RepoFromPaper: An Approach to Extract Software Code Implementations from Scientific Publications

Aleksandar Stankovski[1][0009−0002−4731−7112] and Daniel Garijo[1][0000−0003−0454−7145]

[1]Ontology Engineering Group, Universidad Politécnica de Madrid
`a.stankovski@alumnos.upm.es`, `daniel.garijo@upm.es`

Abstract. An increasing amount of scientists link to their research software code implementations in their academic publications in order to support the reusability of their results. However, research papers usually contain many code links (e.g., from reused tools or existing competing efforts) making it challenging to automatically establish clear links between papers and their corresponding implementations. This paper presents RepoFromPaper, an approach for automatically extracting the main code implementation associated with a research paper, based on the context in which that link is mentioned. Our approach uses fine-tuned language models to retrieve the top candidate sentences where a code implementation may be found, and uses custom heuristics to link candidate sentences back to their corresponding URL (footnote, reference or full-text mention). We evaluated RepoFromPaper on 150 research papers, obtaining an F1 score of 0.94. We also run our approach on nearly 1800 papers from the CS.AI Arxiv category, discovering 604 paper-repository links and making them available to the community.

Keywords: Information extraction · Research Software · Software repository · Open Science.

1 Introduction

Research Software, i.e., the *source code files, algorithms, scripts, computational workflows and executables that were created during the research process* [2] is becoming recognized as a first class citizen in scientific curricula.[1] In order to support the results described in academic publications, scientists often include a link to a code repository (e.g., GitHub, Gitlab) with their technical implementations details.

While efforts have been made by the scientific community to establish principles[16] and formats for software citation [5], detecting the code repository link associated with a publication has two main challenges. First, authors often cite research software inconsistently, employing diverse formats and locations such as

[1] https://sfdora.org/read/

full-text repository mentions (cases where the link is written in the paragraphs), footnotes, or references to refer to a software component [8]. Second, a publication may contain several code repository links (from tools that are reused, or competing with the proposed approach) making it challenging to automatically detect the right code implementation.

This paper introduces a methodology designed to address these challenges by automatically extracting the software implementation repository link associated with a research paper, based on the context in which the link is mentioned. The core contributions of our work include:

1. **Training and validation datasets** of labeled sentences designed to fine-tune and evaluate our approach [17]. The training dataset includes 61 research papers related to software engineering available on the PapersWithCode[2] platform. The validation dataset includes 150 software engineering research articles extracted from Arxiv. Both datasets encompass various types of implementation mention sentences to cover the diverse ways authors reference the implementation repository.
2. **RepoFromPaper**[3], a tool to automatically extract the code implementation repository from a research paper, including PDF-to-Text conversion, sentence extraction, sentence classification and link search, as well as three fine-tuned models.
3. **The results** of the application of our approach on nearly 1800 Arxiv research papers, capturing links between research papers and their software implementations [20].

The rest of the paper is structured as follows. Section 2 describes related work efforts, while Section 3 describes the steps followed by RepoFromPaper to detect implementation links. Section 4 describes the metrics used in our evaluation and Section 5 presents our assessment results on 150 papers. Next, Section 6 describes how we applied our results to nearly 1800 papers, Section 7 discusses the limitations of our approach and Section 8 concludes the paper.

2 Related Work

The landscape of research papers mentioning software is vast and continually expanding. Platforms such as PapersWithCode actively promote the citation and linking of software source code in research papers. The FORCE11 Software Citation Working Group[4] has put forth software citation principles [15], and efforts from Katz et al. have analysed software citation implementation challenges [9], software citation in theory and practice [10], as well as provided a software citation guide [11] for researchers. These initiatives highlight the importance of proper software citation in research.

[2] https://paperswithcode.com/
[3] https://github.com/StankovskiA/RepoFromPaper
[4] https://force11.org/group/software-citation-working-group/

Researchers have taken on the challenge of automatically detecting software mention intent, as exemplified by the work available on GitHub[5] which uses data from SoftCite (Du et al., 2021) [6] and SoMeSci (Schindler et al., 2020) [14]. Their focus is on classifying software mentions based on intent, categorizing them into "Creation" (i.e., a tool is proposed in a paper), "Usage" (i.e., a tool is used in a publication to conduct research), and "Related" (i.e., a tool is mentioned as a related competitive effort). While this work shares similarities with ours, they aim to detect software tool mentions and understand their intent. Instead, our objective is to identify the repository code implementations associated with a research publication.

