SciWING – A Software Toolkit for Scientific Document Processing

Abhinav Ramesh Kashyap
National University of Singapore
abhinav@comp.nus.edu.sg

Min-Yen Kan
National University of Singapore
knmnyn@comp.nus.edu.sg

Abstract

We introduce SciWING, an open-source software toolkit which provides access to state-of-the-art pre-trained models for scientific document processing (SDP) tasks, such as citation string parsing, logical structure recovery and citation intent classification. Compared to other toolkits, SciWING follows a full neural pipeline and provides a Python interface for SDP. When needed, SciWING provides fine-grained control for rapid experimentation with different models by swapping and stacking different modules. Transfer learning from general and scientific documents specific pre-trained transformers (i.e., BERT, SciBERT, etc.) can be performed. SciWING incorporates ready-to-use web and terminal-based applications and demonstrations to aid adoption and development. The toolkit is available from http://sciwing.io and the demos are available at http://rebrand.ly/sciwing-demo[1].

1 Introduction

Automated scientific document processing (SDP) deploys natural language processing (NLP) on scholarly articles. As scholarly articles are long-form, complex documents with conventional structure and cross-reference to external resources, they require specialized treatment and have specialized tasks. Representative SDP tasks include parsing embedded reference strings (Prasad et al., 2018; Thai et al., 2020); identifying the importance, sentiment and provenance for citations (Cohan et al., 2019; Su et al., 2019); identifying logical sections and markup (Luong et al., 2012); parsing of equations, figures and tables (Clark and Divvala, 2016); and article summarization (Qazvinian and Radev, 2008; Qazvinian et al., 2013; Cohan and Goharian, 2015; Cohan et al., 2018; Cohan and Goharian, 2018). SDP tasks, in turn, help downstream systems and assist scholars in finding relevant documents and manage their knowledge discovery and utilization workflows. Next-generation introspective digital libraries such as Semantic Scholar (Ammar et al., 2018), Google Scholar and Microsoft Academic have begun to incorporate such services.

[1]Watch our demo video at https://bit.ly/sciwing-video

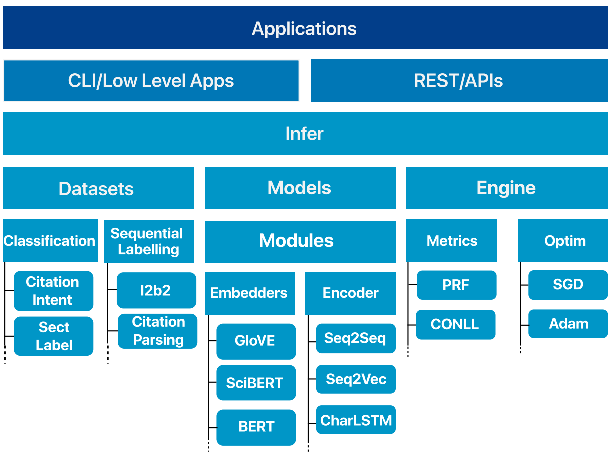


Figure 1: SciWING Components: Text classification and Sequence labelling **Datasets**, **Models** composed from low-level **Modules**, and **Engine** to train and record experiment parameters. **Infer** middleware does the inference and high-level functionality (e.g., developing APIs; low-level and web applications).

While NLP, in general, has seen tremendous progress with the introduction of neural network architectures and general toolkits and datasets to leverage them, their deployment for SDP is still limited. Over the past few years, many open-source software packages have accelerated the development of state-of-the-art (SOTA) NLP models. However, these frameworks have a few limitations concerning SDP. First, most are general purpose frameworks aimed at producing SOTA models for natural language understanding tasks or specific domains such as biomedicine. Sec-

Framework	Pretrained models	SOTA	Neural first	Extensible	Language/Framework
SciSpaCy	✔				Py
AllenNLP	✔		✔	✔	Py(Torch)
FLAIR	✔		✔	✔	Py(Torch)
Grobid	✔				Java
SciWING	✔	✔	✔	✔	Py(Torch)

Table 1: Comparison of SciWING with popular frameworks. **Pretrained models**: availability of pretrained models. **SOTA**: state-of-the-art models for SDP **Neural-Networks first** - framework supports end-end neural networks. **Extensible**: flexibility to allow new datasets and architectures

ond, they do not provide immediately deployable, SOTA models for SDP. Most provide limited or no means for researchers to train models on their datasets, or experiment with model architectures.

