Spring 5-2022

TransParsCit: A Transformer-Based Citation Parser Trained on Large-Scale Synthesized Data

MD Sami Uddin
Old Dominion University, rayan_sami@outlook.com

TRANSPARSCIT: A TRANSFORMER-BASED CITATION PARSER TRAINED ON LARGE-SCALE SYNTHESIZED DATA

by

MD Sami Uddin
B.S. February 2017, Military Institute of Science and Technology, Dhaka, Bangladesh

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2022

Approved by:

Jian Wu (Director)

Vikas Ashok (Member)

Faryaneh Poursardar (Member)

ABSTRACT

TRANSPARSCIT: A TRANSFORMER-BASED CITATION PARSER TRAINED ON LARGE-SCALE SYNTHESIZED DATA

MD Sami Uddin
Old Dominion University, 2022
Director: Dr. Jian Wu

Accurately parsing citation strings is key to automatically building large-scale citation graphs, so a robust citation parser is an essential module in academic search engines. One limitation of the state-of-the-art models (such as ParsCit and Neural-ParsCit) is the lack of a large-scale training corpus. Manually annotating hundreds of thousands of citation strings is laborious and time-consuming. This thesis presents a novel transformer-based citation parser by leveraging the GIANT dataset, consisting of 1 billion synthesized citation strings covering over 1500 citation styles. As opposed to handcrafted features, our model benefits from word embeddings and character-based embeddings by combining the bidirectional long short-term memory (BiLSTM) with the Transformer and Conditional Random Forest (CRF). We varied the training data size from 500 to 1M and investigated the impact of training size on the performance. We evaluated our models on standard CORA benchmark and observed an increase in F1-score as the training size increased. The best performance happened when the training size was around 220K, achieving an F1-score of up to 100% on key citation fields. To our best knowledge, this is the first citation parser trained on a large-scale synthesized dataset. Project codes and documentation can be found on this GitHub repository: `https://github.com/lamps-lab/Citation-Parser`.

Dedicated to my parents!

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES ix

Chapter

1. INTRODUCTION 1
   - 1.1 BACKGROUND 1
   - 1.2 PROBLEM DESCRIPTION 2
   - 1.3 RESEARCH QUESTIONS 3
   - 1.4 RESEARCH GOALS 3

2. BACKGROUND AND RELATED WORKS 5
   - 2.1 REGULAR EXPRESSION AND KNOWLEDGE BASES 5
   - 2.2 TEMPLATE MATCHING 6
   - 2.3 MACHINE LEARNING 7

3. FORMULATION 11
   - 3.1 HIDDEN MARKOV MODELS AND CONDITIONAL RANDOM FIELD 11
   - 3.2 DEEP LEARNING APPROACH FOR SEQUENCE MODELING 12

4. METHODOLOGY 15
   - 4.1 TRANSFORMER COMPONENTS 15
   - 4.2 FEATURES 22

5. DATA 25
   - 5.1 SYNTHESIZED DATASET 25
   - 5.2 DATA PREPROCESSING 26
   - 5.3 TRAINING SAMPLES 29

6. EXPERIMENTS 32
   - 6.1 SETUPS 32
   - 6.2 MODEL SELECTION AND EVALUATION ON CORA 32
   - 6.3 EFFECT OF INCREASING TRAINING SET 34
   - 6.4 COMPARISON WITH CORA 34
   - 6.5 ERROR ANAYLSIS 34

7. DISCUSSION 53
   - 7.1 LIMITATIONS OF THIS WORK 53
   - 7.2 TRANSPARSCIT FOR MULTILINGUAL REFERENCE STRINGS 53

Page

8. CONCLUSIONS AND FUTURE WORKS 54

REFERENCES 55

VITA 60

LIST OF TABLES

Table Page

1. Example reference strings from Computer Science and Biology domains. 2

2. Separated texts and their respective labels extracted by a citation parser. 3

3. A sample knowledge base [20]. 6

4. A sample database of citation templates. 6

5. A snapshot of the GIANT dataset structure. 27

6. A citation string split into its respective tokens an assigned label. 28

7. Schema matching between CORA and datasets that trains TransParsCit. 29

8. A reference string from CORA dataset after preprocessing. 30

9. Samples of training data for the models from GIANT. 31

10. Overall token level performance metrices of the model which was trained on 514 reference strings. 34

11. Entity level performance of the model that trained on 514 reference strings. 35

12. Overall token level performance metrices of the model which was trained on 1095 reference strings. 35

13. Entity level performance of the model that trained on 1095 reference strings. 36

14. Overall token level performance metrices of model trained on 10,074 reference strings. 36

15. Entity level performance of the model that trained on 10,074 reference strings. 37

16. Overall token level performance metrices of model trained on 21,900 reference strings. 37

17. Entity level performance of the model that trained on 21,900 reference strings. 38

18. Overall token level performance metrices of model trained on 100,740 reference strings. 38

Table Page

19. Entity level performance of the model which was trained on 100,740 reference strings. 39

20. Overall token level performance metrices of model trained on 219,000 reference strings. 39

21. Entity level performance of the model which was trained on 219,000 reference strings. 40

22. Overall token level performance metrices of model trained on 342,516 reference strings. 40

23. Entity level performance of the model which was trained on 342,516 reference strings. 41

24. Overall token level performance metrices of model trained on 1,027,548 reference strings. 41

25. Entity level performance of the model which was trained on 1,027,548 reference strings. 42

26. Performance comparison between Neural ParsCit and our model over CORA. 42

27. Model prediction for a CORA sample showing volume mispredicted as Issue. 46

28. Model prediction for a CORA sample showing CT predicted as other (O). 47

29. Another example of showing CT predicted as other (O). 48

30. Model prediction for a CORA sample showing CT mispredicted as VOL. 50

31. Model prediction for a CORA sample showing Title predicted as CT. 51

32. Model prediction for a CORA sample showing CT mispredicted as Page. 52

LIST OF FIGURES

Figure Page

1. Transformer encoder layer consists of the sublayers for self-attention and feed-forward network [44]. 16

2. Scaled Dot-Product Attention in the Encoder sublayer. 17

3. Multi-Head Attention consists of several attention layers running in parallel. The picture depicts only one layer of several parallel layers. 19

4. CBOW and SG configurations in word2vec models. 22

5. A representation of Bi-LSTM that trains the character embeddings and concatenates the outputs with the word embedding obtained from word2vec. 24

6. An unfolded representation of our final model comprising of Transformer-CRF using world level followed by a feed-forward layer and CRF showing how it would work with a reference string (see [8]). 33

7. Performance of each class over the different dataset sizes shows the upward F1-score with an increment of the dataset. 43

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

References are vital to acknowledge a previous scholarly contribution, such as scientific papers. This helps to understand the already developed scientific hypothesis, discoveries, and experiments about a research topic and assists in finding the key ideas and areas for incremental improvements. Correctly parsing and extracting citation strings and identifying tags such as Author, Title, Date, and others is needed for many downstream tasks that match paper records accurately. One example is developing a citation graph that benefits scholarly search engines or calculating the impact factor for a journal. The quality and quantity of citations are commonly accepted as a metric to evaluate a scholar [1, 2, 3].

A typical sequence-labeling model follows the procedures below. The preprocessing step includes obtaining reference strings from portable document format (PDF). The bibliographic section is first extracted to obtain the reference strings. The reference strings are then tokenized. After that, a list of hand-crafted features are extracted for each token, and a sequence labeling model can be trained to procure the labels for each token.

In research works, different academic disciplines have adopted different styles for reference strings. For example, APA and CSE are the commonly used style in biology, whereas MLA style is the primary format for humanities. Besides, different disciplines have different terminology sets and inconsistent entities, which increases error for dictionary-based lookups. For example, as shown in Table 1, a citation string from the Computer Science domain may contain very little similarity in vocabulary compared to a citation string from the Biology or Humanities discipline. Moreover, it is worth noting that documents converted to electronic copies from images using OCR are prone to have errors like missed spelling or character-breaking, which also poses difficulty to accurate citation parsing. Apart from OCR errors, human errors may occur while manually formatting reference strings that include extra spaces, missing punctuations, or style-related errors [4, 5]. These all make the task of citation parsing very challenging.

Although reference strings follow conventions that facilitate the parsing task, such as using delimiters to separate the fields, parsing approaches solely based on delimiting occurrences may fail since delimiters themselves are used inside the areas. Therefore, to build a

TABLE 1: Example reference strings from Computer Science and Biology domains.

Domain	Reference String
Biology	Hodgman, C. E., & Jewett, M. C. (2012). Cell-free synthetic biology: thinking outside the cell. Metabolic engineering, 14(3), 261-269.
Computer Science	Ojokoh, Bolanle, Ming Zhang, and Jian Tang. "A trigram hidden Markov model for metadata extraction from heterogeneous references." Information Sciences 181.9 (2011): 1538-1551.

robust model, it is essential to encode long-range dependencies that include the position of the current word relative to the previous and future sequence of entities. Thus the parsing task can be considered as a sequence problem where each token of the reference string (a sequence) is labeled with one of the possible labels. In natural language processing, models such as the hidden Markov model (HMM) [6] and conditional random field (CRF) [7] have been used for such sequential modeling tasks. HMM, and CRF is typically used to model long-range dependencies under a Markovian assumption, assuming that only a limited local context is enough to make good decisions. There are methods proposed and implemented for reference string labeling that use graphical models in both academic and commercial spaces, such as ParsCit [8], Neural ParsCit[9], CiteSeerX[10], and Mendeley[11].

1.2 PROBLEM DESCRIPTION

A citation parser is expected to extract the metadata fields accurately, given a reference string as input. For example, here is a reference string -

D. Foo, Abbreviations from Bar to Baz, Phys. Rev. 9, 34(2019)

When the citation parser gets an input, the string goes over some processing, the entire string is tokenized, and each token is labeled accordingly. In the end, the citation parser provides an output that shows the different pieces of reference string with designated labels. The outcome can be in XML, JSON, or CSV formats. For the example reference string

TABLE 2: Separated texts and their respective labels extracted by a citation parser.

Label	Text
Author	D. Foo
Title	Abbreviations from Bar to Baz
Journal	Phys. Rev.
Volume	9
Issue	34
Date	2019

above, the output would be like Table 2.

1.3 RESEARCH QUESTIONS

The goal of our project is to address these questions –

- How would training a deep learning citation parsing model on a large, synthesized dataset affect the accuracy of a citation parser?
- How does the training data size affect the performance of a citation parser?
- How is the performance compared with existing deep learning-based citation parsing tools like Neural ParsCit?

1.4 RESEARCH GOALS

Training TransParsCit requires a large and diverse dataset. To suit the requirement of being a capable training dataset for our citation parser, it should have the following properties-

- Incorporate a diverse range of citation styles
- Covers different citation types
- Contains a broad range of disciplines

- Hold some standard formatting error
- Large enough to cover all types of examples

We use the GIANT [12], containing 1 billion labeled synthetic reference strings with varying citation styles and types and bibliographic references from different disciplines. Citations can come from a journal, conference paper, blog, website, etc. Citation types represent what type of article a reference string is from. Citation style denotes how a citation string is arranged. There are different citation styles from IEEE, ACM, Harvard, MLA, etc. GIANT contains 1564 different citation styles and 53 distinct types. This dataset helps to tackle the common problem of the relatively domain-specific and small training dataset.

In this project, we propose a novel approach to develop a citation parsing tool using Transformer-CRF-based architecture, called TransParsCit, to extract the fields of a citation string in any given format. It utilizes GIANT to train models using latent features instead of hand-crafted ones. Our tool employs the transformer structure, and using the parallel processing attribute of the transformer; it can capture the state of a whole sequence into consideration while scoring a token without using convolution or a recurrent network. On top of that, it engages CRF that provides the prediction of a particular token considering the features and labels of neighboring tokens. In this work, we use the term label and tag interchangeably.

We prepare eight subsets of training data from the GIANT dataset to observe how performance changes based on training size on real-world data.

To evaluate the performance of our approach, we experimented with a benchmark dataset named CORA. The CORA data set [13] is widely used as a benchmark for citation parsing. The best performing model using TransParsCit achieved over 84 percent F1-score (micro) on average in key fields.

CHAPTER 2

BACKGROUND AND RELATED WORKS

CiteSeer [14] was one of the first attempts to develop a citation parser for a digital library search engine. It was developed in an effort to establish an automatic citation indexing system. Since then, it has been developed as an established research problem, and many approaches have been developed. These include regular expressions, template matching, knowledge bases, and machine learning approaches.

