Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master in Data Science

Master Thesis

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

Author: Aleksandar Stankovski

Madrid, July, 2024

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis
Master in Data Science

Title: RepoFromPaper: An Approach to Extract Software Code Implementations from Scientific Publications
July, 2024

Author:	Aleksandar Stankovski
Supervisor:	Daniel Garijo Verdejo
	Artificial Intelligence Department
	ETSI Informáticos
	Universidad Politécnica de Madrid

Summary

Introduction

The rapid advancement of scientific research necessitates efficient tools for researchers to stay current with developments and access relevant software implementations. Research software, which includes source code files, algorithms, scripts, computational workflows, and executable files, has become a critical component in research dissemination and education. Despite the growing importance of research software, the automatic extraction of software implementation code repository links from scientific publications remains a significant challenge due to inconsistent citation practices and the presence of multiple repository links within a single paper.

Literature Review

The literature review provides an in-depth analysis of existing initiatives aimed at promoting software citation, automatic detection of software mentions, and link-paper traceability. It highlights the limitations of current approaches, such as the lack of standardization in citation practices and the challenges in maintaining up-to-date repositories. The review also examines advances in NLP techniques that can be leveraged for information extraction, setting the stage for the development of RepoFromPaper.

RepoFromPaper Methodology

RepoFromPaper's methodology is detailed in several key steps:

- **Data Collection**: Gathering a diverse set of scientific publications containing software citations.
- **PDF-to-Text Conversion**: Utilizing tools to convert PDF documents into machine-readable text.
- **Sentence Extraction**: Extracting sentences that potentially contain repository links.
- **Sentence Classification**: Applying BERT models to classify sentences and identify repository links.
  - **Training Data and Fine-Tuning**: Creating a training dataset and fine-tuning BERT models to enhance accuracy.
  - **Inference and Sentence Ranking**: Implementing inference mechanisms to rank sentences based on their likelihood of containing repository links.

RepoFromPaper's tool availability and structure are also discussed, including installation, usage, and integration into the RSEF. The integration allows for bidirectional repository link searches, further enhancing the utility of RepoFromPaper in research environments.

Evaluation

The evaluation chapter details the corpora used for testing RepoFromPaper, the evaluation methods employed, and the results obtained. Key metrics include Mean Reciprocal Rank (MRR), precision, recall, and F1 score. The performance of fine-tuned models is compared against baseline approaches, demonstrating RepoFromPaper's effectiveness in accurately identifying and extracting repository links.

Application and Comparative Analysis

This chapter explores the application of RepoFromPaper to different sets of scientific publications, including AI papers and non-computer science papers. The results of these experiments provide insights into the tool's versatility and robustness. Additionally, a comparative analysis with bi-directional link detection methods is presented, highlighting the strengths and areas for improvement in RepoFromPaper.

Conclusions and Future Work

The thesis concludes by summarizing the key contributions of RepoFromPaper, including its impact on improving research workflows and promoting reproducibility. The challenges and limitations encountered during the study are discussed, such as the need for more extensive training datasets and the improvement of PDF-to-text conversion accuracy. Future work will focus on expanding the training dataset, enhancing the conversion process, and incorporating user feedback to refine the tool further. The broader impact of RepoFromPaper on scientific research and its potential future directions are also considered.

Abstract

The increasing integration of complex software systems in scientific research has amplified the need for efficient methods to access and utilize these software implementations. To enhance reproducibility and transparency, researchers often include links to code repositories within their publications. However, the extraction of these repository links is challenging due to inconsistent citation practices, varied formatting, and the presence of multiple repository links within a single publication. This thesis introduces "RepoFromPaper," an innovative approach designed to automate the extraction of repository links from scientific publications using advanced natural language processing (NLP) techniques.

RepoFromPaper systematically identifies and extracts repository links proposed by authors, enhancing the discoverability and accessibility of research software. The methodology employed includes a multi-step process of data collection, PDF-to-text conversion, sentence extraction, and classification using BERT models. Additionally, RepoFromPaper is evaluated using a gold standard dataset, and its performance is benchmarked against existing approaches using metrics such as Mean Reciprocal Rank (MRR), precision, recall, and F1 score.

This work also explores the integration of RepoFromPaper into the Research Software Extraction Framework (RSEF), enabling bidirectional repository link searches. The evaluation encompasses various scientific domains, highlighting citation practices and the tool's applicability. The findings demonstrate the potential of automated solutions like RepoFromPaper in improving research workflows, promoting reproducibility, and facilitating the use of scientific software.

Contents

Chapter 1

Introduction

The rapid pace of scientific advancements poses significant challenges for researchers striving to stay updated with the latest developments. The proliferation of scientific publications and the integration of complex software systems into research underscore the need for efficient methods to access and utilize these software implementations. Research software encompasses source code files, algorithms, scripts, computational workflows, and executable files developed during the research process [1]. It is increasingly recognized as an essential component in scientific curricula and research dissemination[1].

Adhering to Open Science best practices, researchers usually include links to code repositories in their academic publications to provide details on their technical implementations. However, extracting these software code implementation repository links automatically from scientific papers is fraught with difficulties. One major challenge is the inconsistency in citation practices regarding citation of implementation repositories of proposed software in scientific publications. Authors employ a variety of formats and locations for citing their code repositories, including embedding links within the main text, using footnotes, or listing them in the references section [2].

Another significant challenge is the presence of multiple code repository links within a single publication. Papers may cite several tools that are reused or are alternatives to the proposed approach, making it difficult to identify the primary code implementation. This can be particularly problematic when a paper references various repositories for different components or libraries, adding to the complexity of determining the software proposed in the study. Despite efforts by the scientific community to establish standardized principles [3] and formats for software citation [4], practical implementation remains challenging [5]. This is partly due to the lack of awareness among researchers about these principles and the diverse citation formats requested by conferences. Properly recognizing and extracting the implementation repositories from scientific publications can significantly aid in browsing existing efforts, as demonstrated by platforms such as PapersWithCode[2] and arXiv, which allow authors to manually expose the repositories of their papers. However, many authors do not take advantage of this feature, highlighting the need for automated solutions.

Recognizing the importance of accessing implementation repositories for replicating

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

[2] https://paperswithcode.com/

and building upon research, this thesis introduces "RepoFromPaper," an innovative approach to automate the extraction of these repository links from scientific publications. By leveraging advanced natural language processing (NLP) techniques, RepoFromPaper aims to systematically identify and extract the links to code repositories proposed by the authors, thereby enhancing the discoverability and accessibility of research software. Moreover, RepoFromPaper addresses the challenge of variability in how repository links are cited—whether embedded within the main text, footnotes, or references section—by employing a robust methodology that ensures a higher accuracy of extraction.

A paper [6] summarizing this work has been accepted at the first workshop for Natural Scientific Language Processing and Research Knowledge Graphs (NSLP 2024), highlighting the significance and recognition of the RepoFromPaper project within the scientific community.

1.1 Objectives

The primary objectives of the RepoFromPaper project are as follows:

- Develop a method to automatically extract implementation code repository links of proposed software from scientific publications.
- Create a gold standard dataset to evaluate the performance of the repository link extraction method.
- Use RepoFromPaper to assess citation practices in different scientific domains and identify patterns and inconsistencies in how implementation repositories are cited.

1.2 Scope

The scope of the RepoFromPaper project includes the following assumptions and limitations:

- The focus is on extracting implementation repositories from scientific papers across various domains where software plays a crucial role.
- The project is limited to the extraction of GitHub and GitLab code repositories and does not extend to other forms of supplementary materials such as datasets or figures shared on other platforms.
- The methodology assumes that the extracted repository links are valid and correctly point to the intended software implementations.

By focusing on these objectives and scope, the RepoFromPaper project aims to contribute significantly to the field of research software citation and information extraction, ultimately facilitating more efficient research workflows and enhancing the accessibility of scientific software implementations.

The structure of this thesis is as follows:

- **Chapter 1: Introduction** - Provides an overview of the importance of reproducibility in scientific research, the challenges of software citation, and the aims

of the RepoFromPaper project.

- **Chapter 2: Literature Review** - Reviews existing literature on software citation practices, the role of research software in reproducibility, and advances in NLP techniques for information extraction.
- **Chapter 3: RepoFromPaper** - Describes the methodologies used in the RepoFromPaper project, including data collection, PDF-to-text conversion, sentence extraction and classification, and repository link extraction.
- **Chapter 4: Evaluation** - Presents the results of applying the RepoFromPaper system to a set of scientific publications and evaluates its performance using relevant metrics.
- **Chapter 5: Application and Comparative Analysis** - Discusses the application of RepoFromPaper on AI papers, comparison with bi-directional link detection, and application on non-CS papers.
- **Chapter 6: Conclusions and Future Work** - Summarizes the conclusions, challenges, limitations, and future work, including expanding the training dataset, improving PDF-to-text conversion, and user feedback.

Chapter 2

Literature Review

The landscape of research papers mentioning software is vast and continually expanding. Numerous initiatives and platforms have been developed to promote the citation and linking of software source code in research papers, reflecting the growing recognition of research software as a key asset in scientific progress. This section reviews existing literature and tools related to extracting software implementations from scientific publications, highlighting both the advancements made and the challenges that persist.

2.1 Initiatives Promoting Software Citation

Platforms such as PapersWithCode[1] actively promote the citation and linking of software source code in research papers. This platform lists implementations associated with paper publications based on human feedback, ensuring that the connections between papers and their corresponding code repositories are accurately maintained. Similarly, Heumüller et al. [7] conducted manual curation analysis of hundreds of papers in computer science to assess the availability of their implementations. Although these efforts result in high-quality datasets, they do not address the nuances of code citation practices and require significant manual effort to generate and curate. Additionally, arXiv[2] provides features that allow authors to expose their code repositories. Despite these advancements, the need for automated solutions remains evident, as many authors do not manually link their code repositories.

The FORCE11 Software Citation Working Group[3] has put forth software citation principles [8] aimed at standardizing how software should be cited in academic work. These principles have been pivotal in shaping the discourse around software citation and have been supported by various analyses from researchers such as Katz et al. They have explored software citation implementation challenges [9], discussed software citation in theory and practice [10], and provided a comprehensive software citation guide for researchers [11]. These initiatives underscore the importance of proper software citation in research. The platform GitHub[4] recognizing the impor-

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

tance, added a feature[5][4] which allows users to add a citation file to which other researchers can refer in order to properly cite the repositories.