Lin et al. [12] present a methodology for automatically extracting software source code URLs, reporting a high model accuracy of 0.939. However, their approach has three limitations. First, their methodology does not consider URLs in references. Second, their approach relies on GROBID,[6] a PDF parser that structures nicely the contents of a paper, but may overlook footnotes. Third, their reliance on a regex search for sentences containing URLs may overlook indirect mentions, such as those within references or footnotes.

Finally, our previous work[7] [7] focuses on identifying bi-directional URL mentions between a paper and a repository, i.e., papers which mention a source code repository, and the repository reciprocates by mentioning the paper. While this approach holds the potential for high precision, it falls short in capturing unidirectional repository mentions (i.e., those publications that refer to a code repository but without a link back to that paper), which we aim to address in this work.

3 RepoFromPaper: Methodology

Our approach consists of six steps. Figure 1 provides an overview of the data flow within the pipeline, starting with an input PDF file and concluding with either the discovery of a relevant code implementation link or an empty response. We elaborate on each step of the pipeline below, providing insights into the rationale, processes, and integration of essential components within our methodology.

3.1 PDF-to-Text Conversion

We start with the conversion of PDFs of research papers into text using the Apache Tika PDF reader[8], known for its speed and accurate extraction of text. This initial step enables subsequent text-based processing, facilitating the application of heuristic rules for sentence extraction and input into the models for identifying repository implementation mentions. Although alternatives like GROBID are available, we selected Apache Tika due to its robust performance

[5] https://github.com/karacolada/SoftwareImpactHackathon2023_SoftwareCitationIntent
[6] https://github.com/kermitt2/grobid/
[7] https://github.com/SoftwareUnderstanding/RSEF
[8] https://github.com/chrismattmann/tika-python