A key barrier to entry is accessibility: a non-trivial level of expertise in NLP and machine learning is a prerequisite. Practitioners who wish to deploy SDP on their field's literature may lack knowledge and motivation to learn it for the sake of deployment. Thus there is a clear need for a toolkit that unifies different efforts and provides access to pre-trained, SOTA models for SDP, while also allowing researchers to experiment with models rapidly to create deployable applications.

We introduce SciWING to address this need with respect to other frameworks (cf. Table 1). Built on top of PyTorch, it provides access to neural network models trained for a growing number of SDP tasks which practitioners can easily deploy on their documents. For researchers, these models serve as baselines for experimentation and further construction of complex architectures in a modular manner by swapping or composing different neural network modules. SciWING also allows researchers to declare models in a configuration file without having to write programming code.

SciWING is MIT licensed. All its pre-trained models and freely available datasets are available for easy download. The package runs on Python 3.7 and can be easily installed from Python Packaging Index (PyPI) using `pip install sciwing`. For researchers aiming to further develop SciWING, we provide installation tools that set up the system, alongside documentation.

2 System Overview

Our view is that SDP-specific considerations are best embodied as an abstract layer over existing NLP frameworks. SciWING incorporates AllenNLP, the generic NLP pipeline (Gardner et al., 2017), developing models on top of it when necessary, while using the transformers package (Wolf et al., 2019) to enable transfer learning via its pre-trained general-purpose representations such as BERT (Devlin et al., 2019) and SDP-specific ones like SciBERT (Beltagy et al., 2019). Fig. 1 shows SciWING's Dataset, Model and Engine components facilitating flexible re-configuration. We now describe these components.

Datasets: There are many challenges for the researcher-practitioner to experiment with different SDP tasks. First, researchers are often dealt with the challenge of handling various formats of the datasets: for reference string parsing, the CoNLL format is most common; for text classification, CSV is most common. SciWING enables reading of dataset files in different formats and also facilitates the download of open datasets using command-line interfaces. For example, `sciwing download data --task scienceie` downloads the official openly available dataset for the ScienceIE task.

Current methods for pre-processing are cumbersome and error-prone. Processing can become complex when different models require different tokenisation and numericalisation methods. SciWING unifies these various input formats through a pipeline of pre-processing, tokenisation and numericalisation, via `Tokenisers` and `Numericalisers`. SciWING also handles batching and padding of examples.

Models: The below two subcomponents are combined to build an instance of a neural network model – which are PyTorch classes.

Embedders: Modern NLP models represent natural language tokens as continuous vectors – embeddings. SciWING abstracts this concept via `Embedders`. Generic (non-SDP specific) embeddings such as GlovE (Pennington et al., 2014) are natively provided. Tokens in scientific documents can benefit from special attention, as most are missing from pre-trained embeddings. SciWING includes task-specific trained embeddings for reference strings (Prasad et al., 2018). SciWING also supports contextual word embeddings: ELMo (Peters et al., 2018), BERT (Devlin et al., 2019), SciBERT (Beltagy et al., 2019) etc. State-of-the-art models are easily built by concatenating multiple representations, via SciWING's `ConcatEmbedders` module. For eg., word and character embeddings are combined in NER models (Lample et al., 2016), multiple contextual word embeddings are combined in various clinical and BioNLP tasks (Zhai et al., 2019).

Neural Network Encoders: SciWING consists of commonly-used neural network components that can be composed to form neural architectures for different tasks. For example in text classification, encoding input sentence as a vector using an LSTM is a common task (SciWING's `seq2vecencoder`). Another common operation is obtaining a sequence of hidden states for a set of tokens, often used in sequence labelling tasks and SciWING's `Lstm2seq` achieves this. Further, it also includes attention based modules.

SciWING builds in generic linear classification and sequence labelling with CRF heads that can be attached to the encoders to build models. It provides pretrained state-of-the-art models for particular SDP tasks that work out-of-the-box for prediction or which can be further fine-tuned.

Engine: SciWING handles all the boilerplate code to train the model, monitor the loss and metrics, check-pointing parameters at different stages of training, validation and testing. It helps researchers adopt best practices, such as clipping gradient-norms, saving and deploying best performing models. SciWING's experimentation framework is flexible and users have the flexibility of customizing the following:

Optimisers: SciWING supports all the optimisers supported by PyTorch, and various learning rate schedulers that dynamically manage learning rates based on validation performance.