2.2 Automatic Detection of Software Mentions

Automatic detection of software mention intent has been another area of significant research. For instance, a project available on GitHub[6], utilizing data from SoftCite [12] and SoMeSci [13], focuses on classifying software mentions based on intent. SoftCite and SoMeSci are initiatives aimed at detecting tool names and understanding the role of these tools in research. They categorize mentions into "Creation" (a tool proposed in a paper), "Usage" (a tool used in a publication to conduct research), and "Related" (a tool mentioned as a related effort). While this work shares similarities with ours, as it involves the detection of software tool mentions, it primarily aims to understand the intent behind *tool name* mentions rather than identifying the repository code implementations associated with a research publication.

Lin et al. [14] present a methodology for automatically extracting software source code URLs, reporting a high model accuracy of 0.939. Their objective is to detect URLs to facilitate further research on software impacts. However, their approach has certain limitations. First, their methodology does not account for URLs mentioned in references, which is a common practice. Second, they rely on GROBID[7], a PDF parser that structures the content of a paper but may overlook footnotes, which are another common means for referring to software repositories. Third, their reliance on regex searches for sentences containing URLs may miss indirect mentions, such as those within references or footnotes.

2.3 Link-Paper Traceability and Analysis

Further analyses by researchers like Hata et al. [15] focus on the challenges of link-paper traceability. Wattanakriengkrai et al. [16] analyze GitHub code repositories to find links pointing back to papers, quantifying the frequency of these links. At a more granular level, Inokuchi et al. [17] search for citation links in code comments. These studies provide valuable insights into the relationship between research papers and software, but their scope differs from ours. They aim to detect and study the bidirectional relationships between papers and code.

Recent work by Kelley and Garijo [18] explores the extraction and labeling of software metadata from README files. Nonetheless, their approach does not involve the extraction of implementation repositories from scientific papers. Additionally, the implementation and training data associated with these efforts are not always openly accessible, and the tools used for PDF to text conversion often face limitations in detecting footnotes, which are a common means for referring to software repositories.

[5] https://github.com/citation-file-format/citation-file-format
[6] https://github.com/karacolada/SoftwareImpactHackathon2023_SoftwareCitationIntent
[7] https://github.com/kermitt2/grobid/

2.4 Bidirectional Relationships and Comprehensive Methods

Finally, work of Garijo et al. [8] [5] focuses on identifying bi-directional URL mentions between a paper and a repository. This involves detecting instances where papers 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—those publications that refer to a code repository without a reciprocal link back to the paper. This limitation highlights the need for more comprehensive methods to capture all relevant software mentions in scientific publications.

2.5 Limitations of the State of the Art

Despite the significant progress made in the field of software citation and extraction, several limitations persist. One of the primary limitations is the current approaches' reliance on detecting software citations only in specific formats and locations within publications. Authors often employ diverse formats and embed code repository links in various sections, such as the main text, footnotes, or references. This inconsistency makes it difficult to establish a standardized approach for automatic detection, resulting in many citations being overlooked.

Another limitation is the reliance on manual curation and feedback for identifying and linking software implementations, as seen in platforms like PapersWithCode[9] and the work of Heumüller et al. [7]. While these methods ensure high-quality data, they are not scalable and require substantial effort to maintain. Automated methods, such as those proposed by Lin et al. and other researchers, offer promising solutions but are often constrained by their inability to handle certain types of mentions, such as those in references or footnotes.

The scope of existing studies also varies, with some focusing on detecting tool names and understanding their intent, while others aim to establish bidirectional links between papers and repositories. However, there is a lack of comprehensive methods that can capture all relevant software mentions, categorize them based on their role in the publication, and link them to their corresponding repositories.

Furthermore, the tools and datasets used in these studies are not always openly accessible, limiting their reproducibility and the ability for other researchers to build upon these efforts. This lack of accessibility hinders the progress towards more robust and scalable solutions for software citation detection.

In conclusion, while significant strides have been made in understanding and improving software citation practices, challenges remain in achieving consistent, comprehensive, and automated detection of software implementations from scientific publications. The RepoFromPaper project aims to address these limitations by developing a novel approach that leverages advanced NLP techniques to enhance the identification and linking of research software.

[8] https://github.com/SoftwareUnderstanding/RSEF
[9] https://paperswithcode.com/

Chapter 3

RepoFromPaper

This chapter outlines the systematic approach employed in the RepoFromPaper project to develop a comprehensive methodology for accurately detecting and extracting implementation code repository links of proposed software from research papers. This chapter details each step of the process, starting from data collection to the implementation of the final methodology.

Our approach is designed to handle the complexities associated with scientific literature, specifically focusing on software engineering research papers. By leveraging advanced text processing techniques and state-of-the-art language models, we aim to provide a robust solution for identifying software repository mentions within academic texts.

The methodology is structured into several key components: Data Collection, PDF-to-Text Conversion, Sentence Extraction, Sentence Classification, and Repository Link Extraction. Each component plays a critical role in ensuring the accuracy and efficiency of our detection system.

Firstly, we discuss the Data Collection process, which involves assembling a comprehensive and diverse training corpus from the PapersWithCode platform. This corpus is annotated to create a high-quality dataset essential for training our models.

Next, we delve into the PDF-to-Text Conversion process, where PDF documents are converted into machine-readable text using Apache Tika. This step is crucial for enabling subsequent text-based analysis.

Following this, the Sentence Extraction section describes the methods employed to isolate individual sentences from the converted text. These sentences are then classified to identify potential mentions of software repositories.

The Sentence Classification section details the use of fine-tuned language models, such as BERT [19], SciBERT [20], and RoBERTa [21], to accurately classify sentences based on whether they mention a software repository.

Finally, the Repository Link Extraction section explains how identified repository mentions are processed to extract precise repository links.

Through this detailed and methodical approach, the RepoFromPaper project aims to provide a reliable tool for the automatic extraction of implementation code repository links of proposed software from research papers, contributing to the advancement of

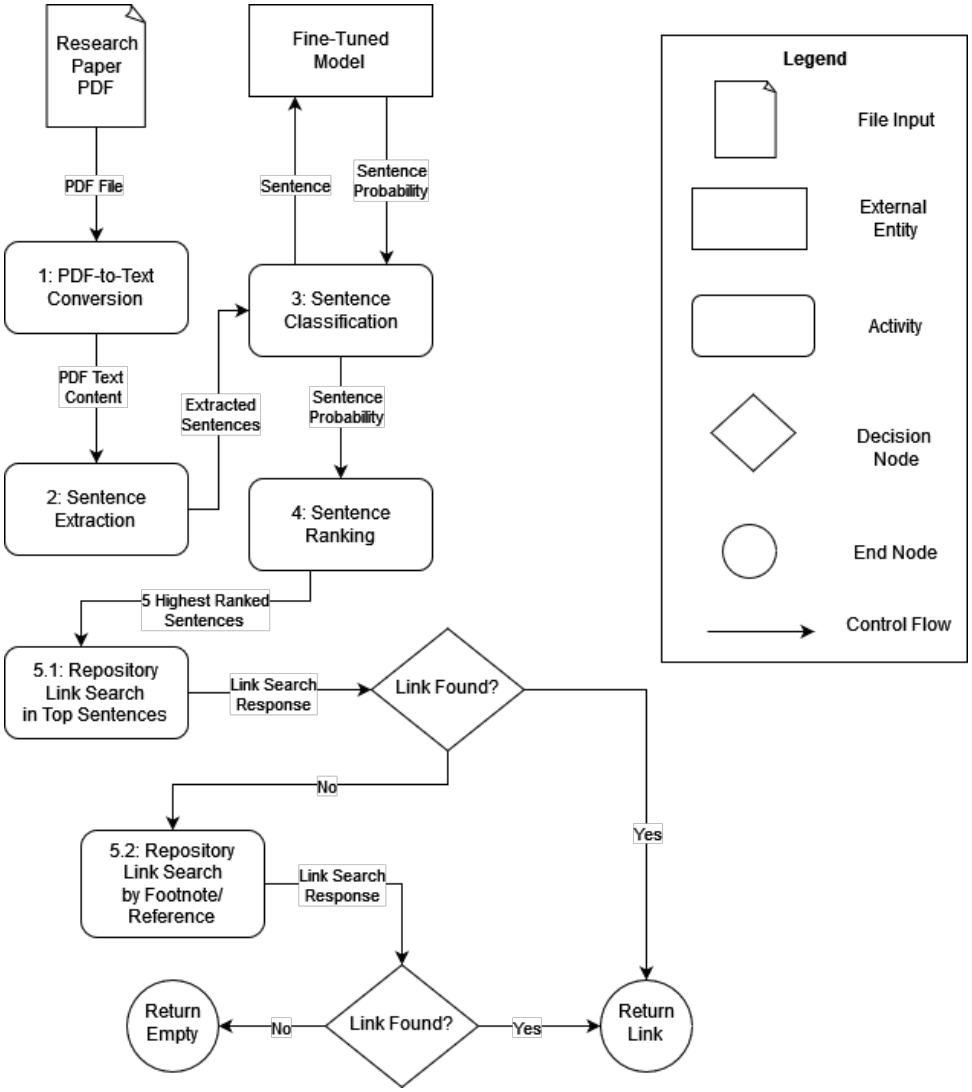

Figure 3.1: RepoFromPaper Methodology Flowchart

open-access research and reproducibility in the software engineering domain. The overall workflow of the RepoFromPaper methodology is illustrated in Figure 3.1, providing a visual representation of the sequential steps involved in the process.

The entire RepoFromPaper methodology is packaged into a tool, which is available on GitHub and Zenodo[1] [22]. This tool, designed for researchers and developers, leverages natural language processing and heuristic techniques to automatically detect and extract repository links from academic papers. The tool's source code, usage instructions, and installation guidelines are accessible online, promoting transparency and ease of use.

The complete methodology is described in detail in the following sections.

[1]RepoFromPaper is available at `https://github.com/StankovskiA/RepoFromPaper`

3.1 Methodology

3.1.1 Data Collection

In order to develop our methodology, we first had to assemble a comprehensive training corpus of research papers on which we would train our models for detecting sentences that mention the implementation repository of proposed software, as well as base our heuristic rules for the sentence extraction step. The primary source for our data collection was the PapersWithCode platform, a well-known repository that links research papers with their corresponding code implementations. This platform was selected due to its extensive coverage of software engineering research, more specifically machine learning research, and its focus on promoting open access to code repositories.

Our data collection targeted research papers in the field of software engineering, encompassing a wide range of topics such as software development, software testing and software maintenance. The diversity of topics ensured that our training corpus captured a broad spectrum of software citation practices and repository mention styles. Specifically, we sourced 61 research papers from PapersWithCode, ensuring that these papers represented various subdomains within software engineering. This diverse selection was intended to make our model robust and generalizable across different types of research within the field.