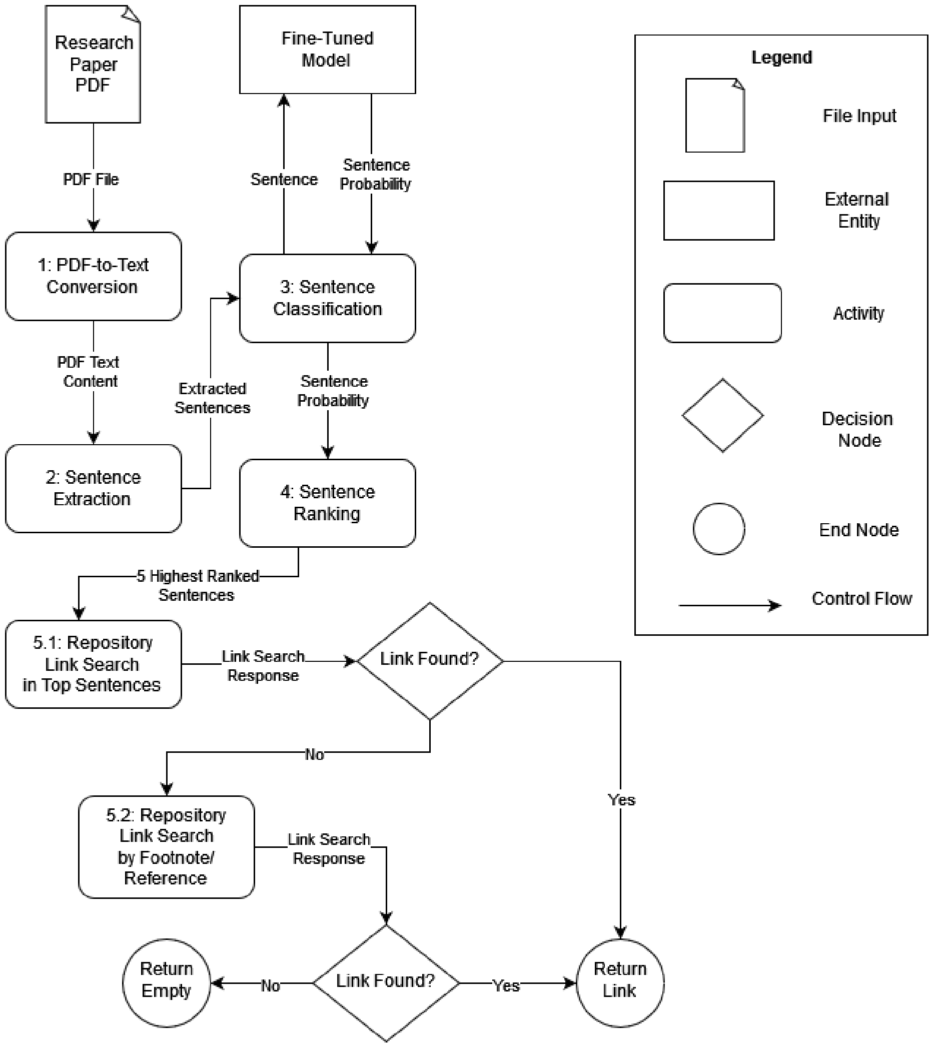

Fig. 1: RepoFromPaper Methodology Flowchart

in handling various PDF formats, processing speed, and accurate representation of footnote content, which is a critical factor in our methodology. The source code for our package, including the integration with Apache Tika, is available online under an MIT license[9], providing transparency and reproducibility for our approach.

[9] https://github.com/StankovskiA/RepoFromPaper

3.2 Sentence Extraction

The Sentence Extraction phase preprocesses the input PDF text and extracts complete sentences for subsequent analysis. Within this phase, various functions contribute significantly to refining and organizing the PDF text, effectively segmenting it into well-formatted sentences (e.g., removing end-of-line dashes) each ready for input into the fine-tuned models to classify them. The decision to extract sentences instead of paragraphs is driven by our findings that paragraphs introduce significant noise, while sentence-level extraction enhances model learning by focusing on key information.

Text cleaning is crucial for ensuring uniformity and clarity in the extracted sentences. This involves removing newline characters, word breaks, extra white space, and inconsistencies in links. Additionally, we extract reference number - reference text pairs, as well as footnote number - footnote text pairs if the footnote text is a link. This approach enables us to utilize this information effectively in the subsequent link search step.

Due to the diverse formats and layouts of research papers, sentences are frequently split across multiple lines, possibly spanning different pages and encountering footnotes in between, oftentimes including hyphenation at the end of lines as a line break. To address this issue, we consolidate fragmented sentences by considering factors such as sentence beginning and ending, new line characters and white spaces, and hyphenation, ensuring the formation of cohesive and complete sentences from fragmented text.

3.3 Sentence Classification

In this section, we delve into the process of classifying sentences extracted from research papers in order to identify implementation links. The classification is performed using fine-tuned BERT, SciBERT and RoBERTa models, chosen for their effectiveness in processing textual data.

Training Data and Fine-Tuning. We approach the problem of distinguishing between implementation mentions sentences and non-implementation sentences as a text classification problem, more specifically a sentence classification problem. Based on the context of the sentence, we aim to assess whether it is proposing an implementation of the paper. For this purpose, we assembled a training corpus [17] consisting of sentences extracted from 61 research papers related to software engineering sourced from the PapersWithCode platform. These sentences encompass various ways of mentioning repositories that authors tend to use, including full-text mentions, footnotes, and references. Each sentence in the corpus was annotated with a binary label indicating whether it mentions an implementation repository (1) or not (0). This annotation process involved one annotator initially labeling the sentences, followed by a review by another annotator. Any conflicts about the annotations were resolved through discussion until agreement was reached, particularly regarding whether to include an implementation mention if it was lacking clear context.