2.1 REGULAR EXPRESSION AND KNOWLEDGE BASES

In early works, regular expressions were used to tackle citation parsing problems, such as the work by Kunnas [15]. However, Tkaczyk et al. [4], and Zhang et al. [16] found that regular expression worked well for the small and clean dataset with predefined citation styles. The regular expression usually does not scale well for real-world data and lacks proper adaptability for new styles, and is non-trivial to maintain due to high complexity[16].

Researchers took several approaches [17, 18, 19] to combine regular expressions with knowledge bases to overcome these limitations. In this approach, a system is developed first with knowledge gathered using available and relevant data sources. This knowledge may include journal titles and authors' names. Citation parsing involves identifying a field extracted from the citation string to be matched against a known knowledge base. Cortez [20] defined the knowledge base as a set of pairs $KB = \{(m_1, O_1), ..., (m_n, O_n)\}$ in which each m_i is a distinct bibliographic metadata field, and O_i is a set of strings $\{o_{i,1}, ..., o_{i,n_i}\}$ called occurrences. The table 3 shows that Oi is set of typical values for field m_i as shown in table.

The approach used by Heckmann et al. [17] is a knowledge-based method that uses Markov logic networks (MLN) [21]. It is evaluated on a new data set including sparse and noisy data, and a 24.8% improvement is reported in the F1 score (0.88) over the popular CRF ML approach. Another knowledge-based system was used by Cortez et al. to design Flux-CiM [18]. Experiments were conducted across three domains: social science, computer science, and health science, matching seven fields: author, title, date, journal, volume, issue, and they found that FLUX-CiM has high precision and recall of over 94%.

TABLE 3: A sample knowledge base [20].

KB	(Author, O_{Author}), (Title, O_{Title})
O_{Author}	“Robert Kiyosaki”, “Napoleon Hill”, “J.K. Rowling”
O_{Title}	"Rich Dad Poor Dad”, “Think and Grow Rich”, “Harry Potter and the Sorcerer’s Stone”

TABLE 4: A sample database of citation templates.

Journal Reference styles	Reference style example
APA style	Mounier-Kuhn, P. (2012). Computer science in French universities: Early entrants and latecomers. Information & Culture: A Journal of History, 47(4), 414–456. https://doi.org/10.7560/IC47402
IEEE style	G. P. Luth, “Chronology and context of the Hyatt Regency Collapse,” Journal of Performance of Constructed Facilities, vol. 14, no. 2, pp. 51-61, 2013.
ACM style	HIRSH, H., COEN, M.H., MOZER, M.C., HASHA, R. and FLANAGAN, J.L. 2004. Room service, AI-style. IEEE intelligent systems, 14 (2), 8-19.

2.2 TEMPLATE MATCHING

Another approach to the problem of citation parsing is template matching. In this approach, citation strings are matched against a list of predefined templates. Regular expressions were usually used to represent these rules.

When citation styles are limited, template-based approaches perform well. Researchers have observed accuracy levels above 90% when experimenting with fewer than 22 citation

styles [22, 5]. However, Chen et al. comment that query processing is more efficient if the template database matches the test data [5]. A study that evaluated the citation parser BibPro using BLAST (Basic Local Alignment Search Tool) reported the highest average field-level accuracy based on a dataset that had only six different citation styles [5].

Scalability is a challenge for template-based approaches that need to handle thousands of citation styles [9, 8]. ParaTools [23] offers 400 templates to match citation strings, but adding new templates or maintaining existing ones is non-trivial. In addition to this potential challenge, a template-based approach may produce citation strings containing errors resulting from human error or OCR. Council et al. state that this approach is unsuitable for high-volume data processing because of its lack of portability [8].

2.3 MACHINE LEARNING

2.3.1 OVERVIEW

Machine learning-based citation parsing approaches significantly outperform rule-based approaches . Support Vector Machines (SVMs) [16], Hidden Markov Models (HMMs) [24, 25, 26], Conditional Random Fields (CRFs) [27, 28, 29] and deep learning [9, 30] are common ML approaches for this task. HMM was applied on this task before 2010 after which CRF became more popular. And deep learning has only been explored in the last four years.

An ML approach does not require expert knowledge to maintain the citation parser, as opposed to template matching or regular expressions. According to Tkaczyk et al. [4], retraining ML models with task-specific data can improve their performance. When retrained on task-specific data, the top three performing citation parsers improved their F1 scores by an average of 10%. Therefore, an ML model should only be retrained on more recent data to improve its performance on a particular citation style or domain-specific language. Therefore, a robust ML model does not need to create any new rules or adapt an existing knowledge base.

2.3.2 SUPPORT VECTOR MACHINES AND HIDDEN MARKOV MODELS

SVM is a supervised ML classifier that can be used in classifying tokens parsed in citation strings. Okada et al. [31] combined SVM and HMM and trained a model using 5-fold cross-validation with 4,651 citations. Their method had recall 14% higher than an SVM approach (0.988 vs. 0.974). A study by Zhang et al. [20] compared structural SVMs with conventional

SVMs.

SVMs based on structural properties use the contextual information contained in neighboring features. On field-level accuracy, structural SVM outperformed SVM (96.95% vs 95.59%). It was trained using 600 references and tested using 1800 reference strings from PubMed [32]. Zhang found that despite the promising results of Okada and Zhang, SVM performed worse compared to the CRF model [16].

Hidden Markov models (HMMs) are another machine-learning approach to citation parsing. An HMM is a probabilistic model in which the system to be modeled is conceived as a Markov process with hidden states. Hetzner [25] introduces a simple HMM-based model with the Viterbi algorithm. The model is trained and evaluated on the popular CORA dataset, demonstrating comparable performance to another benchmark HMM model. However, the results underperformed the results of the CRF model proposed by Peng and McCallum [33]. In a variation of the HMM model, Yin et al. [34] looked at the bigram's sequential relationship and position. Using 4-fold cross-validation on 736 labeled citations, they observed a 3.6% improvement in Bigram HMM performance compared with a regular HMM (F1=0.991 vs. F1=0.868). Using a trigram HMM, Ojokoh et al. [35] made further improvements. A 4-fold cross-validation was performed with three different training datasets ranging from 300 to 712, and the recall, precision, and F1 scores were all above 95%. However, they reported that increasing the training dataset size from 275 citations to 537 citations only resulted in a 0.07% increase in terms of accuracy.

2.3.3 CONDITIONAL RANDOM FIELDS

An analysis of ten existing citation parsing tools was carried out by Tkaczyk et al. [4] in 2018. In their report, they found that three of the most successful tools used a CRF algorithm: GROBID (F1=0.89), CERMINE (F1=0.83), and Parscit (F1=0.75).

The CRF-based tool such as GROBID [36] allows users to parse individual citation strings and offer a wide range of functionality throughout the document processing process. It is trained on 7,800 labeled citations, and Lopez reported a field-level accuracy of 95.7% on the CORA dataset.

Another CRF-based citation parser CERMINE [28], is a part of a larger document processing tool. It can extract citation strings and metadata directly from a PDF.

The open-source CRF-based tool Parscit [8] can locate, parse and retrieve the context of citation strings. It is deployed as part of CiteSeerX, a digital library for computer science.

The performance of Parscit on the CORA dataset was 4.4% higher than Peng and McCallum's [33] earlier work (micro-average F1=0.95 vs. macro-average F1= 0.91). Parscit uses 10-fold cross-validation on the CORA dataset.

2.3.4 DEEP LEARNING

A variety of NLP tasks, such as sequence labeling, have benefited from advances in deep learning techniques in recent years. A CRF prediction layer [37], word embeddings, and character-level embeddings are among the state-of-the-art methods for sequence labeling. Either Recurrent Neural Networks [35] or Convolutional Neural Networks (CNNs) [38] are used to train them. However, CNN is less popular for sequence modeling tasks than RNN. CNN understands a word by gathering the local information from nearby words and treats the terms one after another. Unlike RNN, CNN is independent of the computation of the previous state.

Rodrigues et al. [30] apply both CNN and RNN and compare the architectures for reference mining. Their training and evaluation data was from a corpus of literature on the history of Venice. They used pre-trained word embedding, which was trained using Word2Vec on the entire publications from which they extracted citations. Forty thousand reference strings were used to train the model. Their final model outperformed the CRF baseline by 7.03%, achieving an F1 of 0.896.

Prasad et al. [9] also developed deep-learning systems to parse citations. Using both word-based and character-based embeddings, they conducted extensive model experimentation and tuning. Their model includes a bidirectional Long Short-Term Memory (LSTM) architecture with a CRF layer to tag each token. The authors found a significant ($p < 0.01$) performance improvement compared to the CRF-only citation parser Parscit [8].

Prasad and Rodrigues' results are difficult to compare. The models were trained and evaluated on different datasets. Even though their available training data is relatively small [30], their results demonstrate the potential of a deep-learning approach.

2.3.5 META-LEARNING

Since different citation parsing tools can perform better or worse depending on the citation string given and the fields to be extracted, Tkaczyk et al. investigated a meta-learning approach to citation parsing [39]. ParsRec is a recommender system that suggests the best citation parser based on a citation string. They investigated two approaches to meta-learning recommendations. The first learns the best citation parsing tool for a given

string of citations, whereas the second learns the best tool for a particular field. As per their evaluation of 100,001 Chemistry references, the second approach achieved a 2.6% increase in F1 (0.90 vs. 0.886, $p < 0.001$) compared to GROBID, the best individual parsing tool.

CHAPTER 3

FORMULATION

The parsing of a citation string concludes with converting it into machine-readable forms, such as BibTeX. A citation string is a sequence of tokens, and citation parsing tools can apply sequence labeling solutions to the task. This chapter reviews the background of sequence labeling methods and goes over the fundamental statistical models - HMM and CRF. We also analyze the deep learning approaches that are associated with them.

Problem definition: Let us consider an input sequence $X = (x_1, x_2, \cdots, x_i, \cdots, x_n)$
And possible output (label) sequence $Y = (y_1, y_2, \cdots, y_i, \cdots, y_n)$
The task is to predict an optimal output sequence $\hat{Y} = (\hat{y_1}, \hat{y_2}, \cdots, \hat{y_i}, \cdots, \hat{y_n})$

x_i represents the input, a token in the input sequence X. And y_i represents an output label of x_i. Statistical models can be used during training to obtain a nifty output sequence. These methods and corresponding models are discussed in more detail below.

3.1 HIDDEN MARKOV MODELS AND CONDITIONAL RANDOM FIELD

Hidden Markov models can be used to predict sequential data. HMM can be used for obtaining the output sequence tag by training to maximize the joint log probability of a pair of consecutive token tag positions as:

$$\hat{Y} = \arg\max \log P(X \mid Y) \tag{1}$$

An HMM is used to model a joint probability, when Viterbi decoding is applied to determine the conditional likelihood and thus find the sequence tags during the inference phase. This disparity in testing and training objectives and the possibility of label bias [7] that results from the use of such maximum likelihood models have caused them to be adopted by CRFs in many practical applications. Label bias can cause a model to completely ignore the current observation when predicting the next label. And CRF overcomes it by computing conditional probabilities of global optimal output nodes.

Conditional Random Fields are a class of discriminative models used for predicting sequences. They use contextual information from previous labels, thus increasing the model's amount of information to make a good prediction. Given a set of observed data, the conditional probability of the output labels:

$$\hat{Y} = \arg\max \log P(Y \mid X) \tag{2}$$

Here, the conditional probability, $p(Y|X)$ for the sequence is determined by calculating the product of potential $function(\varphi)$ at each time step as:

$$P(Y \mid X) = \frac{\prod_{i=1}^{N} \varnothing(y_{i-1}, y_i, X)}{\sum_{Y \in Y^*} \prod_{i=1}^{N} \varnothing(y'_{i-1}, y'_i, X)} \tag{3}$$

Hidden Markov Models are generative in the sense that they provide results by modeling the joint probability distribution. Conditional Random Fields, on the other hand, is discriminative in that they model the distribution of probability conditions. CRFs do not assume that the labels are independent of one another. An alternative interpretation of the Hidden Markov Models is that they are a special case of Conditional Random Fields, where instead of using continuous transition probabilities, they use constant probabilities.

CRF models are better compared with HMMs for tasks that require less supervision and have a high degree of class imbalance since CRFs use all their modeling power to discriminate between classes, whereas HMMs divide their modeling power between class discrimination and generation.

