

Detecting Technical Debt Using Natural Language Processing Approaches - A Systematic Literature Review

Edi Sutoyo^{a,b,*}, Andrea Capiluppi^a

^a*Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Nijenborgh 9, Groningen, 9747 AG, The Netherlands*

^b*Department of Information Systems, Telkom University, Bandung, 40257, West Java, Indonesia*

Abstract

Context: Technical debt (TD) is a well-known metaphor for the long-term effects of architectural decisions in software development and the trade-off between producing high-quality, effective, and efficient code and meeting a release schedule. Thus, the code degrades and needs refactoring. A lack of resources, time, knowledge, or experience on the development team might cause TD in any software development project.

Objective: In the context of TD detection, NLP has been utilized to identify the presence of TD automatically and even recognize specific types of TD. However, the enormous variety of feature extraction approaches and algorithms (machine learning or deep learning) employed in the literature often hinders researchers from trying to improve their performance.

Method: In light of this, this systematic literature review proposes a taxonomy of feature extraction techniques, and algorithms used in technical debt detection: its objective is to compare and benchmark their performance in the examined studies.

Results: We selected 55 articles that passed the quality evaluation of the systematic review. We then investigated in depth which feature extractions, and algorithms were employed to identify technical debt in each SDLC phase. All approaches proposed in the analyzed studies were grouped into NLP, NLP+ML, and NLP+DL. This allows us to discuss the performance in three different ways.

*Corresponding author

Email addresses: e.sutoyo@rug.nl (Edi Sutoyo), a.capiluppi@rug.nl (Andrea Capiluppi)

Conclusion: Overall, the NLP+DL group consistently outperforms in precision and F1-score for all projects, and in all but one project for the recall metric. Regarding the feature extraction techniques, the PTWE consistently achieves higher precision, recall, and F1-score for each project analyzed. Furthermore, TD types have been mapped, when possible, to SDLC phases: this served to determine the best-performing feature extractions and algorithms for each SDLC phase. Finally, based on the SLR results, we also identify implications that could be of concern to researchers and practitioners.

Keywords: technical debt, natural language processing, systematic literature review, software development life cycle

1. Introduction

Business needs often require developers to be able to deliver new products and features as rapidly as possible, in order to reach the market. In fact, a developer must be able to balance carefully designing applications and seamlessly coding their features. In the business or corporate world, however, schedules and limited resources frequently hinder developers from generating perfect code before deploying it into a product [1]. This becomes commonplace when stakeholders cut several technical processes to catch up on momentum, especially in startup practices [2, 3]. A tight deadline frequently leads to technical debt, as developers do not have enough time to think long-term or about the best overall solution for a code [4].

Technical debt (TD) is often used to describe the trade-off that occurs when an organization (or a developer) chooses to prioritize the delivery of new features or functionality over the long-term maintenance of the software [5]. TD can occur in any software development project and can be caused by a variety of factors, including lack of resources, time, knowledge, or experience on the part of the development team [6, 7].

A small amount of debt is acceptable; however, a significant amount of debt will hinder the scalability and maintainability of software [7]. At first, it could be possible to get around it by patching the implementation, so that the software or the new features can still be deployed. But over time, TD will reduce productivity: piling more of those shortcuts to add further features will eventually result in effectively not being able to add any more. Even worse, it will be hard to keep a software product maintainable, as remedial patches are all over the place and hard to understand.

As a result, numerous researchers have studied how to manage TD, starting with the early phases of its detection [8, 4]. Due to the significance of TD, a number of empirical studies have investigated TD and provided ways for its detection and management. TD detection is a crucial step to effectively manage TD items and make TD manageable and explicit about controlling the number of TD. In addition, researchers have developed manual and automated approaches to identify TD types using indicators derived from source code metrics, utilizing several techniques and artifacts, such as documentation and architectural reviews, in order to detect documentation debt, test debt, and architecture debt [9, 10].

TD has also been used to characterize many challenges during the software development life cycle (SDLC), including aspects that negatively affect each phase, the evolution of the system, and anything that impedes the progress of its development operations [11]. In addition, Alves et al. [12] recognized and categorized a total of 13 distinct types of TD that were derived from a variety of studies. Moreover, additional pertinent information, like indicators used to identify TD in software projects, is also provided. Therefore, the issue of identifying the existence of technical debt during the SDLC (especially to the point of detecting the specific type of technical debt) becomes increasingly challenging.

Recently, NLP has developed rapidly in both theoretical and practical aspects, and it has expanded into various research areas, including technical debt detection. Despite the fact that experts and academics have focused a significant amount of attention on TD in recent years, there is still no comprehensive review of the current status of technical debt detection using NLP approaches. Two earlier reviews on TD detection were published in [9, 13]: the role of NLP in this endeavor was found to undeniably be a feasible research frontier solution for technical debt identification. Numerous empirical studies have recently concentrated on examining source code comments that indicate technical debt, often known as self-admitted technical debt (SATD). Moreover, SATD detection significantly depends on NLP-based approaches.

In this paper, we produce and evaluate the results of a systematic literature review, that focuses on the identification of technical debt by using machine learning approaches. The objective of our study, and its main contribution, are basically two-fold, and aimed at determining which ML-based technique or algorithm has the better performance: on the one hand, at distinguishing technical debt from non-technical debt; and on the other hand, at detecting technical debt in various phases of the software life cycle.

In addition to the Introduction, the remaining sections of the study are or-

ganized as follows: Section 2 introduces the concepts and related works, and Section 3 describes the methodology utilized to execute the SLR. Sections 4 and 5 summarize and discuss the obtained results, while Section 6 identifies the threats to the validity of this study. Finally, Section 7 offers a conclusion.

2. Related Work

This section is intended to introduce the concepts underpinning this research (NLP and Technical Debt), as well as the works that are relevant to it.

2.1. Technical Debt (TD)

The term "technical debt" was coined by software developer Ward Cunningham. He originally used the metaphor for communicating to non-technical stakeholders the importance of budgeting resources for refactoring [5].

Technical debt, sometimes referred to as tech debt or code debt, is the result of a development team rushing the delivery of projects or features that must later be refactored. In other words, there is a trade-off between delivering high-quality, effective, and optimized code and the short-term benefits of achieving release dates [6, 4]. A small amount of debt is acceptable, but if left uncontrolled, it could end up costing a lot of time and money in the future.

The term "not-quite-right" is commonly used to refer to internal software development activities that are postponed but may lead to future difficulties if not addressed [6, 11]. It represents the debt incurred when development teams opt for quick and easy solutions in the short term, risking long-term negative impacts.

The term "TD" encompasses below-par aspects of the software that developers are aware of but lack resources to address, such as outdated documentation, unperformed tests, complex code needing refactoring, and known defects left unattended [14, 15]. These immature artifacts can lead to challenges in meeting quality standards and cause unforeseen delays in implementing necessary changes. Recently, numerous empirical studies have concentrated on examining source code comments that indicate technical debt, often known as self-admitted technical debt (SATD). Moreover, SATD detection significantly relies on NLP-based approaches.

2.2. Natural Language Processing (NLP)

Natural Language Processing (NLP) is an interdisciplinary field that studies the interaction between computers and human language, encompassing

tasks like speech recognition, natural language understanding, and natural language generation [16, 17]. Chowdhury [17] explains that the objective of NLP researchers is to understand how people perceive and use language so that the appropriate tools and techniques can be developed to assist computers in understanding and using natural languages to complete the necessary tasks.

NLP deals with processing natural language, which is the language used by humans in text or audio form. Computational linguistics, rule-based human language modeling, statistical models, machine learning, and deep learning are combined to enable computer programs to interpret and process human language [18, 19].

The Software Development Life Cycle (SDLC) consists of established phases that guide software development [20]. NLP can be applied to all SDLC phases, especially when dealing with plain text artifacts. For NLP tasks, plain text is used as input, including project artifacts such as source code comments, commit messages, pull requests, and issue tracking systems [21, 22].

Source code comments serve as annotations within the code, providing documentation directly incorporated by programmers [23]. These comments are essential for understanding a system and may contain important information, such as the purpose of code additions, progress status on collective tasks, or reasons for specific changes [24]. Comments can express concerns that need attention in the future (e.g., the TODO keyword), identify emerging issues, and guide future actions [25, 26]. They provide valuable context specific to their authors and are human-readable.

Recently, NLP-based approaches have gained traction in addressing software engineering problems, including the detection of technical debt (TD). NLP has been effectively used to automatically identify the presence of TD and even distinguish specific types of TD. Notably, numerous studies have integrated NLP with machine learning models or deep learning architectures, aiming to enhance algorithm performance (evidence is provided in Subsection 4.2).

2.3. Contributions to the state of the art

Our SLR is an expansion and update of an existing one ([9]), but other contributions will also be offered:

- *Time frame and scope.* Our review is the most extensive systematic literature review (SLR) on the topic, as it includes relevant studies published between 2014 and 2022. Alves et al. [9] conducted a comprehensive

systematic mapping study on technical debt identification, but only between 2010 to 2014. On the other hand, the work published in Sierra et al.'s is based on a survey that covers the 2014-2018 period [13].

- *Systematic approach.* We followed Kitchenham and Charters' [27] original and rigorous procedures for conducting systematic reviews. Alves et al. [9] did not apply the same approach, but rather a systematic mapping approach, as introduced by Petersen et al. [28]. Sierra et al. [13] did not report how they sourced their study, but their approach appeared in the form of a survey study.
- *Comprehensiveness.* We not only rely on search engines for each database used, but we also performed a bidirectional snowballing process. As a result, we analyzed more relevant articles.
- *Analysis.* We provide a more comprehensive overview of each study analyzed. Alves et al. [9] focused on characterizing the categories of TD, identifying indicators that can be used to detect TD, and identifying what visualization techniques have been offered to enhance TD identification and management operations. On the other hand, Sierra et al. [13] only focused on self-admitted technical debt (SATD), and did not focus on the detection of technical debt. Their work categorized their findings into 3 TD activities: detection, comprehension, and repayment. In this paper, we report the type of metric evaluations used, the feature extraction and modeling approaches, and the performance of each algorithm, and synthesize the study findings.

3. Methodology

This study is carried out in accordance with the methodology of conducting a SLR described by Kitchenham *et al.* [29, 30, 27]. The procedures for SLR are broken down into three stages: planning, conducting, and reporting the SLR [27]. We developed a review protocol during the planning stage that included the following steps: i) identification of research topics, ii) creation of a search strategy, iii) study selection criteria, iv) study quality assessment, v) data extraction procedure, and vi) data synthesis process (as illustrated in Figure 1).

The review process involved several steps. First, we formulated a research question for the SLR. Next, we developed a search strategy by identifying search strings and selecting online databases for finding primary studies. The third

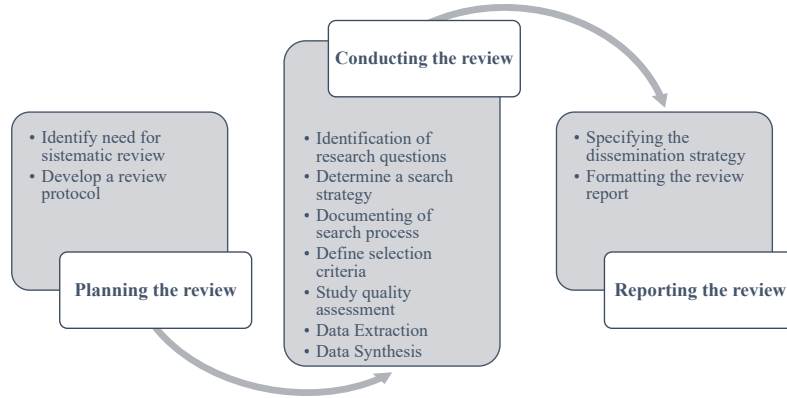


Figure 1: Graphical representation of the SLR process used in this SLR

step was to find relevant studies based on the research questions and establish inclusion and exclusion criteria. Then, we conducted bidirectional snowballing using references from the source articles. After that, we created a data extraction form to answer the research questions. Finally, we designed a data synthesis method to analyze and summarize the findings.

Figure 2 depicts the article selection procedure in determining primary studies used in this SLR. The following subsections provide information about the selection process in detail.

3.1. Research Questions

According to Petticrew and Roberts [31], the Population, Intervention, Comparison, Outcomes, and Context (PICOC) framework, which consists of five components, should be used to formulate research questions. The PICOC for our study topics is displayed in Table 1. All empirical research outlining the various NLP techniques for identifying technical debt is included in our SLR. We did not include a comparison for the NLP approach in PICOC because it cannot be applied to our research objectives.

According to the PICOC in Table 1, the main objective of this SLR is to investigate, identify, and classify all relevant evidence from the primary studies that focus on using NLP to detect technical debt. Therefore, the aim of this SLR is to answer the following questions:

RQ1: What NLP approaches, and its combinations, have been employed to detect technical debt in software engineering and/or software development?

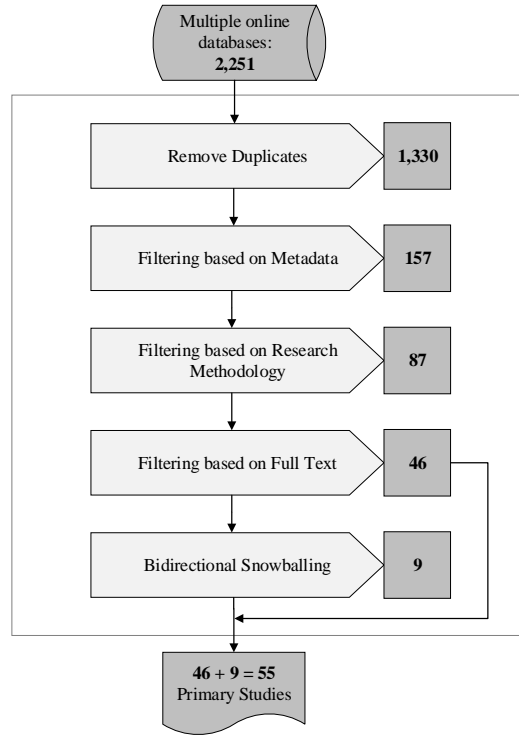


Figure 2: An overview of the steps in determining primary studies

Table 1: PICOC Criteria

Parameter	Description	Keywords Used
Population	Field of application	"technical debt"
Intervention	Approach or method for dealing with a certain problem	"nlp" OR "natural language processing" OR "machine learning"
Comparison	None	None
Outcome	Related to factors of importance to practitioners	detection" OR "identification" OR "prediction"
Context	Context in which intervention take places	"software development" OR "software engineering"

Rationale: This question aims to understand the progress made in detecting technical debt using NLP approaches. By surveying existing studies, we can identify trends and potential future research directions.

RQ2: What feature extraction techniques are utilized in technical debt de-

tection?

Rationale: This research question aims to comprehensively explore feature extraction techniques, analyze their frequency of usage in primary studies, and characterize their prevalent practices among researchers. The goal is to present the current state of feature extraction, identify innovative approaches, and highlight any gaps in the literature to guide future research.

RQ3: In what phases of software development does the NLP approach support technical debt detection?

Rationale: This question aims to identify software development phases where NLP can detect technical debt. It enhances detection accuracy, enabling developers to prioritize resolution and minimize project impacts.

RQ4: What metric evaluations are commonly used for technical debt detection?

Rationale: This research question focuses on the validation methodology and performance of a proposed technical debt prediction algorithm. The objective is to ensure the reliability, accuracy, and precision of the results. A comprehensive evaluation is conducted to assess the algorithm's effectiveness in detecting technical debt and its impact on development processes.

RQ5: Which NLP method performs best for identifying technical debt?

Rationale: This question aims to explore state-of-the-art NLP approaches for detecting technical debt and design a tool for identifying it at each software development stage. It also serves as an open-ended research question to improve upon existing approaches.

3.2. Search Protocol

Before beginning the SLR planning stage, it is necessary to conduct a pilot search on a database such as Scopus to observe if there are any SLRs with the same research theme. There is no need to conduct a new SLR if there are existing SLRs on the same topic [32]. In this systematic review of the literature, no results covering the topic of capabilities of the NLP approach on identifying technical debt were discovered; therefore, the SLR can be conducted as new research.

The SLR's search strategy consists of three stages: pilot search, primary search, and snowballing. In the pilot search, multiple search strings are used on online databases to determine the most effective one. The primary search uses the identified search string to find relevant literature in online source databases. The search concludes with a snowball search [33]. The pilot and primary search

utilized six online databases: Web of Science, Scopus, IEEE Xplore, ScienceDirect, ACM Digital Library, and Springer. These databases, as proved by previous software engineering SLRs, provide an excellent overview of the software engineering literature and are trustworthy to cover our scope of the literature. Moreover, we employed the "Advanced Search" feature in these databases to obtain full-text and metadata results.

Table 2: Search strings utilized during pilot search

String ID	Search String
S1	("technical debt") AND (NLP OR "machine learning")
S2	("technical debt") AND (NLP OR "natural language processing" OR "machine learning") AND ("software development" OR "software engineering") AND (detection OR identification)
S3	("technical debt") AND (NLP OR "natural language processing" OR "machine learning") AND ("software development" OR "software engineering") AND (detect* OR identif* OR predict*)

The results of a pilot search using search string S1 were communicated to the second author by the first author. The second author checked several articles at random [34, 35, 36, 37, 38, 39] to see if they were relevant. The second author proposed altering the search string to S2 because the search string S1 result was still too general. Similar to the first process, iteration is repeated until it is decided to utilize the final search string, namely search string S3.