The models were fine-tuned using a binary sequence classification setup, where the objective was to classify sentences as either implementation mentions or non-implementation sentences. We employed the 'bert-base-uncased' model for BERT [4], 'allenai/scibert_scivocab_uncased' for SciBERT [1] and the 'roberta-base' model for RoBERTa [13], initializing them with pre-trained weights. The fine-tuned BERT[10], SciBERT[11] and RoBERTa[12] models are available on the HuggingFace platform.[13]

3.4 Sentence Ranking

Using our fine tuned models, we classify all the sentences available in an input publication. The models then predict the probability of each sentence belonging to the class of implementation or non-implementation sentences. Based on these probabilities, the sentences are ranked, allowing us to identify the sentences which are most likely to contain implementation mentions. We then retrieve the five sentences with the highest probability as candidates to find implementation links. Our rationale for selecting the top five sentences is that, while the model predicts the probability of each sentence belonging to class 1, we observed that the correct proposal sentence may not consistently have the highest probability. To mitigate this, we opt for a more inclusive strategy, extracting the top five sentences based on their probability scores. This increased the chances of capturing the correct proposal sentence from 80% to 94%, accounting for potential variations in model predictions.

3.5 Repository Link Search

The final step of our methodology aims to link the top ranked sentences with the corresponding link containing the code implementation. We divide this step in two stages:

1. **Repository link search in top sentences**: We use regular expressions to retrieve any code repository links (GitHub, GitLab) that may be found within the candidate sentence itself. As described in Howison & Bullard (2015) [8], inline references are among the common practices for citing software in publications. The rationale behind this step is to establish a direct connection between the predicted proposal sentences and their corresponding repositories. By searching for links within the sentences with the highest probability of being implementation mentions, we aim to streamline the extraction process and efficiently link research papers to their associated repositories. If multiple repository links are present in the top-ranked sentences, we return the first identified link.

[10] https://huggingface.co/oeg/BERT-Repository-Proposal
[11] https://huggingface.co/oeg/SciBERT-Repository-Proposal
[12] https://huggingface.co/oeg/RoBERTa-Repository-Proposal
[13] https://huggingface.co/

2. **Repository link search in footnotes and references**: This step aims to broaden the search scope by examining the sentences that may contain relevant numbers or special characters representing footnote or reference numbers. These characters are indicative of footnote or reference numbers commonly associated with research papers. The order of appearance of these numbers is retained to prioritize classified sentences with higher probability. Once potential footnote or reference numbers are identified, our methodology proceeds to search for candidate sentences containing these numbers. To maximize the chances of finding the correct sentence, we consider both the original appearance of the numbers and variations with or without brackets, particularly when reference numbers are enclosed in brackets.

4 Evaluation Methods

To assess the performance of our methodology, we employ two main evaluation methods, each providing valuable insights into the effectiveness of our approach.

4.1 Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank (MRR) [3] serves as a key evaluation metric for gauging the individual performance of the fine-tuned models. MRR is calculated based on the position of the correct proposal sentence within the list of the top five highest ranked sentences. This metric offers an understanding of how well the models rank the correct proposal sentence relative to other potential candidates. A higher MRR indicates better model performance in isolating and prioritizing the most relevant sentences.

The formula for Mean Reciprocal Rank is given by:

$$MRR = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i} \tag{1}$$

where N is the number of instances, and rank_i is the position of the correct proposal sentence in the ranked list for the i^{th} instance.

4.2 Precision, Recall, and F1 Score

To comprehensively evaluate the overall performance of RepoFromPaper, we employ precision, recall, and F1 score metrics. These metrics are calculated based on the following definitions:

- **True Positive (TP):** The pipeline returns a correct repository implementation link for the target paper.
- **False Positive (FP):** The pipeline returns an incorrect repository implementation link for the target paper.
- **False Negative (FN):** The pipeline fails to identify any repository implementation link, but one is present.

– **True Negative (TN):** The model finds no repository implementation link, and there is no link present in the paper.

Precision measures the accuracy of the identified repository links, recall assesses the ability of the methodology to capture all relevant links, and the F1 score provides a balanced evaluation considering both precision and recall.