3.2 DEEP LEARNING APPROACH FOR SEQUENCE MODELING

When input features are fed to a feed-forward neural network, they go over a sequence of operations in the layers $i = 1$ to L and result in nonlinear transformations of the input feature. Let's say if an input feature is x, then the operation in i-th layer is:

$$h^i = f(W^i \tilde{h}^i(x)) + b^i \tag{4}$$

where,

$$\tilde{h^i}(x) = \begin{cases} x, & \text{if } i = 1 \\ h^{i-1}(x), & \text{otherwise} \end{cases} \tag{5}$$

$$f(x) = \begin{cases} \frac{\exp(x_{\mathrm{k}})}{\sum_{j=0}^{\dim(x)} \exp(x_{\mathrm{j}})}, & \text{if } i = L \\ \tanh(x) = 2\sigma(2x) - 1 = \frac{e^{xk} - e^{-xk}}{e^{xk} + e^{-xk}}, & \text{otherwise} \end{cases} \tag{6}$$

In the hidden layer, nonlinearity is introduced by the tanh function. Softmax enables the nonlinear operation on the output layer ($i = L$) and facilitates multiclass classification. As a result, the output of a simple feed-forward network can be treated as a probability distribution over all classes (i.e., the normalized class prediction for classes $n = 1$ to N can

be treated as the conditional probability $p(y(n)|x)$. L is the final layer that implements the multiclass classification using a softmax function. The other layers allow the modeling process to bypass their input through nonlinear functions (feed-forward layers).

Standard feed-forward models have one notable drawback: they cannot capture temporal dependencies effectively when generating probability distributions. Thus, the conditional probabilities generated by the model are independent of the samples seen previously and the samples seen in the future. Due to this weakness, the standard model is impractical for tasks such as citation parsing, which observe a sequence of tokens. Hybrid techniques have been introduced to overcome this problem by running the output of the neural model through a CRF or an HMM layer in a stacked fashion [9]. A CRF or an HMM layer is then used to accomplish the actual classification based on its own objective function, using the output probability of the neural models as an abstract feature.

However, the neural modeling paradigm also offers native solutions to these problems. The architectures of neural networks, such as recurrent neural networks (RNNs), account for the predictions of the past and/or future (time steps). The strength of RNNs lies in their ability to remember output history through connections to previous time steps and to use such historical evidence to make decisions about the current time step as follows:

$$h(x_{\mathrm{t}}) = \tanh\left(W_{\mathrm{x}} x_{\mathrm{t}} + W_{\mathrm{h}} h(x_{\text{t-1}}) + b\right) \tag{7}$$

Cross-entropy (CE) is usually used as a loss function for such models:

$$CE = -\frac{1}{T} \sum_{t} \sum_{n} \hat{y}_{\mathrm{t(n)}} \log(y_{\mathrm{t(n)}}) \tag{8}$$

By classifying each of the tags correctly, the model reduces CE instead of trying to learn the distribution of tags over the sequence, as is done through HMMs or CRFs.

These models can extend beyond the Markov assumption and predict labels individually without joint decoding. Virtually RNN can pick up historical evidence to the start of a sequence. However, vanilla RNN is not effective at capturing long-term dependencies due to the problem of vanishing gradients [40]. In addition, vanilla RNN captures the dependence in a single direction. This is not always the case. Therefore, it is desirable to run RNN in the opposite direction and concatenate a token's representation generated in either direction. An RNN can be modified to prevent vanishing gradients and exploding gradients if it incorporates memory cells (such as a temporary; [41]). One example is the long short-term memory (LSTM) architecture. A basic memory cell can be trained to retain

relevant information from the input [42]. Both vanilla RNN and LSTM process a sequence of tokens. As a result, they cannot be trained in parallel. To compute the hidden state of a token, we need to calculate the previous token first. A token's representation strongly depends on the representation of the token immediately before it but is less dependent on tokens far from it.

The transformer model was designed to overcome the limitations of sequence processing [43] since it allows parallel computation by avoiding recursion and decreasing performance drop on more extensive sequences where long dependencies are involved. In summary, we utilize the Transformer Encoder, followed by a CRF layer to parse citation strings.

CHAPTER 4

METHODOLOGY

In the following sections, we will describe each component of the deep learning architecture we adopted.

4.1 TRANSFORMER COMPONENTS

A Transformer contains an encoder-decoder structure similar to many neural network models. For our purpose, we only used the encoder layer of the transformer. We use the CRF model as the decoder layer.

The encoder converts a sequence of token representations $(x_1, \cdots, x_n)$ to a new sequence of representation, typically in a different feature space $z = (z_1, \cdots, z_n)$.

4.1.1 ENCODER LAYER

As shown in Fig. 1, the encoding layer can be a stack of encoders, each containing a multi-head self-attention module and a fully connected feed-forward neural network. The input first goes through the encoder's self-attention layer. The outputs of the attention layer are fed to a feed-forward neural network which is applied individually to each position. Each of the two sublayers has a residual connection with a layer normalization.

4.1.2 SELF ATTENTION

When the embeddings are fed to the encoder as input, three different vectors are created to calculate the self-attention. For each input vector, there are Query (Q) vector, a Key (K) vector, and a Value (V) vector. The goal is to get a score for each target word against other words of the input sequence. This score determines the amount of focus needed on other parts of a sequence while encoding a particular word at a certain position. This score is calculated by having a dot product between the Query vector (Q) and Key vector (K) of a target word.

If the position of a particular word is #1, we get the score by a dot product of q1 and k1; if the position is #2, we get the score by the dot product of q2 and k2. The dimension of query and key vectors are d_k, and the dimension of value vectors is d_v. Each score we

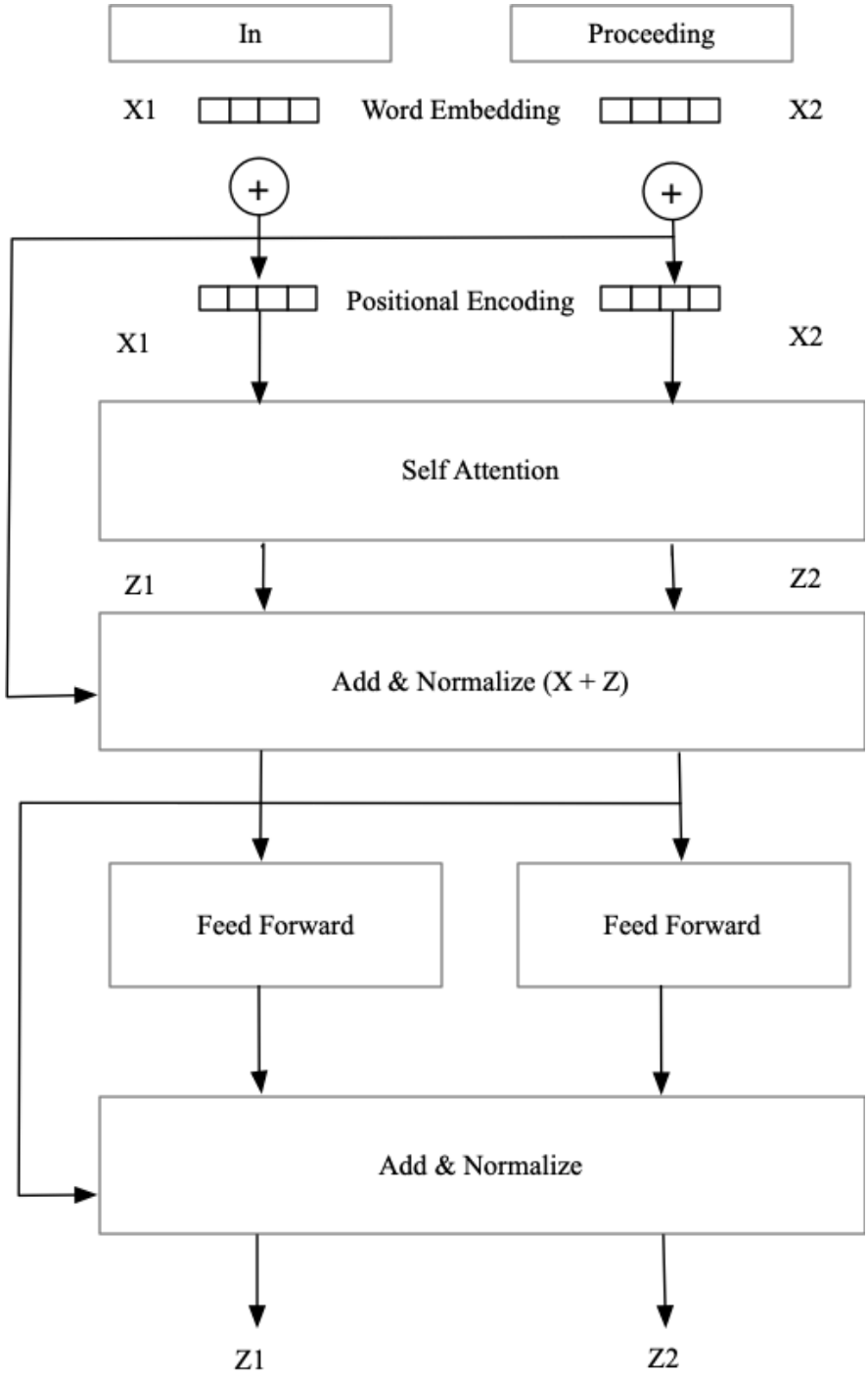

Fig. 1. Transformer encoder layer consists of the sublayers for self-attention and feed-forward network [44].

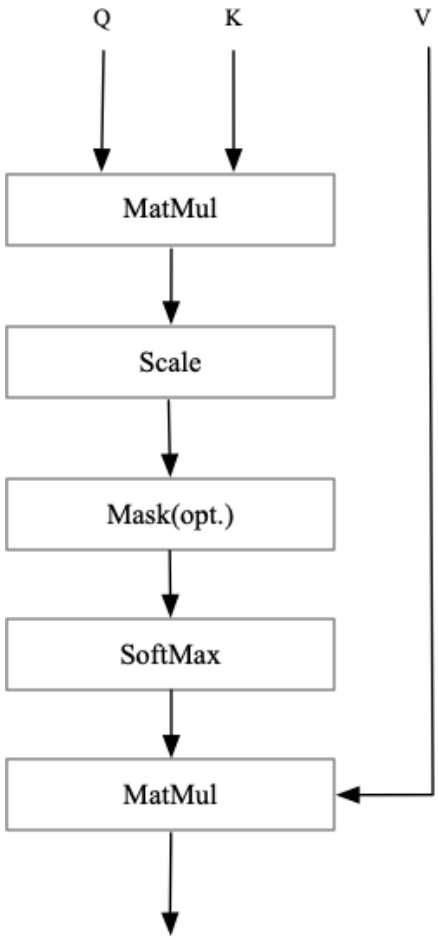

Fig. 2. Scaled Dot-Product Attention in the Encoder sublayer.

obtained after the dot product was normalized by dividing by $\sqrt{d_k}$ and passed through a softmax function that calculated a score, which determined how much each word would contribute to the target word. By definition, the word at the current position will have the highest softmax score, but this score helps to determine the relevance of the other words in the sequence with the present word. Next, each value vector was weighted by the softmax score, which pays more attention to the words that should be focused on and less attention to less relevant words. The weighted summation vector finally produces the self-attention layer output for the target word. The resulting vector goes over the feed-forward neural network.

In practice, the computation of the attention function was imposed simultaneously on a set of queries, which can be denoted as a matrix Q after being grouped together. The keys and values are also grouped together into matrices K and V.

Here's the equation to calculate the output. Let's call it the Z matrix:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_{\mathrm{k}}}})V \tag{9}$$

4.1.3 MULTI-HEAD ATTENTION

In multi-head attention mechanism queries, keys and values get linearly projected for h times with separate learned projections. Each version of these h projections goes over the single attention function parallelly and produces h outputs. These outputs get concatenated and projected to construct the final outcome like Fig. 3.

With the multi-head attention layer, there are multiple sets of Query/Key/Value matrices instead of only one. Each set gets randomly initialized at the beginning and projects the input embeddings into different feature subspaces. These presentation sets are concatenated, resulting in one final value and multiplied with an additional weight matrix, W_0, then fed into the feed-forward layer.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_{\mathrm{h}})W^o \tag{10}$$

$$Where\ head_{\mathrm{i}} = Attention(QW_{\mathrm{i}}{}^Q, QW_{\mathrm{i}}{}^K, QW_{\mathrm{i}}{}^V) \tag{11}$$

Here the parameters are W_i^Q, $W_i^K \epsilon \mathbb{R}^{d_{model} \times d_k}$ and $W_i^V \epsilon \mathbb{R}^{d_{model} \times d_v}$ and $W_i^0 \epsilon \mathbb{R}^{d_{model} \times hd_v}$

For our purpose, we used h = 16 parallel layers or attention heads while d_{model} was 400 on each encoder layer.