Experiment Logging: SciWING adopts current best practices in leveraging logging tools to monitor and manage experiments. SciWING writes logs for every experiment run and facilitates cloud-based experiment logging and corresponding charting of relevant metrics via the third-party API service of *Weights and Biases*[2], with the integration of alternative logging services on the way.

Metrics: Different SDP tasks require their respective metrics. SciWING abstracts a separate `Metrics` module to select appropriate metrics for each task. SciWING includes `PrecisionRecallFMeasure` suitable for text classification tasks, and `TokenClassificationAccuracy` and the official CONLL2003 shared task evaluation metric [3] suitable for sequence labelling.

With these components given, SciWING's **Inference** middleware provides clear abstractions to perform inference once models are trained. The layer runs predictions on the test dataset, user inputs and files. Such abstractions also act as an interface for the development of upstream REST APIs and command-line applications.

2.1 Configuration using TOML files

We have seen the flexible architecture of SciWING. This enables us to equip the users with various functionalities. One of them is a defining feature of SciWING which allows the use of a declarative TOML configuration file. This enables users to declare dataset, model architectures and experiment hyper-parameters in a single place. SciWING parses the TOML file and creates appropriate instances of the dataset, model and engine to run experiments.

A simple configuration file for reference string parsing along with its equivalent model declaration in Python is shown in the listings below. The class declaration and configuration file have a one-to-one correspondence. As deep learning models are made of multiple modules, SciWING automatically instantiates these submodules as needed. SciWING constructs a Directed Acyclic Graph (DAG) from the model definition to achieve this. The DAG's topological ordering instantiates the different submodules to form the final model.

```
[model]
    class="SimpleClassifier"
    encoding_dimension=300
    num_classes=23
    classification_layer_bias=true
```

[2] www.wandb.com.

[3] Based on the official *conlleval* script from CoNLL.

```
    [model.encoder]
        emb_dim=300
        class="BOW_Encoder"
        dropout_value=0.5
        aggregation_type="sum"
        [[model.encoder.embedder]]
        class="VanillaEmbedder"
        embed="word_vocab"
        freeze=False
```

```
class SimpleClassifier(nn.Module):
    def __init__(
        self,
        encoder: nn.Module,
        encoding_dim: int,
        num_classes: int,
        classification_layer_bias: bool)
```

2.2 Command Line Interface

Qualitatively analyzing the results of the model by drilling down to certain training and development instances can be telling and help to diagnose performance issues. SciWING's architecture enables it to provide an interactive inspection of the model for this reason through a command-line interface (CLI). Consider the task of reference string parsing: the confusion matrix for the different classes can be displayed through the provided CLI utility, which also allows finer-grained introspection of (Precision, Recall, F-measure) metrics and the viewing of error instances where one class is confused for another. For example, `sciwing interact neural-parscit` provides introspection utilities for the pre-trained reference string parsing model. Such introspection utilities are also available for other pre-trained models.

SciWING provides commands to run experiments from the configuration file, aiding replication. For example, experiments declared in a file named `experiment.toml`, can be run with the command `sciwing run experiment.toml`. SciWING then saves the best model. Inference is then trivially invoked via `sciwing test experiment.toml` which deploys the best model against the test dataset and displays the resultant metrics.

2.3 End User Interfaces

API service enables the development of various graphical user interfaces. SciWING uses its *Infer* layer and exposes APIs for various tasks including reference string parsing, citation intent classification, extracting abstracts and logical sections of a research articles, identifying entities in clinical notes, using `fastapi`[4]. The API enables the following application families downstream:

Task	SciWING	Best
Reference String Parsing	88.44	90.45
ScienceIE	49.9	48.01
Logical Structure Recovery	73.2	–
Citation Intent Classification	82.16	82.6
I2B2 NER	85.83	86.23

Table 2: SciWING's SDP task performance, compared against other comparable models. Scores are macro F_1.