4.3 Training and Testing Corpora

In the training corpus for our models, we included 75 implementation sentences and approximately 2500 non-implementation sentences from 61 research papers sourced from the PapersWithCode platform. To evaluate the performance of our method, we assembled a separate evaluation corpus [19] consisting of 150 software engineering research papers obtained from Arxiv.org. These papers were carefully selected to ensure heterogeneity and avoid repetitiveness, representing a diverse range of implementation mention styles, authored by various authors. We manually tagged these papers to create a validation set specifically for evaluating our methodology. Importantly, none of the papers included in this validation set were used for training the models, ensuring the integrity of our evaluation process.

We utilized the entire text contents of these papers in our evaluation process. To ensure merit and diversity of repository implementation types, both the training corpus and evaluation set are consisted of papers that encompass the three main mention types: "Full-text" (i.e., inline URLs), "Footnote" and "Reference" mentions. Figure 2 shows the number of mention types present in the training and evaluation set, which follow a similar distribution.

Figure 3 shows the frequency distribution of repository links found in the papers. This distribution sheds light on the challenges associated with automatically identifying and extracting the correct implementation repository links from research papers, as a nearly half of the papers have two or more code links.

Finally, papers that only used hyperlinks to link the implementation repositories were excluded from the set as the link text was not present in the PDF text.

5 Results

Table 1 describes the results obtained from the evaluation of our methodology using the Fine-Tuned BERT, SciBERT, and RoBERTa models over our test set. Our results present an accurate identification of implementation mentions within research papers (0.94 F1), indicating the efficacy of employing fine-tuned language models. The SciBERT model exhibits superior learning capabilities, as reflected in its elevated precision, recall, and F1 score. This implies a more sophisticated grasp of implementation mentions. The model's enhanced performance is in line with its advanced architecture, emphasizing the significance of employing state-of-the-art language models for intricate tasks.

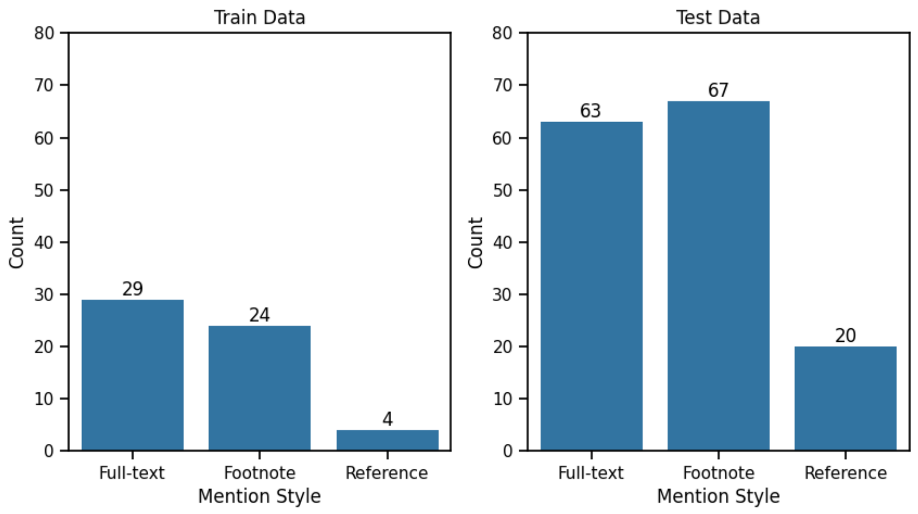

Fig. 2: Distribution of implementation links and their style (within the full text, as a footnote or in a reference) in training and testing corpora

Table 1: Evaluation results for the fine-tuned models

Model	**Precision**	**Recall**	**F1 Score**	**MRR**
BERT	0.864	0.871	0.867	0.55
RoBERTa	0.942	0.929	0.936	0.753
SciBERT	**0.944**	**0.95**	**0.947**	**0.85**

These results provide insights into the performance of the fine-tuned models, as well as the overall methodology, in identifying implementation mentions within the extracted sentences. To better understand the significance of the results, we compare our results in the test set against a baseline method achieved by selecting the most frequent code repository (using a regular expression) in a publication (the first code repository is returned if all code links appear just once).