The next step involved the annotation of the collected data. Each research paper was thoroughly examined to extract sentences that mentioned software repositories. These sentences were categorized based on the common formats and locations where repository mentions typically occur, such as full-text mentions, footnotes, and references. Figures 3.2, 3.3 and 3.4 show examples for each of the three types. This categorization helped in understanding the different contexts and styles of software citation used by researchers.

Each extracted sentence was then annotated with a binary label indicating whether it mentioned an implementation repository (1) or not (0). To ensure the accuracy and reliability of the annotations, the process was carried out in two stages. Initially, one annotator reviewed the sentences and assigned the binary labels.

Sentence	**Class**
Source code for all experiments presented in this paper is publicly available online1.	1
We use crowdsourcing2 to score the retrieved utterances	0

Table 3.1: Example sentences from training set

Subsequently, a second annotator reviewed these labels to verify their accuracy. Any disagreements or conflicts regarding the annotations were resolved through discussion until a consensus was reached.

If a decision could not be reached, a third person was included to facilitate resolution. This collaborative annotation process was essential to minimize errors and biases, ensuring the high quality of our training data.

To maintain the quality and relevance of the training corpus, specific inclusion criteria were established for the sentences. Sentences were included if they contained

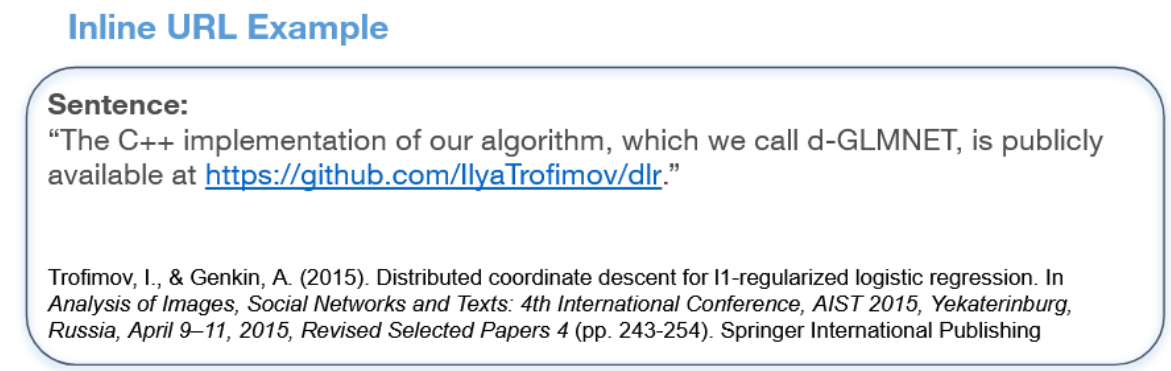

Figure 3.2: Mention sentence example - Inline URL

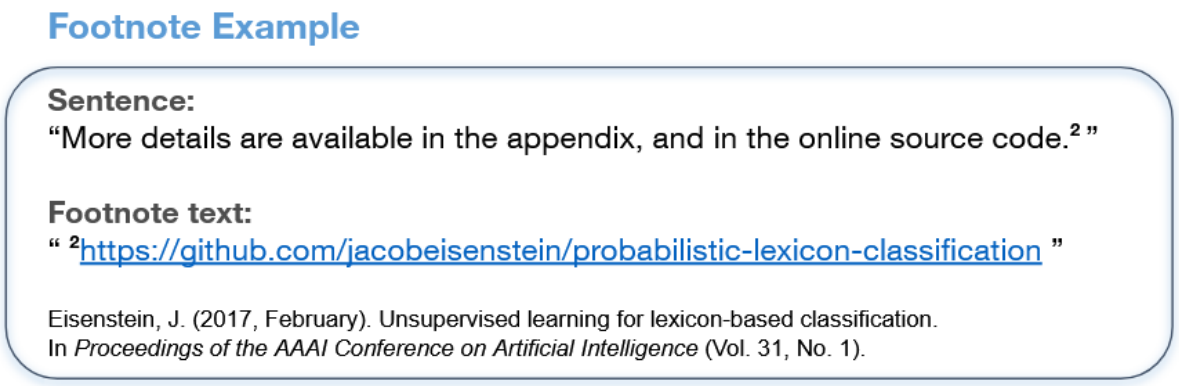

Figure 3.3: Mention sentence example - Footnote

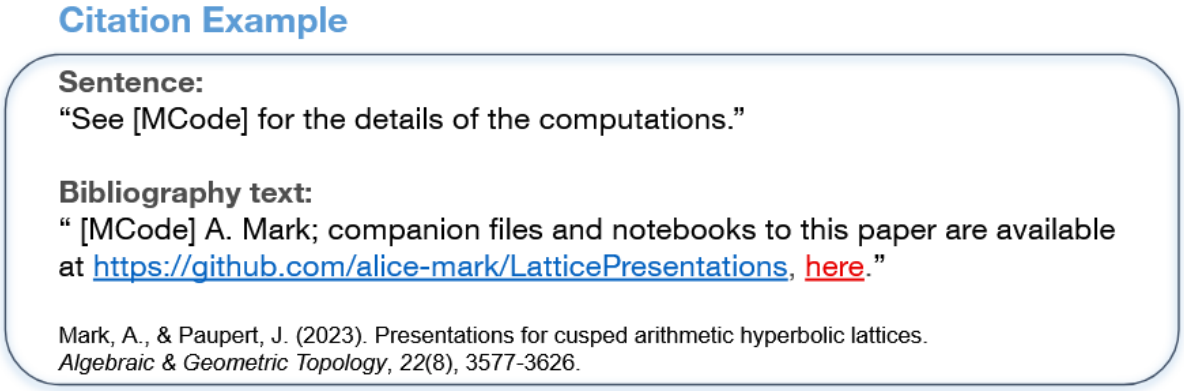

Figure 3.4: Mention sentence example - Citation and Hyperlink

clear mentions of implementation repositories, even if the context was somewhat ambiguous. However, sentences that lacked sufficient context or clarity to confirm a repository mention were excluded. One such example is the sentence "The detailed lists of the tuned parameters, as well as the detailed descriptions of the SUTs and the evaluated workloads, are accessible on the Web [45]" from the research paper that proposes a software called "BestConfig" by Zhu et al. [23] This selective approach ensured that the training data was both precise and representative of real-world citation

practices.

The annotated dataset [24], consisting of 75 sentences labeled as mentioning an implementation repository and approximately 2500 sentences labeled as non-mentioning, served as the foundation for training our language models. The number of sentences that mention the implementation repository are way lower than the non-mentioning sentences, hence the imbalance of the classes. Moreover, the three main mentioning styles are well represented, the distribution of which is shown in Figure 3.5.

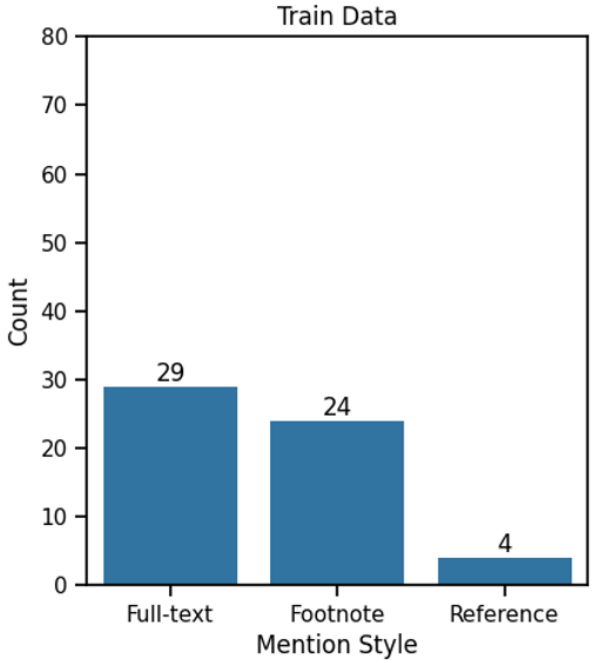

Figure 3.5: Distribution of implementation links and their style in training corpora

In summary, the data collection process for the RepoFromPaper project involved the systematic selection and annotation of research papers from the PapersWithCode platform. By focusing on software engineering topics and employing a rigorous annotation process, we created a high-quality training corpus essential for the development of our software citation detection models. This detailed and methodical approach to data collection is a cornerstone of the project's methodology, ensuring that our models are well-equipped to handle the complexities of software citation in scientific literature.

3.1.2 PDF-to-Text Conversion

The PDF-to-Text conversion step in the RepoFromPaper methodology serves as a crucial initial processing stage aimed at transforming PDF documents of research papers into machine-readable text format. This transformation is essential because it facilitates subsequent text-based analyses and processing steps, enabling the identification of sentences that mention software repository implementations.

The primary goal of the PDF-to-Text conversion is to extract plain text from PDF files obtained from academic repositories like PapersWithCode. These PDFs contain valuable information in the form of scientific text, figures, tables, and references.

By converting them into text format, we prepare the data for further computational analysis, including the application of heuristic rules for sentence extraction and input into models for identifying repository implementation mentions.

For the PDF-to-Text conversion, we utilize Apache Tika PDF reader[2], known for its speed and accurate extraction of text. Apache Tika was chosen over alternatives like GROBID due to its robust performance in handling various PDF formats, processing speed, and accurate representation of footnote content, which is critical for our methodology. While GROBID excels in structured document parsing tasks, Apache Tika's versatility and efficient handling of academic PDF layouts make it well-suited for our needs.

The integration with Apache Tika within our methodology ensures seamless conversion of PDF research papers into text data, preserving the integrity of textual content while discarding formatting elements specific to PDFs. This step is essential as it prepares the data for subsequent processing stages, including sentence extraction and classification using fine-tuned language models.

3.1.3 Sentence Extraction

The Sentence Extraction phase is the step where preprocessing of the input PDF raw text takes place, in order to extract complete sentences for subsequent analysis. During this phase, various functions contribute significantly to refining and organizing the PDF text, effectively segmenting it into well-formatted sentences, ready for input into the fine-tuned models for classification.