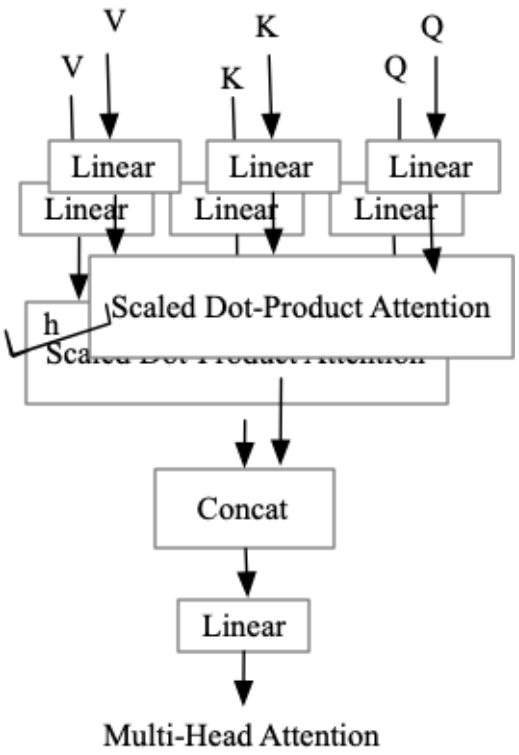

Fig. 3. Multi-Head Attention consists of several attention layers running in parallel. The picture depicts only one layer of several parallel layers.

4.1.4 POINT-WISE FEED FORWARD NETWORK

Each encoder contains a self-attention sublayer, a feed-forward network, and a residual connection.

The output of the multi-head attention sublayer adds up with the positional input embeddings, which is called a residual connection. The layer normalization reduces training time by normalizing the activities of the neurons. The normalized residual output was projected through a point-wise feed-forward network consisting of a couple of linear layers with a ReLU activation in between. The residual output was the input of the point-wise feed-forward network, which was further normalized in the next step.

The normalized residual output was projected through a point-wise feed-forward network. The point-wise feed-forward network consists of a couple of linear layers with a ReLU activation in between. The residual output was the input of the point-wise feed-forward network, which was further normalized in the next step.

By allowing gradients to flow directly through the networks, residual connections help the networks train faster. The layer normalization stabilized the network and significantly reduced the training time. The attention outputs can be converted to a richer representation using the point-wise feed-forward layer.

4.1.5 POSITIONAL ENCODING

Traditional word embeddings, such as word2vec, lack positional information. Softmax obscures any positional information that may exist in self-attention.

The transformer model preserves the positional information by injecting a vector into each embedding. Using positional embedding, the model can predict the location of individual words with respect to each other based on the periodic function (e.g., the combination of sines and cosines with different frequencies).

These vectors follow a specific pattern that the model learns, which helps determine the position of each word or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

As the paper describes, we use sine and cosine functions of different frequencies:

$$PE_{(\mathrm{pos,2i})} = \sin(pos/10000^{2i/d_{\mathrm{model}}}) \tag{12}$$

$$PE_{(\mathrm{pos,2i+1})} = \cos(pos/10000^{2i/d_{\mathrm{model}}}) \tag{13}$$

Here, pos stands for a position, and i stands for dimension. That is, positional encoding is represented as a sinusoid at each dimension. In terms of wavelengths, the geometric progression is from 2π to $10000 \cdot 2\pi$.

4.1.6 CRF LAYER

We use CRF as the top last layer of our architecture. This layer takes the concatenated hidden states from the underlying networks as input and models the joint probability of all the labels in a sequence. The probability of the label sequence is calculated as [7]:

$$P(L \mid S) = \frac{\exp(\sum_i (W_{CRF}^{l_{\mathrm{i}}} h_{\mathrm{i}} + b_{CRF}^{(l_{\mathrm{i\text{-}1}}, l_{\mathrm{i}})}))}{\sum_{L'} \exp(\sum_i (W_{CRF}^{l'_{\mathrm{i}}} h_{\mathrm{i}} + b_{CRF}^{(l'_{\mathrm{i\text{-}1}}, l'_{\mathrm{i}})}))} \tag{14}$$

Here label sequence $L = l_1, l_2, l_3 \ldots, l_n$ and $L^{'}$ represents an arbitrary label sequence. Also $W_{CRF}^{l_i}$ is the model parameter for l_i and $b_{CRF}^{(l_{i-1}, l_i)}$ is a bias specific to l_{i-1} & l_i.

For decoding, a first-order Viterbi algorithm is used to find the most probable label sequence over the input sequence.

In order to get the most probable sequence labels over the input sequence, a first-order Viterbi algorithm is used. For this decoding purpose, the sentence-level log-likelihood loss with L_2 regularization is used to train the model:

$$L = \sum_{i=1}^{N} \log P(y_{\mathrm{i}} \mid s_{\mathrm{i}}) + \frac{\lambda}{2} \|\theta\|^2 \tag{15}$$

Here Θ is the parameter set, λ is the L_2 regularization parameter and $(s_i, l_i)_{i=1}^{N}$ is the set of manually labeled data.

Here Θ stands foris the parameter set, λ is the L_2 regularization parameter and $(s_i, l_i)_{i=1}^{N}$ stands for manually labeled data.

CRF maximizes the likelihood of the whole sequence of decisions, so it can better model cases in which neighboring decisions have to have some kind of interface, allowing them to function together. This is accomplished by using the output of the transformer as an input to the CRF.

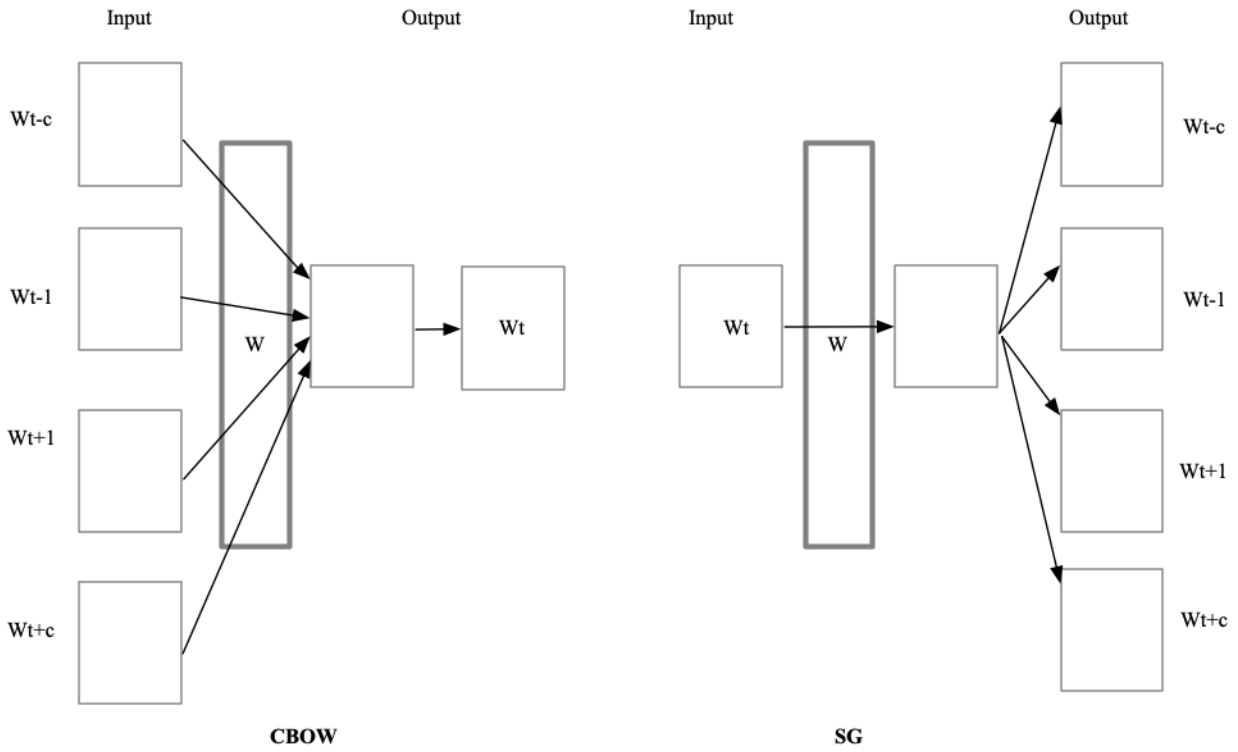

Fig. 4. CBOW and SG configurations in word2vec models.

4.2 FEATURES

1. Word Embeddings are word representation that allows words with similar meanings to reside in the proximity of each other in the feature space. Each word is represented by a real-valued distributed vector with relatively low dimensions, 300 in our case. In contrast, sparse word representations such as one-hot encoding require thousands to millions of dimensions. Distributed representations are learned by capturing the patterns in large-scale text corpora. As a result, words used in similar ways will naturally have similar representations. We use "pre-trained" word embedding provided by Google called word2vec, trained on google news [45].

Two approaches were proposed to train word2vec: namely, the Continuous Bag-of-Words (CBOW) and the Skip-gram (SG) models. CBOW and SG configurations are shown in Fig. 4 [46]. In CBOW, based on an input context, the model predicts the word. In SG, a context is predicted over a fixed vocabulary V based on an input word.

The pre-trained embeddings that we are using were obtained using Skip-gram [47].

2. Character-based Word Embeddings Characters contain unique information about word arrangement and morphology. In a sequence like a reference string, punctuation marks-such as commas, hyphens, parenthesis, or numbers can have evident effects on the token classification. For example, a four-letter number is likely to be a Date, and a numeric field with a hyphen is likely to be a Page Range. To catch the character-level information, we use character-based word embedding. Each character of a word is represented as a numeric vector and further encoded by a Bi-LSTM. The vectors were randomly initialized.

Word embedding can only be used on the word that has been already seen during training. With the character embedding, every word can be formed even when that belongs to out-of-vocabulary words (previously unseen). It handles infrequent words better than word2vec embeddings since the latter may lack training on the uncommon words. These embeddings are small and reduce model complexity while improving performance (speed).

To create a character-based representation of a word, we use Bi-LSTM. Let's say $w = c_1, c_2 \ldots c_i \ldots c_c$ where w denotes a word and c_i denotes a character on that word. The vector representation of each character x_{c_i} is passed sequentially through the LSTM cell, which provides the output vector $[E_{x_{c_1} \ldots x_{c_i}}]$ after performing a series of vector and matrix operations. The output vector represents all the characters of the word w subsequently. Comparably the character vectors are also fed to another independent LSTM network in reverse order, which produces the output vector $[E_{x_{c_i} \ldots x_{c_1}}]$. The resultant vectors from both networks are concatenated to construct a character-based word embedding $[E_{x_{c_1} \ldots x_{c_i}} E_{x_{c_i} \ldots x_{c_1}}]$. The final character-based embedding for w is then concatenated with the respective word embedding W that is obtained from the pretrained word2vec dictionary.

The architecture used for generating character embedding is shown in Fig. 5. Character embeddings are trained in a part of a network among other parts using a common error function. The figure shows how the representation of the token "Proc", a short form for "Proceedings", was generated. The output from the BiLSTM (denoted as CE_{Proc}) is then concatenated with the respective word embedding W (denoted as W_{Proc}).

