be split into "in", "##formation"). Since handcrafted features operate at the word level, we align the two by averaging the embeddings of subwords corresponding to each original word token. This produces a single subword-aware vector per token.
- **Handcrafted Features:** Each token also has categorical handcrafted features (e.g., prefix, caps type). Each of these is treated as a separate vocabulary and embedded using `nn.Embedding` layers in PyTorch. These embeddings are trained alongside the model.

The two components are **aligned** in dimensionality: the subword embedding is passed through a BiLSTM layer, and the handcrafted feature embedding is projected via a linear layer to match the LSTM output's dimensions. The two vectors are then added element-wise and passed to a CRF layer for structured prediction.

Sparse CRF Models (e.g., python-crfsuite)

In models like CRFsuite that do not support dense vector inputs, subword embeddings are converted to feature dictionaries. Each token's embedding is flattened and assigned feature names with numeric indices (e.g., `bert_00̄.123`). If two embedding sources are used (e.g., BERT and hand features), their features are merged using distinct prefixes. Handcrafted features, in this case, are also included as categorical string features. Since CRFsuite works with string-based sparse features, there is no need for alignment between the two embedding sources—each word is treated as a single token, and the embedding representation of a word that contains multiple subwords is averaged beforehand during preprocessing.

This setup ensures that both contextual richness from deep embeddings and structural insight from handcrafted features are available to the model. Dense architectures learn the interaction between the two during training, while sparse CRFs receive both as complementary symbolic inputs.

3.3 Sequence Labeling Models

In this section, we outline the sequence labeling models employed to parse strings of reference into structured fields. The task is posed as a token-level classification problem, with each token or word in a citation given a label like `B-Author`, `I-Title`, `I-Year`, etc. To address this, we look at a series of models, beginning with simple Conditional Random Fields (CRFs) and moving towards more expressive neural models like BiLSTM with CRF. Each model processes features in a different manner, and the design choices they make are in an effort to balance between accuracy, interpretability, and computational cost.

3.3.1 Conditional Random Fields (CRF)

Conditional Random Fields (CRFs) are probabilistic graphical models that are highly used in sequence labeling tasks in natural language processing (NLP). Lafferty, McCallum, and Pereira [4] first introduced CRFs as modeling the conditional probability of a sequence of labels given a sequence of input observations. Unlike generative models such as Hidden Markov Models (HMMs), CRFs do not make strong independence assumptions on the input and instead optimize the conditional likelihood of the output sequence directly.

In CRFs, the label of each sequence is not only a function of the input features of the respective token but also of the labels of neighboring tokens in both directions. This makes CRFs particularly well-suited to applications where output structure is important, i.e., part-of-speech tagging, named entity recognition, and, in this work, bibliographic reference parsing. In this case, tokens in a citation string are labeled with field types (e.g., Author, Title, Journal), and capturing dependencies between consecutive labels greatly enhances prediction.

There are two implementations of CRFs discussed in this thesis:

- A traditional CRF using the CRFsuite library, relying on hand-designed and subword features to facilitate field prediction with no contextual modeling.
- A neural CRF implemented in PyTorch, acting as an output layer in a deep architecture (e.g., BiLSTM + CRF). In this implementation, the CRF is fed dense, contextual embeddings from an earlier neural encoder and learns to capture the transitions of the labels alongside the acquired representations.

CRFs provide a principled and explainable way of solving structured prediction problems. In both versions, the CRF layer guarantees that the output labels have a valid sequence through learning of transition dependencies between output labels.

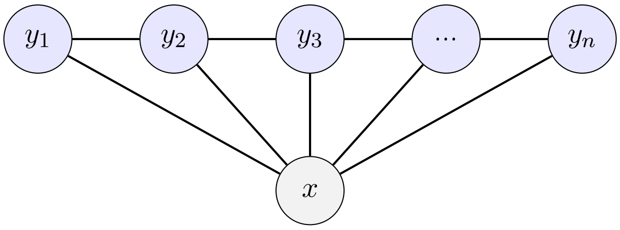


Figure 3.3: Graphical representation of a linear-chain CRF used for sequence labeling.

CRFsuite

At first deployment, Conditional Random Fields had been used using the CRFsuite library, a lightweight yet effective structured prediction task framework for sequence labeling and other structured tasks [31]. This was a non-contextual baseline model, and it used only rich, handcrafted features and subword-level embeddings rather than deep contextual encoders.

The model input was a mixture of

- **Handcrafted features**, borrowed from the AnyStyle parser, that mimic token-level features such as affixes, capitalization, punctuation, semantic category, and position.
- **Byte-Pair Embeddings (BPEmb)**, providing subword-level semantic information for each token for most languages.

These features were combined into a sparse, flat feature vector for each token in a target reference string. CRFsuite does not require vector embeddings as dense tensors such as neural models, but it accepts features in the form of string-labeled attributes, and therefore it is a viable choice for traditional sequence tagging with less computational need. CRFsuite learns label dependencies through the sequence as well as inter-field transitions like `Author`, `Title`, and `Year`. However, it does not learn any context semantics through the sequence except label dependencies; no internal representation exists of the sequence content except in local features. Despite this, the combination of handcrafted linguistic features with BPEmb provided a respectable baseline in terms of accuracy as well as generalization. The deployment was achieved with the assistance of the Python-CRFsuite library, which is a Python binding to CRFsuite offering a clean and expressive API for describing features and training CRF models.

Neural CRF in PyTorch

The second CRF implementation, i.e., Conditional Random Fields (CRF), used in this case is a neural variant integrated in a deep learning pipeline with PyTorch [32]. Unlike the CRFsuite-based model, which is specified over sparse feature vectors only, this implementation uses CRF as the output layer of a neural network. This allows the model to learn and use dense contextual embeddings while retaining the structured output behavior of traditional CRFs. In this setup, each token in the input sequence is represented by a concatenated embedding vector comprising:

- **Pretrained subword embeddings** (e.g., BERT or BPEmb),
- **Trainable embeddings** from handcrafted categorical features.

The pretrained embeddings are input to a Bidirectional LSTM (BiLSTM) encoder, which captures context across the entire sequence in both the forward and backward directions. The output of the BiLSTM — a contextualized representation for each token — is concatenated with the hand features, and then processed by a linear layer that produces emission scores for each label. On top of the final layer, there is a CRF module that decodes the most likely sequence of labels by modeling the compatibility between tags. The module is trained jointly with the rest of the network under a negative log-likelihood loss, allowing the model to simultaneously learn emission as well as transition parameters. This design benefits from both learned representations and structured sequence modeling. While the BiLSTM produces contextualized input embeddings, the CRF layer ensures that label predictions are in line with common citation patterns — e.g., ensuring that a B-Author label is not directly followed by a B-Year without an intervening I-Author.