Table 3: Search results from selected databases

Database	Search Result	Selected for SLR
Web of Science	18	9
Scopus	951	12
IEEE Xplore	384	16
ACM Digital Library	267	10
Springer	474	4
ScienceDirect	157	4
Total	2,251	55

As illustrated in Table 2, through a pilot search with multiple search strings, the most effective one (i.e. S3) was selected. The search string was designed with synonyms, alternative keywords, and boolean operators 'OR' and 'AND'

to retrieve relevant studies. Additionally, wildcard asterisks (*) were used to broaden the search for variations of words [40].

Based on the pilot search results, the final search string (i.e. S3) was applied to the mentioned five online databases to find relevant primary studies for the research questions as described in Subsection 3.1. We utilized the "Advanced Search" feature in these databases to acquire full-text and metadata results.

The final search yielded 2,251 articles (See Table 3). Duplicate articles were removed, and relevant studies were identified based on inclusion and exclusion criteria. The first author executed the review protocol, conducted searches, filtered studies, and extracted data under the supervision of the second author. Any disagreements were resolved through discussions to reach a consensus.

3.3. Selection Study

3.3.1. Inclusion and Exclusion Criteria

Inclusion and exclusion criteria must be precisely defined for study selection. The inclusion criteria should include the NLP approach to identify, detect, and predict technical debt. Consequently, the primary studies were chosen based on a comparison of titles, keywords, abstracts, and full texts (i.e., introduction, methodology, results, and conclusions). We employed 6 inclusion criteria, as shown in Table 4):

Table 4: Inclusion Criteria

No	Inclusion Criteria
1	The article must be peer-reviewed and published at conferences, workshops, or journals
2	The article must be accessible online (i.e., articles can be fully accessed and/or downloaded)
3	For research that has both publication types (journal and proceeding version), only the journal version will be selected
4	The study must discuss approaches for identifying, detecting, predicting, and measuring performance in technical debt using the NLP approach
5	The study must be published between the years 2002 and 2022
6	The study reported must be related to technical debt in software engineering and/or software development

The filters shown in Table 5 were instead utilized as a part of our exclusion criteria, and in order to discard any papers that did not fall within the parameters of our scope.

Table 5: Exclusion Criteria

No	Exclusion Criteria
1	Articles not written in English
2	Works in the form of dissertations, theses, and books
3	Articles not indexed in one of the databases mentioned in the inclusion criteria
4	The list of publications from each database is merged; if duplicate titles/files are found, one will be selected
5	Articles that are still in the form of proposals
6	The article does not answer at least one research question
7	All other secondary and tertiary studies

3.3.2. Selection Process

Exclusion criteria were initially applied to eliminate unnecessary publications from the retrieved sources, as shown in Table 3. The first author reviewed the 2,251 located publications, excluding articles based on titles, abstracts, and keywords. This process resulted in 46 articles remaining, comprising approximately 2.04% of the initial findings. To ensure consistency, the test-retest strategy by Kitchenham and Charters [27] was used instead of inter-rater agreement. Discussions between the first and second authors were held to maintain consistency, and the second author cross-checked the extraction results shared in a spreadsheet¹. Any disagreements were resolved through discussions to reach a consensus.

3.3.3. Snowballing

Snowballing involves using an article’s references or citations to find new sources that might have been missed during the initial search [33]. We employed bidirectional snowballing (forward and backward): firstly, we included all papers cited in relevant articles, and secondly, we checked all articles that cited the relevant papers. The first author reviewed the titles, abstracts, and keywords of the 46 snowball papers, reapplying the inclusion/exclusion criteria. This snowballing process led to the discovery of 9 additional articles (including 2 articles published in 2023) for further investigation (see Table 3). In total, we selected 55 (46 + 9) primary studies after multiple rounds of searches.

¹The spreadsheet is made available in the shared dataset area at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR>

3.4. Quality Assessment Criteria

After finalizing the list of articles for the SLR, we performed a quality assessment phase to ensure that the selected papers provided the necessary information to answer our research questions. The quality of the selected articles was assessed using the questions presented in Table 6, following the procedure recommended by Kitchenham [29]. All 55 relevant articles that we discovered passed the quality evaluation, and thus, our SLR relies on these 55 primary studies.

Table 6: Quality Assessment Criteria

No	Questions
1	Are the research objectives or research questions in the article clear?
2	Does the study include reviewing the relevant literature or background?
3	Does the paper describe a research methodology?
4	Does the article employ an NLP approach?
5	Does the study provide research findings and answer the research questions?
6	Does the article report matrix evaluation results of the approach?
7	Does the article clearly state the dataset used?

All of the listed articles were published to the public between January 2014 and December 2022. Our search for articles was completed at the end of January 2023, supplemented by articles obtained through snowballing. As a result, it is highly unlikely that we missed any articles that were published during our time period due to publication delay.

3.5. Data Extraction

After applying the inclusion and exclusion criteria and conducting the quality evaluation, we selected the final relevant articles. To address our research question, we determined the required information/attributes for data summarization. This facilitated the data extraction process, where we read the full text of each article and extracted the relevant information based on specified attributes. To answer the research questions mentioned in Subsection 3.1, we extracted data items listed in Table 7 from each selected study. The extracted data were then entered into a spreadsheet for further analysis.

4. Results

This section addresses the posed research questions. To that purpose, the outcomes of the 55 primary studies are presented and discussed. To provide

Table 7: Data extracted from each study

No	Data Name	Description	Relevant RQ
1	Title	Title of the study	None
2	Author	Name of the authors	None
3	Year	The publication year of the study	None
4	Venue	The name of the publication venue of the study	None
5	Publication Type	Journal, conference, workshop, or book chapter	None
6	Algorithms used	Machine learning model or deep learning architecture used in the study	RQ1, RQ5
7	NLP Technique used	NLP techniques (feature extraction techniques) used in the study	RQ2, RQ5
8	TD Types	The type of TDs, e.g., design or code TD	RQ3
9	Number of Class	Number of class(es)/TD types used	RQ3
10	Dataset used	Data source/project name and metadata of dataset used	RQ4
11	Metric Evaluation used	Types of metric evaluations used for performance measurement	RQ4
12	Performance Evaluations	Performance measurement results using metric evaluations	RQ5

a clearer context, in the Subsection 4.1 we provide some basic information on machine learning, deep learning, and feature extraction in general, as well as a definition of the terminology used in this paper.

4.1. Terminology

Machine Learning (ML): is the study of algorithms and statistical approaches that enable computer systems to perform tasks effectively without explicit instructions, relying on patterns and inference. It allows for incremental learning and automated predictions based on inputted data using specific algorithms [41]. ML applications train algorithms or statistical processes with "training data" to construct mathematical models for detecting patterns and characteristics in vast volumes of data. The goal is to make decisions or predictions based on this data. ML algorithms can be classified into five types: supervised learning, semi-supervised learning, active learning, reinforcement learning, and unsupervised learning [42]. In technical debt detection, supervised learning is commonly used, as observed in the analyzed studies.

Deep Learning (DL): is a subset of ML that is based on multiple-layer of artificial neural networks (ANNs) that are inspired by the human brain [43]. In particular, deep learning consists of several interconnected artificial neural networks, which distinguishes it from general ML algorithms.

Feature Extraction: NLP is a subfield of computer science and ML focused on training computers to analyze vast amounts of human language data. It enables computers to comprehend human language. To enable ML algorithms to learn from training data with pre-defined features, feature extraction techniques are necessary. These techniques convert raw text into numerical formats (feature matrix or vector) since ML algorithms and DL architectures cannot directly process plain text. Feature extraction (vectorization) of text data is vital for text data processing, as computers can only understand and handle numerical data [16].

Approach: The approach in a study determines how researchers achieve their objectives and address the raised problems. It guides the formulation of the research problem and ensures adherence to scientific principles. In technical debt detection, the approach involves the steps and research methodology, such as pure NLP-based or pure machine learning, used to detect technical debt. The three basic terms we consider as "approaches" are NLP, machine learning (ML), and deep learning (DL).

Group: In the studies analyzed, the NLP approach does not solely use NLP-based, but various NLP approaches combined with ML and DL are also taken into account. However, as explained in the inclusion/exclusion criteria (in Sub-subsection 3.3.1), this SLR focuses on all approaches contained in NLP to detect technical debt, including solely NLP-based only and a combination with ML and DL, whereas an approach using ML- or DL-based only is not in the considered studies.

Technique: In the context of our systematic review, a *technique* refers to a name of a method employed to address a challenge in the identification of technical debt. The names of the techniques utilized in this primary study are categorized into large clusters based on their characteristics and level of complexity. Techniques could include regular expressions (RegExp) and textual patterns, as well as embeddings (both pre-trained or non-pre-trained). A thorough description of the techniques can be found in Subsection 4.3).

4.2. Results - RQ1: What NLP (including NLP-based or a combination of NLP and ML/DL) approaches are employed to detect technical debt in software engineering and/or software development?

As part of the results for RQ1, we found that the papers can be clustered in three general groups, that combine various approaches: papers that only utilize NLP to detect TD (NLP-based); papers that use a mix between NLP and Machine Learning (NLP+MP); and papers that use a mix of NLP and Deep Learning (NLP+DL).

Table 8 shows the composition of these groups: we see that the table is composed mostly by studies that use NLP coupled with other approaches (overall $\approx 70\%$), whereas the NLP-based alone accounts for $\approx 30\%$ (17 out of 55)². In the subsections below we discuss each of these approaches in detail.

Table 8: Approach groups: studies that belong to more than one group are highlighted with an underline.

Group	Articles using the approach
NLP	[48, 49, 50, 51, 52, 53, 54], [46, 47], [55, 56, 57, 58, 59, 60], [44, 45]
NLP+ML	[61, 62, 63, 64, 65, 66, 67, 68], [46], [69, 70, 71, 72, 73, 74, 75, 76, 77], [45, 47], [78], [79, 44], [80, 81], [82]
NLP+DL	[83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94], [79, 44], [95, 96], [45], [97], [82]

4.2.1. NLP

NLP consists of at least three major phases: (i) data preprocessing, (ii) feature extraction, and (iii) modeling. Data preprocessing entails preparing and cleansing text data for computer analysis. Preprocessing transforms data into an executable (numerical) format and extracts text features that an algorithm can utilize. In general, NLP preprocessing can be broken down into the following stages: stop-word removal, stemming, lemmatization, tokenization, and parsing.

This group includes all of the (i) preprocessing phases mentioned above, as well as Regular Expression and Part-of-Speech (POS) tagging. A detailed description of the techniques used in this group will be explained in Subsection

²The four studies ([44, 45, 46, 47]) in this NLP group are also included in other groups.

4.3 in order to answer RQ2. This group is focused on approaches dealing only with text syntax, and no prediction occurs. For instance, Potdar and Shibab [48] utilized the comment patterns approach to identify as many as 62 patterns (words and phrases) after analyzing more than one hundred thousand source code comments in four open source software projects: Eclipse, Chromium OS, Apache HTTP Server, and ArgoUML. The amount of SATD ranged from 2.4% to 31% of the files, according to the study. Furthermore, Maldonado and Shihab [51] classified SATD using regular expressions into five categories: design, defects, testing, requirements, and documentation debt. Bavota and Russo [49] mined the change history of open-source Java systems to find SATD instances and manually analyzed comments to identify SATD types.

In addition, for techniques that use pre-trained models, we deliberately include them in this group as well because this approach utilizes models that have been previously trained to be used to classify new datasets. Therefore, it is possible to use it directly without having to do data training beforehand. For example, Sharma *et al.* [45] used ALBERT to automatically detect SATD in the R programming. The (ii) feature extraction and (iii) modeling phases are covered in the NLP+ML and NLP+DL groups.

4.2.2. NLP+ML

The feature extraction phase's primary objective is to convert words into numeric vectors for computation using a vector space model (VSM). Next, the model is developed and trained on pre-processed data with extracted features. The supervised learning approach is used to create the model, where the dataset is divided into training data with labels used for model creation. The model will then predict or detect technical debt (TD) or non-TD based on the training process. Evaluation is performed using test data without labels to assess the model's performance.

4.2.3. NLP+DL

In principle, this group has similarities with the NLP+ML group, what distinguishes is the type of algorithm used. In general, deep learning employs the results of feature extraction as input to be further processed as training data. Therefore, supervised deep learning is the most commonly used for technical debt detection.

4.2.4. Temporal trends within groups

Figure 3 illustrates the number of papers published per group as detected in our SLR. The combination of approaches (NLP+ML and NLP+DL) is a relatively new but influential factor, driving a large and increasing number of studies focusing on TD detection using NLP. On the other hand, Figure 4 depicts the temporal trends in techniques used in the relevant studies (details in Subsection 4.3).

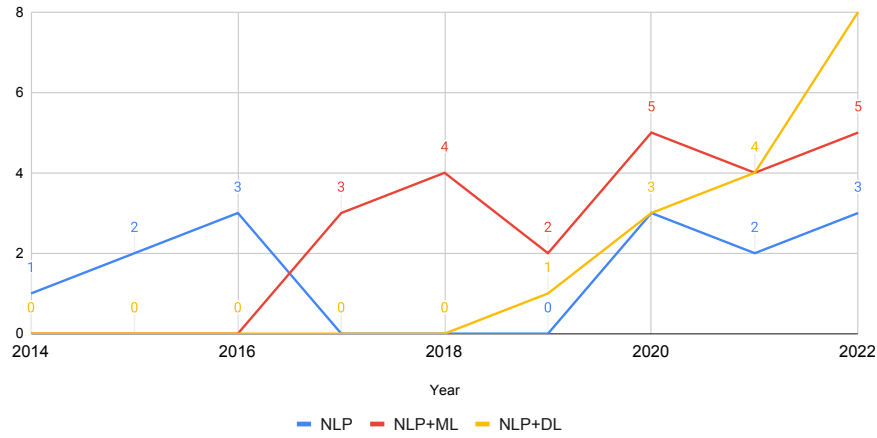


Figure 3: Articles count throughout the years based on the group name

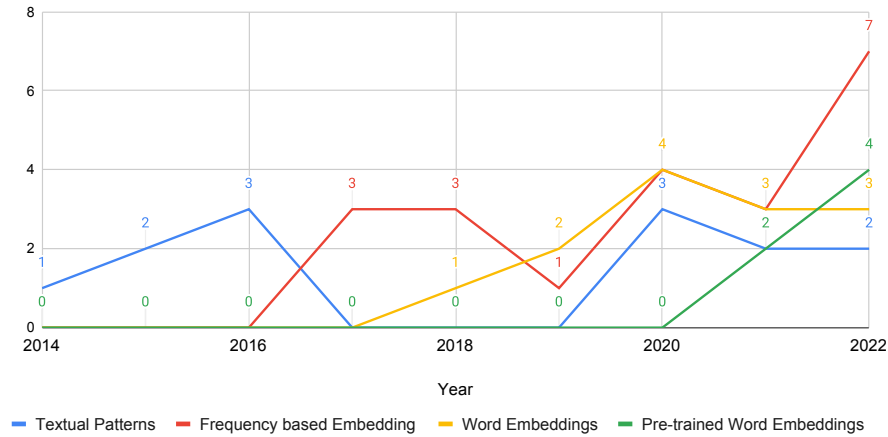


Figure 4: Articles count throughout the years based on the technique name

4.3. Results - RQ2: What feature extraction techniques have been utilized in technical debt detection?

We categorize feature extractions based on their characteristics and complexity into three categories in order to provide more intuitive categories. The difference between these technique categories is determined by the complexity and extent to which the techniques can understand the context (semantic) in the text. The categorization of the techniques is based on the following criteria:

- We extract three sections from each article: Introduction, Abstract, and Methodology. If any of these sections are missing, two available sections suffice.
- The NLP utilization is arranged in the following order based on proposed techniques. Most solutions are outlined in the Abstract or Introduction sections. If it is unclear how NLP is implemented, we identify keywords and techniques in the Methodology section to understand the complexities of the NLP tasks. If the NLP technique remains unidentified, we analyze the source code (if available) to determine the techniques.
- When multiple steps of NLP techniques are employed, only the technique with the highest degree of complexity is specified. We carefully examine every NLP technique used in each article. Generally, we refer to the research methodology section to determine the NLP technique, which helps categorize the technique under the appropriate group. For example, if an article uses preprocessing and then utilizes Term Frequency-Inverse Document Frequency (TF-IDF) also, it will be included in the Frequency-based Embedding (FBE) technique.

Figure 5 provides the mapping between group names and NLP techniques. Circles represent group names, rectangles indicate technique names. For instance, NLP (circle) connects to Textual Patterns and Pre-trained Word Embeddings (rectangles), showing two techniques within the NLP group. Numbers on arrows indicate the count of papers using that combination (e.g., 15 papers use NLP and Textual Patterns). The mapping does not have a one-to-one correspondence between group and technique names.

In the following sections, we provide an explanation of each categorization of NLP techniques.