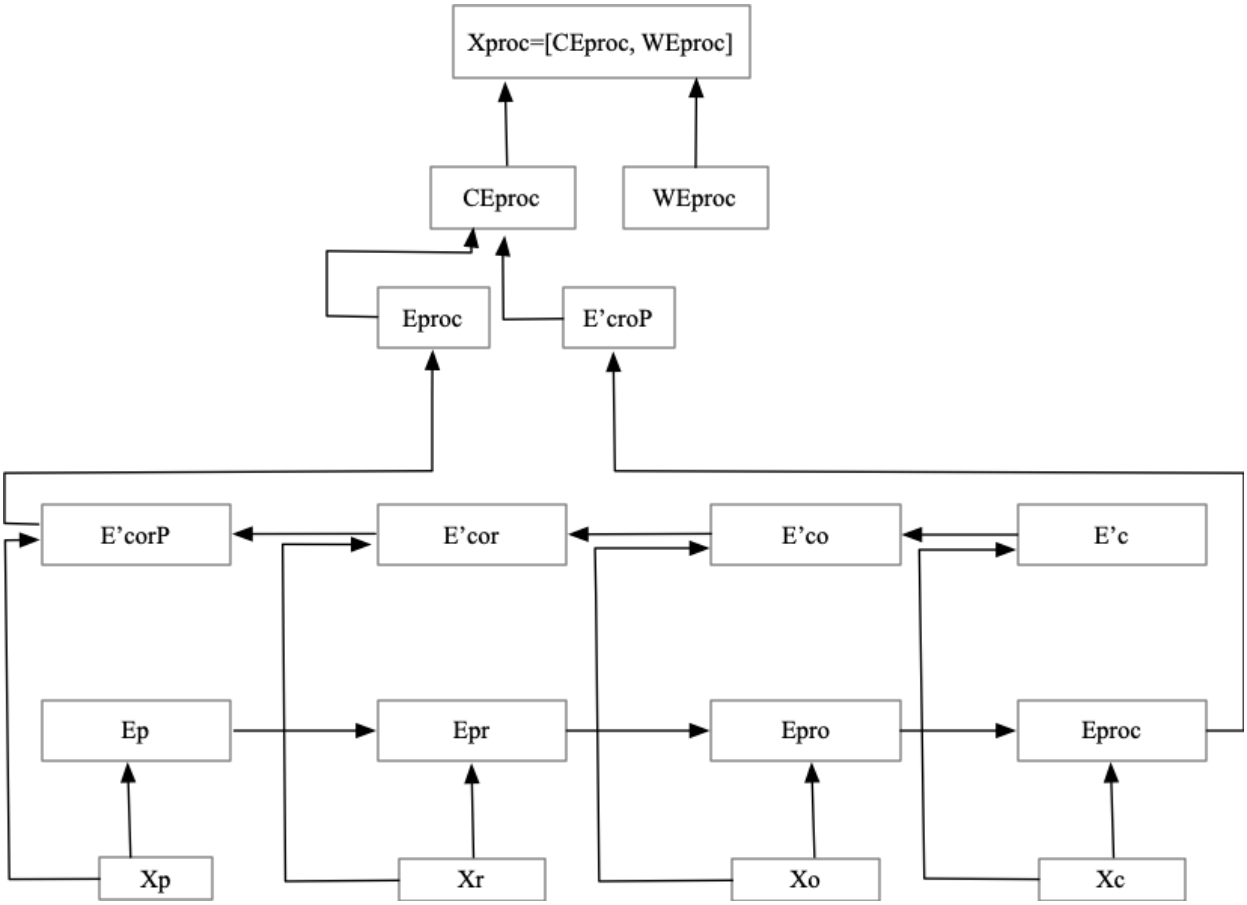

Fig. 5. A representation of Bi-LSTM that trains the character embeddings and concatenates the outputs with the word embedding obtained from word2vec.

CHAPTER 5

DATA

We acquire two types of datasets – synthesized data for training and validation and manually annotated data for evaluation. Due to the enormous size of our synthesized data, we sample the synthesized data randomly to develop the training datasets. And we use a widely used citation parsing evaluation dataset, CORA, to evaluate our model. This human-annotated dataset contains 500 citation strings; 90% are from computer science papers.

5.1 SYNTHESIZED DATASET

The synthesized dataset is generated by parsing citation styles described by the Citation Style Language (CSL) with the bibliographic records in the digital library metadata and then combining them into a processed dataset. These metadata are relatively accurate because, unlike automating the extraction of the metadata, they were initially input by the authors and so can be used as the ground truth. Each citation style specifies how each metadata field in a citation string should be arranged within the citation string. As part of this process, the CSL processor is used, which accepts a bibliographic record (in the format of a .bib or .ris file) and a citation style (i.e., CSS) as inputs, and outputs a reference string.

An existing synthesized dataset is GIANT, consisting of 1 billion reference strings [12]. The author took advantage of Crossref [48] to create this dataset, which has freely available scholarly metadata from various academic fields. There were 677,000 unique records obtained from Crossref using their public API. Each record was then converted into a labeled citation string with 1,564 different citation styles. For every citation record, a total of 1,564 citation strings were generated in GIANT. Citation styles were obtained from the official CSL repository on GitHub [49]. To generate the desired citation string, the author used an open-source CSL processor such as citeproc-js [50].

The GIANT dataset contains 219 CSV files. Each file has around 3000 unique citation strings on 1564 citation styles. Each file has three pieces of metadata, with each labeled citation. These are:

1. The DOI of the citation

2. The citation type (book, journal article, etc.)

3. The citation style (Harvard, MLA, etc.)

Here is a sample citation string:

<author><family>Nakamura</family>, <given>R.</given>. & <given>T.</given> <family> Kenzaka</family>.</author> <issued><year>2015</year></issued>. <title >Magnetic resonance imaging of cardiac amyloidosis</title> <container-title>QJM</co ntainer-title> <volume>109</volum e>. <publisher>Oxford University Press (OUP)</ publish -er>: <page>63-63</page>. <URL>htt p://dx.doi.org/10.1093/qjmed/hcv155</URL>

Each labeled reference string has up to ten different entities, depending on the citation style. Author, title, container-title, publisher, volume, issue,issued-year,issued-month, page, and URL/DOI. Moreover, each name inside the author label is further divided into family names and given names. The <title> tag labels the article or chapter title of a reference. <container-title> is used for labeling journals, books, and non-bibliographic titles.

5.2 DATA PREPROCESSING

In an existing citation parser Neural-Parscit, the tokenizer used spaces as the delimiter. Although this delimiter was applicable for most fields, certain tokens resulting from this tokenizer may contain more than one field, such as "1(1)", which combines volume and issue numbers. In addition, the above tokenizer ignored punctuation marks, which could be helpful for classifying tokens. Therefore, we developed a new tokenizer to process a reference string before it is passed to the transformer-based parser. This tokenizer allowed us to consider both alphanumeric and punctuation. For example, the following citation string:

<author><family>Gabbay</family> <given>Dov M.</given></author> <title> Index</title>. <container-title>Vistas in Astronomy</container-title> <issued><year> 1975 </year></issued>; <volume>18</volume>: <page>1017-1034</page>

Here the individual tokens would be: C = {Gabbay, Dov, M,., Index,., Vistas, in, Astronomy, 1975,;, 18,:, 1017, -, 1034}

Once the string has been tokenized, the preprocessor assigns each token the label based on their belonging class. Our preprocessor considers everything else as "Other" which are

TABLE 5: A snapshot of the GIANT dataset structure.

Doi	Article Type	Citation Style	Annotated Citation String
10.1016 /j.carrev .2009.04 .081	3	0	<author><family>Mehrinfar</family>, <given>Ramona</given> and <family>Shah</family>, <given>Atman Prabodh</given></author> (<issued><year>2009</year></issued>) <title>Bivalirudin versus heparin and eptifibatide in patients undergoing rescue percutaneous coronary intervention.</title> <container-title>Cardiovascular Revascularization Medicine</container-title>, <volume>10</volume>(<issue>4</issue>), p. <page>267</page>. [online] Available from: <URL>http://dx.doi.org/10.1016/j.carrev.2009 .04.081</URL>
10.1016 /j.jlumin .2015.09 .036	3	0	<author><family>Zmojda</family>, <given>J.</given>, <family>Kochanowicz</family>, <given>M.</given>, <family>Miluski</family>, <given>P.</given>, <family>Dorosz</family>, <given>J.</given>, et al.</author> (<issued><year>2016</year></issued>) <title>Investigation of upconversion luminescence in antimony–germanate double-clad two cores optical fiber co-doped with Yb3+/Tm3+ and Yb3+/Ho3+ ions</title> <container-title>Journal of Luminescence</container-title>, <volume>170</volume>, pp. <page>795-800</page>. [online] Available from: <URL>http://dx.doi.org/10.1016/j.jlumin.2015 .09.036</URL>

TABLE 6: A citation string split into its respective tokens an assigned label.

Token	Label
Gabbay	B-AUTHOR
Dov	I-AUTHOR
M	I-AUTHOR
.	B-PUNC
Index	B-TITLE
.	B-PUNC
Vistas	B-CT
In	I-CT
Astronomy	I-CT
1975	B-DATE
;	B-PUNC
18	B-VOL
1017	B-PAGE
-	B-PUNC
1034	I-PAGE

not part of an XML tag, and consider all type of punctuation (regardless of their position inside the XML tag or not) as a part of the "Punc" class.

We follow the CoNLL-2003 IOB convention to format the input. Each line contains a token followed by the corresponding label. Individual citation strings are separated by an empty line. Each starting word inside an XML tag gets a "B" prefix on its label name. Other words in the same field were labeled with "I" followed by the field name (Table 6). We removed URL/DOI from the training samples before preprocessing because regular expressions could easily capture them.

5.2.1 TRAINING AND VALIDATION DATA

Table 6 shows the exact citation string previously shown with tokens and their corresponding labels. We do not consider spaces, so that is removed.

TABLE 7: Schema matching between CORA and datasets that trains TransParsCit.

CORA Entity	TransParsCit
Author	Author
Booktitle / journal	CT
Date	Date
Pages	Page
Publisher	Publisher
Volume	Vol
Title	Title
Punctuation	Punc
Location	<remove>
Tech	<remove>
Institute	<remove>
Editor	<remove>
note	<remove>

5.2.2 EVALUATION DATA

We used the standard CORA dataset for evaluation. The schemas between GIANT and CORA do not match perfectly, so we made some changes to preprocess the evaluation data by removing a location, tech, institute, editor, and note fields from the annotations of CORA. Besides, we combined booktitle and journals under the same entity called CT.

Here is a sample from CORA dataset:

> <author> J. J. Koenderink. </author> <title> The structure of images. </title> <journal> Biological Cybernetics, </journal> <volume> 50 </volume> <pages> 363-396, </pages> <date> 1984. </date>

5.3 TRAINING SAMPLES

We prepared eight subsets of GIANT data. We sampled the same number of citation strings from each file and combined them for each subset. For example, if we want to develop

TABLE 8: A reference string from CORA dataset after preprocessing.

Token	Label
J	B-Author
.	B-PUNC
J	I-Author
.	B-PUNC
Koenderink	I-Author
The	B-TITLE
Structure	I-TITLE
Of	I-TITLE
images	I-TITLE
.	B-PUNC
Biological	B-CT
Cybernetics	I-CT
50	B-VOL
363	B-PAGE
-	B-PUNC
396	I-PAGE
,	B-PUNC
1984	B-DATE
.	B-PUNC

TABLE 9: Samples of training data for the models from GIANT.

Token	Label
514	2 ref strings from 200 files and 6 from 19 files
1095	5 ref strings from 219 CSV files
10,074	46 ref strings from 219 CSV files
21,900	100 ref strings from 219 CSV files
100,740	460 ref strings from 219 CSV files
219,000	1000 ref strings from 219 CSV files
342,516	1564 ref strings from 219 CSV files
1,027,548	4692 ref strings from 219 CSV files

a training dataset of 100,740 citation strings, we randomly choose 460 reference strings from 219 files.

CHAPTER 6

EXPERIMENTS

Here we describe the experimental setups. We also describe our findings and compare our results with the Neural ParsCit. We develop our architecture using building blocks based on the Pytorch framework [51].

6.1 SETUPS

The pre-trained word embedding we used encodes a word to a 300-dimensional vector. Our architecture uses word embedding as the first layer, followed by the character embedding layer. The dimension of each character vector is set to 25. The BiLSTM layer used for training the character embedding has 50-dimensional hidden layers.

We only use one layer of the transformer encoder layer. The multi-head attention sub-layer inside the transformer encoder layer contains 16 attention heads. The concatenated word features have the dimension of 400, which is set to d_{model}. The intermediate layers inside the transformer encoder use ReLU for activation. The feed-forward neural network contains 2048 neurons.

Our Transformer-CRF network is shown in Fig. 6 shows the workflow for a part of the reference string "string parsing package".

All models are trained and tested on Tesla V100-SXM2-16GB. It takes a few minutes to a few hours to train a model based on the dataset sizes. The model contains about 2.5 million parameters. We set the learning rate to be 0.001. The loss function was negative log-likelihood. We used one layer of transformer encoder on the transformer components, which got 16 attention heads and 200 hidden layers on a feed-forward neural network. We train eight different models and examine the effects of large-scale data on performance.

We describe our experiments, starting with a small dataset and then increasing the size of the dataset to investigate how the performance changed with the size of the training data.