The neural CRF was implemented using the torchcrf package, a widely used CRF layer for PyTorch that offers both training and decoding using efficient forward–backward algorithms.

3.3.2 BiLSTM + CRF

The BiLSTM + CRF architecture is among the most widespread and effective models for sequence labeling tasks, particularly if both label dependencies and contextual information are relevant. In this work, the BiLSTM is used to generate contextual embeddings of each token from subword representations, and the CRF layer embeds the dependencies between the output labels so that coherent predictions can be generated.

BiLSTM Encoding

LSTM networks are recurrent neural networks (RNNs) used to process long-range dependencies using gating mechanisms regulating information flowing through time [33]. Bidirectional LSTM reads the sequence in both directions, accessing context from future and previous tokens for each word [34]. It is particularly useful in parsing the reference string, where the label of a token can depend on both the context before and after it. In this design, the input to the BiLSTM is subword embeddings obtained from Byte-Pair Encoding (BPEmb). They offer semantics at the subword level and generalize for citation styles and languages. The BiLSTM outputs a sequence of hidden states for every token, representing its context-aware embedding.

Integration of Handcrafted Features

In addition to the subword embeddings, manually crafted features inspired by AnyStyle [10] are used. Such categorical features are processed into a trainable embedding layer and projected to the same space as the BiLSTM representations. The two representations are combined using element-wise addition to form a fused vector that includes both learned and manually crafted knowledge.

CRF Output Layer

The combined representations are fed into a linear classifier to predicts scores for each label. These are fed as input to a CRF layer that models label transitions and produces the most probable sequence using the Viterbi algorithm. The CRF is trained on a negative log-likelihood loss, which encourages it to produce high scores for valid and correct label sequences.

This model allows the system to take advantage of both deep contextualized representations and sequence-level coherence, and it is best suited for annotating structured references with highly diverse formatting styles.

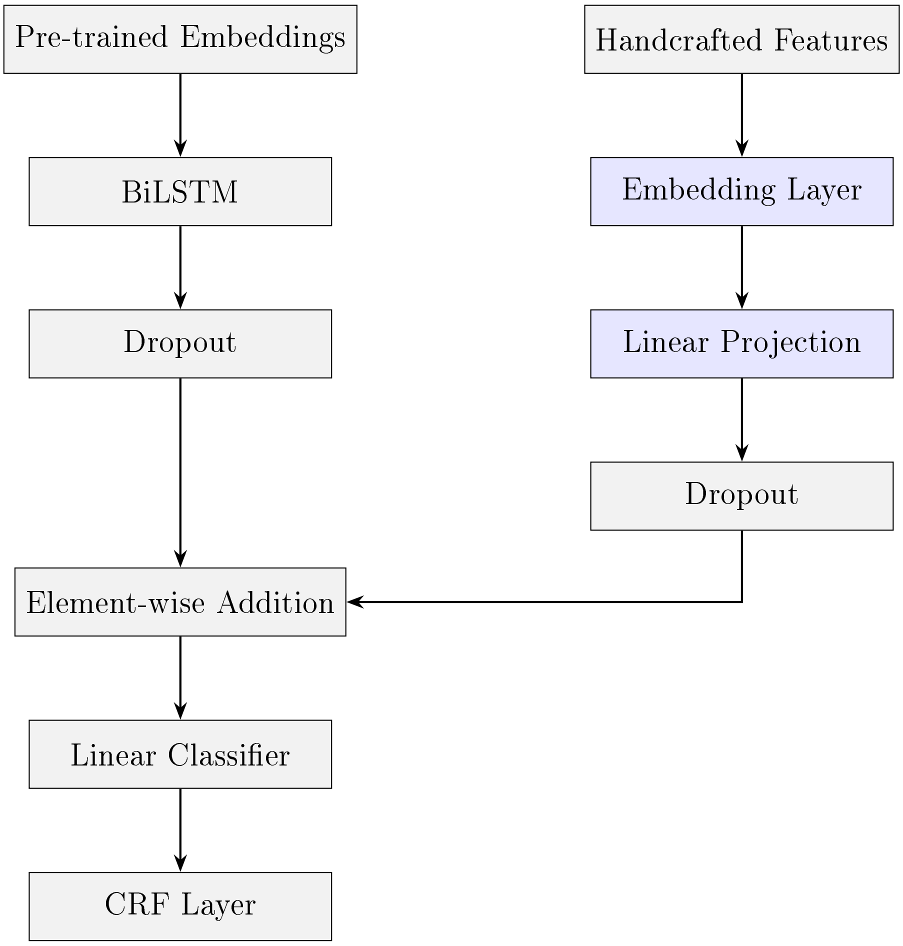


Figure 3.4: Architecture of the BiLSTM + CRF model combining pre-trained embeddings and handcrafted features.

3.4 Dataset and Preprocessing

The accuracy of any machine learning algorithm, especially in sequence labeling problems, heavily relies on the quality and consistency of the input data. In this work, we use a vast corpus of citations to train and evaluate models to split and label reference strings into structured fields. This section describes the dataset used, the annotation and label formatting process, and the preprocessing pipeline that was developed to convert raw reference data into model inputs. Special attention is given to converting XML annotations into a standardized BIO tagging scheme and integrating both handcrafted and learned features.

3.4.1 Dataset Description

We employed the GIANT dataset [11] to train as well as evaluate models for citation parsing in this research because it represents the largest public dataset of reference strings with annotations. The GIANT dataset came into existence to solve the problems of existing datasets due to their restricted nature of small sample size and limited domain applicability, while demonstrating minimal citation style variety. GIANT contains more than 991 million XML-labeled citation strings, allowing it to support modern deep learning approaches for reference string segmentation.

The research team developed GIANT through the amalgamation of three components:

- 677,000 bibliographic records from Crossref,
- The CSL repository contains 1,564 distinct citation styles, which form the basis of GIANT's data generation.
- A widespread citation processor named citeproc-js served as the tool for formatting citations.