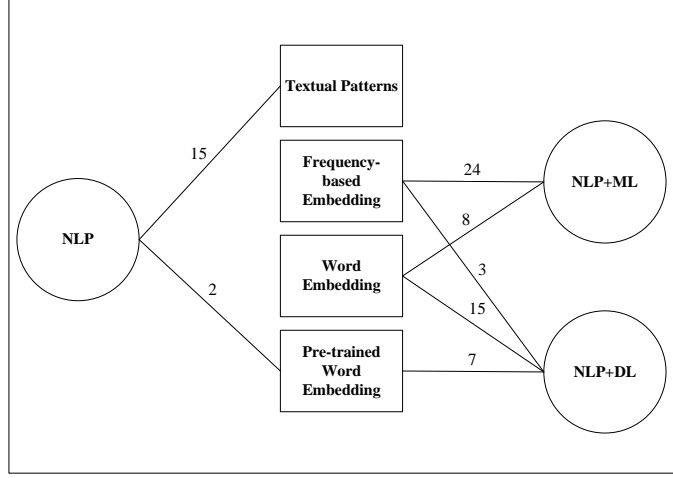


Figure 5: Mapping between groups (circles) and techniques (rectangles) used. The numbers on the arrows represent the numbers of research studies in the intersection group+technique

4.3.1. Textual Patterns (TP)

The TP technique is a straightforward approach for processing text data: it involves using regular expressions to determine if a string contains a specific character pattern. These patterns consist of meta-characters, which have special meanings, and characters (non-meta-characters). This technique focuses on basic complexity processing tasks such as stemming, pattern matching, lemmatization, tokenization, task annotation, Regular Expression (Reg-Exp), and Part-of-Speech (POS) tagging. It primarily deals with textual syntax and doesn't involve prediction (as shown in the first row of Table 9). An example of this technique is searching for "TODO" in a string, which is often related to technical debt as it represents work that has been deferred to a later time.

Some research has successfully identified the presence of technical debt and different forms of technical debt contained in source code comments by creating and implementing heuristic filtering using regular expressions [48, 51, 49, 50]. Furthermore, Farias [59, 52] used part-of-speech (POS) tagging to label each word in the source code comments sentence, then analyzed the POS tags and filtered comments using terms included in the vocabulary. Similarly, Guo *et al.* [53] suggested a simple heuristic method that uses Matches task Annotation Tags (MAT) in source code comments to identify SATD.

4.3.2. Frequency-based Embedding (FBE)

This NLP technique involves calculating the frequency of occurrence of a term; specifically, we categorize features based on the frequencies associated with single words in text or/and n-grams (n-grams are a sequence of n contiguous items from a given sample of text or speech). Several techniques that fall into this category are Bag-of-Words (BoW), Latent Dirichlet allocation (LDA), TF-IDF, N-gram IDF, Count Vectorizer (CV), chi-square (CHI), Information Gain (IG), Mutual Information (MI), and TextRank. The studies using this category are listed in Table 10.

For instance, Huang [98] employed the VSM, also called the BoW, to turn text data from natural languages into numerical representations. Moreover, IG was also employed for feature selection in order to address the *curse-of-dimensionality* problem. Other NLP techniques, such as TF-IDF were also adopted by several researchers, such as [62, 63, 80, 92, 65, 45, 44, 70, 93, 82]. Combination feature extraction techniques such as N-Gram IDF are also used by [80, 67, 68, 73]. In addition, Rantala and Mäntylä [72] utilized a popular topic modeling technique called LDA to find topics from texts that are used to predict the appearance of SATD.

4.3.3. Word Embeddings (WE)

WE are a model-defined representation of a word, letter, or sentence. Embedding is a low-dimensional space in which high-dimensional vectors can be translated. Word Embedding makes it easy to apply machine learning to sparse vectors that represent words. Word2Vec, WordNet, and FastText are three examples of word embedding techniques that have been put to use in the process of detecting technical debt.

Word2Vec is applied to a variety of machine learning algorithms in the research that is presented in [64, 66, 85, 87, 89, 94, 90, 83] in order to obtain the best possible performance in detecting technical debt. Similarly, the research carried out by [75, 44] made an effort to identify technical debt by utilizing the WordNet technique. In addition, Sabbah and Hanani [82] recently completed a study in which they compared the effectiveness of the tools Word2Vec and FastText in identifying instances of technical debt within source code comments. Details of the utilization of the word embedding technique in several machine learning algorithms and deep learning architectures are shown in Table 10 and Table 11, respectively.

4.3.4. Pre-trained Word Embeddings (PTWE)

PTWE refers to a technique of training utilizing a previously trained model and then applying it to a new dataset [99]. A pre-trained word embedding is one that has been trained on a dataset and has weights and biases that correspond to the features of the dataset used to train it. The acquired characteristics are transferable to other data sets. Pre-trained models are able to deliver optimal accuracy and solve the challenge of limited datasets and cross-domain issues by collecting necessary information from existing data and applying it to the target data.

Pre-trained word embedding has achieved remarkable success in NLP, resulting in a paradigm change from supervised learning to pre-training followed by fine-tuning [100]. This includes all models that require no training as they have already been trained and show good performance on average on various data sets in different domains.

Pre-trained models have the advantage of being a reliable starting point for building customized models that perform better in domain-specific tasks, including on technical debt detection. There are several pre-trained word embeddings that have been used to solve technical debt detection problems, namely using Bidirectional Encoder Representations from Transformers (BERT) [44], Robustly Optimized BERT Approach (RoBERTa) [45], and A Lite BERT (ALBERT) [45] (as reported in the second row of Table 9).

In addition to the conventional pre-trained word embedding described above, there are other works combining pre-trained word embedding with deep learning architecture. Zhuang *et al.* [95], and Sabbah and Hanani [82] have combined Global Vectors for word representation (GloVe) with CNN to produce a binary classification of source comments (TD-related comments or non-TD-related comments), and multi-class classes (defect, test, documentation, design, and requirement) of technical debt, respectively. This mixed technique can be seen in detail in the third row of Table 11.

Table 9: Studies in the NLP-based group

Technique	Reference no.
Textual Patterns	[48, 49, 50, 51, 52, 53, 54, 46, 47, 55, 56, 57, 58, 59, 60]
Pre-trained Word Embeddings	[44, 45]

Table 10: Articles in the NLP+ML group, combining NLP and one or more machine learning algorithms. Studies using more than one ML algorithm are highlighted with an underline.

NLP Technique	ML Algorithm	Reference no.
Frequency-based Embedding	Naïve Bayes (NB)	[98, 65, 46, 66] , [81, 82, 77] , [69, 62]
	Support Vector Machine (SVM)	[45, 46, 74, 98, 66] , [44]
	Random Forest (RF)	[46, 47, 82, 74] , [68]
	Logistic Regression (LR)	[46, 47, 45, 72] , [70]
	k-Nearest Neighbor (kNN)	[98]
	eXtreme Gradient Boosting (XGBoost)	[78]
	Auto-Sklearn	[67, 73, 71, 80]
	Decision Tree	[46]
	Maximum Entropy (ME)	[63, 61] , [66]
	Sequential Minimal Optimization (SMO)	[76]
Word Embeddings	SVM	[64] , [66, 79]
	NB	[66, 79, 82, 81, 77]
	LR	[75, 79, 72]
	RF	[82, 75, 79]
	ME	[66]
	XGBoost	[75]
	kNN	[79]

4.4. Results - RQ3: In what phases of software development does the NLP approach support technical debt detection?

In order to answer research question RQ3, we first collected and listed all types of technical debt from our primary studies. We found that there are 23 types of technical debt (including on-hold, TD, and non-TD types). The first and second authors then discussed and observed the various types of technical debt discovered. Then, we had a discussion that led to an agreement to place similar types of technical debt in the same bucket, such as "code" and "build" debt, or "design" and "architecture". Then, we arranged them according to the phases of the software development life cycle (SDLC). The results of discussions and agreements regarding the placement of TD types and also the mapping of TD types to the SDLC phase are shown in Table 12. Moreover, we did not place TD and non-TD types into any of the SDLC phases, but we placed them in the "Other" category. Through discussion, disagreements over the placement and sequencing were resolved.

Table 11: Articles in the NLP+DL group, combining NLP and one or more deep learning algorithms. Studies using more than one DL algorithm are highlighted with an underline.

NLP Technique	DL Algorithm	Reference no.
Frequency-based Embedding	Convolutional Neural Network (CNN)	[92, 82]
	Long Short-Term Memory (LSTM)	[93]
	Graph Neural Network (GNN)	[92]
	Feed-Forward Neural Network (FNN)	[92]
Word Embeddings	GNN	[94]
	LSTM	[85] , [84, 87, 89]
	CNN	[83] , [96, 85, 79] , [45]
	BiLSTM	[44]
	GCN	[79]
	CNN-BiLSTM	[90, 96]
Pre-trained Word Embeddings	GloVe+CNN	[95] , [82]
	GloVe+BiLSTM	[88] , [44]
	GloVe+CNN-GRU	[97]
	BERT+CNN	[82]
	FastText+CNN	[82]
	ELMo+BiLSTM	[86]
	TextING+GGNN	[91]

As illustrated in Table 13, the definitions of the letters used to represent the types of TD are as follows, A=requirement; B=usability; C=compatibility; D=security; E=performance; F=design; G=architecture; H=algorithm; I=process; J=versioning; K=on-hold; L=user interface; M=code; N=build; O=infrastructure; P=people; Q=implementation; R=test; S=defect; T=documentation; U=service; V=TD (SATD); and W=Non-TD.

As an example of this mapping, we consider the work published by Maldonado and Shibab [51]: in the third row of Table 13 there are 5 (five) check-marks (columns A, F, S, T, and R), meaning that this study focuses on detecting 5 (five) types of TD, namely requirement, design, defect, documentation, and test debt, respectively.

We then grouped these types into clusters that represent where in the SLDC they are most likely to occur³; for example, the A+B+C+D+E types were all as-

³Some categories are not immediately easy to assign: for example, the 'on-hold' type of TD refers to accumulated issues, problems, or sub-optimal code or design decisions that have been identified as problematic but are not being actively addressed or resolved at the moment. We

signed to the *Requirement* phase (under the REQ header of Table 13) of the SDLC. In this way, we have categorized the TD approaches, groups, and techniques observed in the papers and based on the SDLC phases. This will be used in the sections below to evaluate the performances of the ML and DL algorithms based on where in the SDLC phase they are utilized.

Using the groupings as mapped in Table 12, we also observed that the research works are evenly spread to represent the various SDLC phases, apart from the testing one, for which we found only 12 papers. We also observed that only a handful of papers gathered data for all the phases of the SDLC [59, 49, 69, 52, 58, 77, 45], whereas the majority of the retrieved papers focused on one or two SDLC phases.

4.5. Results - Q4: What metric evaluations are commonly used for technical debt detection?

We evaluated the effectiveness of various techniques using precision, recall, and F1-score obtained from confusion matrix calculations. Accuracy was not used due to imbalanced datasets, and it was not reported in most articles. We also considered the number of articles and datasets that evaluated the algorithm or feature extraction.

The box plots reported below show metric evaluation results from only 39 out of 55 articles, as only a subset of those have fully reported their performance scores. There are 14 papers that did not report findings, and 2 articles used ROC-AUC, which cannot be compared for box plots. The following formulas are used to calculate precision, recall, and F1-score:

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F1-score = $\frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$

When available, we have recorded all these performance metrics in a spreadsheet and used them to compare the chosen NLP approaches and algorithms.

assigned it to the *Implementation* phase after internal discussion.

Table 12: Mapping Technical Debt (TD) Types to SDLC Phase

Alias	TD Name	SDLC Phase	Abbreviation
A	Requirement Debt	Requirement	REQ
B	Usability Debt		
C	Compatibility Debt		
D	Security Debt		
E	Performance Debt		
F	Design Debt	Design	DES
G	Architecture Debt		
H	Algorithm Debt		
I	Process Debt		
J	Versioning Debt	Implementation	IMP
K	On-hold		
L	User Interface Debt		
M	Code Debt		
N	Build Debt		
O	Infrastructure Debt		
P	People Debt		
Q	Implementation Debt		
R	Test Debt	Testing	TES
S	Defect Debt	Maintenance	MAI
T	Documentation Debt		
U	Service Debt		
V	TD (SATD)	Other	OTH
W	non-TD		

4.6. Results - RQ5: Which NLP method performs best for identifying technical debt?

In this section we summarize, by means of box plots, the results that we obtained on the performance metrics of the groups, approaches, feature extraction techniques and ML/DL algorithms⁴.

We are aware that it would be misleading to put all results and all datasets in the same visualizations, and compare ML approaches and techniques without prior clustering of our data. In order to avoid wrong representations, we performed two steps: first, we classified the relevant published articles into ei-

⁴Some of the box plots presented here might be too small to visualize. We have made all the images available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR>.

Table 13: Types of Technical Debt (TD)

No	REQ					DES				IMP									TES		MAI				OTH			Σ	
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W						
[48]																													2
[59]	✓					✓	✓		✓				✓	✓	✓	✓		✓	✓	✓	✓								12
[51]	✓					✓							✓	✓	✓	✓		✓	✓	✓									5
[49]	✓					✓							✓					✓	✓	✓									6
[50]																									✓	✓			2
[56]																									✓	✓			2
[63]	✓					✓																			✓	✓			4
[61]																									✓	✓			2
[62]																									✓	✓			2
[67]	✓					✓																							2
[65]																										✓	✓		2
[64]																										✓	✓		2
[98]																										✓	✓		2
[83]																										✓	✓		2
[66]																										✓	✓		2
[68]	✓					✓																					✓	✓	3
[73]												✓																✓	2
[71]											✓																	✓	2
[85]																											✓	✓	2
[72]																											✓	✓	2
[70]																											✓	✓	2
[87]	✓					✓																						✓	3
[84]	✓					✓																						✓	3
[69]	✓			✓	✓							✓	✓	✓				✓		✓									8
[52]	✓					✓	✓						✓	✓	✓		✓	✓	✓	✓									9
[58]	✓					✓	✓						✓	✓	✓			✓	✓	✓									8
[60]						✓				✓				✓	✓														4
[88]																									✓	✓			2
[86]																										✓	✓		2
[90]	✓					✓														✓							✓	✓	4
[89]																											✓	✓	2
[77]	✓		✓			✓		✓										✓	✓	✓									7
[75]																											✓	✓	2
[74]																											✓	✓	2
[76]	✓					✓												✓	✓	✓									5
[53]																											✓	✓	2
[57]	✓	✓				✓	✓			✓			✓	✓					✓	✓							✓	✓	11
[45]	✓	✓				✓	✓			✓			✓	✓		✓		✓	✓	✓							✓	✓	12
[80]																											✓	✓	2
[44]	✓	✓				✓	✓			✓			✓	✓					✓	✓									9
[95]																											✓	✓	2
[96]																											✓	✓	2
[78]						✓												✓		✓									3
[91]						✓												✓		✓									3
[92]	✓					✓													✓	✓	✓								5
[94]																										✓	✓		2
[47]																										✓	✓		2
[93]																										✓	✓		2
[81]																										✓	✓		2
[46]																										✓	✓		2
[79]																										✓	✓		2
[54]																										✓	✓		2
[55]																										✓	✓		2
[82]	✓					✓												✓	✓	✓									5
[97]	✓					✓												✓	✓	✓									5
Σ	20	3	1	1	1	21	7	1	1	4	2	1	8	8	2	3	2	12	16	14	1	32	39						

ther "TD or non-TD" types, and studies that identify specific TD types individually (requirements debt, design debt and so on). Second, we classified each of these classes into subgroups according to the dataset employed, for instance, studies employing the datasets from Maldonado⁵ and studies employing other datasets.

In Figure 6, the pie chart section "TD and non-TD" contains 32 articles (i.e., studies that detect if an item can be considered TD or not): 18 of those utilise the Maldonado dataset, the rest use other datasets. In addition, there were 23 articles in the "Specific TD Types" pie chart section, of which 14 used the Maldonado dataset, and the remaining 9 used other different datasets.

As a result, this section is articulated in 4 parts, and based on the groupings that we analyzed: 4.6.1 aggregates the results for all the research works that used the Maldonado dataset; 4.6.2 summarises the results from the papers that attempted a binary classification (TD or non-TD) from the extracted source comments; 4.6.3 on the contrary shows the results for the papers that attempted a classification of TD in various types; lastly, 4.6.4 aggregates the previous results in larger categories, according to the phases of the SDLC.

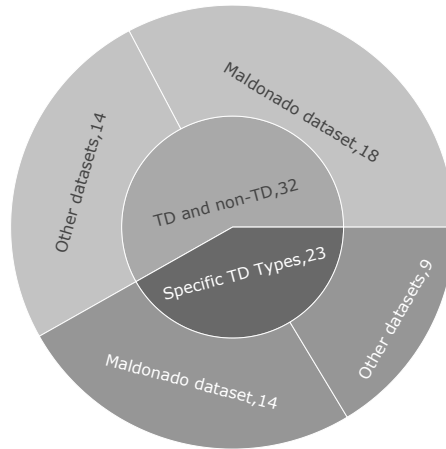


Figure 6: Distribution of the datasets used in the studies analyzed

⁵The dataset is available at <https://github.com/maldonado/tse.satd.data>. From this point onwards, we refer to this resource as the "Maldonado dataset".

4.6.1. Performance results based on the Maldonado dataset (project-level)

In this section, we report the results of the metric evaluations for the studies using the same dataset, namely studies using the Maldonado dataset. Maldonado et al. [63] created a dataset derived from source code comments of 10 open-source projects: Apache Ant, ArgoUML, Columba, EME, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and Squirrel. In total, they manually classified 62,566 comments into 5 SATD types, namely design debt, defect debt, documentation debt, requirement debt, and test debt classes.

We first visualize all the metric evaluations (precision, recall, and F1-score) in a comprehensive box plot: Figure 7 summarises the metric evaluations of the Maldonado dataset, that form a shared dataset for 19 of the analyzed articles.

As depicted in the box plots, the results for precision, recall, and F1-score aggregated per project have a wide distribution and a large number of outliers for each performance metric. This is understandable since Figure 7 contains all the results per project, and regardless of group, technique, and ML or DL algorithm. In the following sections, we cluster the same results based on *group* (NLP, NLP+ML, NLP+DL), *technique* (FBE, word embeddings, etc) and ML or DL *algorithm* (CNN, LSTM, etc).