6.2 MODEL SELECTION AND EVALUATION ON CORA

We used the CORA dataset to evaluate our models. We performed 10-fold cross-validation by splitting this dataset into ten random groups, each containing 10% of the standard dataset, and iterating through them to see the performance of each group. The

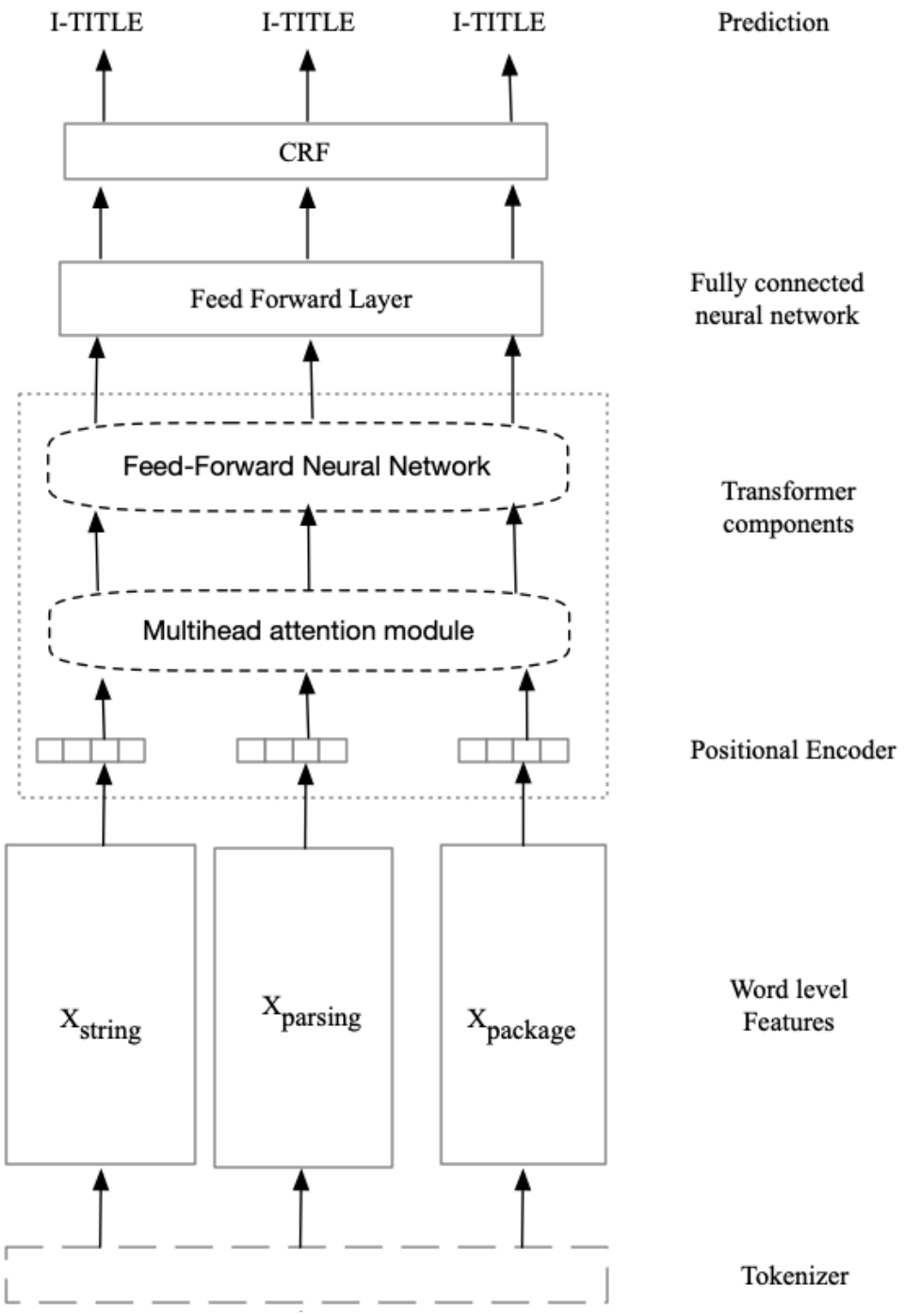

Fig. 6. An unfolded representation of our final model comprising of Transformer-CRF using world level followed by a feed-forward layer and CRF showing how it would work with a reference string (see [8]).

TABLE 10: Overall token level performance metrices of the model which was trained on 514 reference strings.

Performance Metrices	Scores
F1 score	84.7
Recall	83.6
Precision	83.6

table below shows the performance of our best model by averaging evaluation metrics of all fields. The table below shows the performance of our best model for each metadata field.

Here are the performance reports for our models:

6.3 EFFECT OF INCREASING TRAINING SET

Our observation shows that, in general, the larger the training data, the better the model's performance. However, we also notice that model performance does not improve after a certain point containing 220K training samples. The decreased performance with a more extensive training set may be due to overfitting.

We notice that Author, Date, Page, Publication, Title, Punc (punctuation) achieves F1-score over 87%. In contrast, CT (container-title) and Vol (Volume) perform poorly across all the models. For CT, we get the F1-score of 40.762.1%, and for Vol, it is 39.465.2%.

6.4 COMPARISON WITH CORA

We combine Journal and Booktitle into CT and take the average score. By directly comparing the results reported in the Neural-ParsCit paper, we found that Neural ParsCit outperformed our model in all fields.

There can be several reasons behind our model performance compared with Neural ParsCit. Neural ParsCit has been trained using tenfold cross-validation, splitting 80% for train and 10% for validation and test. Also, the performance results that are shown here were evaluated over the same dataset, CORA. However, we trained our model with GIANT, a synthesized dataset, while evaluating using a different CORA dataset. Therefore, the results above indicate that TransParsCit underperforms NeuralParsCit on the CORA dataset. However, because most of CORA samples were extracted from computer science

TABLE 11: Entity level performance of the model that trained on 514 reference strings.

Class	Precision	Recall	F1 score
Author	97.2	91.3	94.2
CT	40.1	42.2	40.7
Date	64.7	39.1	48.8
Page	52.4	55.0	53.7
Publisher	34.9	85.2	48.2
Punc	100	99.1	99.6
Title	60.2	73.7	65.9
Vol	34.2	47.9	39.4
Micro avg	83.7	86.0	84.9
Macro avg	53.6	59.3	54.5
Weighted avg	86.6	86.0	85.9

TABLE 12: Overall token level performance metrices of the model which was trained on 1095 reference strings.

Performance Metrices	Scores
F1 score	87.4
Recall	87.2
Precision	87.2

TABLE 13: Entity level performance of the model that trained on 1095 reference strings.

Class	Precision	Recall	F1 score
Author	96.4	94.4	95.3
CT	50.7	47.5	48.7
Date	78.6	50.1	61.0
Page	61.4	63.9	62.4
Publisher	46.9	92.9	61.2
Punc	100	98.2	99
Title	69.8	75.5	72.4
Vol	44.9	49.6	46.6
Micro avg	87.3	87.5	87.5
Macro avg	61.0	63.6	60.8
Weighted avg	89.4	87.5	88.2

TABLE 14: Overall token level performance metrices of model trained on 10,074 reference strings.

Performance Metrices	Scores
F1 score	87.7
Recall	87
Precision	87

TABLE 15: Entity level performance of the model that trained on 10,074 reference strings.

Class	Precision	Recall	F1 score
Author	97.2	95.5	96
CT	50.8	56.2	53
Date	90.5	33.8	48.5
Page	54.3	69.0	60.6
Publisher	44.7	93.6	59
Punc	100	100	100
Title	73.2	74.1	73.5
Vol	36.9	43.4	39.6
Micro avg	87.1	88.3	87.7
Macro avg	60.8	62.7	58.9
Weighted avg	89.7	88.3	88.1

TABLE 16: Overall token level performance metrices of model trained on 21,900 reference strings.

Performance Metrices	Scores
F1 score	92.2
Recall	92.5
Precision	92.5

TABLE 17: Entity level performance of the model that trained on 21,900 reference strings.

Class	Precision	Recall	F1 score
Author	98.7	97.7	98
CT	56.2	56.3	56.1
Date	84.5	82	83.1
Page	91.6	58.6	71.1
Publisher	79.6	86	81.7
Punc	100	100	100
Title	86.1	88.8	87.5
Vol	50.1	53.8	51.5
Micro avg	92.5	91.3	92.3
Macro avg	71.8	69.3	69.9
Weighted avg	93.7	91.9	92.6

TABLE 18: Overall token level performance metrices of model trained on 100,740 reference strings.

Performance Metrices	Scores
F1 score	93.5
Recall	93.6
Precision	93.6

TABLE 19: Entity level performance of the model which was trained on 100,740 reference strings.

Class	Precision	Recall	F1 score
Author	97.6	98.4	98
CT	59.2	61.1	60.2
Date	91.8	90.7	91.2
Page	95.1	90.7	91.2
Publisher	88.2	87.5	87.3
Punc	100	100	100
Title	86.7	88.9	87.8
Vol	58.4	52.3	54.9
Micro avg	93.5	93.5	93.4
Macro avg	75.1	72.1	73.6
Weighted avg	94.6	93.5	93.9

TABLE 20: Overall token level performance metrices of model trained on 219,000 reference strings.

Performance Metrices	Scores
F1 score	94.2
Recall	94.3
Precision	94.3

TABLE 21: Entity level performance of the model which was trained on 219,000 reference strings.

Class	Precision	Recall	F1 score
Author	98.9	98.8	98.8
CT	59.6	64.7	62.1
Date	86.9	88.7	87.7
Page	94.3	76.7	84.4
Publisher	89.8	94.3	91.4
Punc	100	100	100
Title	92.0	89.6	90.7
Vol	80.5	93.9	65.2
Micro avg	94.4	93.9	94.2
Macro avg	78.1	74.2	75.6
Weighted avg	95.3	93.9	94.5

TABLE 22: Overall token level performance metrices of model trained on 342,516 reference strings.

Performance Metrices	Scores
F1 score	93.6
Recall	93.9
Precision	93.9

TABLE 23: Entity level performance of the model which was trained on 342,516 reference strings.

Class	Precision	Recall	F1 score
Author	99.2	98.6	98.8
CT	57.7	65.1	61.3
Date	84.3	85.1	84.5
Page	93.6	66.5	77.5
Publisher	92.7	93.5	92.9
Punc	100	100	100
Title	89.6	90.9	90.2
Vol	77.9	52.5	62.5
Micro avg	93.9	93.5	93.6
Macro avg	77.2	72.5	74.1
Weighted avg	94.9	93.5	93.9

TABLE 24: Overall token level performance metrices of model trained on 1,027,548 reference strings.

Performance Metrices	Scores
F1 score	93.4
Recall	93.5
Precision	93.5

TABLE 25: Entity level performance of the model which was trained on 1,027,548 reference strings.

Class	Precision	Recall	F1 score
Author	98.7	98.4	98.5
CT	57.3	63.6	60.1
Date	86.7	85.4	86
Page	95.2	64.2	76.3
Publisher	84.9	95.6	89.6
Punc	100	100	100
Title	91.5	90.8	91.2
Vol	62.6	55.5	58.4
Micro avg	93.5	93.2	93.3
Macro avg	75.3	72.4	73.5
Weighted avg	94.7	93.2	93.7

TABLE 26: Performance comparison between Neural ParsCit and our model over CORA.

Model Class	Neural ParsCit			TransParsCit		
	Precision	Recall	F1	Precision	Recall	F1
Author	99.72	98.48	99.1	98.9	98.8	98.8
Date	99.37	98.5	98.93	86.9	88.7	87.7
Title	98.15	96.76	97.45	92	89.6	90.7
CT	93.95	93.54	93.73	59.6	64.7	62.1
Publisher	91.64	94.96	93.27	89.8	94.3	91.4
Page	98.34	97.74	98.04	94.3	76.7	84.4
Volume	94.31	94.41	94.36	80.5	93.9	65.2
Micro Avg	-	-	90.45	-	-	94.2
Macro Avg	-	-	95.68	-	-	75.6

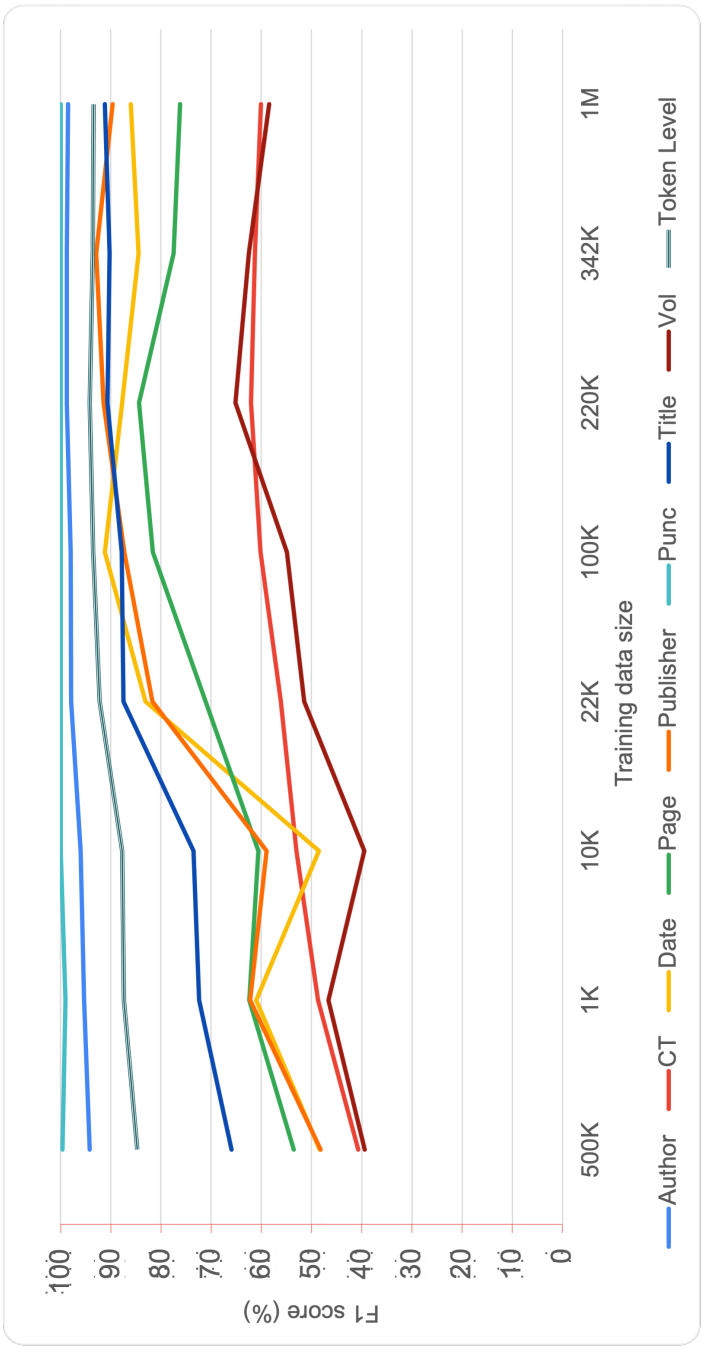

Fig.. 7. Performance of each class over the different dataset sizes shows the upward F1-score with an increment of the dataset.