Through citeproc-js, the entire Crossref database received transformation into every supported citation style. The produced citation strings underwent XML-style labeling that marked fields including `<author>`, `<title>`, `<year>`, and `<journal>` together with other tags. The team changed citation styles by hand to add the necessary tags that surrounded important bibliographic content for sequence labeling model use. By converting the documents into machine-readable format, the dataset contains a versatile collection of citation styles that range from journal articles to books and chapters, in addition to conference papers. In total, the dataset includes:

- **991,411,100 labeled citation strings**
- Derived from **633,895 unique citations**
- Spanning citation types such as:
  - Journal articles (75.9%)
  - Book chapters (12.4%)
  - Conference papers (5.6%)
  - Others (e.g., datasets, reference entries)

All citations are in English, in the U.S. locale, and compressed for efficient retrieval. Every record also includes metadata such as citation type, citation style, and DOI. The metadata facilitates indexing and filtering, and researchers can focus on particular subdomains or types of citations when experimenting.

The scale and diversity of GIANT make it especially suited for training sequence labeling models such as CRFs and neural networks, which benefit from having both large numbers and varied examples. In this work, we used a 5 million sample subsample of GIANT due to computational constraints, selecting records uniformly across citation styles and types in order to preserve diversity.

3.4.2 Label Set and Annotation Scheme

The process of training citation parsing models needs each reference string to receive bibliographic field-specific labels. For our work, we have selected the BIO tagging format because it functions as a standard annotation method in sequence labeling tasks. The BIO scheme helps identify token positions within fields by using three labels, which represent **Begin**, **Inside**, and **Outside**. The starting words of `'Title'` receive label `B-Title`, yet succeeding words within the title use the label `I-Title`. Tokens without specification fall under the category `O`. This data format proves being helpful in such applications because:

- By using BIO it becomes simple to distinguish fields that extend across multiple consecutive tokens in cases such as lengthy author names and journal titles.
- By using this format, models automatically acquire better capabilities for moving between different field categories.
- CRF-based and neural sequence models with BIO-style labeling help the model to find complete compatibility when using this format.

References in the original GIANT dataset receive full XML annotation for every listed field. A sample snippet includes different elements like `<author>`, `<given>`, `<family>`, `<title>`, `<issued>`, `<volume>`, `<container-title>`, `<page>` and `<URL>`, for example:

```
<author>
  <family>Ritchie</family>, <given>E.</given> and
  <family>Powell</family>, <given>Elmer Ellsworth</given>
</author>
(<issued><year>1907</year></issued>)
<title>Spinoza and Religion.</title>
<container-title>The Philosophical Review</container-title>,
<volume>16</volume>(<issue>3</issue>), p.
<page>339</page>. [online] Available from:
<URL>http://dx.doi.org/10.2307/2177340</URL>
```

The detailed annotation scheme becomes complex for training models since multiple fields (such as `<given>` and `<family>`) need to merge into the `Author` category. For this work, the annotated reference was simplified in an automated manner to retain only the most essential labels. These simplified labels were chosen to reflect the minimum necessary structure required to uniquely identify a cited work, while also keeping the annotation task manageable for a learning algorithm.

Label	**Description**
Author	Names of authors, editors, or contributors
Title	Title of the work (book, article, etc.)
Container-Title	Journal, conference, or book series
Year	Year of publication
Volume	Volume number
Issue	Issue number (if available)
Page	Page number or page range
Publisher	Publishing organization or institution
DOI	Digital Object Identifier
ISSN	International Standard Serial Number (if available)
ISBN	International Standard Book Number
URL	Web link to the cited item

Table 3.2: Simplified label set used for BIO tagging.

3.4.3 Preprocessing Steps

Before training, a chain of preprocessing operations had been done on the raw GIANT dataset in order to prepare reference strings for sequence labeling. The data originally came in the form of XML-annotated data with mixed nested tags and unnecessary metadata. The pipeline below was applied in order to preprocess the training data into something clean and homogeneous:

1. Tag Filtering and Simplification

The original XML annotations contained a wide range of nested tags like `<given>`, `<family>`, and other irrelevant elements. A regular expression pattern was applied to extract all XML tags along with their values. A filtering step retained only a selected subset of allowed labels (e.g., `Author`, `Title`, `DOI`), and all the other tags were automatically removed. This step of reducing tags made sure that only meaningful fields remained, but the job of annotation was still feasible for models. To remove nested tags, a recursive function was called, which left the valuable content and removed everything else.

2. Cleaning Text

Following tag filtering, the content left was flattened to plain text with labels implicit through XML tags for the permitted classes. Tokens were then processed for BIO annotation by regular expressions that:

- Detect punctuation
- Strip whitespace
- Collapse nested or badly formed tags

Each cleaned string was stored as an additional column alongside the original one, so that raw and processed inputs could be compared.

3. BIO Tagging

Although not illustrated here in this phase, the reference strings that were cleaned were then tokenized and tagged in BIO format. The tokens were matched against the collapsed XML tags and tagged as `B-<TAG>` or `I-<TAG>`, based on their position within the field. Tokens that were not labeled were tagged with `O`.

4. Splitting the Dataset

Initial data files were read in from a subset of the GIANT dataset, shuffled using a set seed for reproducibility. Data was divided into:

- 5 million training
- 200k validation
- 200k test

This splitting was done similarly for all the samples to preserve type and style diversity.

3.5 Training and Evaluation Setup

Two different training pipelines were used for the CRFsuite-based model and the BiLSTM+CRF model implemented in PyTorch. Both models were trained on processed data derived from the GIANT dataset, using subsets of varying size based on model complexity and resource constraints.

3.5.1 CRFsuite Model

The CRFsuite model was implemented using the sklearn-crfsuite Python library. Feature vectors for each token were constructed from a combination of handcrafted features and Byte-Pair Embeddings (BPEmb). These features were extracted and formatted into the string-keyed dictionary structure expected by CRFsuite.

- **Training size:** Training was done using both 1 million and 5 million annotated reference strings
- **Test size:** Testing used both 100,000 and 200,000 references according to the training size
- **Optimizer:** Limited-memory BFGS (lbfgs)
- **Regularization:** c1 = 0.1, c2 = 0.1
- **Max iterations:** 100
- **Transitions:** Enabled `all_possible_transitions` for richer label modeling
- **Number of features:** 600 when using BPEmb, and 619 using BPEmb and Hand features

The model was trained using default BIO-annotated labels. Model performance was evaluated using token-level metrics (precision, recall, F1-score) computed by flattening the sequences. The trained CRF model was serialized and stored using pickle for reuse and reproducibility.

3.5.2 BiLSTM + CRF Model (Neural CRF)

The BiLSTM + CRF model was implemented in PyTorch. It uses dense embeddings and contextual modeling, and was trained on a larger subset of the GIANT dataset due to its capacity to model long-range dependencies.