When aggregated around the three basic groups (NLP, NLP+ML, and NLP+DL), we started to observe some interesting patterns (see Figure 8): although the performance metrics have still a large variance in terms of results, the NLP+DL group does consistently score better (using the medians of the box plot distributions) in precision and F1-score for all the projects in the Maldonado dataset, and in all but one project for the recall metric.

We then aggregated the results based on groups and both feature extractions and ML/DL techniques. These groupings are shown in Figure 9. In terms of techniques used, we also observed recurring patterns, based on each analyzed project: the PTWE consistently scores better precision, recall, and F1-score for each of the analyzed projects. This is also understandable as the embeddings used to classify TD have been pre-trained using context-relevant data and examples.

4.6.2. Performance Results based on "TD or non-TD" Detection

In this section, we present the aggregated summary of performance metrics from previous papers that aimed to detect binary TD from source artifacts: as shown in Figure 6. Out of these studies, 18 papers used the Maldonado dataset as a benchmark to distinguish TD comments from non-TD comments. Additionally, 14 studies focused on binary TD detection but did not explicitly use

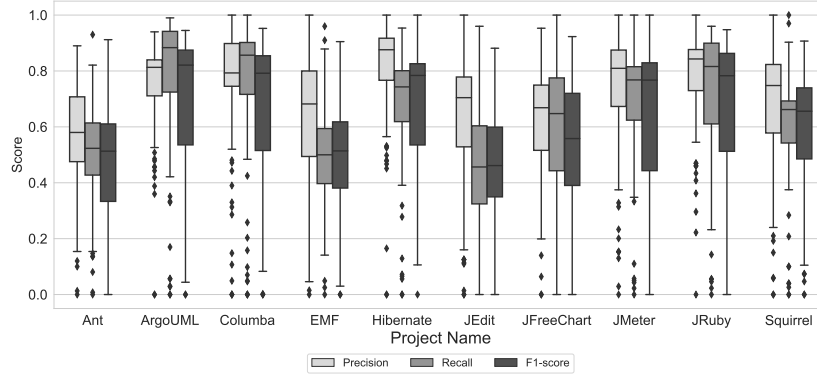


Figure 7: Box plots of performance results for all the papers using the dataset provided by Maldonado et al. [63]

the Maldonado dataset. In total, we analyzed 32 studies that investigated the detection of TD and non-TD comments.

Similarly to above, as visible in Figure 10 the NLP+DL group consistently generates the highest performance scores (i.e., the highest median values for precision, recall, and F1-score) compared to the NLP and NLP+ML groups, when attempting to detect whether source artefacts contain (or not) TD. It is noteworthy to observe that the NLP-based studies obtain a comparable high precision but a low recall: this means that very few results are returned with this approach, but most of its predicted labels (TD or not-TD) are correct.

Considering the techniques used, we do not have a comprehensive set of results: the available papers attempting a binary classification of TD do not show all the experimental variations shown in Figure 5 (for example, no papers in this subset use an NLP approach using PTWE). The resulting summaries are shown in Figure 11: a combination of NLP+DL and PTWE achieves better scores in terms of precision, recall, and F1-score, although the PTWE technique has not been so far used (or reported) with NLP+ML and NLP-based approaches. The observation made for the NLP results above is confirmed in the approach too: TP has, in general, a high precision, but at the expense of a very low recall.

Finally, considering the feature extraction methods and the ML or DL algorithms, we observed that the Text classification method for INductive word representations via Graph Neural Networks (in short, TextING) and the Embeddings from Language Models (ELMo) obtain the highest scores for precision and recall, respectively. On the F1-score, the TextING and ELMo algorithms achieved the highest scores with a rather small gap between them (as shown in

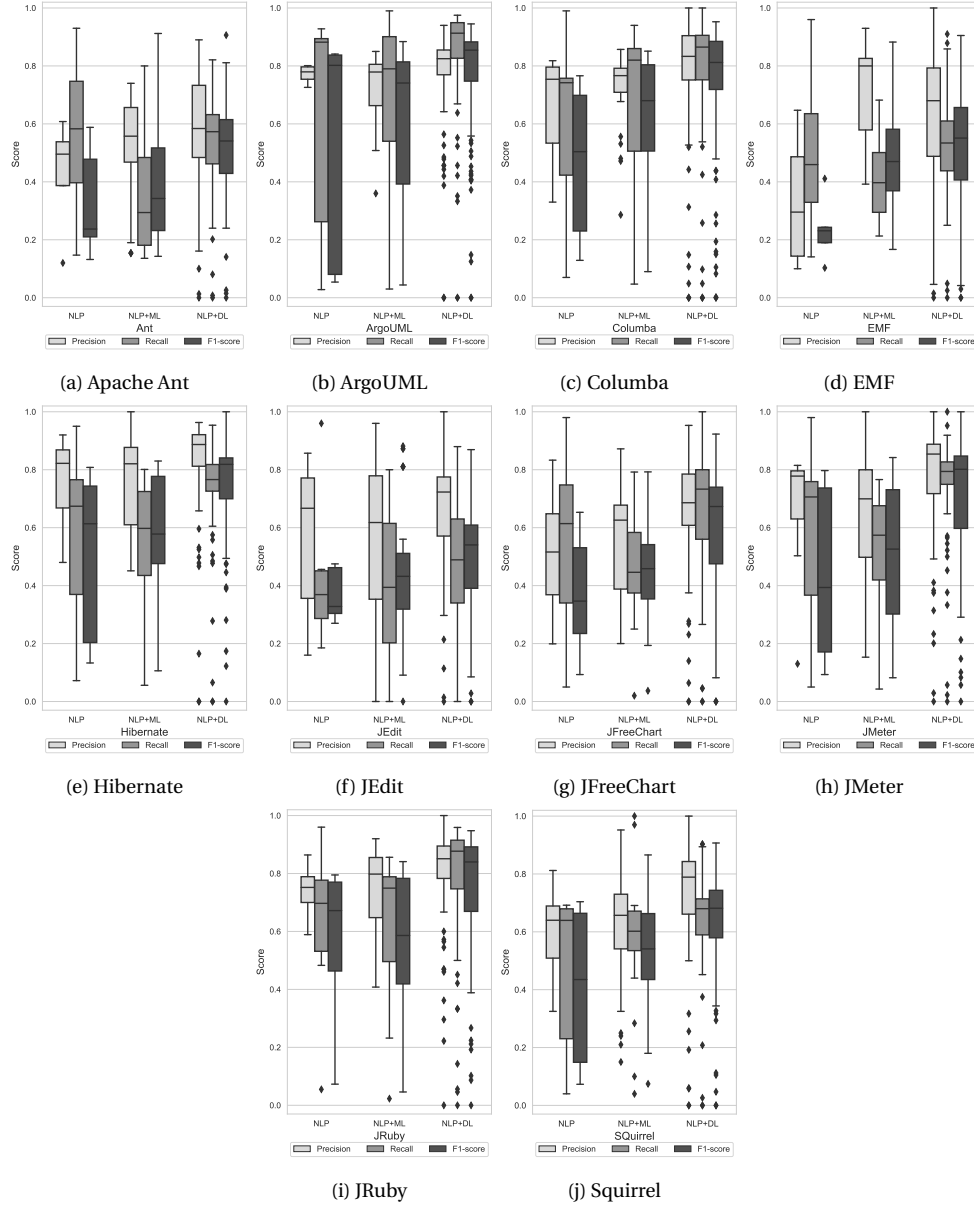


Figure 8: Box plots of metric evaluations using the Maldonado dataset, and clustered by group (NLP, NLP+ML, and NLP+DL). Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

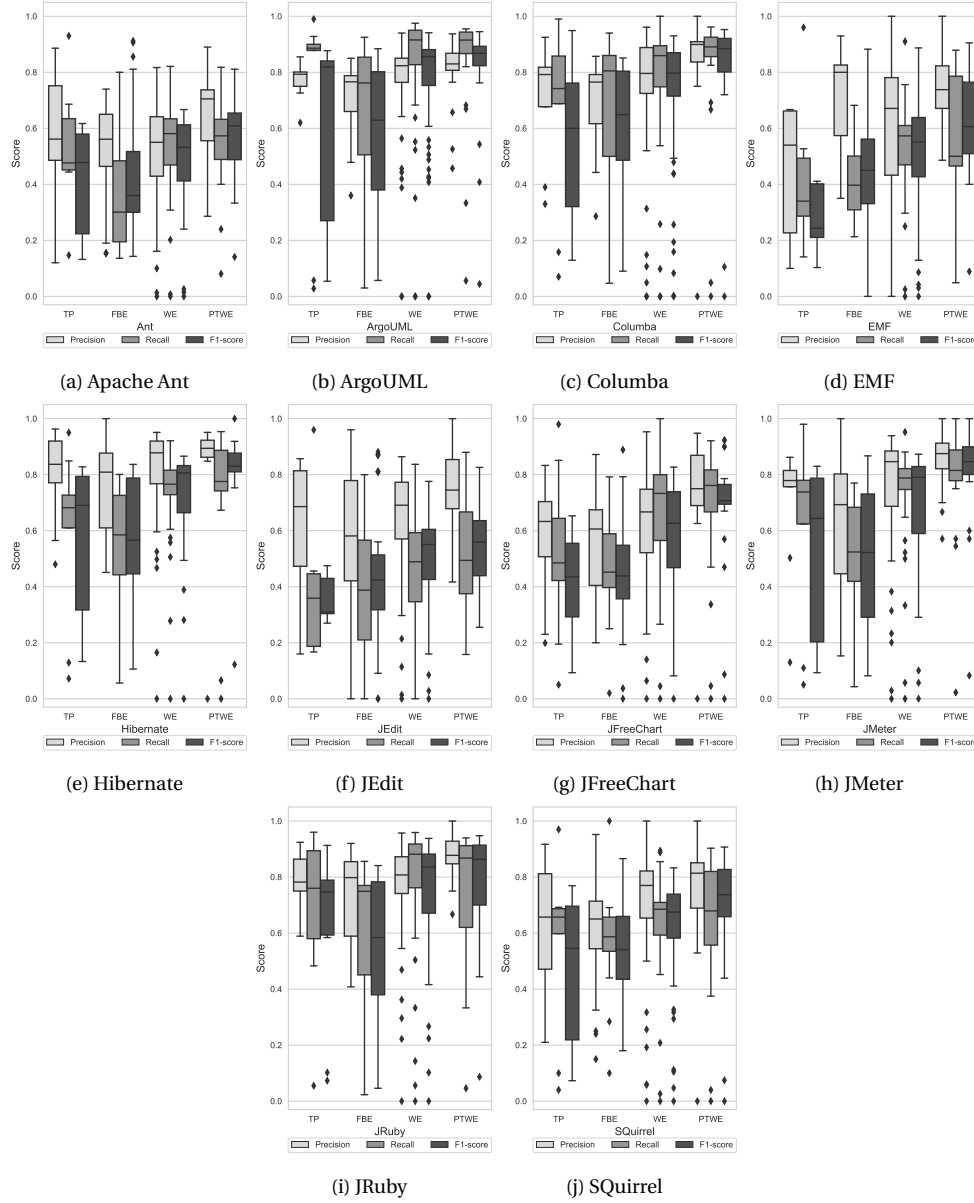


Figure 9: Box plot of metric evaluation results produced from the Maldonado dataset, grouped by project and the NLP technique used. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

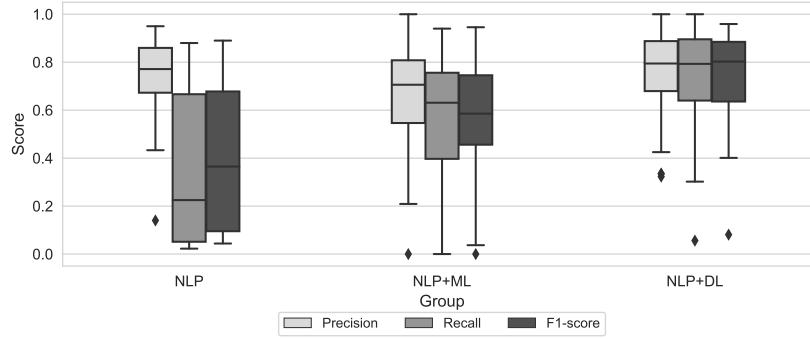


Figure 10: Performance results based on NLP groups - binary detection of TD or non-TD types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

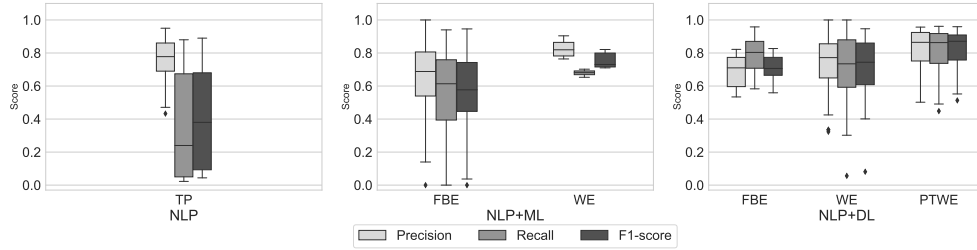


Figure 11: Performance results based on techniques - binary detection of TD or non-TD types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

Figure 12). On the other hand, SVM and Gated Graph Neural Networks (GGNN) come out on top when it comes to precision at the algorithm level. Afterward, the best scores were obtained by BiLSTM and GGNN for recall and F1-score (as shown in Figure 13).

4.6.3. Performance Result based on "Specific TD" Types Detection

In this third batch of results, we gathered the performance metrics for all the research studies that not only detected TD in software artefacts, but they also classified them using different types of TD (e.g., requirement TD, design TD and so on). At the aggregated level (Figure 14) the performance results are sensibly different from the performance of the binary detection, TD or non-TD. The results at this level show that the NLP group is more dominant in precision and recall.

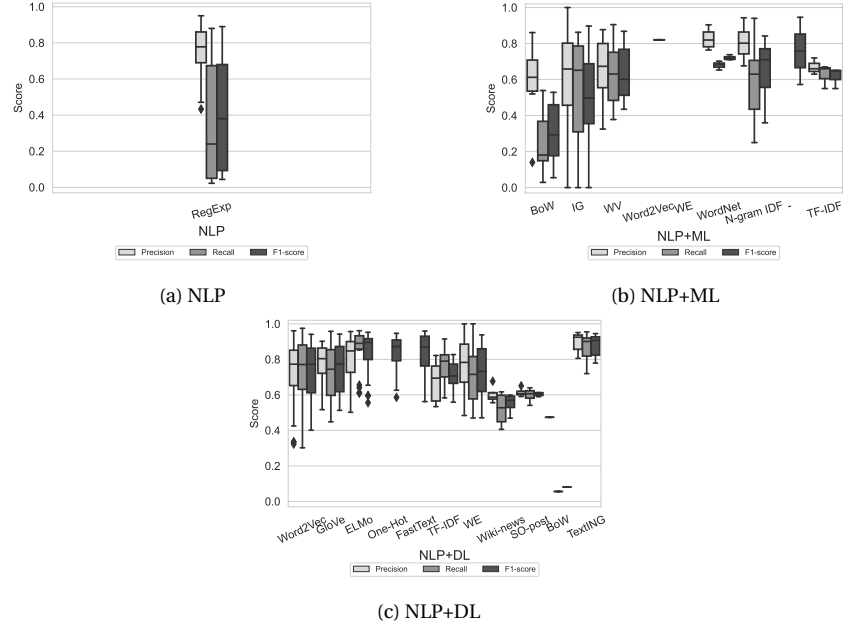


Figure 12: Performance results using feature extraction - binary detection of TD or non-TD types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

The performance comparison results at the technique level are more detailed than those at the group name level, but confirm the trends: Figure 15 illustrates the overall results of the techniques used in each group, and in the NLP group, PTWE performed significantly better than TP. In contrast, within the NLP+ML group, WE performed better than FBE, while in the NLP+DL group, PTWE is higher in terms of precision and F1-score, whereas FBE is superior in terms of recall.

Figure 16 displays the performance results of various feature extraction methods in detecting different types of technical debt, organized by group name. In the NLP group, the PTWE technique, particularly BERT, achieved superior results compared to ALBERT and RoBERTa. However, RegExp performed significantly worse than other PTWE techniques. In the NLP+ML group, IG exhibited better performance in terms of precision and recall, while TF-IDF outperformed other techniques in terms of F1-score. In the NLP+DL group, Word Vector, TF-IDF, and GloVe demonstrated excellent performance in terms of precision, recall, and F1-score, respectively.