The decision to extract sentences rather than paragraphs is driven by the findings that paragraphs introduce significant noise, whereas sentence-level extraction enhances model learning by focusing on key information. These findings are the conclusion derived from the results of our initial model training and testing. The models were unable to differentiate the sentences when the whole paragraphs were used, while that was not the case with the other approach. Moreover, text cleaning is essential for ensuring uniformity and clarity in the extracted sentences for achieving superior performance. This involves removing newline characters, word breaks, extra white spaces, and inconsistencies in links. Additionally, reference number-text pairs, as well as footnote number-text pairs, are extracted if the footnote text is a link. This approach allows for the effective utilization of this information 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, often including hyphenation at the end of lines as a line break. To address this issue, fragmented sentences are consolidated by considering factors such as sentence beginnings and endings, newline characters, white spaces, and hyphenation, ensuring the formation of cohesive and complete sentences from fragmented text.

This step is based on heuristic rules developed from the patterns we observed in the training set and generalized as much as possible. These heuristic rules are designed to manage the various formats and idiosyncrasies found in research papers. However,

[2]https://github.com/chrismattmann/tika-python

they are still prone to errors due to the inherent complexity and variability of PDF documents and the ways in which text can be structured within them.

The sentence extraction process is implemented through a series of functions designed to handle these complexities. The key functions include:

- `extract_references`: This function identifies and extracts references from the PDF text, returning a dictionary with references and a list of non-references.
- `extract_footnotes`: This function extracts footnotes by looking for numbers in the text followed by a link, returning a dictionary with the number as the key and the link as the value.
- `extract_full_sentences`: This function extracts full sentences from paragraphs, identifying sentences that begin with a capital letter and end with a period.
- `combine_split_sentences`: This function combines split sentences by checking for sentences that begin with a capital letter but do not end with a period, and the subsequent sentence begins with a lowercase letter and ends with a period.

In the Table 3.2 below, we provide an example of what the expected extracted sentences following the aforementioned steps would be given the raw paragraph from a research paper [6].

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

Table 3.2: Full sentence extraction example

By employing these functions, the Sentence Extraction phase ensures that the input text is clean and well-structured, providing a robust foundation for the subsequent steps in the RepoFromPaper methodology.

3.1.4 Sentence Classification

Classifying sentences that contain implementation repository links is a key aspect of our methodology for extracting software repositories from scientific publications. In

this section, we explore the techniques used to identify and classify these sentences accurately.

We leverage state-of-the-art natural language processing (NLP) models, including BERT, SciBERT, and RoBERTa, which have been fine-tuned for this specific task. These models are renowned for their ability to understand and process complex textual data, making them well-suited for our needs. The following subsections will detail the training and fine-tuning processes of these models, the data preparation and annotation steps required for effective model training, and the evaluation metrics used to assess model performance.

3.1.4.1 BERT Models

The task of identifying sentences that contain implementation links in research papers is a binary text classification problem. Specifically, we aim to classify sentences into two categories: those that propose a software implementation and those that do not. To address this challenge, we employ advanced natural language processing (NLP) models that are well-suited for such classification tasks.

BERT [19], which stands for Bidirectional Encoder Representations from Transformers, is a groundbreaking NLP model developed by Google. It introduced a new way of pre-training language representations by jointly conditioning on both left and right context in all layers, allowing it to achieve state-of-the-art results on various NLP tasks. BERT models are pre-trained on a large corpus of text and then fine-tuned on specific tasks, such as sentence classification, named entity recognition, or question answering.

In our approach, BERT based models are fine-tuned to classify sentences extracted from scientific publications. By leveraging their ability to understand the context from both directions, BERT based models can accurately distinguish between sentences that contain implementation links and those that do not. This fine-tuning process is critical to adapting the pre-trained BERT based model to the specific nuances of scientific text, ensuring high precision and recall in our classification task.

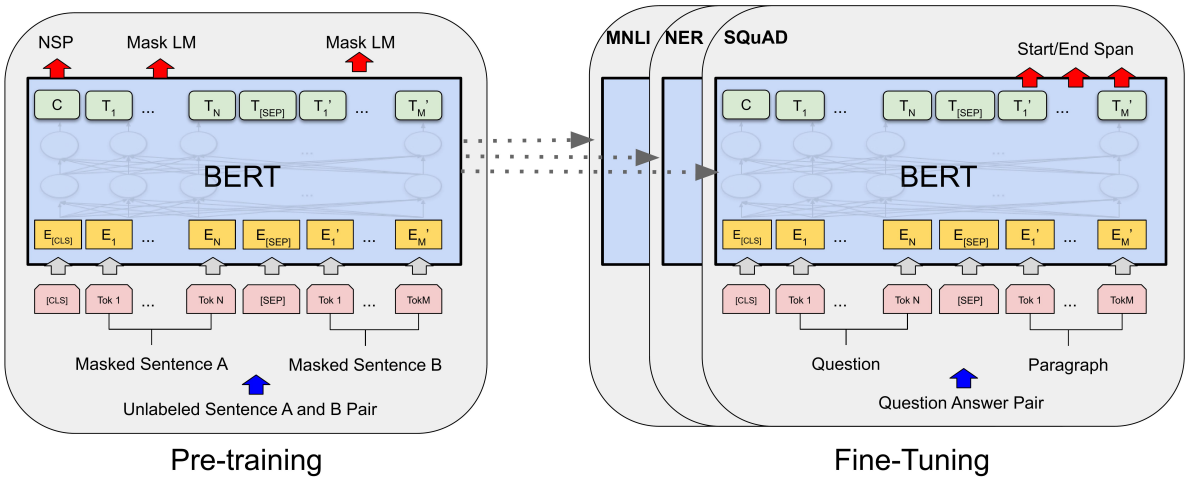

Figure 3.6: Bidirectional Encoder Representations from Transformers (BERT)[3]

We chose to experiment with three different models to ensure a comprehensive eval-

uation of our sentence classification task. BERT serves as a baseline model due to its well-established performance and wide usage in various NLP tasks. RoBERTa was selected because it offers improved performance over the original BERT by optimizing key hyperparameters and removing the next sentence prediction objective, making it more robust for general text processing. Finally, SciBERT was included to assess whether a model specialized in scientific text could outperform the other models, given that our input consists of sentences extracted from scientific publications. This comparative analysis allows us to determine the most effective model for accurately identifying implementation links in research papers.

BERT (Bidirectional Encoder Representations from Transformers): The BERT [19] model used in our methodology is the 'bert-base-uncased' variant. It consists of 12 layers, 768 hidden units, and 12 attention heads, totaling 110 million parameters. This model is pre-trained on the BookCorpus and English Wikipedia, making it highly effective at understanding the nuances of the English language.

RoBERTa (A Robustly Optimized BERT Pretraining Approach): RoBERTa [21] is an optimized version of BERT developed by Facebook AI. It modifies key hyperparameters in BERT, such as removing the next sentence prediction objective, training with much larger mini-batches and learning rates, and training on a larger corpus for a longer duration. The 'roberta-base' model used in our methodology follows this optimized approach, providing improved performance on various NLP tasks.

SciBERT (Scientific BERT): SciBERT [20] is a variant of BERT designed specifically for scientific text. It uses the same architecture as BERT but is pre-trained on a large corpus of scientific literature from the Semantic Scholar corpus, which includes papers from the domains of computer science, biomedicine, and other scientific fields. The 'allenai/scibert_scivocab_uncased' model used in our methodology leverages this domain-specific pre-training to better handle scientific terminology and structures commonly found in research papers.

3.1.4.2 Training Data and Fine-Tuning Process

We approach the problem of distinguishing between implementation mention 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 proposes an implementation of the paper. To achieve this, we utilized our pre-assembled training corpus, comprising sentences annotated with binary labels indicating the presence or absence of implementation mentions. The training data encompasses a variety of sentence structures and contexts, ensuring the models can generalize well to different ways researchers refer to repositories.

For model training, we employed the BERT, SciBERT, and RoBERTa models. These models were fine-tuned using a binary sequence classification setup, where the aim was to classify sentences as either implementation mentions or non-implementation sentences.

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, 'allenai/scibert_scivocab_uncased' for SciBERT, and the 'roberta-base' model for RoBERTa, initializing them with pre-trained weights.

The fine-tuning process involves several key steps:

- **Data Preparation**: Sentences from the training corpus are tokenized and converted into the format required by the models. This includes adding special tokens, padding, and creating attention masks.
- **Model Training**: The pre-trained models are fine-tuned on the training data using a binary cross-entropy loss function. The models are trained for a number of epochs, with the learning rate and other hyperparameters optimized for best performance.
- **Evaluation**: The performance of the models is evaluated using metrics such as accuracy, precision, recall, and F1 score. This helps in assessing how well the models can distinguish between implementation mention sentences and non-implementation sentences.

The fine-tuned BERT[4], SciBERT[5], and RoBERTa[6] models are available on the HuggingFace platform, providing easy access for researchers and developers to replicate and build upon our work. HuggingFace[7] is a leading AI company that provides state-of-the-art tools and libraries for natural language processing, making it easy to build and deploy machine learning models. Their platform includes a repository of pre-trained models, which can be fine-tuned for specific tasks, facilitating advanced NLP research and application development.

3.1.4.3 Inference and Sentence Ranking

Once our models, including BERT, SciBERT, and RoBERTa, have been fine-tuned on the annotated training corpus, the next step is inference and sentence ranking. This phase involves applying these models to classify all sentences extracted from a given research publication. The primary goal is to assess the likelihood of each sentence proposing an implementation of the paper, distinguishing between sentences that mention software repositories and those that do not.

Each sentence undergoes classification based on its contextual cues and linguistic features identified during the model training. BERT models, which leverage transformer architectures, SciBERT designed for scientific text, and RoBERTa known for robust pre-training, excel in understanding nuanced language patterns crucial for identifying implementation mentions. This process enables us to assign probabilities to each sentence, indicating its likelihood of belonging to the implementation or non-implementation class.

Following classification, the sentences are ranked according to the predicted probabilities of belonging to the positive class. This ranking strategy allows us to prioritize sentences that are most likely to contain implementation mentions. Specifically, we extract the top five sentences with the highest probability scores. This approach mitigates the risk of solely relying on the highest probability sentence, which may not consistently correlate with accurate proposal sentences due to varying linguistic contexts and model intricacies. An example of this scenario is given in Table 3.3 obtained

[4]https://huggingface.co/oeg/BERT-Repository-Proposal
[5]https://huggingface.co/oeg/SciBERT-Repository-Proposal
[6]https://huggingface.co/oeg/RoBERTa-Repository-Proposal
[7]https://huggingface.co/

by one of our initial model testing experiments on the research paper "An Empirical Study on Quality of Android Applications written in Kotlin language" by Bruno Gois Mateus and Matias Martinez [25]. In the provided example, where the sentences are given in ascending order, we can see that the sentence containing the implementation code repository link appears fourth i.e. has rank 4.