papers, it was unclear whether the model worked equally well for citations in other domains. The GIANT dataset contains citation strings from multiple disciplines, so the model may be more robust when tested on citation strings in other domains. Further investigation should also be performed to reveal performance dependency on tokenizers and hyperparameters. The transformer architecture has superior performance in machine translation and natural language inference tasks. However, many recent NLP tasks found that it underperforms the BiLSTM-CRF in named entity recognition (NER) tasks. The citation parsing task has the same paradigm as the NER task. Therefore, if TransParsCit still underperforms Neural-ParsCit, we experiment with alternative tokenizers and hyperparameters. Our results may indicate that the transformer model is not suitable for the citation parsing task.

6.5 ERROR ANAYLSIS

Although TransParsCit achieved relatively good performance on entities such as Author, Publisher, Date, Title, and Page, CT and Vol are performing poorly. We investigate the errors by inspecting the automatically parsed citation strings by TransParsCit. Our inspection indicated that many errors are not because of the wrong predictions by TransParsCit, but of how the ground truth data was annotated. We categorize the errors into two types: true error and false error. The true error happens when a model truly makes the wrong prediction. The false error happens when the ground truth labels have incorrect labels or insignificant character offsets (e.g., a space).

The reference string below in our training data shows that volume is annotated as a numerical followed by the issue number enclosed in a pair of parentheses. The page numbers are annotated in a similar way, without including "pp." that may appear before the values. Below we show that corresponding fields were annotated in the CORA dataset.

> <author><family>Lomangino</family>, <given>Kevin</given></author> (<issued > <year>2008</year></issued>) <title>Does Caffeine Cause Miscarriages? Depends on the Study.</title> in <container-title>Clinical Nutrition INSIGHT</container-title>, <volume>34</volume>(<issue>4</issue>), pp. <page>6-7</page>. [online] Available from: <URL>http://dx.doi.org/10.1097/01.nmd.000031

6.5.1 FALSE ERROR - VOL MISPREDICTED AS ISSUE

There are cases where volume has two sets of digits like <volume> 2(1) </volume>. Both sets of digits are marked as the volume in the standard gold data. However, as shown

in Table 27 our model predicts 2 as volume and 1 as an issue, which is correct because, in our training data, the issue number typically stays inside a bracket followed by a volume. We categorize this as a false error, as shown below.

CORA annotation for Table 27:

> <author> Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. </author> <title> WorldWideWeb: the information universe. </title> <booktitle> Electronic Networking: Research, Applications and Policy, </booktitle> <volume> 2(1) </volume> <pages> 5258, </pages> <date> Spring 1992 </date>

On Table 27, *True Tag* refers to the labels from CORA sample and *True Tag* refers to the labels predicted by TransParsCit.

6.5.2 FALSE ERROR - CT PREDICTED AS O

In our training sample, the letter "In" is typically located before the CT tag and labeled as "O" (Other). However, in the CORA dataset, the journal tag includes the token "In". We categorize this as a false error. The example is presented on Table 28.

CORA Sample for Table 28:

> <author> Ancona, D., and Zucca, E. </author> <title> An algebraic approach to mixins and modularity. </title> <booktitle> In Proc. Conference on Algebraic and Logic Programming </booktitle> <location> (Berlin, </location> <date> 1996), </date>

This is a frequent error in our evaluation. Here is another example on Table 29:

CORA sample for 29:

> <author> Y. Sagiv and M. Yannakakis. </author> <title> Equivalence among relational expressions with the union and difference operators. </title> <journal> In J. ACM </journal> <volume> 27(4) </volume> <pages> pp. 633-655, </pages> <date> 1981. </date>

6.5.3 TRUE ERROR - CT PREDICTED AS VOL

The GIANT dataset annotates both journal and booktitle as CT. Sometimes, our model mistakenly predicts numbers at the end of CT as VOL, as shown in Table 30. We notice

TABLE 27: Model prediction for a CORA sample showing volume mispredicted as Issue.

Word	Predicted Tag	True Tag
Jean	I-AUTHOR	I-AUTHOR
-	B-PUNC	B-PUNC
Francois	I-AUTHOR	I-AUTHOR
Groff	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
and	I-AUTHOR	I-AUTHOR
Bernd	I-AUTHOR	I-AUTHOR
Pollermann	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
WorldWideWeb	B-TITLE	B-TITLE
:	B-PUNC	B-PUNC
the	I-TITLE	I-TITLE
information	I-TITLE	I-TITLE
universe	I-TITLE	I-TITLE
.	B-PUNC	B-PUNC
Electronic	B-CT	B-CT
Networking	I-CT	I-CT
:	B-PUNC	B-PUNC
Research	I-CT	I-CT
,	B-PUNC	B-PUNC
Applications	I-CT	I-CT
and	I-CT	I-CT
Policy	I-CT	I-CT
,	B-PUNC	B-PUNC
2	B-VOL	B-VOL
(	B-PUNC	B-PUNC
1	B-ISSUE	I-VOL
)	B-PUNC	B-PUNC
52	B-PAGE	B-PAGE
-	B-PUNC	B-PUNC
58	O	I-PAGE
Spring	B-CT	B-DATE
1992	B-DATE	I-DATE

TABLE 28: Model prediction for a CORA sample showing CT predicted as other (O).

Word	Predicted Tag	True Tag
Ancona	B-AUTHOR	B-AUTHOR
,	B-PUNC	B-PUNC
D	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
,	B-PUNC	B-PUNC
and	I-AUTHOR	I-AUTHOR
Zucca	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
E	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
An	B-TITLE	B-TITLE
algebraic	I-TITLE	I-TITLE
approach	I-TITLE	I-TITLE
to	I-TITLE	I-TITLE
mixins	I-TITLE	I-TITLE
and	I-TITLE	I-TITLE
modularity	I-TITLE	I-TITLE
.	B-PUNC	B-PUNC
In	O	B-CT
Proc	B-CT	I-CT
.	B-PUNC	B-PUNC
Conference	I-CT	I-CT
on	I-CT	I-CT
Algebraic	I-CT	I-CT
and	I-CT	I-CT
Logic	I-CT	I-CT
Programming	I-CT	I-CT
1996	B-DATE	B-DATE
)	B-PUNC	B-PUNC
,	B-PUNC	B-PUNC

TABLE 29: Another example of showing CT predicted as other (O).

Word	Predicted Tag	True Tag
Y	B-AUTHOR	B-AUTHOR
.	B-PUNC	B-PUNC
Sagiv	I-AUTHOR	I-AUTHOR
and	I-AUTHOR	I-AUTHOR
M	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Yannakakis	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Equivalence	B-TITLE	B-TITLE
among	I-TITLE	I-TITLE
relational	I-TITLE	I-TITLE
expressions	I-TITLE	I-TITLE
with	I-TITLE	I-TITLE
the	I-TITLE	I-TITLE
union	I-TITLE	I-TITLE
and	I-TITLE	I-TITLE
difference	I-TITLE	I-TITLE
operators	I-TITLE	I-TITLE
.	B-PUNC	B-PUNC
In	O	B-CT
J	B-CT	I-CT
.	B-PUNC	B-PUNC
ACM	I-CT	I-CT
27	B-VOL	B-VOL
(	B-PUNC	B-PUNC
4	B-ISSUE	I-VOL
)	B-PUNC	B-PUNC
633	B-PAGE	B-PAGE
-	B-PUNC	B-PUNC
655	I-PAGE	I-PAGE
1981	B-DATE	B-DATE
.	B-PUNC	B-PUNC

that it happens because the training reference strings volumes typically come after container-titles. We categorize this as a true error.

On following CORA sample (see Table 30) *95* comes at the end of the journal, which presents the year. Although the dates in our training set are a 4-digit numeric values

> <author> K. H. Wolf, K. Froitzheim and P. Schulthess, </author> <date> (1995) </date> <title> Multimedia Application Sharing in a Heterogeneous Environment, </title> <journal> ACM Multimedia 95, </journal> <pages> Pages 57-64. </pages>

6.5.4 TRUE ERROR - TITLE PREDICTED AS CT

Table 31 is an example in which the title is mistakenly predicted as CT, which is a true error.

CORA annotation for Table 31:

> <author> R. Lipsett, C. Schaefer, C. Ussery, </author> <title> VHDL: Hardware Description and Design, </title> <publisher> Kluwer Academic Publishers, </publisher> <date> 1989. </date>

6.5.5 TRUE ERROR - CT MISPREDICTED AS PAGE

In the example in Table 32, "CT" is predicted as "PAGE" by TransParsCit. However, the b́ooktitleín the CORA sample has a digit at the end. This is a true error.

CORA sample for Table 32:

> <author> Thrun, T., Schwartz, A. </author> <date> (1995) </date> <title> F̈inding Structure in Reinforcement Learning,̈</title> <booktitle> in Advances in Neural Information Processing Systems, 7. </booktitle> <location> San Mateo: </location> <publisher> Morgan Kaufmann </publisher>

TABLE 30: Model prediction for a CORA sample showing CT mispredicted as VOL.

Word	Predicted Tag	True Tag
K	B-AUTHOR	B-AUTHOR
.	B-PUNC	B-PUNC
H	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Wolf	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
K	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Froitzheim	I-AUTHOR	I-AUTHOR
and	I-AUTHOR	I-AUTHOR
P	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Schulthess	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
(	B-PUNC	B-PUNC
1995	B-DATE	B-DATE
)	B-PUNC	B-PUNC
Multimedia	B-TITLE	B-TITLE
Application	I-TITLE	I-TITLE
Sharing	I-TITLE	I-TITLE
in	I-TITLE	I-TITLE
a	I-TITLE	I-TITLE
Heterogeneous	I-TITLE	I-TITLE
Environment	I-TITLE	I-TITLE
,	B-PUNC	B-PUNC
ACM	B-CT	B-CT
Multimedia	I-CT	I-CT
95	B-VOL	I-CT
,	B-PUNC	B-PUNC
57	B-PAGE	B-PAGE
-	B-PUNC	B-PUNC
64	I-PAGE	I-PAGE

TABLE 31: Model prediction for a CORA sample showing Title predicted as CT.

Word	Predicted Tag	True Tag
R	B-AUTHOR	B-AUTHOR
.	B-PUNC	B-PUNC
Lipsett	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
C	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Schaefer	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
C	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
Ussery	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
VHDL	B-CT	B-TITLE
:	B-PUNC	B-PUNC
Hardware	I-CT	I-TITLE
Hardware	I-CT	I-TITLE
Description	I-CT	I-TITLE
and	I-CT	I-TITLE
Design	I-CT	I-TITLE
,	B-PUNC	B-PUNC
Kluwer	B-PUBLISHER	B-PUBLISHER
Academic	I-PUBLISHER	I-PUBLISHER
Publishers	I-PUBLISHER	I-PUBLISHER
,	B-PUNC	B-PUNC
1989	B-DATE	B-DATE
.	B-PUNC	B-PUNC

TABLE 32: Model prediction for a CORA sample showing CT mispredicted as Page.