- **Training size:** 5 million annotated reference strings
- **Validation size:** 200,000 references

- **Test size:** 200,000 references
- **Batch size:** 32
- **Dropout:** 0.5
- **Optimizer:** Adam (torch.optim.Adam)
- **Loss function:** Negative Log-Likelihood (from CRF layer)
- **Embedding:**
  - Subword embeddings from BPEmb
  - Trainable embeddings from handcrafted feature classes
- **Architecture:**
  - Bidirectional LSTM followed by dropout
  - Projected handcrafted features added to BiLSTM output
  - Linear layer to generate emission scores
  - CRF layer for structured decoding
  - **Model Specification:**
    - LSTM hidden dim: 128
    - Hand Features embedding size: 50
    - BPEmb featurs: 600
    - Dropout rate: 0.5
    - Number of tags (labels): 26

Datasets were loaded and batched using PyTorch's `DataLoader`, with a custom collate function that aligns token lengths and encodes categorical feature indices. Label-to-index mappings were handled dynamically based on the full BIO label set used in preprocessing. Evaluation was conducted on the test set using the token-level F1-score and classification reports generated by `sklearnṁetricsċlassification_report`.

3.5.3 BERT-based CRF Model

The BERT-based CRF models were implemented in sklearn-crfsuite Python library. Instead of training a contextual encoder from scratch, the system used pretrained BERT-like models to extract high-quality embeddings, which were then directly fed into a CRF layer for structured prediction over tokens.

Two types of encoders were experimented with:

- **Linq-Embed-Mistral:** A modern, lightweight, high-performance multilingual encoder.
- **DistilBERT Multilingual Cased:** A distilled, faster version of multilingual BERT, providing strong token representations with reduced computational cost.

The training setup was as follows:

- **Training size:** 5 million annotated reference strings
- **Test size:** 200,000 references
- **Optimizer:** Limited-memory BFGS (lbfgs)
- **Regularization:** c1 = 0.1, c2 = 0.1
- **Max iterations:** 100
- **Transitions:** Enabled `all_possible_transitions` for richer label modeling
- **Embedding:**
  - Contextual embeddings from the frozen BERT encoder
  - Number of features: 4096 when using `Linq-Embed-Mistral`, and 768 when using `DistilBERT`

Datasets were processed similarly to other models. Evaluation was performed on the flattened token sequences using macro-averaged and weighted F1-scores, as reported by `sklearn.metrics.classification_report`.

Chapter 4

Results

4.1 Evaluation Metrics

For model performance evaluation, we employ the standard measures of **precision**, **recall**, and **F1-score** from sequence labeling tasks. These are calculated at the token level such that each token prediction is compared against its respective ground truth label. A prediction is considered correct only if both the BIO tag and the corresponding field label (e.g., `B-TITLE`, `I-YEAR`) match the reference string annotated.

The metrics are computed with the `classification_report` function of the `sklearn.metrics` module, which provides a per-class breakdown, together with macro- and weighted averages across all labels:

- **Precision** is the proportion of predicted tokens for a label that were accurate.
- **Recall** is the proportion of actual tokens correctly predicted.
- **F1-score** is the mean of precision and recall, offering a balance of performance.

Because of class frequency imbalance (for example, `O` tokens and `I-TITLE` are far more common than `B-ISSN` or `I-ISSUE`), we present **macro averages** (which are treating all labels equally) and **weighted averages** (which treats class frequency), because due to given class frequency imbalance we want to emphasize both overall model robustness and infrequent field type performance. For example, in the CRF model, token-wise weighted F1-score was 0.91 with particularly high performance for often occurring fields `B-AUTHOR`, `I-TITLE`, and `B-PAGE`. More underrepresented labels `I-ISSUE` and `I-ISBN` achieved modestly lower recall as might be expected relative to their frequency within the corpus.

4.2 Model Comparison

4.2.1 CRF Configurations

During the first phase of experimentation, we experimented with a baseline CRF model with input features having solely **Byte-Pair Embeddings (BPEmb)**. It was trained on a dataset of 1 million labeled reference strings and served as the baseline to study the performance of semantic subword embeddings in isolation for citation parsing.

To examine if the mere encoded token-level cues would improve performance, we enriched the feature set by adding **hand-engineered features** derived from the AnyStyle parser [10]. These included affixes, punctuation type, character case, semantic class, and position. The new model, again trained on 1 million samples, utilized a concatenation of BPEmb and these additional features. The result was substantial improvement across all field tags with the exception of the most common ones. Precisely, F1-scores on structurally ambiguous or poorly documented fields (e.g., `B-ISSUE`, `B-DOI`, `I-CONTAINER-TITLE`) increased, which suggests that hand-engineered features helped the model detect syntactic boundaries and semantic roles harder to acquire from embeddings.

Encouraged by this progress, we ramped up CRF training to leverage **5 million** annotated examples and preserve both handcrafted features and BPEmb. Increased training size brought additional improvements, particularly in label consistency and low-frequency tag recall. This line of models — from embeddings-only, to hybrid features, to large-scale training — illustrates how both feature sparsity and training size lead to better structured prediction performance on citation parsing tasks.

The following table showes the results of each of the models and its performance on the test set:

Model	**Training Size**	**Features Used**	**F1-score (Weighted Avg)**
CRF	1 million	BPEmb only	0.87
CRF	1 million	BPEmb + Handcrafted	0.90
CRF	5 million	BPEmb + Handcrafted	0.91

Table 4.1: Comparison of CRF models with different features and dataset sizes using weighted F1-score.

As shown in Table 4.1, we observe a consistent improvement in F1-score as we increase the feature richness and training data size. Adding handcrafted features to the BPEmb baseline improves the model's ability to capture structural cues in citation strings. Further increasing the training data to 5 million references yields additional gains, highlighting the importance of both quality and quantity in feature-based CRF models.

4.2.2 BiLSTM + CRF

Two variants of the model were trained using **5 million reference strings**: one using only BPEmb, and the other using BPEmb with additional handcrafted features. Both models were evaluated on the same test set using token-level precision, recall, and F1-score. The results are summarized in Table 4.2.

Metric	**BPEmb only**	**BPEmb + Hand**
Precision (Weighted Avg)	0.80	0.83
Recall (Weighted Avg)	0.93	0.93
F1-score (Weighted Avg)	0.86	0.87

Table 4.2: Comparison of BiLSTM + CRF models trained on 5 million samples using different feature sets, the results are measured on the test set.