Top 5 Sentence Example
For that reason, we decided to study the open-source application available on GitHub
The rest of the apks files (11%) were not available
They analyzed 5,713 Android applications available on GitHub and published on Google Play
All the data presented in this paper is publicly available in our appendix: https://github.com/UPHF/kotlinandroid
To our knowledge, those are the three largest publicly available datasets of Android applications that contain applications recently released

Table 3.3: Ranked sentences example

This inclusive strategy accommodates potential deviations in model predictions, thereby improving the reliability and effectiveness of our approach in identifying software repository links within scientific literature. The sentence ranking step not only optimizes the extraction process but also ensures that the proposed software implementations are accurately captured, aligning with the objectives of the RepoFromPaper project.

In summary, the sentence classification step in the RepoFromPaper methodology leverages the power of state-of-the-art NLP models—BERT, SciBERT, and RoBERTa—to accurately identify sentences that mention implementation repositories. This process involves careful preparation of training data, fine-tuning of pre-trained models, and rigorous evaluation to ensure high performance. By utilizing these advanced models, we can effectively filter out non-relevant sentences and focus on those that propose implementation links, facilitating the accurate extraction of repository links from research papers.

3.1.5 Repository Link Extraction

The final step in our methodology is the extraction of repository links from the top-ranked sentences, footnotes, and references. This crucial phase is divided into two primary stages: searching for repository links within the top sentences and extending the search to footnotes and references.

3.1.5.1 Repository Link Search in Top Sentences

In the first stage, we focus on identifying repository links directly within the highest-ranked sentences. To achieve this, we utilize regular expressions designed to detect repository URLs hosted on GitHub and GitLab. Regular expressions are patterns used to match character combinations in strings, making them ideal for identifying structured URLs within text. Inline references to repositories are a common practice in academic papers, as noted by Howison and Bullard (2015) [2]. By focusing on

the top-ranked sentences that are most likely to mention repositories, we streamline the extraction process and enhance the precision of linking research papers to their corresponding software repositories.

The process involves applying a set of predefined regular expressions to each of the top-ranked sentences. If a repository link is found, it is extracted and recorded. In cases where multiple repository links are present within the top-ranked sentences, the first identified link is returned. This approach ensures that we capture the most relevant repository link directly associated with the top-probability sentences, thus establishing a clear connection between the predicted proposal sentences and their corresponding repositories.

3.1.5.2 Repository Link Search in Footnotes and References

In the second stage, we extend our search scope to include footnotes and references. This step aims to capture repository links that may not be directly mentioned within the top sentences but are instead referenced through footnote or reference numbers. Research papers often use these numbers to point readers to supplementary information, including software repositories. This second step is executed only if the first step yields no results, ensuring that direct mentions within the top-ranked sentences are prioritized before expanding the search to these additional elements.

To achieve this, we first identify sentences that contain footnote or reference numbers. These numbers are indicative of additional information sources commonly associated with research papers. We retain the order of appearance of these numbers to prioritize sentences with higher classification probabilities. Once potential footnote or reference numbers are identified, we proceed to search for repository links within the corresponding footnotes and references.

The search process involves examining each identified footnote or reference number and retrieving any associated repository links. This is accomplished by matching the numbers to their corresponding entries in the footnotes and references sections. If a repository link is found, it is extracted and recorded. We also handle variations in how reference numbers are presented, such as with or without brackets, ensuring a comprehensive search.

Additionally, we consider sentences that contain footnotes directly referencing repository links. For instance, a sentence might include a footnote number followed by a repository link. By extracting such links, we ensure that all potential sources of repository information are covered.

To integrate these steps into a cohesive process, we employ a function that consolidates sentences containing footnotes with potential repository links. This function ensures that all relevant sentences are considered, maximizing the likelihood of accurately extracting repository links from the publication.

Through these comprehensive steps, our methodology effectively links top-ranked sentences to their corresponding code repositories, ensuring a robust and reliable extraction of implementation links from academic papers.

3.2 RepoFromPaper Tool Availability and Structure

To facilitate the practical application of the RepoFromPaper methodology, we have packaged the entire process into a tool that is available on GitHub. This tool is designed to help researchers and developers automatically detect and extract proposed repository links from academic papers using a combination of natural language processing and heuristic techniques. The tool's source code, usage instructions, and installation guidelines are accessible online, promoting transparency and ease of use[8] [22].

3.2.1 Installation and Usage

The package has only three dependencies to the external tools transformers[9], tika[10] and torch[11]. To install the package, the following command can be used from the root folder:

```
pip install .
```

The main function of the package, `extract_repo_links_from_pdf`, takes the local path to the PDF as input and returns a list of the best-matched sentences and the repository link if found. Below is an example of how to use the package:

```
from RSEF.repofrompaper.rfp.main import extract_repo_links_from_pdf

pdf_path = "path/to/pdf"
best_sentences, repo_link = extract_repo_links_from_pdf(pdf_path)
```

3.2.2 Integration in RSEF

The Research Software Extraction Framework (RSEF)[12] [5] is a tool designed to find and verify the connection between scientific papers and software repositories. RSEF accomplishes this by locating URLs of software repositories within research papers, extracting metadata from these repositories, and checking for any associated links back to the original papers. This process establishes either a unidirectional or bidirectional link, depending on whether the repository metadata points back to the paper. The bidirectional metric indicates a stronger validation, as it confirms that both the paper mentions the repository and the repository acknowledges the paper.

3.2.2.1 Bidirectional Repository Link Search

The bidirectional link search in RSEF aims to find instances where the research paper mentions the repository and, reciprocally, the repository's metadata cites the research paper. This is achieved by scanning both the text of the paper and the repository metadata. The search targets repository platforms like GitHub and Zenodo, ensuring comprehensive coverage of commonly used research repositories.

[8] RepoFromPaper is available at `https://github.com/StankovskiA/RepoFromPaper`
[9] https://pypi.org/project/transformers/
[10] https://pypi.org/project/tika/
[11] https://pypi.org/project/torch/
[12] `https://github.com/SoftwareUnderstanding/RSEF/`

3.2.2.2 Integration of RepoFromPaper into RSEF

RepoFromPaper (RFP) is seamlessly integrated into RSEF to enhance the framework's capability in extracting repository links from research papers. The integration process involves utilizing RFP's natural language processing and heuristic techniques to identify repository links within the text of the papers.

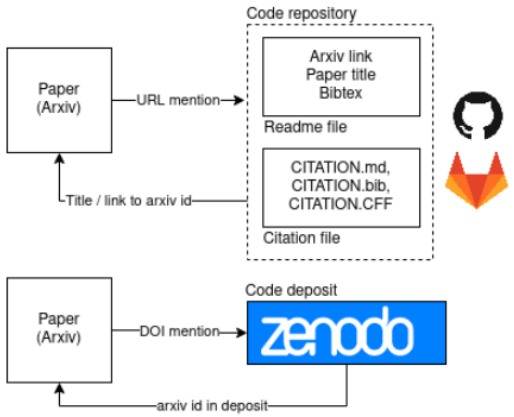

Figure 3.7: Bidirectional Repository Link Search

Usage in RSEF

RFP is seamlessly integrated into RSEF and is invoked using the "-U" flag within the "assess" command to evaluate the presence of repository links in research papers. Utilizing the SciBERT model, RFP classifies and ranks sentences based on their likelihood of mentioning a repository. It then employs regular expressions to extract repository URLs directly from the top-ranked sentences. If no direct links are found, RFP extends its search to include footnotes and references, ensuring that even indirectly mentioned repository links are captured. This comprehensive approach enhances RSEF's capability to locate unidirectional links, complementing the bidirectional link search and improving the overall accuracy and validation of linking papers to repositories. RFP either returns the repository link along with the sentence where it was found, or indicates that no link has been identified.

Benefits of RFP in RSEF

Integrating RFP into RSEF offers several advantages: RFP's advanced NLP models improve the accuracy of identifying sentences that mention repositories, ensuring that relevant links are not overlooked. By extending the search to footnotes and references, RFP ensures that repository links mentioned indirectly are also captured, providing comprehensive coverage. Moreover, RFP operates solely on the text of the PDF, requiring no external resources to detect links. This integration allows RSEF to effectively locate unidirectional links that might be overlooked by the bidirectional approach, thereby improving the overall validation process by capturing all possible connections between papers and repositories.

The integration of RepoFromPaper into RSEF significantly enhances the framework's ability to link research papers with their software repositories accurately and efficiently, thereby supporting the broader goal of improving research transparency and reproducibility.

RSEF Link Search Output

The RSEF assess command offers flexibility to run unidirectional, bidirectional, or both types of searches simultaneously. This command evaluates the presence of repository links in research papers and outputs the results in a structured format. Below is an example JSON structure illustrating the output format of the analysis for one paper:

```
{
    "title": "",
    "implementation_urls": [
        {
            "identifier": "",
            "type": "",
            "paper_frequency": N,
            "extraction_methods": [
                {
                    "type": "",
                    "location": "",
                    "location_type": "",
                    "source": "",
                    "source_paragraph": ""
                }
            ]
        }
    ],
    "doi": "",
    "arxiv": "",
    "abstract": "",
    "file_name": "",
    "file_path": ""
}
```

Each field in the JSON represents specific information associated with the research paper and its related repository links. The "implementation_urls" array captures details such as the repository identifier, type (e.g., Git, Zenodo), frequency of occurrence in the paper, and the extraction methods used (e.g., regex, bidirectional). This structured output facilitates further analysis and validation of the links identified by RSEF, supporting the goal of enhancing research transparency and reproducibility.

Chapter 4

Evaluation

To assess the performance of our methodology, we conducted multiple experiments over a validation corpus. These experiments allowed us to evaluate and understand the effectiveness and robustness of our approach.

4.1 Evaluation Corpora

For training our models, as described in Section 3.1.1, 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 [26] 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.