Figure 17 presents the performance results of different algorithms used in

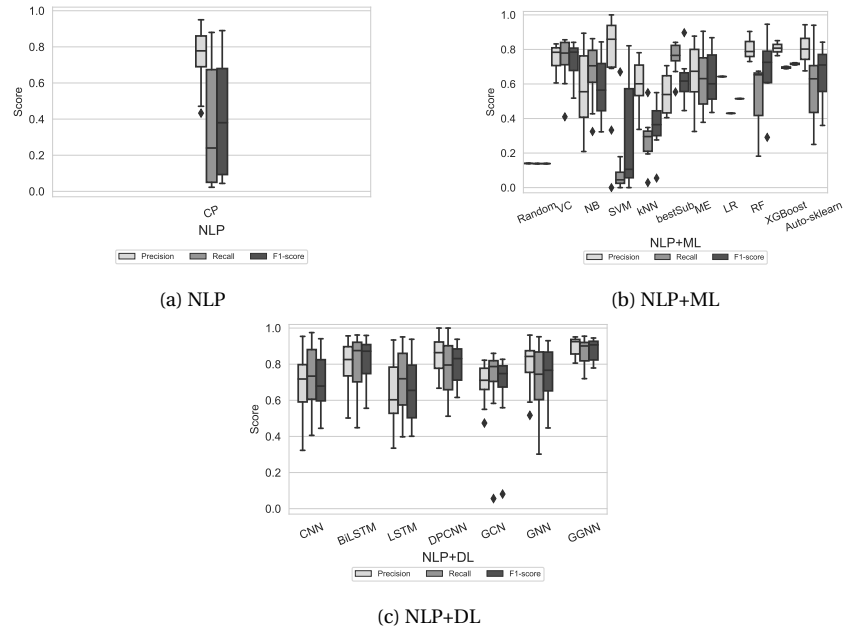


Figure 13: Performance results using ML and DL algorithms - binary detection of TD or non-TD types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

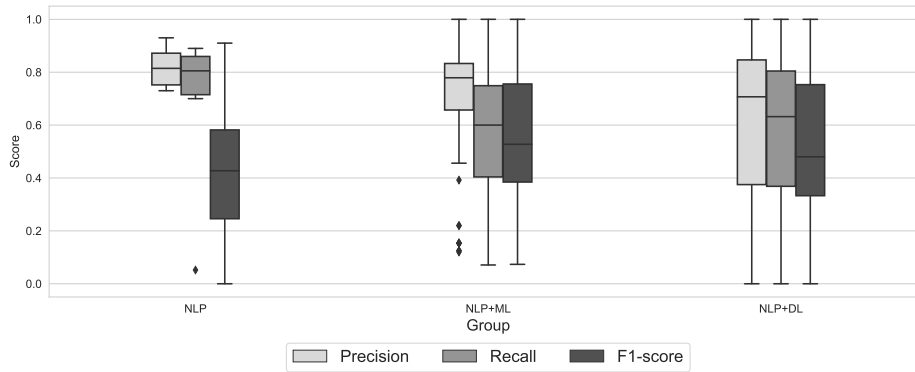


Figure 14: Box plot of metric evaluation results based on the group name for Specific TD Types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

detecting different types of technical debt, organized by group name. In the NLP group, BERT outperformed all other algorithms. In the NLP+ML group, SMO achieved the highest results in terms of precision, recall, and F1-score.

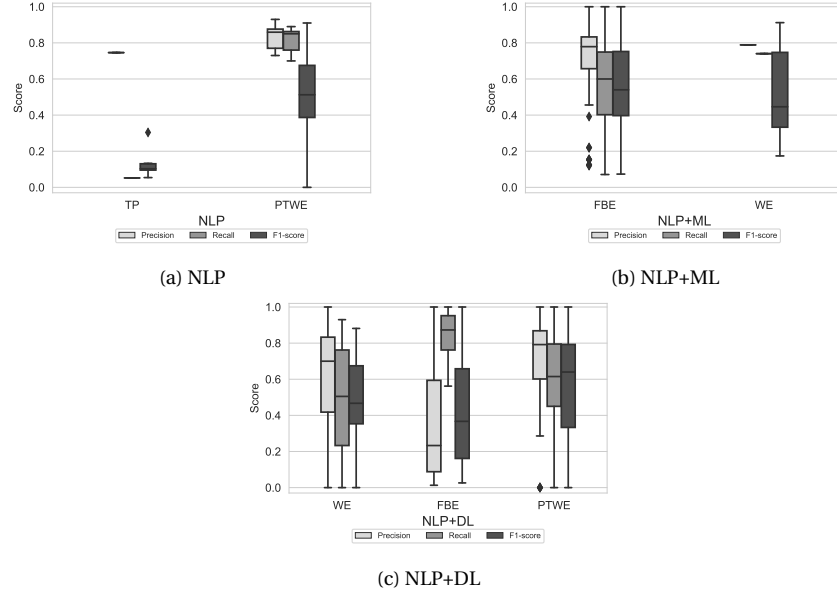


Figure 15: Box plot of metric evaluation results based on the technique name for Specific TD Types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

Auto-Sklearn also performed well in terms of precision. In the NLP+DL group, the combination of CNN and gated recurrent unit (CNN-GRU) showed excellent precision and F1-score, while LSTM performed best in terms of recall.

4.6.4. Performance of Technical Debt Detection in Relation to SDLC Phases

Figure 18 shows the performance of the group approach in relation to technical debt detection in each SDLC phase. We use a general 5-phase paradigm in the SDLC, namely requirement, design, implementation, testing, and maintenance phases [101, 102].

In the requirement phase, six articles reported their evaluation metric results [63, 87, 67, 92, 97, 45]. In the design phase, seven articles reported their results [63, 87, 67, 92, 97, 78, 45]. In the implementation, testing, and maintenance phases, there were two [78, 45], one [45], and two [78, 45] articles reporting their results, respectively.

In the requirement and design phases, all articles reported precision, recall, and F1-score metrics, while in the implementation, testing, and maintenance phases, all relevant studies reported only their F1-score. In Figure 19, we break down the results of metric evaluations based on the technique names within

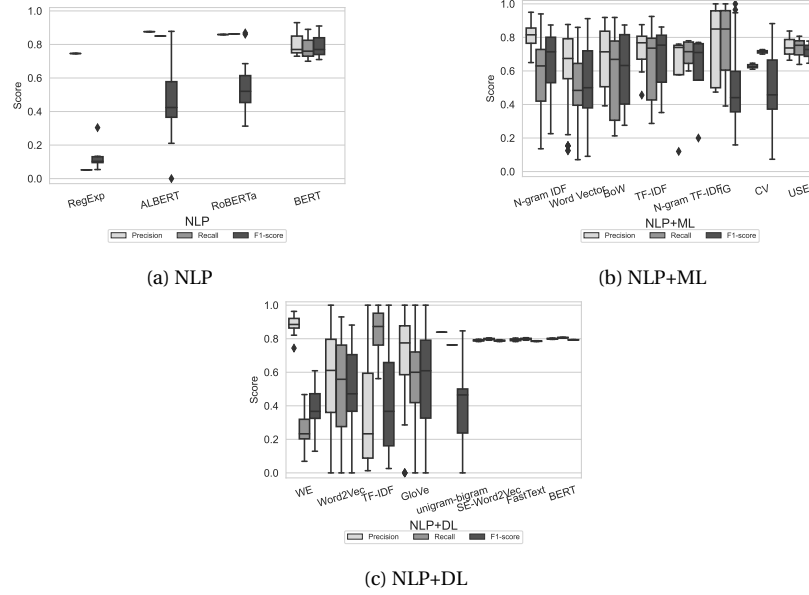


Figure 16: Box plot of metric evaluation results based on the feature extraction name for Specific TD Types. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

each group approach in each SDLC phase.

As seen in Figure 20, in the requirement and design phases, the NLP+DL group outperforms in terms of precision, recall, and F1-score compared to the NLP and NLP+ML groups. However, in the implementation, testing, and maintenance phases, the NLP group performs better than the NLP+ML and NLP+DL groups in terms of F1-score.

Figure 21 allows us to delve into the details of algorithms used for detecting technical debt in each SDLC phase. The box plots are grouped based on the three approaches: NLP, NLP+ML, and NLP+DL. This facilitates a comprehensive analysis of algorithm performance in individual studies, within the same group, and across all phases of the SDLC.

Most studies reported using at least one metric evaluation for their algorithms. While individual studies often determine their top-performing algorithms, we can compare the performance of these algorithms across studies. This provides insights into how well different algorithms perform in various contexts.

Figure 19 provides a detailed breakdown of the metric evaluation results for

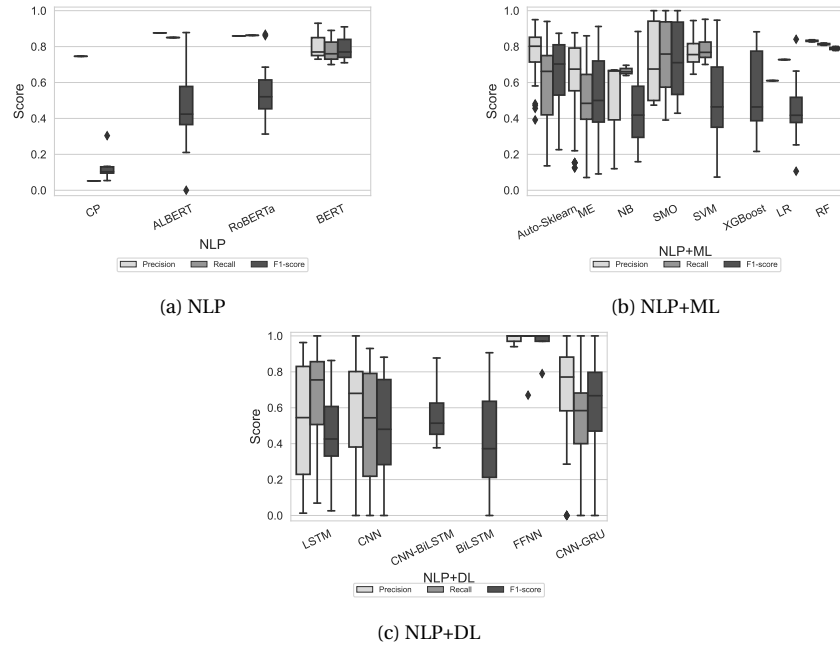


Figure 17: Box plot of metric evaluation results based on the algorithm name for Specific TD Types

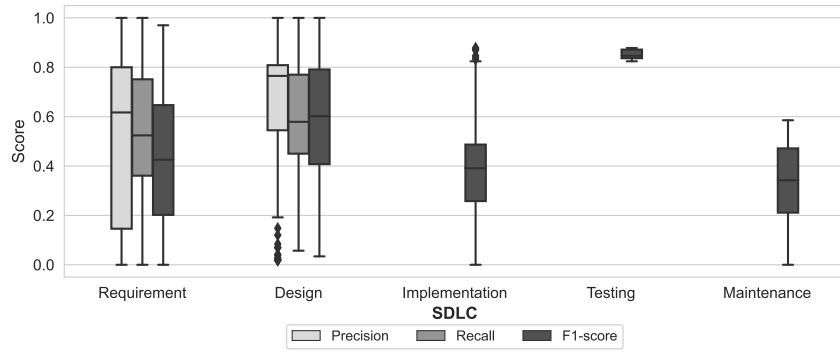


Figure 18: Evaluation metric results on technical debt detection in the SDLC phase. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

each technique in different phases of the SDLC. In the requirements phase, the PTWE technique outperforms the other three techniques in precision, recall, and F1-score. However, in the design phase, the PTWE technique performs best only in precision, while WE and FBE techniques achieve the highest recall

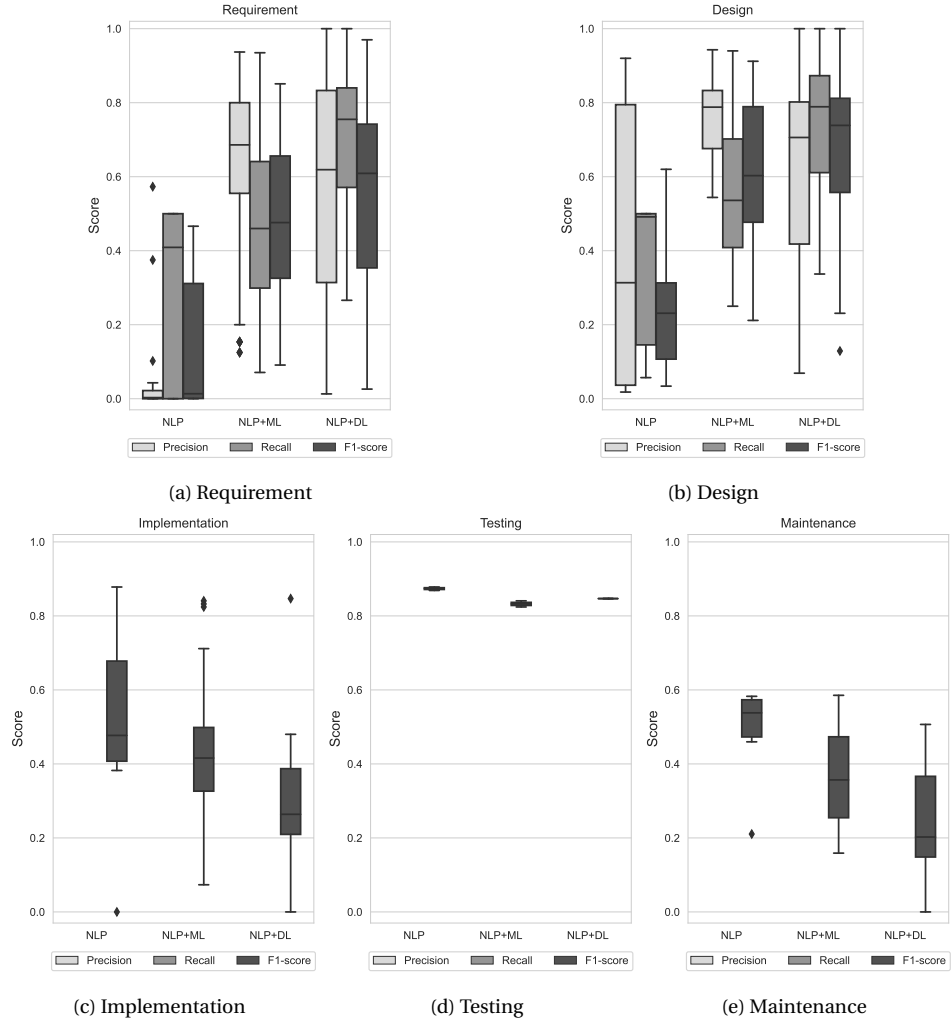


Figure 19: Evaluation metric results on technical debt detection in the SDLC phase. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

and F1-score, respectively.

In the implementation, testing, and maintenance phases, the PTWE technique exhibits the highest F1-score, surpassing the other three techniques. On the other hand, TP does not perform as well as the other techniques in the requirements and design phases, and it is not utilized in the implementation, testing, and maintenance phases across the selected articles.

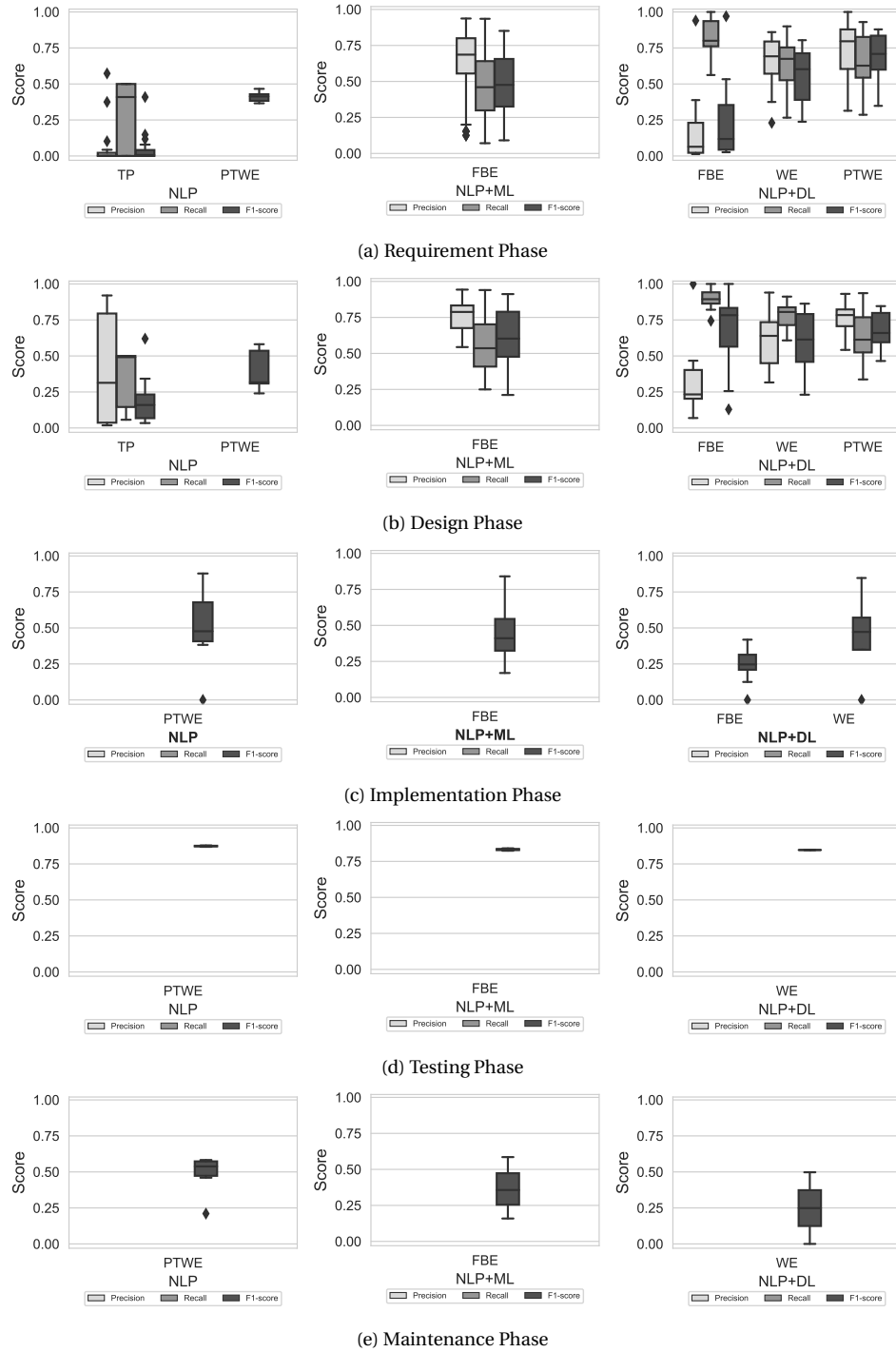


Figure 20: Evaluation metric results on technical debt detection in the SDLC phase. Larger graphs for all box plots are available at <https://github.com/edisutoyo/TD-Detection-using-NLP-SLR> for better visualization