As shown in Table 4.2, incorporating handcrafted features alongside BPEmb yielded a modest but measurable improvement in overall token-level F1-score. While both models benefited from the expressive power of bidirectional LSTMs, the inclusion of explicit structural cues contributed to more accurate field segmentation.

4.2.3 BERT-based CRF Model

To explore the effect of more advanced embeddings, we trained a CRF model using token-level embeddings generated by a modern `BERT-style encoder decoder`. Specifically, two different transformer-based architectures were evaluated:

- **Linq-Embed-Mistral:** A recent multilingual model ranked highly on the Hugging Face embedding leaderboard for retrieval and semantic similarity tasks.
- **DistilBERT Multilingual Cased:** A distilled and efficient version of BERT, offering faster inference with a modest trade-off in performance.

Both models were used without additional handcrafted features or LSTM layers. The goal of these experiments was to determine whether high-quality contextual embeddings alone could match or surpass hybrid models that explicitly combine contextual and structural information.

The CRF models were trained on 5 million annotated reference strings using the BIO tagging scheme and evaluated using token-level precision, recall, and F1-score. The results, summarized below, show that both BERT-based CRF models achieve competitive performance, with Linq-Embed-Mistral slightly outperforming DistilBERT in weighted F1-score. In particular, the Linq-Embed-Mistral model demonstrated superior performance on complex multi-token fields such as I-TITLE and I-AUTHOR, while DistilBERT achieved robust overall generalization with lower computational overhead. These findings suggest that leveraging strong pre-trained multilingual BERT based embeddings can effectively substitute for manual feature engineering when trained at scale.

Metric	**Linq-Embed-Mistral**	**DistilBERT**
Precision (Weighted Avg)	0.92	0.91
Recall (Weighted Avg)	0.94	0.93
F1-score (Weighted Avg)	0.93	0.92

Table 4.3: Token-level weighted F1-score performance of the CRF models using different BERT-style embeddings, the results are measured on the test set.

4.2.4 Final Comparison of Best Model Variants

After analyzing each architecture independently, we summarize the performance of the best-performing configuration from each category in Table 4.4. The results clearly demonstrate the benefit of rich contextual embeddings provided by transformer-based models. The BERT + CRF configuration achieved the highest overall F1-score, followed by the traditional CRF using both BPEmb and handcrafted features. Interestingly, the BiLSTM + CRF model underperformed despite its ability to model sequential context, likely due to the difficulty of learning both token context and structure from scratch. This comparison highlights that pre-trained embeddings can substantially reduce the reliance on handcrafted features and outperform more complex neural architectures when applied effectively.

	CRF	BiLSTM + CRF	BERT + CRF
Metric	**BPEmb + Handcrafted**	**BPEmb + Handcrafted**	**Linq-Embed-Mistral only**
Precision (Weighted Avg)	0.91	0.83	0.92
Recall (Weighted Avg)	0.91	0.93	0.94
F1-score (Weighted Avg)	0.91	0.87	0.93

Table 4.4: Final comparison between the best-performing models from each architecture family.

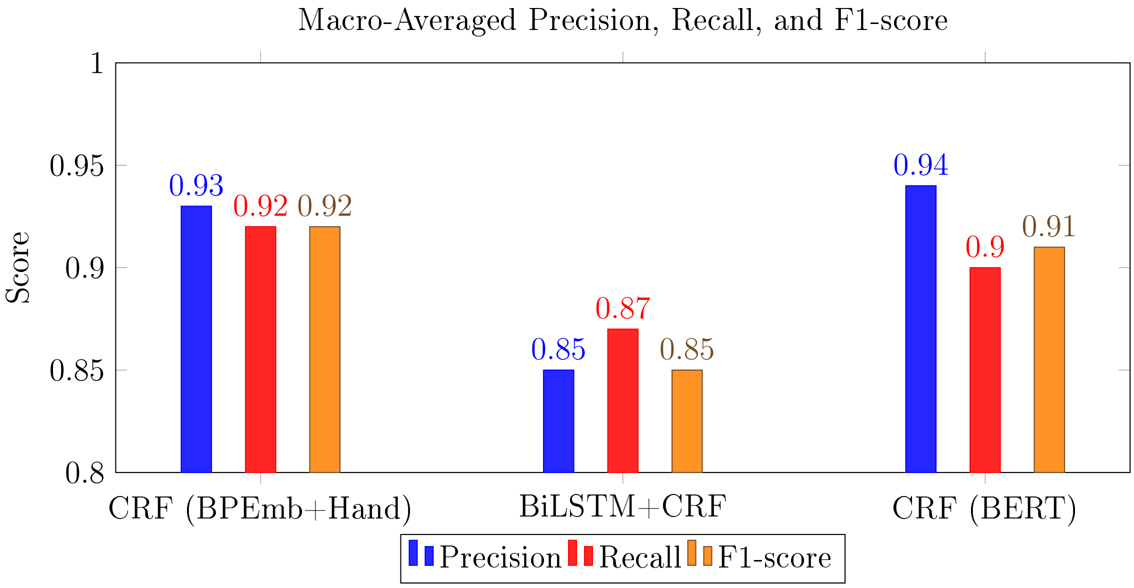


Figure 4.1: Macro-level comparison of Precision, Recall, and F1-score across best-performing models.

4.2.5 Analysis of Results

Based on the evaluation of the best-performing configurations across the CRF, BiLSTM + CRF, and BERT + CRF models, several insights can be drawn at both the macro and field-specific levels:

- **Model-Level Observations:**
  - **BERT + CRF**: achieved the highest overall weighted F1-score (0.93), along with the best precision (0.92) and recall (0.94). This confirms the effectiveness of transformer-based contextual embeddings, especially when used without additional handcrafted features.
  - **CRF (BPEmb + Handcrafted)**: performed surprisingly well with a weighted F1-score of 0.91, showcasing the strength of explicit token-level features when deep contextual embeddings are not available. Its performance was competitive with neural models across many fields.

  - **BiLSTM + CRF**, while offering contextual modeling through LSTM, slightly underperformed (F1-score: 0.87). This may be attributed to challenges in optimizing both the sequential context and structural features in a unified neural pipeline.