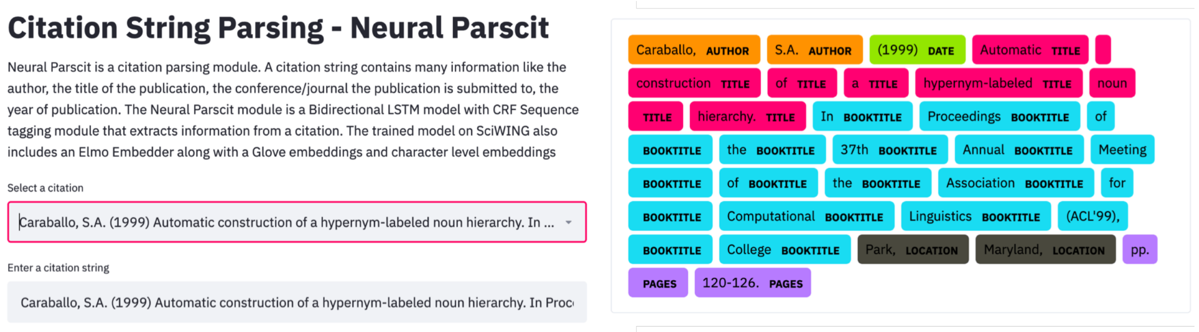
- **Web Demonstrations**: To provide quick access to predictions from state-of-the-art models, fulfilling one key aim of SciWING, we develop an interactive demo using streamlit[5]. An instance of the demo is available at `http://rebrand.ly/sciwing-demo`. Pre-specified data can be chosen or user data can be entered and quickly processed using the distributed models (Figure 2). Both API services and demos can also be run by installing SciWING locally.
- **Programmatic Interfaces** in SciWING provisions more advanced usage. Users can make predictions for data stored in .pdf or text files. For example, to parse a text file's citations, SciWING provides a `NeuralParscit` class that has methods to parse all the strings in a file, storing them in a new file. Such a programmatic interface helps the practitioner make predictions easily.

3 Tasks

SciWING prepackages models for various SDP tasks. The examples demonstrate how to use the framework effectively. These models have performance close or comparable to state-of-the-art models (Table 2). They are production-ready, but also can be used as baselines for further research.

- **Reference String Parsing** assigns one of 13 classes to tokens of a reference string that correspond with a in-document citation: *author*, *journal* and *year* of publication, among them. Neural sequence labelling models, combining a bidirectional LSTM with CRF currently yield top results (Prasad et al., 2018). SciWING's distributed model implements the same model architecture,

[4]https://fastapi.tiangolo.com/
[5]http://www.streamlit.io


Citation String Parsing - Neural Parscit

Neural Parscit is a citation parsing module. A citation string contains many information like the author, the title of the publication, the conference/journal the publication is submitted to, the year of publication. The Neural Parscit module is a Bidirectional LSTM model with CRF Sequence tagging module that extracts information from a citation. The trained model on SciWING also includes an Elmo Embedder along with a Glove embeddings and character level embeddings

Select a citation

Caraballo, S.A. (1999) Automatic construction of a hypernym-labeled noun hierarchy. In ...

Enter a citation string

Caraballo, S.A. (1999) Automatic construction of a hypernym-labeled noun hierarchy. In Proc

Caraballo, AUTHOR S.A. AUTHOR (1999) DATE Automatic TITLE construction TITLE of TITLE a TITLE hypernym-labeled TITLE noun TITLE hierarchy. TITLE In BOOKTITLE Proceedings BOOKTITLE of BOOKTITLE the BOOKTITLE 37th BOOKTITLE Annual BOOKTITLE Meeting BOOKTITLE of BOOKTITLE the BOOKTITLE Association BOOKTITLE for BOOKTITLE Computational BOOKTITLE Linguistics BOOKTITLE (ACL'99), BOOKTITLE College BOOKTITLE Park, LOCATION Maryland, LOCATION pp. PAGES 120-126. PAGES

Figure 2: Sample SciWING's demonstration (https://bit.ly/sciwing-demo) for reference string parsing model, where input (l) is then classified into 13 output classes (r). We utilize the displaCy visualization toolkit (www.spacy.io) and streamlit (www.streamlit.io).

but adding ELMo embeddings.

- **ScienceIE:** identifies typed keyphrases, originally from chemical documents: *Task* keyphrases that denote the end task or goal, *Material* keyphrases indicate any chemical, and *Dataset* that is being used by the scientific work and the process includes any scientific model or algorithm. The state-of-the-art system from 2017 includes a word and character embeddings and a bidirectional LSTM with CRF and uses language model (LM) embeddings (Ammar et al., 2017). SciWING includes a reference implementation without using LM embeddings and the results are comparable.

- **Logical Structure Recovery** identifies the logical sections of a document: introduction, related work, methodology, and experiments. This drives the relevant, targeted text to downstream tasks such as summarization, citation intent classification, among others. Currently, there are no neural network methods for this task, so SciWING's models can serve as strong baselines.