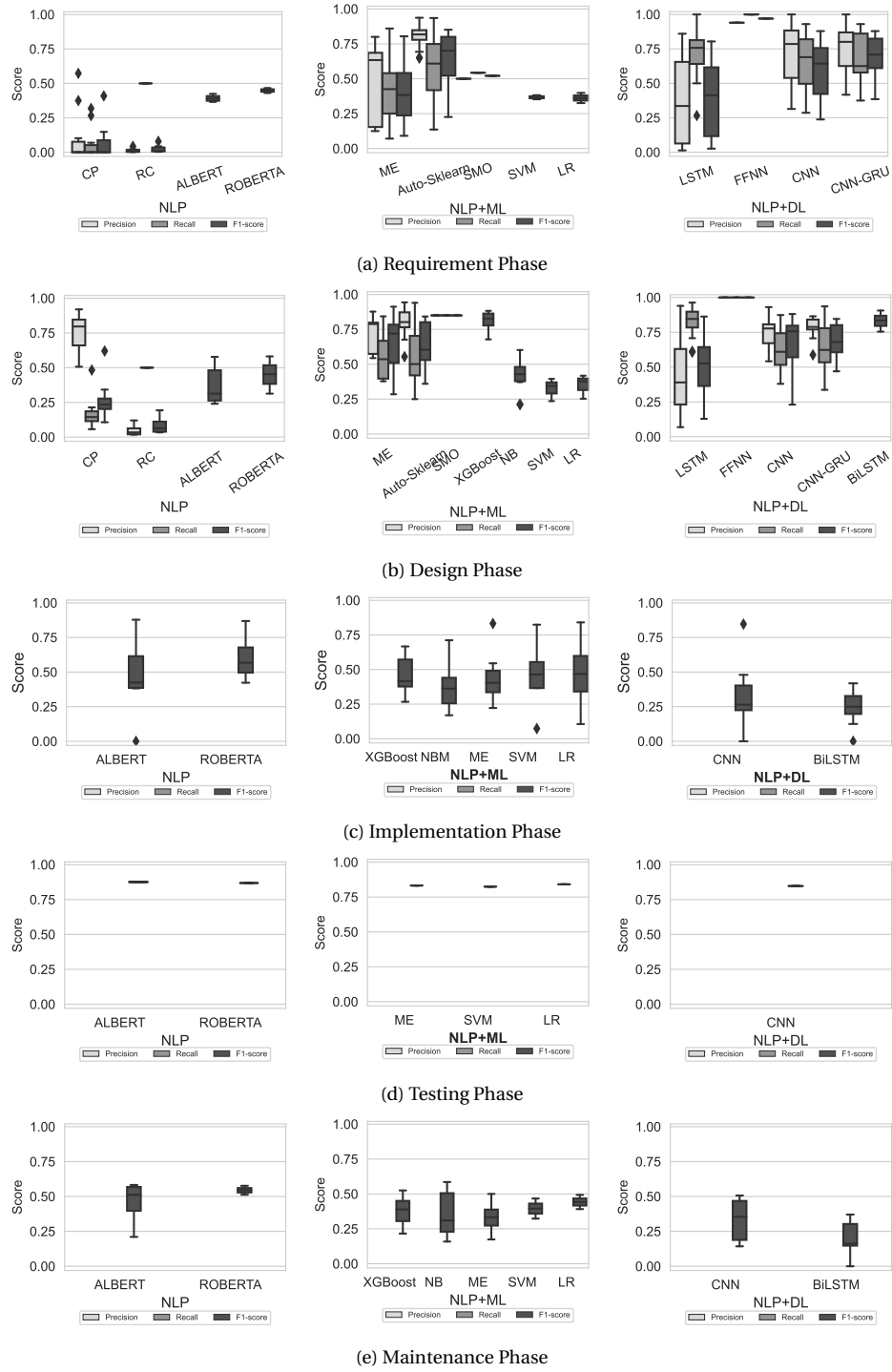


Figure 21: Evaluation metric results of the algorithms used in every SDLC phase

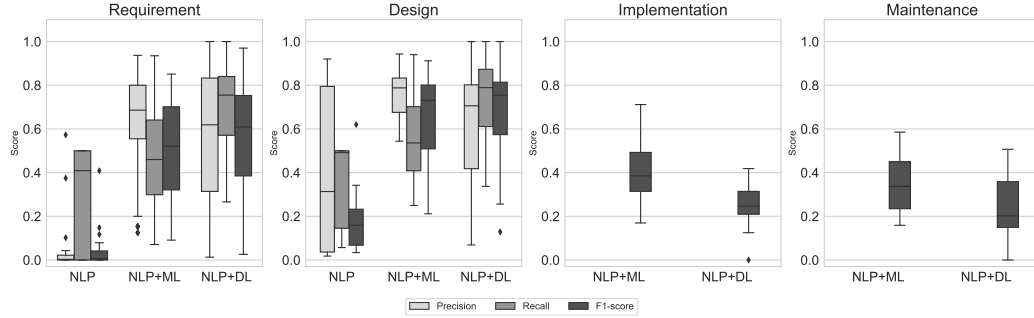


Figure 22: Box plots based on group name on technical debt detection in the SDLC phases from Maldonado dataset

Figure 19 depicts the SDLC phase performance results for all relevant articles. We may further refine these results by separating them according to the dataset used. Because there are seven papers in total throughout all SDLC phases, we divide them into two categories: studies that used the Maldonado dataset and those that used other dataset. This separation is carried out because six studies used the Maldonado dataset, and one used the R source code comments dataset.

Figure 22 depicts a box plot presenting the overall findings of Maldonado dataset performance metric analyses in terms of the precision, recall, and F1 scores of the relevant articles in each phase of the SDLC from the perspective of their technique name displayed by approach group name. To gain a deeper understanding, Figure 23 depicts the outcomes of metric evaluations for each SDLC phase. Precision, recall, and F1-score values were reported in the requirements and design phases. However, only the F1 score was available in the implementation and maintenance phase, and there was no result in the testing phase.

While study using another dataset is shown in Figure 24, which derived from just one study, namely from Sharma et al. [45]. Even though there is only one study, and it only reports the F1-score, the results from this article can be mapped to all phases of the SDLC. Furthermore, detailed performance results based on technique name can be seen in Figure 25.

5. Discussion

In this study, 55 primary studies were analyzed, each of which employed at least one of the described *groups*, namely NLP, NLP+ML, and NLP+DL, to solve

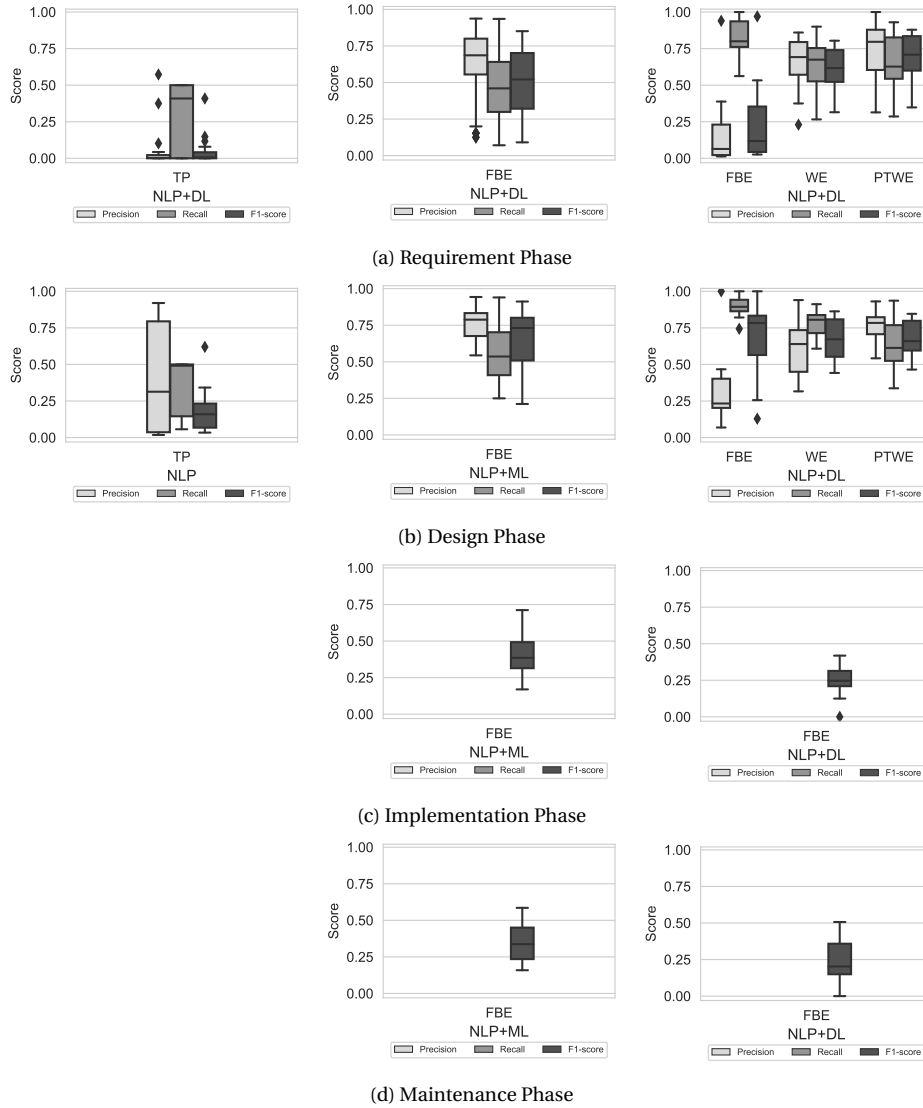


Figure 23: Evaluation metric results on technical debt detection in the SDLC phases from Maldonado dataset

the problem of detecting technical debt. Our main contribution is a summary of the findings of these studies by providing a detailed description of the feature extraction, algorithms, NLP group approach that supports technical debt detection in the SDLC phase, and metric evaluations used in the literature while also providing researchers with a recommendation for the best performing fea-

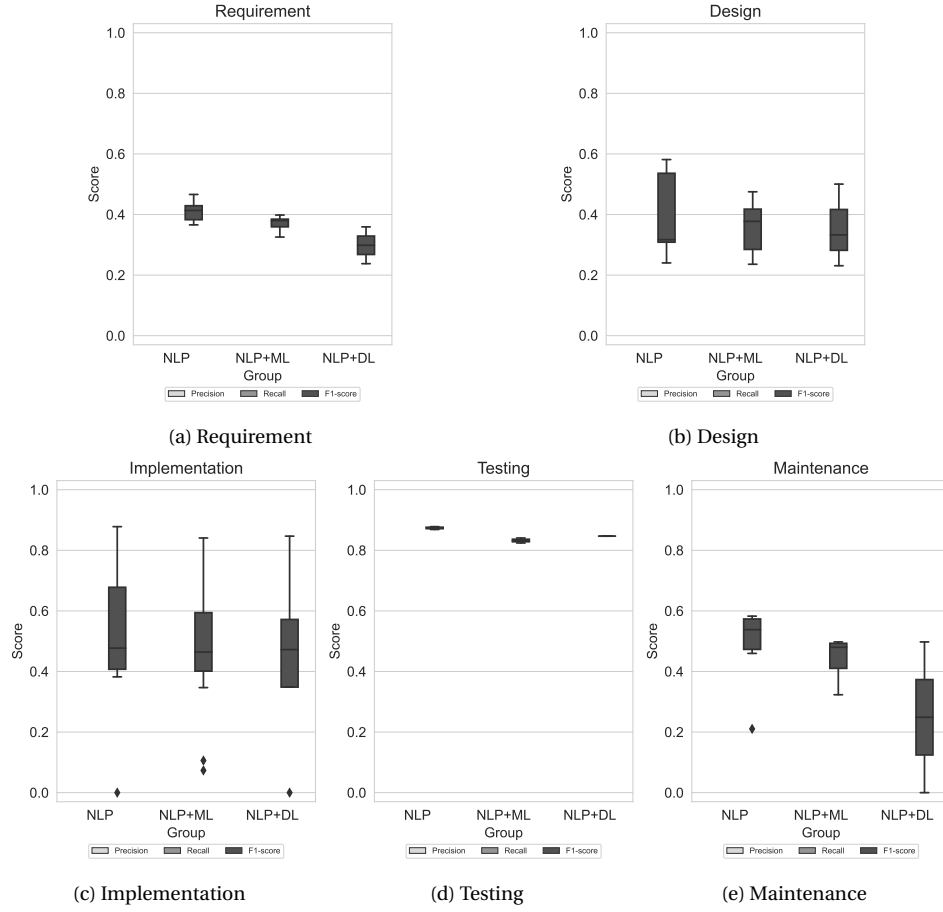


Figure 24: Evaluation metric results on technical debt detection in the SDLC phase from other datasets

ture extraction and algorithms.

5.1. RQ1: What NLP approaches are employed to detect technical debt in software engineering and/or software development?

Based on the studies analyzed, we concur by dividing the NLP approach into 3 large groups namely, NLP, NLP+ML, and NLP+DL. This group aims to provide a clear distinction of the approaches that have been proposed across the relevant articles.

Based on Figure 10, we provide a box plot to display the overall results for each of these groups. This figure provides an overview that the performance

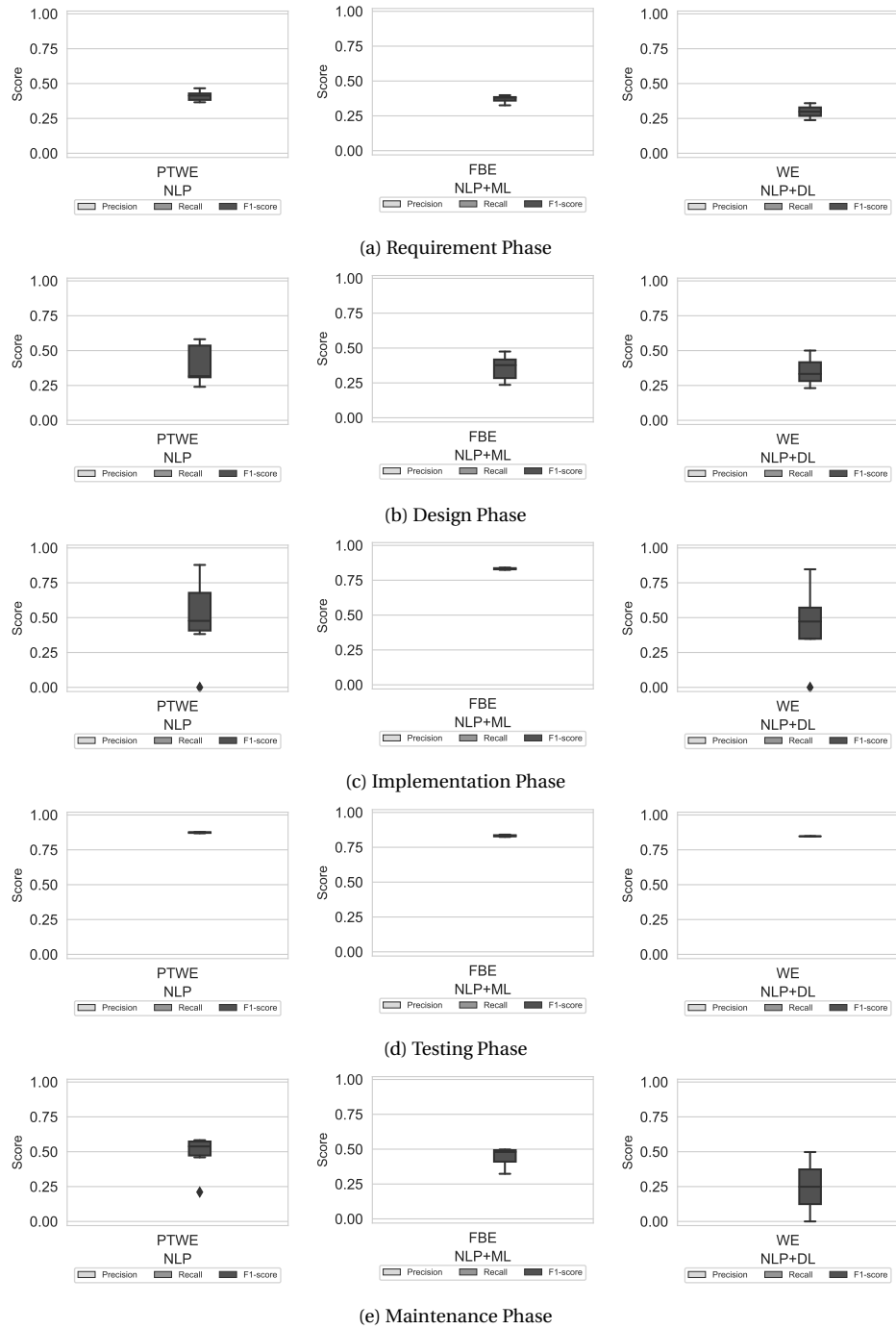


Figure 25: Evaluation metric results on technical debt detection in the SDLC phases from other datasets

of the NLP+DL group approach outperforms the NLP and NLP+ML group approaches in terms of precision, recall, and F1-score. These results are in line when associated with the trend of the approach taken: as seen in Figure 3, although the NLP and NLP+ML group approaches continue to increase from year to year, the studies included in the NLP+DL group approach have experienced a significant increase, even though they only started in 2018. In 2022, there were 8 articles utilizing one of the NLP+DL approaches, while there were 3 and 5 studies on the NLP and NLP+ML approaches, respectively.