- **Field-Specific Insights:**
  - **Author Fields** (`B-AUTHOR`, `I-AUTHOR`): All models performed well on author detection, especially the BERT + CRF model, which reached F1-scores of 0.88 and 0.93 respectively. BiLSTM + CRF showed slightly lower precision on `B-AUTHOR`, indicating some sensitivity to name boundaries.
  - **Title Fields** (`B-TITLE`, `I-TITLE`): These fields consistently benefited from contextual embeddings. BERT + CRF achieved the best performance (`I-TITLE`: 0.94 F1), capturing long spans more effectively. The BiLSTM model also did well on `I-TITLE` due to its sequential memory, but struggled slightly with B-TITLE.
  - **URL Fields** (`B-URL`, `I-URL`): All models performed exceptionally on URLs, with F1-scores above 0.95. These fields are often easier to capture due to distinctive formatting (e.g., `http`, `www`, slashes), making them reliably detectable even by simpler models.
  - **DOI** and **PAGE**: Both BERT and CRF-based models achieved high performance on `B-DOI`, `B-PAGE`, and `I-PAGE`. These tokens also follow consistent patterns (numbers, slashes, etc.), allowing even non-contextual CRFs to generalize well.
  - **Container Title** (`I-CONTAINER-TITLE`): BiLSTM + CRF struggled here, achieving only 0.58 F1, while **CRF** and **BERT** models performed significantly better (0.87 and 0.85 respectively). This indicates that handcrafted features or pre-trained embeddings provide better signals for longer semantic chunks like journal names.
  - **Publisher** (`B-PUBLISHER`, `I-PUBLISHER`): The `I-PUBLISHER` tag was consistently well-handled across models (F1 near 0.90), but B-PUBLISHER showed more variability, reflecting challenges in distinguishing the beginning of organization names, especially when punctuation or unusual casing is involved.

4.2.6 Comparison with External Systems

To place the performance of the new models in context, we compare their performance with previous work on bibliographic reference parsing. Despite potentially different conditions for datasets as well as evaluation policies, comparison provides an intuition of the relative position of the described methods.

Evaluation Against a Large Language Model (GPT-4o)

Recent advances in large language models (LLMs) have enabled strong performance on many natural language understanding tasks, even without explicit task-specific fine-tuning. To explore how a general-purpose LLM would perform on bibliographic reference parsing, we used GPT-4o to label a sample of 100 reference strings. The comparison offers insights into the strengths and limitations of applying a state-of-the-art LLM to a specialized sequence labeling task, without model retraining or task-specific adaptation.

Despite not having any task-specific tuning, GPT-4o did well on zero-shot general fields like `TITLE`, `PUBLISHER`, and `URL` with F1-scores greater than 0.90. Its performance was highly variable, however, on structured fields like `VOLUME`, `ISSUE`, `DOI`, and `ISBN`, where critical formatting details must be captured precisely. Our specialized models, however, like CRF with BPE and Handcrafted Features, BiLSTM+CRF, and CRF with BERT embeddings consistently outperformed GPT-4o across structured metadata fields. Overall, GPT-4o achieved a weighted F1-score of 0.89, while our best-performing models surpassed 0.91-0.93, confirming the value of task-specialized models for highly structured sequence labeling tasks like reference parsing.

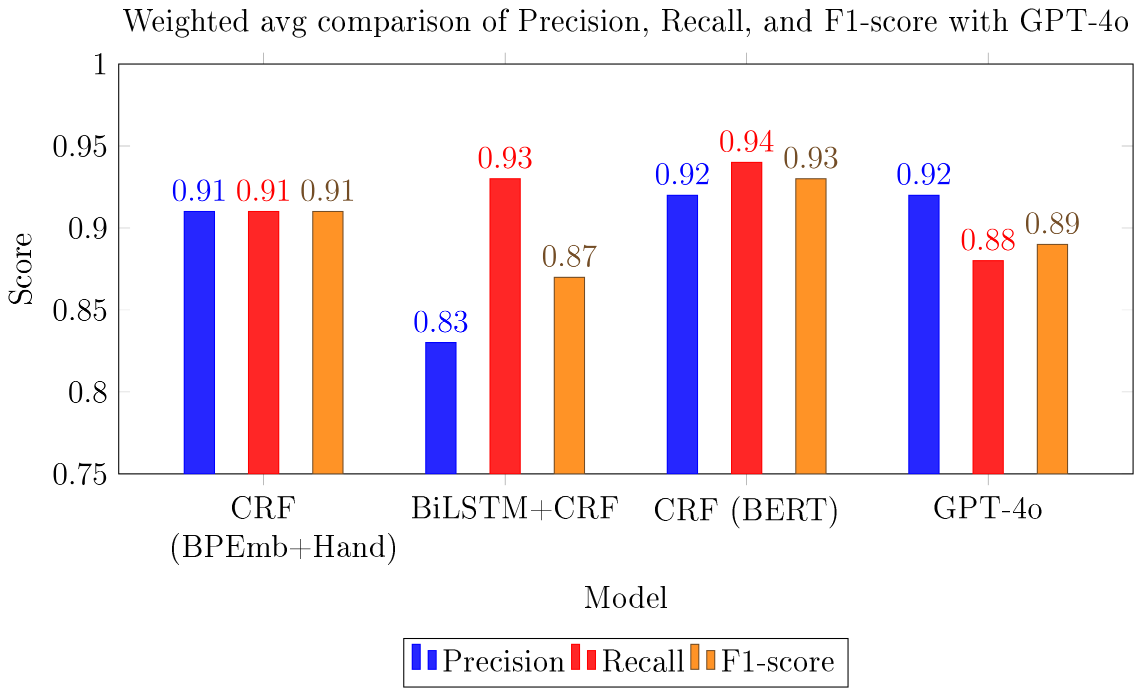


Figure 4.2: Weighted-averaged Precision, Recall, and F1-score for the best performing models and GPT-4o.

Comparison with External Published Work

To put our models' performance into better perspective, we compare them to numbers published in independent studies. In particular, we compare our models' performance to the study by Cuéllar Hidalgo et al. [17], who have conducted an extensive evaluation of bibliographic reference parsing models. While their work shares some methodological similarity, the following key differences should be noted: their models were trained on much larger corpus (the full GIANT corpus), used no handcrafted features, and were evaluated on the CORA corpus — an independent test set from that employed in our experiments. Despite these differences, the comparison is worthwhile in so far as it gives insight into the relative design and performance of both approaches.

In their work, the BiLSTM+CRF model had a reported weighted F1-score of 0.96 while tested on the CORA dataset [17] compared to both CRF-only and Transformer+CRF models. On our experiments, our top-performing BiLSTM+CRF model, which was trained using BPEmb embeddings and handcrafted features, achieved a weighted F1-score of 0.87 on our in-house large-scale test set. While our score is actually lower, note that our training set was much smaller (5 million references versus almost 200 million), and that hand-designed features were employed to augment performance instead of using learned embeddings exclusively. Besides, while their evaluation used a standard dataset (CORA), our dataset more accurately sampled diverse and realistic styles of citations, with noisy and partial samples. This means our models retained good generalization behavior under more challenging situations.