The comparison between the regex baseline and our best performing model can be seen in Table 2. Our method outperforms the baseline by achieving 15% higher precision and 6% increase in F1 Score, while having a 5% lower recall. We consider this an adequate trade off for the problem at hand.

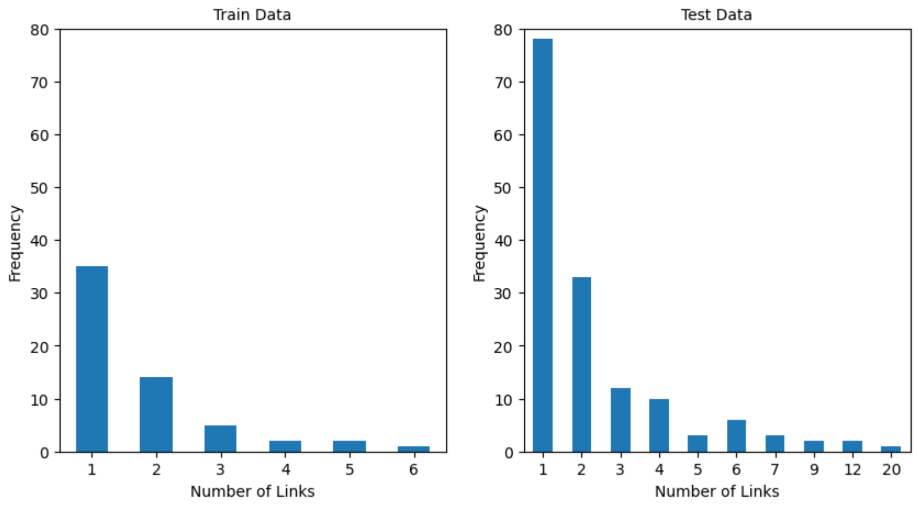

Fig. 3: Number of code repository links in training and test papers

Table 2: Comparison with baseline performance

Model	Precision	Recall	F1 Score
Regex Baseline	0.793	1	0.88
SciBERT	**0.944**	**0.95**	**0.947**

6 A Corpus of Papers and their Corresponding Implementations

We packaged our method into RepoFromPaper[14] [21] and applied it on nearly 1800 research papers submitted in the years 2022 and 2023 found on the Artificial Intelligence section on Arxiv.org.[15] We applied our method both with the fine-tuned RoBERTa and SciBERT models over the papers. The leading performance of the SciBERT model approach was once again confirmed as it led to detection and extraction of 604 implementation repository links, while the RoBERTa model approach detected 585. We make the outcomes of this application of our method public [20].

Finally, we compared our approach using the SciBERT model against a method that detects bi-directional links between papers and code repositories [7] on 150 research papers from the Software Engineering category on Arxiv.org[16]. These 150 articles were selected randomly, from the year 2023 (selected papers may or may not include a code implementation link). In summary, the number of implementation links found only by the bi-directional approach was 4, the

[14] RepoFromPaper is available at https://github.com/StankovskiA/RepoFromPaper
[15] https://arxiv.org/list/cs.AI/recent
[16] https://arxiv.org/list/cs.SE/recent

number of implementation links found only by RepoFromPaper was 41 and the number of implementation links found by both approaches was 16. In 89 publications none of the approaches found an implementation link. The results of the comparison are available online [18].

Our findings show that our approach is able to extract 25% more implementation repository links when compared to the bi-directional approach. Furthermore, we observe an expected overlap in the extracted links between both approaches, i.e., both approaches successfully extract the same implementation link. However, we also observe unique links extracted by each method. The reason for this divergence is twofold. On the one hand, while our approach is able to detect uni-directional links between a research paper and a repository, the bi-directional approach requires the repository to also point back to the paper, therefore missing these links. On the other hand, the bi-directional approach is able to return multiple confirmed links (many authors separate code implementation and evaluation results in different repositories) whereas our approach currently returns only one implementation repository link. The ability of both approaches to detect unique links suggests that they can complement each other, aiming to extract as many correct implementation repository links from the research paper as possible.