We consider that studies utilizing the NLP+DL approach will grow rapidly, especially for the problem of technical debt detection: this approach will be increasingly supported by other researchers, especially for proposing new deep learning architectures to achieve even better performance. Moreover, work that incorporates multiple deep learning architectures will also increasingly attract researchers for use specifically on the topic of technical debt, and generally in the domain of software engineering.

Furthermore, Figure 20 shows the results of mapping the use of three group approaches in each SDLC phase. Overall results on the requirements and design phases show that the performance of the NLP+DL approach group is higher than the other two group approaches. Whereas in the implementation, testing, and maintenance phases the NLP group approach is better than the other two group approaches. However, these results were only derived from 2 (in the implementation and maintenance phase) articles and 1 (in the testing phase) article which only report the F1-score.

5.2. RQ2: What feature extractions are utilized in technical debt detection?

We can see in Figure 11 to answer this RQ 2. We can compare each technique names in the same group approach, as well as compare each technique names across group approaches.

According to the recent trend (2014–2022) as illustrated in Figure 4, studies employing TP and WE as feature extraction techniques fluctuated before leveling off. Up until 2022, the use of FBE is projected to increase. This occurs, in our perspective, since the FBE technique can be employed for both NLP+ML and NLP+DL group approaches. PTWE, which was recently employed by academics to identify technical debt in 2020, has also grown. Although PTWE can only be employed in the NLP+DL group method, the utilization of PTWE has increased as the number of deep learning architectures has grown.

5.3. RQ3: *In what phases of software development does the NLP approach support technical debt detection?*

In order to answer research question RQ3, we first collected and listed all types of technical debt from our primary studies. We found that there are 23 types of technical debt (including on-hold, TD, and non-TD types). The first and second authors then discussed and observed each type of technical debt discovered. Then, we had a discussion that led to an agreement to place similar types of technical debt close together, such as code and build debt or design and architecture. Then, we arranged them according to the SDLC phases. The results of discussions and agreements regarding the placement of TD types and also the mapping of TD types to the SDLC phase are shown in Table 12. Moreover, we did not place TD, and non-TD types into any of the SDLC phases, but we placed them in the "Other" category. Through discussion, disagreements over the placement and sequencing were resolved.

Table 13 provides a general overview of the analyzed studies which are mapped into the 5 SDLC phases (excluding the OTH column). In total, there are 18 studies across all phases of the SDLC, with 6, 7, 2, 1, and 2 studies in the requirements, design, implementation, testing, and maintenance phases, respectively.

5.4. RQ4: *What metric evaluations are commonly used for technical debt detection?*

In evaluating the performance of ML algorithms and DL architecture in the case of supervised learning, we use the confusion matrix as a reference. The confusion matrix represents the predictions and actual conditions of the data generated by the ML algorithm. Based on the confusion matrix, we can determine accuracy, precision, recall, and F1-score.

Based on the studies that were analyzed, all of them used unbalanced datasets, so all researchers did not use accuracy as a reference for algorithm performance. In terms of technical debt detection, we prefer True Positives (TP) and really do not want False Positives (FP) to occur. For instance, we prefer a source code comment that is actually non-TD but is predicted to be TD, rather than a source code comment that is actually TD but predicted to be non-TD (so that the comment is missed and not be addressed). Therefore, for technical debt detection, precision, recall, and F1-score are only used.

Furthermore, in the article by Sharma et al. [45] just utilized the F1-score. In our opinion, they only used F1-score because they employ 12 classes in their work, and imbalanced classes appeared in their dataset. However, in the studies we analyzed, only their work could we map to all phases of the SDLC.

Therefore, we invite researchers to create and share datasets that can be used specifically for technical debt detection that can be mapped to all SDLC phases. This dataset is very useful, especially for the development of research in the technical debt area, and generally for the development of the software engineering domain.

5.5. RQ5: Which NLP method performs best for identifying technical debt?

In this section, we collect all the results that we provided in the form of box plots above: our objective is to evaluate which NLP group, technique, feature extraction methods, and ML/DL algorithms provide the best performance in detecting TD, as reported from the relevant literature. We articulate this section as follows: 5.5.1 reports, at a general level, which groups and techniques show the best performance to detect technical debt using NLP. These first results aggregate all the software projects analyzed so far, and all NLP techniques, so they are necessarily very coarse, and need more refinement. Therefore in 5.5.2 we summarise the most promising performances obtained either when a binary detection (TD or non-TD) is sought, or when a specific TD type needs to be detected from software artefacts. Finally, in 5.5.3 we present the best reported performances, once clustered around the basic blocks of the SDLC.

5.5.1. Best NLP Performances - Overall View

We have thoroughly covered all of the metric evaluation results in order to address RQ 5. The performance results are summarized in Table 14 based on the name of the group and technique. The overall findings demonstrate that the precision, recall, and F1-score values are consistently highest in the NLP+DL group. Meanwhile, the results based on the technique name, the PTWE technique constantly obtains the highest precision, recall, and F1-score. In addition, the FBE technique can also be an alternative because it is capable of producing recall and F1-score that are close to PTWE performance.

Table 14: Summary of performance results

Category	Metric Evaluations		
	Precision	Recall	F1-score
Group	NLP+DL	NLP+DL	NLP+DL
Technique	PTWE	FBE, PTWE	FBE, PTWE

5.5.2. Best Performance based on TD Types

Table 15 illustrates the performance results depending on the group, technique, feature extraction, and algorithms. These categories are separated into two groups based on their class, which are referred to as "TD and non-TD" types and "Specific TD" types, respectively.

The findings for the "TD and non-TD" type reveal that, generally, at the group and technical level, NLP+DL and PTWE generate the highest precision, recall, and F1-score values. These findings suggest that when it comes to detecting technical debt in source comments, the combined performance of NLP+DL employing the PTWE technique is the best option, regardless of the type of technical debt being detected. Moreover, according to the studies that were reviewed, the best performance can be achieved by applying feature extraction techniques such as Word Embedding (WE), WordNet, ELMo, and FastText to various models such as SVM, Deep Pyramid CNN (DPCNN), and Bidirectional Long Short-Term Memory Neural Network (BiLSTM).

Table 15: Summary of performance results based on TD types

Category	Metric Evaluations		
	Precision	Recall	F1-score
TD and non-TD Types			
Group	NLP+DL	NLP+DL	NLP+DL
Technique	PTWE	PTWE	PTWE
Feature Extraction	WE, WordNet	ELMo	FastText, ELMo
Algorithm	SVM, DPCNN	BiLSTM	BiLSTM
Specific TD Types			
Group	NLP+DL	NLP+DL	NLP+DL
Technique	PTWE	FBE, PTWE	FBE, PTWE
Feature Extraction	IG	IG, TF-IDF	BoW, TF-IDF, N-gram IDF
Algorithm	FNN, SMO, CNN-GRU	SMO, LSTM, BERT	SMO, FNN, BERT

While the results for the "Specific TD" type tend to be more varied at the group name level, NLP and NLP+DL achieve the best results in terms of precision and recall. On the other hand, NLP+ML achieves the highest results in terms of the F1-score. Similarly, in terms of the technical name, PTWE is superior to the other two techniques in terms of precision. In contrast, in terms of recall and F1-score, the techniques of FBE and PTWE can be utilized as the top choices because they provide the most significant scores. Furthermore, in feature extraction, IG is able to achieve the highest precision values. In the recall, there are IG and TF-IDF, and in the F1-score there are BoW, TF-IDF, and N-gram

IDE, which can produce the highest scores. In terms of precision, FNN, SMO, and CNN-GRU produce the highest scores. Moreover, SMO, LSTM, and BERT algorithms are superior to other algorithms since they have the highest recall scores. On the F1-score, SMO, FNN, and BERT achieved the highest scores. This "Specific TD" type has a broader range of algorithm models since each metric evaluation yields a distinct best algorithm (except for SMO, which appears in all metric evaluations).

5.5.3. Best Performance in Relation to the SDLC Phases

A summary of the performance results of technical debt detection in relation to the SDLC phases can be shown in Table 16. This table provides a comprehensive overview of the group names as well as the techniques used in the studies analyzed in each SDLC phase.

Table 16: Summary of performance results in relation to SDLC phase

SDLC Phase	Group	Metric Evaluations		
		Precision	Recall	F1-score
REQ	NLP+DL	PTWE	FBE	PTWE
DES	NLP+ML, NLP+DL	PTWE, FBE	FBE, WE	FBE
IMP	NLP	-	-	PTWE
TES	NLP	-	-	PTWE
MAI	NLP	-	-	PTWE

The NLP+DL group performs better than NLP+ML and NLP in the Requirement (REQ) phase. PTWE achieves the highest precision and F1-score values, whereas FBE achieves the highest recall. The best performance results in the Design (DES) phase can use either NLP+ML or NLP+DL group approach. Meanwhile, the best techniques that can produce the highest precision and recall are PTWE and FBE, but for the recall is FBE. Furthermore, the NLP group outperforms in the Implementation (IMP) phase, while the technique that can be used as the first choice is PTWE because it achieves the highest F1-score. In addition, the NLP group was able to come out as the winner in the Testing (TES) and Maintenance (MAI) phases. Also, in both phases, the highest scores of precision, recall, and F1-score were achieved by PTWE.

Furthermore, when viewed in terms of the algorithms used, CNN-GRU outperforms in all metric evaluations used in the REQ phase. In addition, there were also alternatives such as CNN, which has a high score in precision and recall, and Auto-Sklearn, which has a high score in F1-score. In the DES phase,

Table 17: Summary of performance results of the algorithms used in relation to SDLC phase

SDLC Phase	Metric Evaluations		
	Precision	Recall	F1-score
REQ	CNN, CNN-GRU	CNN, CNN-GRU	Auto-Sklearn, CNN-GRU
DES	Auto-Sklearn	LSTM	BiLSTM
IMP	-	-	RoBERTa
TES	-	-	ALBERT
MAI	-	-	ALBERT

Auto-Sklearn, LSTM, and BiLSTM outperform the other algorithms in terms of precision, recall, and F1-score, respectively. In the last 3 phases, there were only F1-score values reported, with the best results in the IMP, TES, and MAI phases being RoBERTa, ALBERT, and ALBERT, respectively. The detailed results are shown in Table 17.

Table 18: Summary of performance results of the feature extractions used in relation to SDLC phase

SDLC Phase	Metric Evaluations		
	Precision	Recall	F1-score
REQ	GloVe, N-gram IDF	TF-IDF	GloVe, N-gram IDF
DES	GloVe, N-gram IDF	TF-IDF	GloVe, N-gram IDF
IMP	-	-	RoBERTa
TES	-	-	ALBERT
MAI	-	-	ALBERT

Table 18 illustrates the most effective feature extractions utilized in each SDLC phase. The GloVe and N-gram IDF are best at precision and recall, while TF-IDF is best at F1-scored in the REQ and DES phases. In the last three phases, only F1-score values are reported. In the final three phases, only F1-score values were reported, with RoBERTa, ALBERT, and ALBERT achieving the highest results in the IMP, TES, and MAI phases, respectively.

5.6. Implications for Researchers and Practitioners

On the basis of the SLR results shown above, we identified the following implications that can be brought to the attention of researchers.

Validity and representativeness of datasets. Researchers play an essential role in validating the accuracy of TD detection approaches. To ensure the accu-

racy and reliability of the proposed approaches, they must establish benchmark datasets and evaluate the performance of various tools and methods against real-world scenarios. The reliability of detection approaches is heavily dependent on the quality and representativeness of the datasets. Currently, there are only a few accessible datasets, as the work underneath each is incredibly complex and takes a lot of work. The implications of our findings are related to these shared datasets, but further real-world samples from software repositories and projects must be mined.

Data-balancing methods. Imbalanced data is a critical issue in machine learning and deep learning. Data imbalances have a negative impact on classification results, as minority classes are frequently misclassified as majority classes. In the Maldonado dataset, for example, which is frequently used for SATD detection, there is a very substantial proportion imbalance between the SATD and non-SATD classes, and extremely imbalanced for the specific types of SATD. The dataset for the classification task should ideally have a balanced proportion of classes. As a result, a variety of data-balancing methods are required to make the data between classes more balance.

ML- and DL-based approaches specialized for data types. Researchers must continue to innovate and develop improved techniques for detecting technical debt. This requires the development of specialized algorithms and NLP techniques to extract instances of technical debt from source code comments, issue trackers, pull requests, commit messages, and other possible artifacts.

Combining techniques for efficiency. The selection of feature extraction and algorithm has a significant impact on the efficiency of the proposed method. Self-admitted technical debt detection can be combined with static and dynamic analysis techniques for a more complete understanding of technical debt in software projects. Researchers can investigate methods for combining multiple detection methods to increase the metric evaluation results.

Meanwhile, the implications of this finding for practitioners are described below.

Continuous Monitoring and Improvement. The detection of technical debt is a continual effort. In order to keep technical debt under control throughout the SDLC, practitioners need to incorporate it into their strategies for continuous monitoring and improvement.

Early TD Detection. By leveraging the rapidly growing research in the technical debt detection area, practitioners can identify potential issues early in the SDLC. This enables teams to take preventative measures and forestall the accumulation of substantial technical debt over time.

Process enhancement. Practitioners can use the knowledge acquired from technical debt detection to enhance their development processes. Developer teams can adjust their coding guidelines and practices to minimize future and recurring occurrences of technical debt if they are aware of its common patterns.

6. Threats to Validity

Search string. A suitable search query is one of the most critical aspects of properly conducting an SLR. In the pilot search (S1), only "nlp" was used. Although we used the "Full text and Metadata" search option, it became apparent from the search results that we had overlooked some articles. Therefore, "natural language processing" was also employed in S2. In addition, in S2, we added "identification" as a synonym for "detection," but after obtaining the results of this query, we realized that there are articles that use other forms of these terms. Therefore, we utilized S3 with the asterisk (*) as a wildcard to ensure that no articles were missed and to mitigate this threat.

Studies included or excluded. In addition, the inclusion and exclusion criteria used in this SLR would necessitate additional analysis to determine their impact on the article's validity. However, Kitchenham and Charters [27] provided detailed guidelines for conducting SLR by planning, conducting, and reporting in order to enhance the quality of the SLR employed in this article. Since the first author carried out the data extraction processes independently, there is a chance of personal or author bias. Therefore, in order to ensure that data was acquired objectively, discussions were always conducted between the first and second authors to resolve disagreements or conflicts of opinion during meetings.

Inaccuracies in data extraction. Concerning data extraction, there may be prejudices that influence the grouping and analysis results of the selected studies. To reduce this bias, the authors in this SLR discussed the extracted data items and reached a consensus on their respective meanings. In addition, pilot extraction and selection of data were performed by the authors, and disagreements regarding the data results were discussed and resolved. Therefore, the authors evaluated the extracted data items and resolved any disagree-

ments that arose. With measures implemented to reduce bias, the extracted data items' precision was further enhanced.

Technical debt type. There is a chance that we might miss particular types of technical debt. Our SLR has identified and accumulated 23 types of technical debt (including on-hold, TD, and non-TD). However, according to studies conducted by Alves et al. [12], there were 13 (thirteen) types of TD. The results of our study demonstrate that we have identified a type of technical debt that encompasses all types of TD mentioned in the article (with the exception of Test Automation Debt) and have accumulated more types than those listed.

7. Conclusion

This article presents a systematic literature review of 55 articles published from 2014 to 2022, which analyzed and identified state-of-the-art feature extractions, machine learning models, and deep learning architectures for detecting technical debt. Then, we investigated in depth which feature extraction, model, and architecture were employed to identify technical debt in each SDLC phase. All approaches proposed in the analyzed studies were grouped into NLP, NLP+ML, and NLP+DL. This allowed us to discuss the performance in three different ways: first, we examined the performance of each study to determine the performance of the algorithms reported within the study; second, we compared the performances within the same group name; and third, we were able to compare all algorithms reported in the 18 studies across all group names and all phases of the SDLC.

Overall results showed that the NLP+DL group consistently outperformed the other groups in terms of precision, recall, and F1-score. Meanwhile, based on the name of the technique, the PTWE technique approach consistently achieved high performance. In addition, the FBE technique, on the other hand, could provide recalls and F1-score results comparable to those produced by the PTWE; therefore, it can be considered an alternative.

Data availability

Supplementary material related to this article, including all the box plots as images, can be found online at

<https://github.com/edisutoyo/TD-Detection-using-NLP-SLR>

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the Indonesian Education Scholarship Program, Ministry of Education, Culture, Research, and Technology, Indonesia, Ref. Number: 1036/J5.2.3/BPI.LG/VIII/2022.