Word	Predicted Tag	True Tag
Thrun	B-AUTHOR	B-AUTHOR
,	B-PUNC	B-PUNC
T	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
,	B-PUNC	B-PUNC
Schwartz	I-AUTHOR	I-AUTHOR
,	B-PUNC	B-PUNC
A	I-AUTHOR	I-AUTHOR
.	B-PUNC	B-PUNC
(	B-PUNC	B-PUNC
1995	B-DATE	B-DATE
)	B-PUNC	B-PUNC
"	B-PUNC	B-PUNC
Finding	B-TITLE	B-TITLE
Structure	I-TITLE	I-TITLE
in	I-TITLE	I-TITLE
Reinforcement	I-TITLE	I-TITLE
Learning	I-TITLE	I-TITLE
,	B-PUNC	B-PUNC
"	B-PUNC	B-PUNC
in	O	B-CT
Advances	B-CT	I-CT
in	I-CT	I-CT
Neural	I-CT	I-CT
Information	I-CT	I-CT
Processing	I-CT	I-CT
Systems	I-CT	I-CT
,	B-PUNC	B-PUNC
7	B-PAGE	I-CT
.	B-PUNC	B-PUNC
Morgan	B-PUBLISHER	B-PUBLISHER
Kaufmann	I-PUBLISHER	I-PUBLISHER

CHAPTER 7

DISCUSSION

Here we discuss some issues that are related to our work.

7.1 LIMITATIONS OF THIS WORK

TransParsCit can not work with converted texts from OCR images. It can not deal with misspelled or character-breaking reference strings.

We only evaluated our model using the CORA dataset, which is dominated by reference strings from the Computer Science domain. However, in order to get the full picture of the generic performance, we need to evaluate TransParsCit with FLUX-CiM, ICONIP, GROBID, etc.

TransParsCit was trained only on ten different classes. It can not determine some classes like *Editor*, *Location*, *Institute* which were present in CORA dataset.

7.2 TRANSPARSCIT FOR MULTILINGUAL REFERENCE STRINGS

We have not tested, but we can assume that the current model can not work with a different language out of the box since it was trained on word-level features in English. For parsing a reference string, not in English, we may need to use translation first to get the strings into English. This may require integrating a third-party software or calling an API during the reference parsing stage. However, if embeddings for other languages and reference strings are available, TransParsCit can also be useful for multilingual tasks.

CHAPTER 8

CONCLUSIONS AND FUTURE WORKS

Our approach to parsing reference strings uses deep learning. The most significant benefit of deep learning is that it can induce features on its own, which is fitting for parsing reference strings. For TransParsCit, we used core components from Transformer Encoder, which helps to understand better the words in the reference strings with CRF for prediction. We also developed a custom tokenizer that retains the word orders and punctuations to comprehend a reference string better while training.

We trained our models on the GIANT dataset and evaluated our models over CORA, and achieved up to 94.2% F1 score on our best model. We also performed a detailed error analysis and found that the actual performance by taking account of false errors could be better than reported. We believe the performance can be further improved following some steps -

In-domain WE training: Currently, we only use pre-trained embeddings provided by Google. Authors on Neural ParsCit trained in-domain word embeddings using the word2vec toolkit on 4.3M reference strings from the Association of Computing Machinery (ACM) and observed performance gain over the augmented word embeddings.

Parallel Training: We use a single GPU to train our models. If the dataset is too large (1M), it takes a couple of hours to more than a day for the training. Utilizing multiple GPUs, our model may get the advantage of the whole GIANT dataset.

Features: We train our models by lowering the words and only considering the embeddings as features. Considering casing information like what Neural ParsCit did may provide better performance.

Transformer components: We use only one layer of transformer encoder in our architecture. Performance can be improved by stacking multiple transformer layers.

GitHub repository for the project: `https://github.com/lamps-lab/Citation-Parser`.

REFERENCES

[1] E. Garfield and R. K. Merton, *Citation indexing: Its theory and application in science, technology, and humanities*, vol. 8. Wiley New York, 1979.

[2] F. Radicchi, S. Fortunato, B. Markines, and A. Vespignani, "Diffusion of scientific credits and the ranking of scientists," *Physical Review E*, vol. 80, no. 5, p. 056103, 2009.

[3] F. Radicchi, S. Fortunato, and C. Castellano, "Universality of citation distributions: Toward an objective measure of scientific impact," *Proceedings of the National Academy of Sciences*, vol. 105, no. 45, pp. 17268–17272, 2008.

[4] D. Tkaczyk, A. Collins, P. Sheridan, and J. Beel, "Machine learning vs. rules and out-of-the-box vs. retrained: An evaluation of open-source bibliographic reference and citation parsers," in *Proceedings of the 18th ACM/IEEE on joint conference on digital libraries*, pp. 99–108, 2018.

[5] C.-C. Chen, K.-H. Yang, C.-L. Chen, and J.-M. Ho, "Bibpro: A citation parser based on sequence alignment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 2, pp. 236–250, 2010.

[6] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[7] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[8] I. G. Councill, C. L. Giles, and M.-Y. Kan, "Parscit: an open-source crf reference string parsing package.," in *LREC*, vol. 8, pp. 661–667, 2008.

[9] A. Prasad, M. Kaur, and M.-Y. Kan, "Neural parscit: a deep learning-based reference string parser," *International journal on digital libraries*, vol. 19, no. 4, pp. 323–337, 2018.

[10] "Citeseerx." `https://citeseerx.ist.psu.edu/`. (Accessed on 04/07/2022).

[11] A. Medaille, "Mendeley: www. mendeley. com," *Public Services Quarterly*, vol. 6, no. 4, pp. 360–362, 2010.

[12] M. Grennan, M. Schibel, A. Collins, and J. Beel, "Giant: The 1-billion annotated synthetic bibliographic-reference-string dataset for deep citation parsing.," in *AICS*, pp. 260–271, 2019.

[13] A. McCallum. `https://people.cs.umass.edu/~mccallum/data.html`. (Accessed on 04/07/2022).

[14] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in *Proceedings of the third ACM conference on Digital libraries*, pp. 89–98, 1998.

[15] E. Kunnas, "Pdf structure and syntactic analysis for meta-data extraction and tagging: https://code.google.com/p/pdfssa4met/-eliask/pdfssa4met," in *Proceedings of the third ACM conference on Digital libraries*, pp. 89–98, 2018.

[16] X. Zhang, J. Zou, D. X. Le, and G. R. Thoma, "A structural svm approach for reference parsing," in *2010 Ninth International Conference on Machine Learning and Applications*, pp. 479–484, IEEE, 2010.

[17] D. Heckmann, A. Frank, M. Arnold, P. Gietz, and C. Roth, "Citation segmentation from sparse & noisy data: A joint inference approach with markov logic networks," *Digital Scholarship in the Humanities*, vol. 31, no. 2, pp. 333–356, 2016.

[18] E. Cortez, A. S. da Silva, M. A. Gonçalves, F. Mesquita, and E. S. de Moura, "A flexible approach for extracting metadata from bibliographic citations," *Journal of the American Society for Information Science and Technology*, vol. 60, no. 6, pp. 1144–1158, 2009.

[19] A. Constantin, S. Pettifer, and A. Voronkov, "Pdfx: fully-automated pdf-to-xml conversion of scientific literature," in *Proceedings of the 2013 ACM symposium on Document engineering*, pp. 177–180, 2013.

[20] E. Cortez, A. S. da Silva, M. A. Gonçalves, F. Mesquita, and E. S. de Moura, "Flux-cim: flexible unsupervised extraction of citation metadata," in *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pp. 215–224, 2007.

[21] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1, pp. 107–136, 2006.

[22] M.-Y. Day, R. T.-H. Tsai, C.-L. Sung, C.-C. Hsieh, C.-W. Lee, S.-H. Wu, K.-P. Wu, C.-S. Ong, and W.-L. Hsu, "Reference metadata extraction using a hierarchical knowledge representation framework," *Decision Support Systems*, vol. 43, no. 1, pp. 152–167, 2007.

[23] M. Jewell, "Paratools reference parsing toolkit - version 1.0 released." `http://www.dlib.org/dlib/february03/02inbrief.html#JEWELL`, 2003. (Accessed on 04/08/2022).

[24] K. Seymore, A. McCallum, R. Rosenfeld, *et al.*, "Learning hidden markov model structure for information extraction," in *AAAI-99 workshop on machine learning for information extraction*, pp. 37–42, 1999.

[25] E. Hetzner, "A simple method for citation metadata extraction using hidden markov models," in *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, pp. 280–284, 2008.

[26] V. Borkar, K. Deshmukh, and S. Sarawagi, "Automatic segmentation of text into structured records," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 175–186, 2001.

[27] Q. Zhang, Y.-G. Cao, and H. Yu, "Parsing citations in biomedical articles using conditional random fields," *Computers in biology and medicine*, vol. 41, no. 4, pp. 190–194, 2011.

[28] D. Tkaczyk, P. Szostek, P. J. Dendek, M. Fedoryszak, and L. Bolikowski, "Cermine–automatic extraction of metadata and references from scientific literature," in *2014 11th IAPR International Workshop on Document Analysis Systems*, pp. 217–221, IEEE, 2014.

[29] M. Romanello, F. Boschetti, and G. Crane, "Citations in the digital library of classics: extracting canonical references by using conditional random fields," in *Proceedings of the 2009 Workshop on Text and Citation Analysis for Scholarly Digital Libraries (NLPIR4DL)*, pp. 80–87, 2009.

[30] D. Rodrigues Alves, G. Colavizza, and F. Kaplan, "Deep reference mining from scholarly literature in the arts and humanities," *Frontiers in Research Metrics and Analytics*, p. 21, 2018.

[31] T. Okada, A. Takasu, and J. Adachi, "Bibliographic component extraction using support vector machines and hidden markov models," in *International Conference on Theory and Practice of Digital Libraries*, pp. 501–512, Springer, 2004.

[32] "Medline/pubmed data documentation." `https://www.nlm.nih.gov/databases/download/pubmed_medline_documentation.html`. (Accessed on 04/07/2022).

[33] F. Peng and A. McCallum, "Information extraction from research papers using conditional random fields," *Information processing & management*, vol. 42, no. 4, pp. 963–979, 2006.

[34] P. Yin, M. Zhang, Z. Deng, and D. Yang, "Metadata extraction from bibliographies using bigram hmm," in *International Conference on Asian Digital Libraries*, pp. 310–319, Springer, 2004.

[35] B. Ojokoh, M. Zhang, and J. Tang, "A trigram hidden markov model for metadata extraction from heterogeneous references," *Information Sciences*, vol. 181, no. 9, pp. 1538–1551, 2011.

[36] P. Lopez, "Grobid: Combining automatic bibliographic data recognition and term extraction for scholarship publications," in *International conference on theory and practice of digital libraries*, pp. 473–474, Springer, 2009.

[37] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[38] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," *arXiv preprint arXiv:1603.01354*, 2016.

[39] D. Tkaczyk, P. Sheridan, and J. Beel, "Parsrec: Meta-learning recommendations for bibliographic reference parsing," *arXiv preprint arXiv:1808.09036*, 2018.

[40] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[42] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[44] J. Alammar, "The illustrated transformer." `https://jalammar.github.io/illustrated-transformer/`, 06 2018. (Accessed on 04/17/2022).

[45] "word2vec." `https://code.google.com/archive/p/word2vec/`. (Accessed on 04/07/2022).

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

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[48] "Crossref." `https://www.crossref.org/about/`. (Accessed on 04/07/2022).

[49] "Official repository for citation style language (csl) citation styles." `https://github.com/citation-style-language/styles`. (Accessed on 04/10/2022).

[50] F. Bennett, "A javascript implementation of the citation style language (csl)." `https://citeproc-js.readthedocs.io/en/latest/`, 2016. (Accessed on 04/07/2022).

[51] "Pytorch." `https://pytorch.org/`. (Accessed on 04/07/2022).

VITA

MD Sami Uddin
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

e-mail: rayan_sami@outlook.com

Education

Bachelor of Science in Computer Science (16 February, 2017)
Military Institute of Science and Technology, Dhaka, Bangladesh

Employment

2021 - 2022 – Graduate Research Assistant
Old Dominion University Web Science and Digital Libraries Research Group

2020 - 2021 – Graduate Teaching Assistant
Old Dominion University Computer Science Department

2017 - 2019 – Systems Developer
Bluebnc, Dhaka, Bangladesh

Selected Publications

S. Uddin, B. Banerjee, J. Wu, W. A. Ingram, E. A. Fox, "Building A large collection of multi-domain electronic theses and dissertations", 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 2021.

H.-N. Lee, S. Uddin, V. Ashok, "TableView: Enabling Efficient Access to Web Data Records for Screen-Magnifier Users", in The 22nd International ACM SIGACCESS Conference on Computers and Accessibility, 2020, pp. 1–12.

An updated list of publications is available at https://scholar.google.com/citations?user=9GJoF4AAAAAJ&hl=en

Typeset using LaTeX.