7 Discussion

Our approach produces high evaluation results, presents several limitations. Firstly, the model training dataset was limited to the 75 implementation mention sentences present in 61 research papers, which may restrict its ability to generalize in other domains. Additionally, when the proposal link was not found in the top-ranked sentences, our methodology searched for footnote or reference mentions, but the abundance of numbers in some sentences introduced potential noise. Moreover, the order of sentences containing footnote/reference numbers posed complexity, occasionally leading to false positive links. Another limitation is that our methodology currently returns only one implementation link, even when multiple correct links may exist in a publication. Our approach also does not extract links embedded as hyperlinks, defined in tables or present in metadata. Despite these challenges and limitations, our methodology demonstrates robust performance by effectively detecting and extracting implementation repository links from PDFs of research papers, irrespective of their formats or the diverse ways in which implementation repositories are mentioned.

8 Conclusions and Future Work

In this paper we introduced RepoFromPaper, a methodology and tool for the automatic extraction of implementation repository links from research papers. Our evaluation demonstrates promising results, showcasing the efficacy of using fine-tuned language models. The achieved precision, recall, and F1 scores, particularly with the SciBERT model, signify a considerable success in identifying implementation mentions within research papers.

However, while our approach has shown good performance, there remain areas for improvement, particularly in the pre-processing step of converting PDFs to text. Enhancements in this phase may lead to more accurate sentence extraction, reducing noise and further refining the pipeline's effectiveness.

Moving forward, there are several avenues for future work and improvements. First, expanding the training dataset and fine-tuning the models with a more extensive range of proposal mention variations may enhance their ability to recognize diverse ways of mentioning repositories. Second, investigating more advanced PDF-to-Text conversion techniques may contribute to better sentence extraction, overcoming challenges posed by varied PDF formats. Finally, applying our method on research papers from different domains will help us generalizing our approach, gaining better insights into the current practices regarding code and data repository mentions in disciplines other than Computer Science, ranging from Astronomy to Geology or Computational Biology.

Acknowledgements

This work is supported by the Madrid Government (Comunidad de Madrid - Spain) under the Multiannual Agreement with Universidad Politécnica de Madrid in the line Support for R&D projects for Beatriz Galindo researchers, in the context of the VPRICIT, and through the call Research Grants for Young Investigators from Universidad Politécnica de Madrid. The authors would also like to acknowledge European Union's Horizon Europe Programme under GA 101129744 — EVERSE — HORIZON-INFRA-2023-EOSC-01-02.

References

1. Beltagy, I., Lo, K., Cohan, A.: Scibert: A pretrained language model for scientific text. arXiv preprint arXiv:1903.10676 (2019). https://doi.org/10.48550/arXiv.1903.10676
2. Chue Hong, N.P., Katz, D.S., Barker, M., Lamprecht, A.L., Martinez, C., Psomopoulos, F.E., Harrow, J., Castro, L.J., Gruenpeter, M., Martinez, P.A., Honeyman, T., Struck, A., Lee, A., Loewe, A., van Werkhoven, B., Jones, C., Garijo, D., Plomp, E., Genova, F., Shanahan, H., Leng, J., Hellström, M., Sandström, M., Sinha, M., Kuzak, M., Herterich, P., Zhang, Q., Islam, S., Sansone, S.A., Pollard, T., Atmojo, U.D., Williams, A., Czerniak, A., Niehues, A., Fouilloux, A.C., Desinghu, B., Goble, C., Richard, C., Gray, C., Erdmann, C., Nüst, D., Tartarini, D., Ranguelova, E., Anzt, H., Todorov, I., McNally, J., Moldon, J., Burnett, J., Garrido-Sánchez, J., Belhajjame, K., Sesink, L., Hwang, L., Tovani-Palone, M.R., Wilkinson, M.D., Servillat, M., Liffers, M., Fox, M., Miljković, N., Lynch, N., Martinez Lavanchy, P., Gesing, S., Stevens, S., Martinez Cuesta, S., Peroni, S., Soiland-Reyes, S., Bakker, T., Rabemanantsoa, T., Sochat, V., Yehudi, Y., WG, R.F.: FAIR Principles for Research Software (FAIR4RS Principles) (Jun 2022). https://doi.org/10.15497/RDA00068