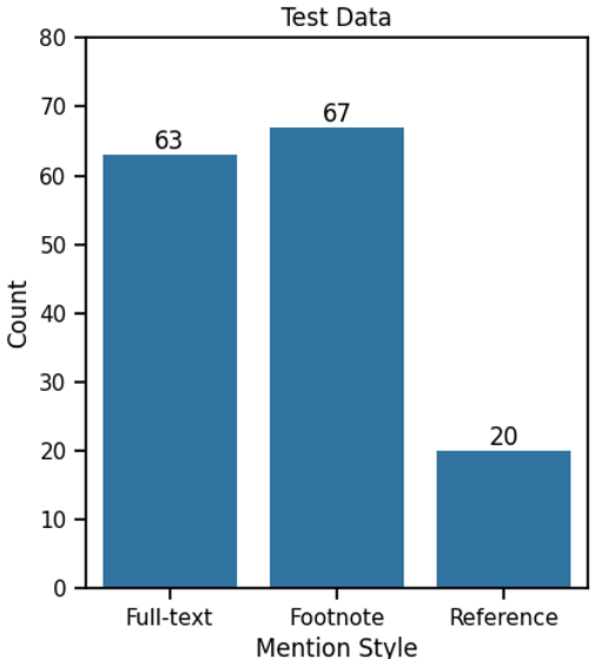

Figure 4.1: Distribution of implementation links and their style in testing corpora

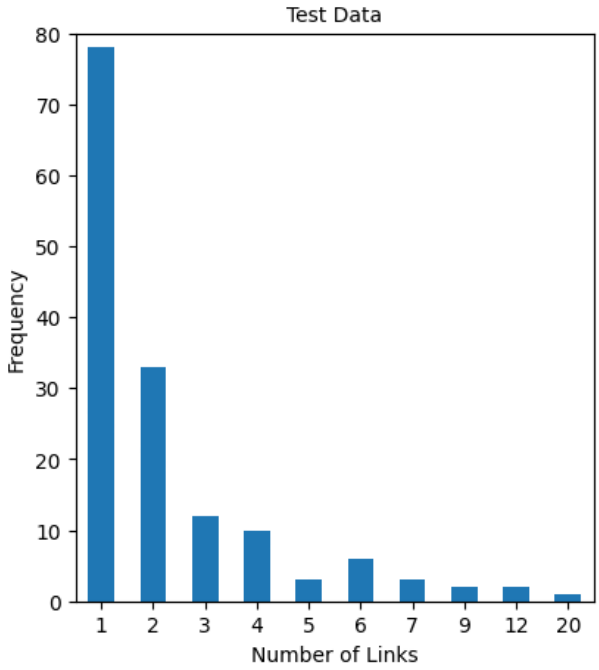

Figure 4.2: Number of code repository links in test papers

The evaluation set was manually tagged 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 consist of papers that encompass the three main mention types: *Full-text* (i.e., inline URLs), *Footnote*, and *Reference* mentions. Figure 4.1 shows the number of mention types present in the test set, which follow a similar distribution. Figure 4.2 shows the frequency distribution of repository links found in the papers of the testing corpora. This distribution sheds light on the challenges associated with automatically identifying and extracting the correct implementation repository links from research papers, as 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.

4.2 Evaluation Methods

To evaluate the performance of RepoFromPaper (RFP), we employ two main evaluation methods: Mean Reciprocal Rank (MRR) and the combined metrics of Precision, Recall, and F1 Score. Each of these metrics provides valuable insights into the effectiveness of our approach.

4.2.1 Mean Reciprocal Rank (MRR)

Mean Reciprocal Rank (MRR) [27] serves as a key evaluation metric for gauging the individual performance of our 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:

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i} \tag{4.1}$$

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.

Example:

Consider a scenario where we have five research papers, and the correct proposal sentences for these papers are ranked as follows:

- Paper 1: Correct sentence ranked 1st
- Paper 2: Correct sentence ranked 2nd
- Paper 3: Correct sentence ranked 5th
- Paper 4: Correct sentence ranked 3rd
- Paper 5: Correct sentence not in the top 5 (rank considered as infinity)

To calculate the MRR, we first compute the reciprocal rank for each paper:

- Paper 1: $\frac{1}{1} = 1$
- Paper 2: $\frac{1}{2} = 0.5$
- Paper 3: $\frac{1}{5} = 0.2$
- Paper 4: $\frac{1}{3} = 0.33$
- Paper 5: $\frac{1}{\infty} = 0$

Next, we sum these reciprocal ranks and divide by the number of papers (N = 5):

$$\text{MRR} = \frac{1}{5} (1 + 0.5 + 0.2 + 0.33 + 0) = \frac{1}{5} \times 2.03 = 0.406 \tag{4.2}$$

Thus, the Mean Reciprocal Rank for this set of papers is 0.406, indicating the average performance of the model in ranking the correct proposal sentences. A higher MRR number signifies better model performance, with MRR of 1 being the highest i.e. best a model can perform and MRR of 0 being the lowest i.e. the worst a model can perform.

4.2.2 Precision, Recall, and F1 Score

To comprehensively evaluate the overall performance of RepoFromPaper, we employ precision, recall, and F1 score metrics. These metrics are defined as follows:

- **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 pipeline 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 Results

To evaluate the performance of our methodology, we conducted a series of experiments using fine-tuned BERT, SciBERT, and RoBERTa models on our test set. This section presents the detailed results of these experiments.

4.3.1 Evaluation of Fine-Tuned Models

The evaluation focused on the models' ability to accurately identify implementation mentions within research papers. The results, summarized in Table 4.3.1, indicate a high level of accuracy, with the SciBERT model demonstrating superior performance. Additionaly, by selecting the top five sentences, our methodology enhanced the robustness of implementation mention extraction from 80% to 94% accuracy.

BERT Model

The fine-tuned BERT model achieved a precision of 0.864, recall of 0.871, F1 score of 0.867, and an MRR of 0.55. This indicates a solid performance in identifying relevant sentences that mention repositories. The BERT model's performance is noteworthy given its foundational architecture, which has been widely adopted for various NLP tasks.

RoBERTa Model

RoBERTa, another model evaluated in our experiments, showed improved results compared to BERT. It achieved a precision of 0.942, recall of 0.929, F1 score of 0.936, and an MRR of 0.753. The higher precision and recall scores of RoBERTa suggest its robustness in accurately identifying implementation mentions, which is reflected in its higher F1 score and MRR. This improvement can be attributed to RoBERTa's enhanced pre-training strategies, which involve more extensive data and longer training times compared to BERT.

SciBERT Model

The SciBERT model outperformed both BERT and RoBERTa, achieving the highest precision of 0.944, recall of 0.950, F1 score of 0.947, and MRR of 0.85. SciBERT's superior performance is a result of its architecture, which is specifically optimized for scientific text. The high F1 score and MRR indicate that SciBERT is particularly effective at ranking the correct implementation sentences at the top, thus facilitating accurate repository link identification.

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**

Table 4.1: Evaluation results for the fine-tuned models

4.3.2 Comparison with Baseline Performance

To further understand the significance of these results, we compared the performance of our best model, SciBERT, against a baseline method. The baseline method employed regular expressions to identify the most frequent code repository link in a publication, defaulting to the first code repository if all links appeared only once. The baseline performance metrics are presented in Table 4.3.2.

The regex baseline method achieved a precision of 0.793, recall of 1, and F1 score of 0.88. While the recall was perfect, indicating that the baseline method identified all relevant links, its precision was relatively lower. This suggests that the baseline method, while thorough, also introduced a higher number of false positives, leading to a lower precision score.

In contrast, the SciBERT model exhibited a precision of 0.944, recall of 0.950, and F1 score of 0.947. Although the recall was slightly lower than the baseline, the precision was significantly higher by 15%, and the F1 score improved by 6%. This comparison highlights the effectiveness of the SciBERT model in not only identifying the relevant links but also in reducing the number of false positives, thus achieving a better balance between precision and recall.

Model	Precision	Recall	F1 Score
Regex Baseline	0.793	1	0.88
SciBERT	**0.944**	**0.950**	**0.947**

Table 4.2: Comparison with baseline performance

Overall, the results from these experiments provide valuable insights into the performance of our fine-tuned models and the overall methodology in accurately identifying implementation mentions within research papers. The SciBERT model, in particular, stands out for its superior precision and balanced performance, demonstrating the benefits of using advanced, domain-specific language models for this intricate task.

Chapter 5

Application and Comparative Analysis

5.1 Application of RepoFromPaper on AI Papers

In this section, we detail the creation and analysis of a corpus of research papers along with their corresponding implementation repositories. This work was facilitated by our method, RepoFromPaper[1] [22], which we applied to nearly 1800 research papers submitted in the years 2022 and 2023 from the Artificial Intelligence section on Arxiv.org.[2] Our method was tested using both fine-tuned RoBERTa and SciBERT models.

We began by applying RepoFromPaper with the fine-tuned RoBERTa and SciBERT models to the selected AI papers. The results reaffirmed the leading performance of the SciBERT model. The SciBERT model successfully detected and extracted 604 implementation repository links, whereas the RoBERTa model identified 585 links. These results are made publicly available for further research and verification [28].

5.1.1 Results of AI Papers Experiment

The application of RepoFromPaper on nearly 1800 AI research papers yielded the following results:

- The SciBERT model detected and extracted 604 implementation repository links.
- The RoBERTa model detected and extracted 585 implementation repository links.

These results indicate that the SciBERT model, which is specifically designed for scientific text, once again had superior performance over the more generalized RoBERTa model in identifying implementation links within AI research papers.

5.2 Comparison with Bi-Directional Link Detection

To evaluate the effectiveness of our approach, we compared it against the RSEF method that detects bi-directional links between papers and code repositories [5].

[1] RepoFromPaper is available at `https://github.com/StankovskiA/RepoFromPaper`

[2] `https://arxiv.org/list/cs.AI/recent`

This comparison was performed on 150 research papers from the Software Engineering category on Arxiv.org.[3] These papers were randomly selected from the year 2023 and may or may not include a code implementation link.

5.2.1 Comparison Methodology

We applied both our method (using the SciBERT model) and the bi-directional link detection method to these 150 papers. The results were as follows:

- The bi-directional approach found 4 unique implementation links.
- RepoFromPaper identified 41 unique implementation links.
- Both methods identified 16 common implementation links.
- In 89 publications, neither method found an implementation link.

The full results of this comparison are available online [29].

5.2.2 Analysis of Results

Our findings indicate that our approach can extract 25% more implementation repository links compared to the bi-directional approach. This is a significant improvement, demonstrating the robustness and effectiveness of our methodology.

We observed an expected overlap in the extracted links between both approaches, where both methods successfully extracted the same implementation link. However, we also identified unique links detected by each method. This divergence arises from two main reasons:

1. Our approach can detect uni-directional links between a research paper and a repository, whereas the bi-directional approach requires the repository to also point back to the paper. This requirement can lead to missed links in the bi-directional approach.
2. The bi-directional approach can return multiple confirmed links, as many authors separate code implementation and evaluation results into different repositories. In contrast, 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. By combining the strengths of both methods, we can aim to extract as many correct implementation repository links from research papers as possible. This complementary nature enhances the overall capability of linking research papers with their corresponding implementation repositories, contributing to better research transparency and reproducibility.