- **Citation Intent Classification** identifies the purpose of a citation. Some citations refer to another work for *background* knowledge, a few to a related work's *results* and others to *compare and contrast* their methods or results. Such citation intents get used in Semantic Scholar[6]. We train a bi-LSTM with ELMo on the Scicite dataset and achieve an F-score of 82.16 (compare verses 82.6 from Cohan et al. (2019)).

- **I2B2 Named Entity Recognition** identifies three kinds of entities from clinical notes; problems (e.g., a disease), treatments (e.g., a drug) and tests (e.g., diagnostic procedures). We use a similar model of Bi-LSTM CRF with ELMo embeddings, achieving 85.83 F_1.

[6]www.semanticscholar.org

4 Use Cases

SciWING caters to both use cases of practitioners looking to deploy pre-trained models as well as researchers looking to refine model architectures and perform fine-tuning domain adaptation on top of state-of-the-art contextual word embedding models. We now examine both use cases.

4.1 Using Pretrained Reference String Parser

SciWING provisions out-of-the-box access to pretrained models for direct deployment. Citation string parsing can be deployed with just a few lines of code as shown below.

```
from sciwing.models.neural_parscit
    import NeuralParscit

# instantiate the best model for
    reference string parsing.
neural_parscit = NeuralParscit()

# predict for some reference
neural_parscit.predict_for_text("
    reference")

# predict for a file containing one
    reference per line
neural_parscit.predict_for_file(/
    path/to/file)
```

4.2 Building a Reference String Parser from Scratch

State-of-the-art models (SOTA) are built by stacking up multiple components. We illustrate how to construct such SciWING models, building up to such SOTA model by simple modifications. Such simple step-by-step building of models facilitates ablation which is a common part to empirical studies.

1. **Bi-LSTM tagger**: Our base model is a bi-LSTM with a GLoVE embedder. Every input token is classified into one of 13 different classes.

```
# initialize a word embedder
word_embedder = WordEmbedder(
    embedding_type = "glove_6B_100")

# initialize a LSTM2Seq encoder
lstm2seqencoder = LSTM2SeqEncoder(
    embedder = word_embedder,
    hidden_dim = 100,
    bidirectional = True)

# initialize a tagger without CRF
model = SimpleTagger(
    rnn2seqencoder = lstm2seqencoder
    , encoding_dim = 200)
```

2. Bi-LSTM Tagger with CRF: We then make a single modification to the above code, swapping the simple tagger with one that uses a CRF.

```
...

# an RNN tagger with CRF on top
model = RnnSeqCrfTagger(
    rnn2seqencoder = lstm2seqencoder
    , encoding_dim = 200
)
```

3. Bi-LSTM tagger with character and ELMo Embeddings: We modify the code to include a bidirectional LSTM character embedder. We use the ConcatEmbedders module to create the final word embeddings (Line 16), which concatenates the character embeddings with those from the previous word embedding and a pretrained ELMo contextual word embedding. This final model is the provisioned model for the reference string parsing task provided in SciWING.

```
1  ...
2  word_embedder = WordEmbedder(
3     embedding_type = "glove_6B_100"
4  )
5
6  # LSTM character embedder
7  char_embedder = CharEmbedder(
8     char_embedding_dimension = 10,
9     hidden_dimension = 25,
10 )
11
12 # ELMo embedder
13 elmo_embedder = ElmoEmbedder()
14
15 # Concatenate the embeddings
16 embedder = ConcatEmbedders([
       word_embedder, char_embedder,
       elmo_embedder])
```

5 Related Work

Grobid (GRO, 2008–2020) is the closest to a general workbench for scientific document processing. Similarly to SciWING, Grobid also performs document structure classification, reference string parsing, among other tasks. But Grobid is architected in the traditional, manual feature engineering approach, leading to performance losses for many SDP tasks, and difficulties in retrofitting neural models into its framework.

SciSpaCy (Neumann et al., 2019) focuses on biomedical related tasks such as POS-tagging, syntactic parsing and biomedical span extraction. However, SciSpaCy primarily caters for practitioners; it does not easily allow for the development and testing of new models and architectures.

Task- and domain-agnostic frameworks also exist. NCRF++ (Yang and Zhang, 2018) is a tool for performing sequence tagging using Neural Networks and Conditional Random Fields and FLAIR (Akbik et al., 2018) is a framework for general-purpose NLP and mainly provide access to different embeddings and ways to combine them.