3. Craswell, N.: Mean reciprocal rank. pp. 1703–1703. Springer US (2009). https://doi.org/10.1007/978-0-387-39940-9˙488
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018). https://doi.org/10.48550/arXiv.1810.04805
5. Druskat, S., Spaaks, J.H., Chue Hong, N., Haines, R., Baker, J., Bliven, S., Willighagen, E., Pérez-Suárez, D., Konovalov, O.: Citation File Format (Aug 2021). https://doi.org/10.5281/zenodo.5171937
6. Du, C., Cohoon, J., Lopez, P., Howison, J.: Softcite dataset: A dataset of software mentions in biomedical and economic research publications **72**(7), 870–884 (jun 2021). https://doi.org/10.1002/asi.24454
7. Garijo, D., Arroyo, M., Gonzalez, E., Treude, C., Tarocco, N.: Bidirectional paper-repository tracing in software engineering. In: 21st International Conference on Mining Software Repositories (MSR '24). ACM, Cham (April 2024). https://doi.org/10.1145/3643991.3644876
8. Howison, J., Bullard, J.: Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. Journal of the Association for Information Science and Technology **67**(9), 2137–2155 (2016). https://doi.org/10.1002/asi.23538
9. Katz, D.S., Bouquin, D., Hong, N.P.C., Hausman, J., Jones, C., Chivvis, D., Clark, T., Crosas, M., Druskat, S., Fenner, M., et al.: Software citation implementation challenges. arXiv preprint arXiv:1905.08674 (2019). https://doi.org/10.48550/arXiv.1905.08674
10. Katz, D.S., Chue Hong, N.P.: Software citation in theory and practice. In: Mathematical Software–ICMS 2018: 6th International Conference, South Bend, IN, USA, July 24-27, 2018. pp. 289–296. Springer (2018). https://doi.org/10.1007/978-3-319-96418-8˙34
11. Katz, D.S., Hong, N.P.C., Clark, T., Muench, A., Stall, S., Bouquin, D., Cannon, M., Edmunds, S., Faez, T., Feeney, P., et al.: Recognizing the value of software: a software citation guide. F1000Research **9** (2020). https://doi.org/10.12688/f1000research.26932.1
12. Lin, J., Wang, Y., Yu, Y., Zhou, Y., Chen, Y., Shi, X.: Automatic analysis of available source code of top artificial intelligence conference papers. International Journal of Software Engineering and Knowledge Engineering **32**(07), 947–970 (2022). https://doi.org/10.1142/S0218194022500358
13. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019). https://doi.org/10.48550/arXiv.1907.11692
14. Schindler, D., Bensmann, F., Dietze, S., Krüger, F.: Somesci- a 5 star open data gold standard knowledge graph of software mentions in scientific articles. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. p. 4574–4583. CIKM '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3459637.3482017
15. Smith, A.M., Katz, D.S., Niemeyer, K.E.: Software citation principles. PeerJ Computer Science **2**, e86 (2016). https://doi.org/10.7717/peerj-cs.86
16. Smith, A.M., Katz, D.S., Niemeyer, K.E., FORCE11 Software Citation Working Group: Software citation principles. PeerJ Computer Science **2**, e86 (Sep 2016). https://doi.org/10.7717/peerj-cs.86
17. Stankovski, A.: PapersWithCode-Corpus Repository Proposal Sentences Training Dataset (Feb 2024). https://doi.org/10.5281/zenodo.10701846

18. Stankovski, A.: Repofrompaper comparison with bidir method (Apr 2024). https://doi.org/10.5281/zenodo.10988947
19. Stankovski, A.: RepoFromPaper Repository Implementation Link Testing Dataset (Apr 2024). https://doi.org/10.5281/zenodo.10980368
20. Stankovski, A.: Rfp output on csai papers from 2022/23 (Apr 2024). https://doi.org/10.5281/zenodo.10975879
21. Stankovski, A.: Stankovskia/repofrompaper: v1.0.1 (Apr 2024). https://doi.org/10.5281/zenodo.10988913