5.3 Application of RepoFromPaper on Non-CS Papers

In order to assess the robustness of RepoFromPaper, we applied it on a total of 2,000 non-computer science papers from the quantitative biology section of arXiv[4] and pub-

[3] https://arxiv.org/list/cs.SE/recent
[4] https://arxiv.org/list/q-bio/recent

lished in the years 2023 and 2024. We managed to find a total of 335 repository links. The list of the papers which was used as input for the RSEF analysis alongside the RSEF outputs of the analysis are available on Zenodo [30].

We ran RepoFromPaper on the collected papers using the assess command from RSEF, performing both the bidirectional (bidir) and unidirectional (unidir) link searches. The bidirectional search identifies links where both the paper and the repository reference each other, while the unidirectional search detects links where only the paper mentions the repository.

5.3.1 Observations

Number of Links Found Per Paper

We found a total of 335 repository links across the 2,000 papers. Out of these, 258 papers had only one link, 37 papers had 2 links, and one paper had 3 links. This distribution is illustrated in Figure 5.1. It is important to note that the unidirectional RepoFromPaper approach can return at most one link per paper, while the bidirectional approach can return more than one confirmed link.

Type of Repository Links

Among the 335 repository links found, 305 were Git-based (Github/Gitlab), and 30 were Zenodo links. This distribution is shown in Figure 5.2. The unidirectional approach can only find Git-based links, while the bidirectional approach can find both types of links.

Links Found by Each Method

We analyzed how many links were found by each method (Bidirectional and Unidirectional RepoFromPaper), and how many were found by both approaches. The results are shown in Figure 5.3. Specifically, 205 links were found by the unidirectional method only, 71 links were found by both methods, and 59 links were found by the bidirectional method only.

Locations of Bidirectional Links

The bidirectional method not only analyzes the raw text from the research paper PDF but also searches for a link back to the paper in multiple locations. The frequencies of the bidirectional link confirmations in various locations are illustrated in Figure 5.4. The locations include Citation in README, README text, Description, Repository title, Zenodo, Related papers, README BibTeX, Citation file, and others.

The specific counts are as follows: Citation in README (43), README text (32), Description (42), Repository title (35), Zenodo (40), Related papers (52), README BibTeX (14), Citation file (3), File CFF (3), and File BibTeX (2).

These results indicate that citation practices in research papers are still developing and vary significantly. Even though some locations, such as the README or description, are more popular than others, there is still heterogeneity in the locations where citations are found. This distribution of confirmed links highlights the lack of standardized practices for citing repositories, despite the recognized importance of proper

citation for research reproducibility and transparency.

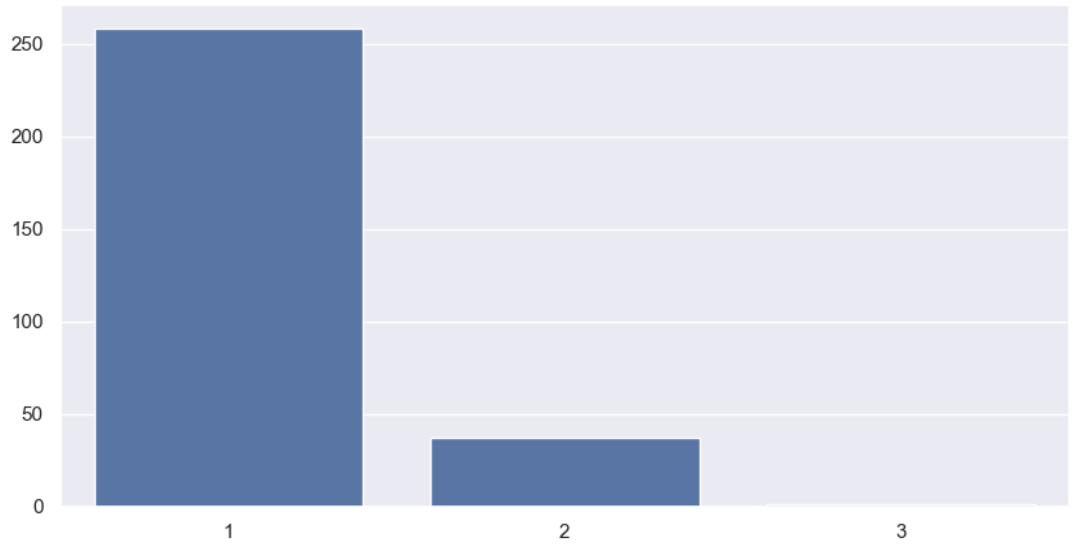

Figure 5.1: Number of Links Found Per Paper

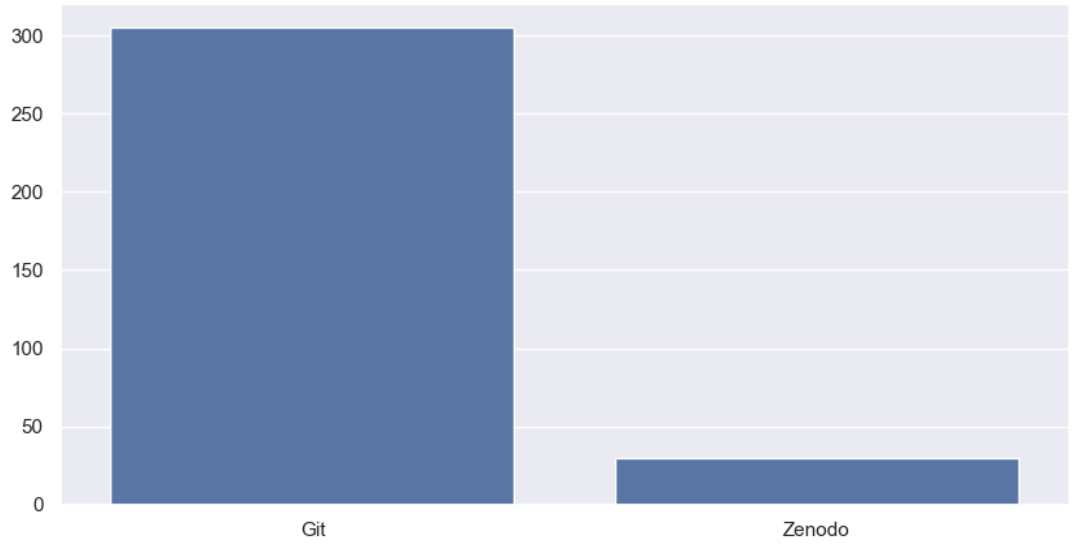

Figure 5.2: Types of Repository Links Found

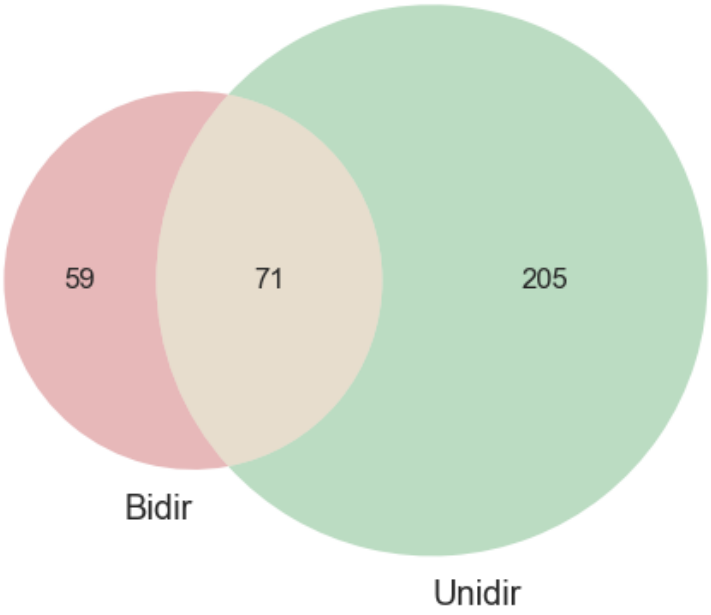

Figure 5.3: Comparison of Links Found by Each Method

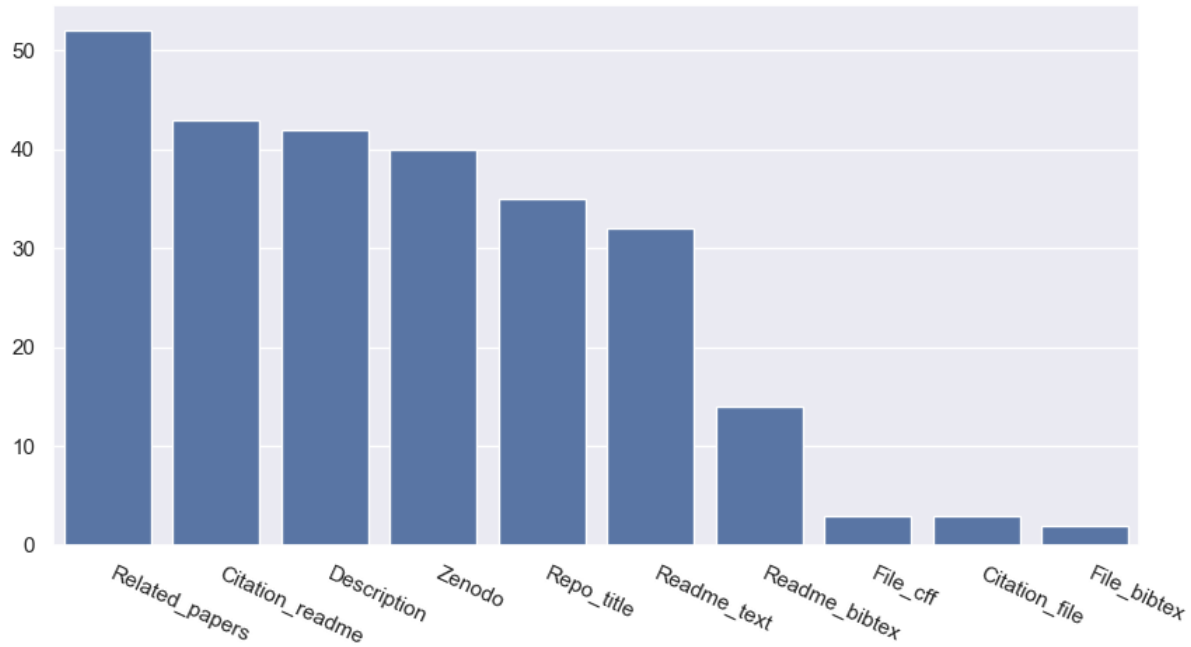

Figure 5.4: Frequencies of Bidirectional Link Confirmations by Location

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We introduced RepoFromPaper, a comprehensive methodology and tool for automatic extraction of implementation repository links from research papers. By leveraging fine-tuned language models, we aim to enhance the identification and linking of research software. Our evaluation results demonstrate the efficacy of our approach, particularly with the SciBERT model, which achieved high precision, recall, and F1 scores. This indicates a significant improvement in identifying implementation mentions within research papers compared to traditional regex-based methods.