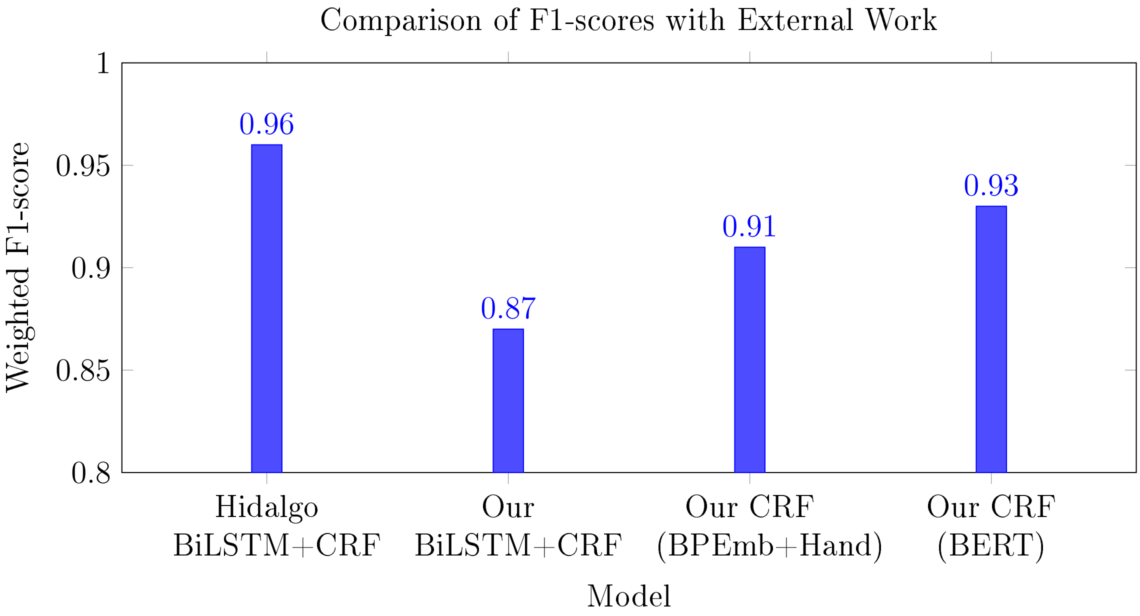


Figure 4.3: Comparison of weighted F1-scores between our models and the BiLSTM+CRF model from[17].

Chapter 5

Conclusion

In this thesis, the issue of reference parsing to support token-level field labeling of reference strings has been explored. By integrating manually crafted feature engineering and the newest subword embedding technology, we aimed to provide robust sequence labeling models to tackle the high variability and noise levels in citation format. Properly parsing references is a highly pertinent issue for citation network construction using automated methods, academic metadata retrieval, and scientific information retrieval.

The proposed system combined traditional sequence labeling models like Conditional Random Fields (CRFs) with neural models like Bidirectional LSTM (BiLSTM) networks and more recent Transformer-based encoders. One of the major contributions of this research was the creation and integration of a rich set of hand-crafted features, based on AnyStyle parsing techniques, that captured fine-grained linguistic, structural, and positional patterns in reference strings. Apart from these features, the use of Byte-Pair Embeddings (BPEmb) and BERT-inspired embeddings added contextualized subword-level representations, significantly enhancing model understanding of citation elements.

Heavy experimentation revealed that classical CRF models, when combined with both BPEmb embeddings and hand-crafted features, were as good as deeper neural architectures. The BiLSTM + CRF model also improved generalization to unseen data by capturing sequential dependencies at the expense of extra computational resources and training time. Finally, the pairing of a modern BERT-like encoder, i.e., Linq-Embed-Mistral, with CRF decoding achieved the highest overall weighted F1-score, demonstrating the ability of high-capacity language models to model citation structure without explicit feature engineering.

Comparison against external baselines, including a ChatGPT-4o-powered extraction system and prior work on bibliographic parsing, found the suggested models achieved high performance considering the relatively small synthetic training dataset and lack of domain adaptation. On more difficult multi-token fields like `I-TITLE`, `I-AUTHOR`, and `I-URL`, the models maintained high recall and precision, with both the structural features and deep contextual embeddings found to add complementary value.

Despite these promising outcomes, some limitations remain. The system was trained exclusively on synthetic data (the GIANT dataset) and has yet to be validated on truly noisy, real-world references extracted from diverse document types. Furthermore, the models were evaluated using token-level metrics rather than full-reference reconstruction accuracy, which could be a more practical downstream measure in citation graph building tasks. Resource limitations also restricted the use of larger datasets and broader hyperparameter tuning, particularly for transformer-based models.

Future work can expand the system in several directions. First, applying domain adaptation techniques to transfer models on real-world reference datasets may enhance robustness. Second, using recent advances in retrieval-augmented language models or semi-supervised learning can enhance performance without intensive manual annotation. Finally, extrapolating the models to collectively parse references and deduce citation context (in-text citation extraction) could provide an end-to-end solution to automatic bibliography graph construction.

Overall, this thesis demonstrated that structured feature design in combination with powerful neural embeddings yields good performance on the bibliographic reference parsing task. The intersection of linguistic understanding, statistical learning, and modern deep language models presents a generalizable platform for handling one of the most fundamental bottlenecks to the automated study of scholarly discourse. As scientific ecosystems continue to expand, accurate and scalable reference parsing will remain a fundamental component in scientific discovery of knowledge.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, **Simonyi András**, for his continuous support and insightful guidance throughout the course of this research. His expertise was instrumental in shaping the project and bringing it to completion. I am also deeply thankful to the faculty and staff of the Department of Computer Science at Eötvös Loránd University, whose support and resources were essential to the successful execution of this work. Finally, I extend my appreciation to the open-source community and the developers whose tools and libraries were integral to the implementation of this project.

Bibliography

[1] Isaac Councill, C. Lee Giles, and Min-Yen Kan. "ParsCit: an Open-source CRF Reference String Parsing Package". In: *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. Ed. by Nicoletta Calzolari et al. Marrakech, Morocco: European Language Resources Association (ELRA), May 2008. URL: `https://aclanthology.org/L08-1291/`.