6 Conclusion and Future Work

We introduce SciWING, an open-source scholarly document processing (SDP) toolkit, targeted at practitioners and researchers interested in rapid experimentation. It provisions pre-trained models for key SDP tasks that achieve state-of-the-art performance and aids practitioners to deploy models directly on their community's literature.

SciWING's modular design also greatly facilitates SDP researchers in model architecture development, speeding train/test cycles for architecture search, and supporting transfer learning for use cases with limited annotated data. SciWING allows declaration of models, datasets and experiment parameters in a single configuration file.

SciWING is actively being developed. We consider the following improvements in our roadmap:

• SciWING has yet to incorporate natural language generation related models. We would like to consider sequence to sequence neural models which have proven useful for scientific document summarization tasks, among others.

• Scientific document processing involves minimal training data and has found benefits in incorporating document structure, both of which are tackled using multi-task learning. Multi-task learning is thus a future milestone in SciWING.

• We would like SciWING to foster collaboration among the SDP community and encourage assistance with these goals through contributions to our Github repository in the form of models, datasets and improvements to the framework.

References

2008–2020. Grobid. https://github.com/kermitt2/grobid.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, Rodney Kinney, Sebastian Kohlmeier, Kyle Lo, Tyler Murray, Hsu-Han Ooi, Matthew Peters, Joanna Power, Sam Skjonsberg, Lucy Wang, Chris Wilhelm, Zheng Yuan, Madeleine van Zuylen, and Oren Etzioni. 2018. Construction of the literature graph in semantic scholar. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 84–91, New Orleans - Louisiana. Association for Computational Linguistics.

Waleed Ammar, Matthew Peters, Chandra Bhagavatula, and Russell Power. 2017. The AI2 system at SemEval-2017 task 10 (ScienceIE): semi-supervised end-to-end entity and relation extraction. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 592–596, Vancouver, Canada. Association for Computational Linguistics.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. SciBERT: A pretrained language model for scientific text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Christopher Clark and Santosh Kumar Divvala. 2016. Pdffigures 2.0: Mining figures from research papers. *2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL)*, pages 143–152.

Arman Cohan, Waleed Ammar, Madeleine Van Zuylen, and Field Cady. 2019. Structural scaffolds for citation intent classification in scientific publications. In *NAACL*.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Arman Cohan and Nazli Goharian. 2015. Scientific article summarization using citation-context and article's discourse structure. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 390–400, Lisbon, Portugal. Association for Computational Linguistics.

Arman Cohan and Nazli Goharian. 2018. Scientific document summarization via citation contextualization and scientific discourse. *Int. J. Digit. Libr.*, 19(2–3):287–303.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Minh-Thang Luong, Thuy Dung Nguyen, and Min-Yen Kan. 2012. Logical structure recovery in scholarly articles with rich document features. In *Multimedia Storage and Retrieval Innovations for Digital Library Systems*, pages 270–292. IGI Global.

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. Scispacy: Fast and robust models for biomedical natural language processing.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, Doha, Qatar.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL-HLT*.

Animesh Prasad, Manpreet Kaur, and Min-Yen Kan. 2018. Neural parscit: A deep learning-based reference string parser. *Int. J. Digit. Libr.*, 19(4):323–337.

Vahed Qazvinian and Dragomir R. Radev. 2008. Scientific paper summarization using citation summary networks. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 689–696, Manchester, UK. Coling 2008 Organizing Committee.

Vahed Qazvinian, Dragomir R. Radev, Saif M. Mohammad, Bonnie Dorr, David Zajic, Michael

Whidby, and Taesun Moon. 2013. Generating extractive summaries of scientific paradigms. *J. Artif. Int. Res.*, 46(1):165–201.

Xuan Su, Animesh Prasad, Min-Yen Kan, and Kazunari Sugiyama. 2019. Neural multi-task learning for citation function and provenance. *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 394–395.

Dung Thai, Zhiyang Xu, Nicholas Monath, Boris Veytsman, and Andrew McCallum. 2020. Using bibtex to automatically generate labeled data for citation field extraction. In *Automated Knowledge Base Construction*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Jie Yang and Yue Zhang. 2018. Ncrf++: An open-source neural sequence labeling toolkit. In *Association for Computational Linguistics*.

Zenan Zhai, Dat Quoc Nguyen, Saber Akhondi, Camilo Thorne, Christian Druckenbrodt, Trevor Cohn, Michelle Gregory, and Karin Verspoor. 2019. Improving chemical named entity recognition in patents with contextualized word embeddings. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 328–338, Florence, Italy. Association for Computational Linguistics.