The integration of RepoFromPaper into the Research Software Extraction Framework (RSEF) further validated its effectiveness. By applying our method to a diverse set of research papers, we demonstrated that RepoFromPaper not only excels in identifying unidirectional links but also complements the bidirectional link search within RSEF. This dual approach ensures a more comprehensive validation of the connections between research papers and their corresponding software repositories.

6.2 Challenges and Limitations

One of the primary challenges is the pre-processing step of converting PDFs to text. Accurate sentence extraction is critical for the pipeline's performance, and variations in PDF formatting can introduce noise, leading to potential errors in identifying relevant sentences. While our current method performs well, there is room for improvement in handling diverse PDF formats more effectively.

Another limitation lies in the structure and diversity of sentences used in the training data. Although SciBERT has shown superior performance in our experiments, the model's effectiveness is heavily dependent on the variety and representativeness of the training sentences. If the training data lacks sufficient examples from different research fields or fails to capture the diverse ways in which implementation repositories are mentioned, the model's ability to generalize across all domains may be compromised. Ensuring that the training data includes a comprehensive range of sentence structures and mention styles is crucial for the models to adapt effectively to varied research contexts.

Additionally, RepoFromPaper is currently limited to returning only one repository link per paper, even though there might be multiple correct implementation code repository links mentioned. This restriction can lead to incomplete results, especially for papers that cite several tools or libraries. Expanding the capability to identify and return multiple relevant repository links is a necessary enhancement for future iterations of the project.

Moreover, RepoFromPaper is restricted to identifying repository links hosted on GitHub and GitLab. This limitation excludes other platforms such as Bitbucket, Zenodo, SourceForge, and institutional repositories, potentially omitting relevant implementation links. Broadening the scope to include various repository hosting services would improve the comprehensiveness and utility of the tool.

6.3 Future Work

There are several avenues for future work that can enhance the robustness and applicability of RepoFromPaper:

6.3.1 Expanding the Training Dataset

Expanding the training dataset to include a broader range of proposal mention variations can improve the model's ability to recognize diverse ways of referencing repositories. This expansion should encompass different research domains, styles of writing, and types of implementation mentions. A more extensive and varied training dataset will enhance the models' generalization capabilities, making them more effective across different disciplines.

6.3.2 Improving PDF-to-Text Conversion

Investigating more advanced PDF-to-Text conversion techniques is another crucial area for future work. Current methods can struggle with the heterogeneity of PDF formats, leading to inaccuracies in text extraction. Exploring state-of-the-art conversion tools and techniques can help overcome these challenges, ensuring that the extracted text is as clean and accurate as possible. Improved conversion will lead to more reliable sentence extraction, ultimately enhancing the overall effectiveness of RepoFromPaper.

6.3.3 User Feedback and Iterative Improvement

Finally, incorporating user feedback into the development cycle can drive iterative improvements. By engaging with researchers and practitioners who use RepoFromPaper, we can gather valuable insights into its strengths and weaknesses. This feedback loop will allow us to continuously refine and enhance the tool, ensuring that it meets the evolving needs of the research community.

6.4 Broader Impact and Future Directions

The broader impact of RepoFromPaper extends beyond the immediate scope of this project. By facilitating easier access to implementation repositories, we contribute

to the advancement of open science and reproducible research. One significant impact is the ability to conduct extended analyses to assess best practices in different scientific communities. This can provide valuable insights into how various fields approach software citation and implementation sharing, potentially guiding the development of standardized practices.

Future directions include not only improving the current methodology but also exploring its application in new and emerging fields. By continuously refining the extraction process and expanding the scope to include diverse repository hosting services, we aim to push the boundaries of what is possible in automated research tool development. Additionally, enhancing the tool to handle multiple repository links and incorporating more sophisticated natural language processing techniques will further improve its accuracy and utility.

Intelligent systems have the potential to address inconsistencies in citation practices by automating the identification and standardization of citations. Our work, through RepoFromPaper, lays the groundwork for such systems. By developing a bot that detects repository links and automatically fills in citation information according to best practices, we can improve the uniformity and accessibility of research software citations. This would enhance the discoverability and use of research software, contributing to the integrity and transparency of scientific research.

By addressing these challenges and exploring these avenues, RepoFromPaper can significantly contribute to the landscape of scientific research, promoting transparency, accessibility, and collaboration across disciplines.

Bibliography

[1] N. P. Chue Hong, D. S. Katz, M. Barker, A.-L. Lamprecht, C. Martinez, F. E. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, T. Honeyman, A. Struck, A. Lee, A. Loewe, B. van Werkhoven, C. Jones, D. Garijo, E. Plomp, F. Genova, H. Shanahan, J. Leng, M. Hellström, M. Sandström, M. Sinha, M. Kuzak, P. Herterich, Q. Zhang, S. Islam, S.-A. Sansone, T. Pollard, U. D. Atmojo, A. Williams, A. Czerniak, A. Niehues, A. C. Fouilloux, B. Desinghu, C. Goble, C. Richard, C. Gray, C. Erdmann, D. Nüst, D. Tartarini, E. Ranguelova, H. Anzt, I. Todorov, J. McNally, J. Moldon, J. Burnett, J. Garrido-Sánchez, K. Belhajjame, L. Sesink, L. Hwang, M. R. Tovani-Palone, M. D. Wilkinson, M. Servillat, M. Liffers, M. Fox, N. Miljković, N. Lynch, P. Martinez Lavanchy, S. Gesing, S. Stevens, S. Martinez Cuesta, S. Peroni, S. Soiland-Reyes, T. Bakker, T. Rabemanantsoa, V. Sochat, Y. Yehudi, and R. F. WG, "FAIR Principles for Research Software (FAIR4RS Principles)," June 2022.

[2] J. Howison and J. Bullard, "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*, vol. 67, no. 9, pp. 2137–2155, 2016.

[3] A. M. Smith, D. S. Katz, K. E. Niemeyer, and FORCE11 Software Citation Working Group, "Software citation principles," *PeerJ Computer Science*, vol. 2, p. e86, Sept. 2016.

[4] S. Druskat, J. H. Spaaks, N. Chue Hong, R. Haines, J. Baker, S. Bliven, E. Willighagen, D. Pérez-Suárez, and O. Konovalov, "Citation File Format," Aug. 2021.

[5] D. Garijo, M. Arroyo, E. Gonzalez, C. Treude, and N. Tarocco, "Bidirectional paper-repository tracing in software engineering," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pp. 642–646, IEEE, 2024.

[6] A. Stankovski and D. Garijo, "Repofrompaper: An approach to extract software code implementations from scientific publications," 2024.

[7] R. Heumüller, S. Nielebock, J. Krüger, and F. Ortmeier, "Publish or perish, but do not forget your software artifacts," *Empirical Software Engineering*, vol. 25, no. 6, pp. 4585–4616, 2020.

[8] A. M. Smith, D. S. Katz, and K. E. Niemeyer, "Software citation principles," *PeerJ Computer Science*, vol. 2, p. e86, 2016.

[9] D. S. Katz, D. Bouquin, N. P. C. Hong, J. Hausman, C. Jones, D. Chivvis, T. Clark, M. Crosas, S. Druskat, M. Fenner, *et al.*, "Software citation implementation challenges," *arXiv preprint arXiv:1905.08674*, 2019.

[10] D. S. Katz and N. P. Chue Hong, "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.

[11] D. S. Katz, N. P. C. Hong, T. Clark, A. Muench, S. Stall, D. Bouquin, M. Cannon, S. Edmunds, T. Faez, P. Feeney, *et al.*, "Recognizing the value of software: a software citation guide," *F1000Research*, vol. 9, 2020.

[12] C. Du, J. Cohoon, P. Lopez, and J. Howison, "Softcite dataset: A dataset of software mentions in biomedical and economic research publications," vol. 72, p. 870–884, jun 2021.

[13] D. Schindler, F. Bensmann, S. Dietze, and F. Krüger, "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*, CIKM '21, (New York, NY, USA), p. 4574–4583, Association for Computing Machinery, 2021.

[14] J. Lin, Y. Wang, Y. Yu, Y. Zhou, Y. Chen, and X. Shi, "Automatic analysis of available source code of top artificial intelligence conference papers," *International Journal of Software Engineering and Knowledge Engineering*, vol. 32, no. 07, pp. 947–970, 2022.

[15] H. Hata, J. L. Guo, R. G. Kula, and C. Treude, "Science-software linkage: the challenges of traceability between scientific knowledge and software artifacts," *arXiv preprint arXiv:2104.05891*, 2021.

[16] S. Wattanakriengkrai, B. Chinthanet, H. Hata, R. G. Kula, C. Treude, J. Guo, and K. Matsumoto, "Github repositories with links to academic papers: Public access, traceability, and evolution," *Journal of Systems and Software*, vol. 183, p. 111117, 2022.

[17] A. Inokuchi, Y. S. Nugroho, S. Wattanakriengkrai, F. Konishi, H. Hata, C. Treude, A. Monden, and K. Matsumoto, "From academia to software development: publication citations in source code comments," *arXiv preprint arXiv:1910.06932*, 2019.

[18] A. Kelley and D. Garijo, "A framework for creating knowledge graphs of scientific software metadata," *Quantitative Science Studies*, vol. 2, no. 4, pp. 1423–1446, 2021.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[20] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[22] A. Stankovski, "Stankovskia/repofrompaper: v1.0.1," Apr. 2024.

[23] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, ACM, Sept. 2017.

[24] A. Stankovski, "PapersWithCode-Corpus Repository Proposal Sentences Training Dataset," Feb. 2024.

[25] B. Góis Mateus and M. Martinez, "An empirical study on quality of android applications written in kotlin language," *Empirical Software Engineering*, vol. 24, p. 3356–3393, June 2019.

[26] A. Stankovski, "RepoFromPaper Repository Implementation Link Testing Dataset," Apr. 2024.

[27] N. Craswell, "Mean reciprocal rank," pp. 1703–1703, Springer US, 2009.

[28] A. Stankovski, "Rfp output on csai papers from 2022/23," Apr. 2024.

[29] A. Stankovski, "Repofrompaper comparison with bidir method," Apr. 2024.

[30] A. Stankovski, "Rsef output on non-cs papers from 2023/24," July 2024.