[2] Animesh Prasad, Manpreet Kaur, and Min-Yen Kan. "Neural ParsCit: A Deep Learning-Based Reference String Parser". In: *International Journal on Digital Libraries* 19.4 (2018), pp. 323–337. DOI: `10.1007/s00799-018-0242-1`. URL: `https://doi.org/10.1007/s00799-018-0242-1`.

[3] L. Rabiner and B. Juang. "An introduction to hidden Markov models". In: *IEEE ASSP Magazine* 3.1 (1986), pp. 4–16. DOI: `10.1109/MASSP.1986.1165342`.

[4] John D. Lafferty, Andrew McCallum, and Fernando 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 '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ISBN: 1558607781.

[5] CiteSeerX. *CiteSeerX: Scientific Literature Digital Library and Search Engine*. Accessed: 2025-04-17. 2024. URL: `https://citeseerx.ist.psu.edu`.

[6] Dominika Tkaczyk et al. "Machine Learning vs. Rules and Out-of-the-Box vs. Retrained: An Evaluation of Open-Source Bibliographic Reference and Citation Parsers". In: May 2018, pp. 99–108. DOI: `10.1145/3197026.3197048`.

[7] Mark Grennan and Joeran Beel. *Synthetic vs. Real Reference Strings for Citation Parsing, and the Importance of Re-training and Out-Of-Sample Data*

for Meaningful Evaluations: Experiments with GROBID, GIANT and Cora. Apr. 2020. DOI: 10.48550/arXiv.2004.10410.

[8] Vidhi Jain, Niyati Baliyan, and Shammy Kumar. "Machine Learning Approaches for Entity Extraction from Citation Strings". In: *Lecture Notes in Electrical Engineering: Decision Intelligence.* Accessed: 2025-04-18. Springer Nature Singapore, 2023, pp. 287–297. DOI: 10.1007/978-981-99-5997-6_25. URL: https://ouci.dntb.gov.ua/en/works/9Zwj6R37/.

[9] Wonjun Choi et al. "Building an annotated corpus for automatic metadata extraction from multilingual journal article references". In: *PLOS ONE* 18.1 (Jan. 2023), pp. 1–22. DOI: 10.1371/journal.pone.0280637. URL: https://doi.org/10.1371/journal.pone.0280637.

[10] Sylvester Keil. *AnyStyle: A Parser for Bibliographic References.* Accessed: 2025-04-18. 2025. URL: https://anystyle.io/.

[11] Mark Grennan et al. "GIANT: The 1-Billion Annotated Synthetic Bibliographic-Reference-String Dataset for Deep Citation Parsing." In: *AICS.* Ed. by Edward Curry et al. Vol. 2563. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 260–271. URL: http://dblp.uni-trier.de/db/conf/aics/aics2019.html#GrennanSCB19.

[12] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. *Digital libraries and autonomous citation indexing.* English (US). June 1999. DOI: 10.1109/2.769447.

[13] Dominika Tkaczyk et al. "CERMINE: automatic extraction of structured metadata from scientific literature". In: *International Journal on Document Analysis and Recognition (IJDAR)* 18 (July 2015). DOI: 10.1007/s10032-015-0249-8.

[14] Chien-Chih Chen et al. "BibPro: A Citation Parser Based on Sequence Alignment Techniques". In: Jan. 2008, pp. 1175–1180. DOI: 10.1109/WAINA.2008.125.

[15] Mohammad Shahrear Uddin. "TransParsCit: A Transformer-Based Citation Parser Trained on Large-Scale Synthesized Data". ODU Master's Theses, Paper 133. Master's Thesis. Norfolk, Virginia, USA: Old Dominion University,

2022. URL: https://digitalcommons.odu.edu/computerscience_etds/133/.

[16] Zhen Yin and Shenghua Wang. “Enhancing bibliographic reference parsing with contrastive learning and prompt learning”. In: *Engineering Applications of Artificial Intelligence* 133 (2024), p. 108548. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2024.108548. URL: https://www.sciencedirect.com/science/article/pii/S0952197624007061.

[17] Rodrigo Cuéllar Hidalgo et al. “Neural Architecture Comparison for Bibliographic Reference Segmentation: An Empirical Study”. In: *Data* 9.5 (2024). ISSN: 2306-5729. DOI: 10.3390/data9050071. URL: https://www.mdpi.com/2306-5729/9/5/71.

[18] Dominika Tkaczyk et al. “ParsRec: A Novel Meta-Learning Approach to Recommending Bibliographic Reference Parsers”. In: *26th Irish Conference on Artificial Intelligence and Cognitive Science (AICS)*. Vol. 5. 1. 2018, pp. 31–42.

[19] Patrice Lopez. “GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction for Scholarship Publications”. In: vol. 5714. Sept. 2009, pp. 473–474. ISBN: 978-3-642-04345-1. DOI: 10.1007/978-3-642-04346-8_62.

[20] Benjamin Heinzerling and Michael Strube. “BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Ed. by Nicoletta Calzolari et al. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://aclanthology.org/L18-1473/.

[21] Jacob Devlin et al. “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, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[22] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162/.

[23] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2017). Ed. by Lillian Lee, Mark Johnson, and Kristina Toutanova, pp. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://aclanthology.org/Q17-1010/.

[24] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: (2018).

[25] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202/.

[26] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 6000–6010. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[27] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv preprint arXiv:1609.08144* (2016). URL: https://arxiv.org/abs/1609.08144.

[28] Junseong Kim et al. *Linq-Embed-Mistral: Elevating Text Retrieval with Improved GPT Data Through Task-Specific Control and Quality Refinement.* https://huggingface.co/Linq-AI-Research/Linq-Embed-Mistral. Linq AI Research Blog. 2024.

[29] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *ArXiv* abs/1910.01108 (2019).

[30] Victor Sanh et al. *distilbert-base-multilingual-cased*. https://huggingface.co/distilbert/distilbert-base-multilingual-cased. Accessed: 2025-04-25. 2019.

[31] Shintaro Takahashi. *python-crfsuite: Python binding for CRFsuite*. https://github.com/scrapinghub/python-crfsuite. Accessed: 2025-04-19. 2015.

[32] Kamal Kurniawan. *torchcrf: Conditional Random Field for PyTorch*. https://github.com/kmkurn/pytorch-crf. Accessed: 2025-04-19. 2018.

[33] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[34] Zhiheng Huang, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF Models for Sequence Tagging". In: *ArXiv* abs/1508.01991 (2015). URL: https://api.semanticscholar.org/CorpusID:12740621.

List of Figures

List of Tables