References

- [1] N. Rios, M. G. de Mendonça Neto, R. O. Spínola, A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners, *Information and Software Technology* 102 (2018) 117–145. [doi:10.1016/j.infsof.2018.05.010](https://doi.org/10.1016/j.infsof.2018.05.010).
- [2] T. Klinger, P. Tarr, P. Wagstrom, C. Williams, An enterprise perspective on technical debt, *Proceedings - International Conference on Software Engineering* (2011) 35–38 [doi:10.1145/1985362.1985371](https://doi.org/10.1145/1985362.1985371).
- [3] T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe, J. Bosch, Embracing technical debt, from a startup company perspective, *Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018* (2018) 415–425 [doi:10.1109/ICSME.2018.00051](https://doi.org/10.1109/ICSME.2018.00051).
- [4] P. Avgeriou, P. Kruchten, I. Ozkaya, C. Seaman, Managing Technical Debt in Software Engineering, *Dagstuhl Reports* 6 (4) (2016) 110–138. [doi:10.4230/DagRep.6.4.110](https://doi.org/10.4230/DagRep.6.4.110).
- [5] W. Cunningham, The WyCash portfolio management system, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA Part F1296* (October) (1992) 29–30. [doi:10.1145/157709.157715](https://doi.org/10.1145/157709.157715).

- [6] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, N. Zazworka, Managing technical debt in software-reliant systems, in: Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010, 2010, pp. 47–51. doi:[10.1145/1882362.1882373](https://doi.org/10.1145/1882362.1882373).
- [7] E. Tom, A. Aurum, R. Vidgen, An exploration of technical debt, Journal of Systems and Software 86 (6) (2013) 1498–1516. doi:[10.1016/j.jss.2012.12.052](https://doi.org/10.1016/j.jss.2012.12.052).
- [8] Z. Li, P. Avgeriou, P. Liang, A systematic mapping study on technical debt and its management, Journal of Systems and Software 101 (2015) 193–220. doi:[10.1016/j.jss.2014.12.027](https://doi.org/10.1016/j.jss.2014.12.027).
- [9] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, C. Seaman, Identification and management of technical debt: A systematic mapping study, Information and Software Technology 70 (2016) 100–121. doi:[10.1016/j.infsof.2015.10.008](https://doi.org/10.1016/j.infsof.2015.10.008).
- [10] L. Xiao, Y. Cai, R. Kazman, R. Mo, Q. Feng, Identifying and quantifying architectural debt, Proceedings - International Conference on Software Engineering 14-22-May- (2016) 488–498. doi:[10.1145/2884781.2884822](https://doi.org/10.1145/2884781.2884822).
- [11] P. Kruchten, R. L. Nord, I. Ozkaya, Technical debt: From metaphor to theory and practice (2012). doi:[10.1109/MS.2012.167](https://doi.org/10.1109/MS.2012.167).
- [12] N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, R. O. Spínola, Towards an ontology of terms on technical debt, in: Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014, Institute of Electrical and Electronics Engineers Inc., 2014, pp. 1–7. doi:[10.1109/MTD.2014.9](https://doi.org/10.1109/MTD.2014.9).
- [13] G. Sierra, E. Shihab, Y. Kamei, [A survey of self-admitted technical debt](https://doi.org/10.1016/j.jss.2019.02.056), Journal of Systems and Software 152 (2019) 70–82. doi:[10.1016/j.jss.2019.02.056](https://doi.org/10.1016/j.jss.2019.02.056).
URL <https://doi.org/10.1016/j.jss.2019.02.056>
- [14] N. Zazworka, R. O. Spínola, A. Vetro, F. Shull, C. Seaman, A case study on effectively identifying technical debt, in: ACM International Conference Proceeding Series, 2013, pp. 42–47. doi:[10.1145/2460999.2461005](https://doi.org/10.1145/2460999.2461005).

- [15] R. O. Spínola, A. Vetrò, N. Zazworka, C. Seaman, F. Shull, Investigating technical debt folklore: Shedding some light on technical debt opinion, 2013 4th International Workshop on Managing Technical Debt, MTD 2013 - Proceedings (2013) 1–7 [doi:10.1109/MTD.2013.6608671](https://doi.org/10.1109/MTD.2013.6608671).
- [16] E. D. Liddy, Natural language processing (2001).
- [17] G. G. Chowdhury, Natural language processing, Annual Review of Information Science and Technology 37 (2003) 51–89. [doi:10.1002/ARIS.1440370103](https://doi.org/10.1002/ARIS.1440370103).
- [18] N. Indurkha, F. J. Damerau, N. Indurkha, F. J. Damerau, Handbook of natural language processing, Chapman and Hall/CRC, 2010. [doi:10.1201/9780824746346](https://doi.org/10.1201/9780824746346).
- [19] K. R. Chowdhary, Natural Language Processing, in: Fundamentals of Artificial Intelligence, Springer, 2020, pp. 603–649. [doi:10.1007/978-81-322-3972-7_{_}19](https://doi.org/10.1007/978-81-322-3972-7_{_}19).
- [20] P. Yalla, N. Sharma, Integrating natural language processing and software engineering, International Journal of Software Engineering and its Applications 9 (11) (2015) 127–136. [doi:10.14257/ijseia.2015.9.11.12](https://doi.org/10.14257/ijseia.2015.9.11.12).
- [21] S. Haiduc, V. Arnaoudova, A. Marcus, G. Antoniol, The use of text retrieval and natural language processing in software engineering, in: Proceedings of the 38th International Conference on Software Engineering Companion, 2016, pp. 898–899. [doi:10.1145/2889160.2891053](https://doi.org/10.1145/2889160.2891053).
- [22] Y. Li, M. Soliman, P. Avgeriou, Automatic Identification of Self-Admitted Technical Debt from Different Sources, arXiv preprint arXiv:2202.02387 (2022). [doi:10.48550/arXiv.2202.02387](https://doi.org/10.48550/arXiv.2202.02387).
- [23] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, J. Singer, Todo or to bug: exploring how task annotations play a role in the work practices of software developers, in: Proceedings of the 30th international Conference on Software Engineering, 2008, pp. 251–260. [doi:10.1145/1368088.1368123](https://doi.org/10.1145/1368088.1368123).
- [24] S. C. B. de Souza, N. Anquetil, K. M. de Oliveira, Which documentation for software maintenance?, Journal of the Brazilian Computer Society 12 (2006) 31–44. [doi:10.1007/BF03194494](https://doi.org/10.1007/BF03194494).

- [25] W. Maalej, H.-J. Happel, Can development work describe itself?, in: 2010 7th IEEE working conference on mining software repositories (MSR 2010), IEEE, 2010, pp. 191–200. [doi:10.1109/MSR.2010.5463344](https://doi.org/10.1109/MSR.2010.5463344).
- [26] R. Shokripour, J. Anvik, Z. M. Kasirun, S. Zamani, Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation, in: 2013 10th Working Conference on Mining Software Repositories (MSR), IEEE, 2013, pp. 2–11. [doi:10.1109/MSR.2013.6623997](https://doi.org/10.1109/MSR.2013.6623997).
- [27] B. Kitchenham, S. M. Charters, Guidelines for performing Systematic Literature Reviews in Software Engineering, Tech. rep. (2007).
- [28] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic Mapping Studies in Software, in: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12, Vol. 17, 2008, pp. 1–10. [doi:10.14236/ewic/EASE2008.8](https://doi.org/10.14236/ewic/EASE2008.8).
- [29] Barbara Kitchenham, Procedures for Performing Systematic Reviews, Keele University Technical Report 33 (2004) (2004) 1–26.
- [30] B. Kitchenham, Evidence-based software engineering and systematic literature reviews, in: Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 4034 LNCS, Springer Verlag, 2006, p. 3. [doi:10.1007/11767718_3](https://doi.org/10.1007/11767718_3).
- [31] M. Petticrew, H. Roberts, Systematic reviews in the social sciences: A practical guide, John Wiley & Sons, 2008.
- [32] J. Popay, H. Roberts, A. Sowden, M. Petticrew, N. Britten, L. Arai, K. Roen, M. Rodgers, Developing guidance on the conduct of narrative synthesis in systematic reviews, J Epidemiol Community Health 59 (Suppl 1) (2005) A7.
- [33] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of The 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10. [doi:10.1145/2601248.2601268](https://doi.org/10.1145/2601248.2601268).

- [34] N. Ramasubbu, C. F. Kemerer, Managing technical debt in enterprise software packages, *IEEE Transactions on Software Engineering* 40 (8) (2014) 758–772. [doi:10.1109/TSE.2014.2327027](https://doi.org/10.1109/TSE.2014.2327027).
- [35] L. Rantala, Towards better technical debt detection with nlp and machine learning methods, in: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, 2020, pp. 242–245.
- [36] D. Albuquerque, E. Guimaraes, G. Tonin, M. Perkusich, H. Almeida, A. Perkusich, Comprehending the use of intelligent techniques to support technical debt management, in: *Proceedings of the International Conference on Technical Debt*, 2022, pp. 21–30. [doi:10.1145/3524843.3528097](https://doi.org/10.1145/3524843.3528097).
- [37] D. Pina, A. Goldman, C. Seaman, Sonarlizer Xplorer: a tool to mine Github projects and identify technical debt items using SonarQube, in: *Proceedings of the International Conference on Technical Debt*, 2022, pp. 71–75. [doi:10.1145/3524843.3528098](https://doi.org/10.1145/3524843.3528098).
- [38] M. Albarak, R. Bahsoon, I. Ozkaya, R. Nord, Managing technical debt in database normalization, *IEEE Transactions on Software Engineering* 48 (3) (2020) 755–772. [doi:10.1109/TSE.2020.3001339](https://doi.org/10.1109/TSE.2020.3001339).
- [39] T. Sharma, V. Efstathiou, P. Louridas, D. Spinellis, Code smell detection by deep direct-learning and transfer-learning, *Journal of Systems and Software* 176 (2021) 110936. [doi:10.1016/j.jss.2021.110936](https://doi.org/10.1016/j.jss.2021.110936).
- [40] E. Aromataris, D. Riitano, Constructing a search strategy and searching for evidence, *Am J Nurs* 114 (5) (2014) 49–56. [doi:10.1097/01.NAJ.0000446779.99522.f6](https://doi.org/10.1097/01.NAJ.0000446779.99522.f6).
- [41] I. El Naqa, M. J. Murphy, *What is machine learning?*, Springer, 2015. [doi:10.1007/978-3-319-18305-3_{_}1](https://doi.org/10.1007/978-3-319-18305-3_{_}1).
- [42] P. Langley, Human and machine learning, *Machine Learning* 1 (1986) 243–248. [doi:10.1007/BF00116892](https://doi.org/10.1007/BF00116892).
- [43] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *nature* 521 (7553) (2015) 436–444. [doi:10.1038/nature14539](https://doi.org/10.1038/nature14539).

- [44] J. Y. Khan, G. Uddin, Automatic Detection and Analysis of Technical Debts in Peer-Review Documentation of R Packages, in: Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, 2022, pp. 765–776. [doi:10.1109/SANER53432.2022.00094](https://doi.org/10.1109/SANER53432.2022.00094).
- [45] R. Sharma, R. Shahbazi, F. H. Fard, Z. Codabux, M. Vidoni, Self-admitted technical debt in R: detection and causes, Automated Software Engineering 29 (2) (2022). [doi:10.1007/s10515-022-00358-6](https://doi.org/10.1007/s10515-022-00358-6).
- [46] Z. Yu, F. M. Fahid, H. Tu, T. Menzies, Identifying Self-Admitted Technical Debts With Jitterbug: A Two-Step Approach, IEEE Transactions on Software Engineering 48 (5) (2020) 1676–1691. [doi:10.1109/TSE.2020.3031401](https://doi.org/10.1109/TSE.2020.3031401).
- [47] H. Tu, T. Menzies, DebtFree: minimizing labeling cost in self-admitted technical debt identification using semi-supervised learning, Empirical Software Engineering 27 (4) (2022). [doi:10.1007/s10664-022-10121-w](https://doi.org/10.1007/s10664-022-10121-w).
- [48] A. Potdar, E. Shihab, An exploratory study on self-admitted technical debt, in: Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014, 2014, pp. 91–100. [doi:10.1109/ICSME.2014.31](https://doi.org/10.1109/ICSME.2014.31).
- [49] G. Bavota, B. Russo, A large-scale empirical study on self-admitted technical debt, Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016 (2016) 315–326. [doi:10.1145/2901739.2901742](https://doi.org/10.1145/2901739.2901742).
- [50] S. Wehaibi, E. Shihab, L. Guerrouj, Examining the impact of self-admitted technical debt on software quality, 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016 1 (2016) 179–188. [doi:10.1109/SANER.2016.72](https://doi.org/10.1109/SANER.2016.72).
- [51] E. D. S. Maldonado, E. Shihab, Detecting and quantifying different types of self-admitted technical Debt, 2015 IEEE 7th International Workshop on Managing Technical Debt, MTD 2015 - Proceedings (2015) 9–15. [doi:10.1109/MTD.2015.7332619](https://doi.org/10.1109/MTD.2015.7332619).

- [52] M. A. d. F. Farias, M. G. d. M. Neto, M. Kalinowski, R. O. Spínola, Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary, *Information and Software Technology* 121 (January 2019) (2020). [doi:10.1016/j.infsof.2020.106270](https://doi.org/10.1016/j.infsof.2020.106270).
- [53] Z. Guo, S. Liu, J. Liu, Y. Li, L. Chen, H. Lu, Y. Zhou, How Far Have We Progressed in Identifying Self-admitted Technical Debts? A Comprehensive Empirical Study, *ACM Transactions on Software Engineering and Methodology* 30 (4) (2021). [doi:10.1145/3447247](https://doi.org/10.1145/3447247).
- [54] H. Azuma, S. Matsumoto, Y. Kamei, S. Kusumoto, An empirical study on self-admitted technical debt in Dockerfile, *Empirical Software Engineering* 27 (2) (2022) 49. [doi:10.1016/j.infsof.2022.106855](https://doi.org/10.1016/j.infsof.2022.106855).
- [55] B. Russo, M. Camilli, M. Mock, WeakSATD: Detecting Weak Self-admitted Technical Debt, *Proceedings - 2022 Mining Software Repositories Conference, MSR 2022* (2022) 448–453 [doi:10.1145/3524842.3528469](https://doi.org/10.1145/3524842.3528469).
- [56] S. Bellomo, R. L. Nord, I. Ozkaya, M. Popeck, Got technical debt? Surfacing elusive technical debt in issue trackers, *Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016* (2016) 327–338 [doi:10.1145/2901739.2901754](https://doi.org/10.1145/2901739.2901754).
- [57] Z. Codabux, M. Vidoni, F. H. Fard, Technical debt in the peer-review documentation of r packages: A ropensci case study, *Proceedings - 2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021* (2021) 195–206 [doi:10.1109/MSR52588.2021.00032](https://doi.org/10.1109/MSR52588.2021.00032).
- [58] Y. Li, M. Soliman, P. Avgeriou, Identification and Remediation of Self-Admitted Technical Debt in Issue Trackers, in: *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, 2020, pp. 495–503. [doi:10.1109/SEAA51224.2020.00083](https://doi.org/10.1109/SEAA51224.2020.00083).
- [59] M. A. D. F. Farias, M. G. D. M. Neto, A. B. D. Silva, R. O. Spinola, A Contextualized Vocabulary Model for identifying technical debt on code comments, in: *2015 IEEE 7th International Workshop on Managing Technical Debt, MTD 2015 - Proceedings, IEEE, 2015*, pp. 25–32. [doi:10.1109/MTD.2015.7332621](https://doi.org/10.1109/MTD.2015.7332621).

- [60] Y. S. Al-Slais, Towards a comprehensive self-admitted technical debt extraction technique from source code comments, in: 3rd Smart Cities Symposium (SCS 2020), 2021, pp. 109–114. [doi:10.1049/icp.2021.0881](https://doi.org/10.1049/icp.2021.0881).
- [61] E. S. Da Maldonado, R. Abdalkareem, E. Shihab, A. Serebrenik, An empirical study on the removal of Self-Admitted Technical Debt, in: Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, 2017, pp. 238–248. [doi:10.1109/ICSME.2017.8](https://doi.org/10.1109/ICSME.2017.8).
- [62] K. Dai, P. Kruchten, Detecting technical debt through issue trackers, CEUR Workshop Proceedings 2017 (QuASoQ) (2017) 59–65.
- [63] E. D. S. Maldonado, E. Shihab, N. Tsantalis, Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt, IEEE Transactions on Software Engineering 43 (11) (2017) 1044–1062. [doi:10.1109/TSE.2017.2654244](https://doi.org/10.1109/TSE.2017.2654244).
- [64] J. Flisar, V. Podgorelec, Enhanced feature selection using word embeddings for self-admitted technical debt identification, Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018 (2018) 230–233 [doi:10.1109/SEAA.2018.00045](https://doi.org/10.1109/SEAA.2018.00045).
- [65] Z. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, S. Li, SATD detector: A text-mining-based self-Admitted technical debt detection tool, in: Proceedings - International Conference on Software Engineering, 2018, pp. 9–12. [doi:10.1145/3183440.3183478](https://doi.org/10.1145/3183440.3183478).
- [66] J. Flisar, V. Podgorelec, Identification of Self-Admitted Technical Debt Using Enhanced Feature Selection Based on Word Embedding, IEEE Access 7 (2019) 106475–106494. [doi:10.1109/ACCESS.2019.2933318](https://doi.org/10.1109/ACCESS.2019.2933318).
- [67] S. Wattanakriengkrai, R. Maipradit, H. Hata, M. Choetkiertikul, T. Sunetnanta, K. Matsumoto, Identifying design and requirement self-admitted technical debt using N-gram IDF, in: Proceedings - 2018 9th International Workshop on Empirical Software Engineering in Practice, IWESEP 2018, IEEE, 2019, pp. 7–12. [doi:10.1109/IWESEP.2018.00010](https://doi.org/10.1109/IWESEP.2018.00010).
- [68] S. Wattanakriengkrai, N. Srisermphoak, S. Sintoplertchaikul, M. Choetkiertikul, C. Ragkhitwetsagul, T. Sunetnanta, H. Hata,

- K. Matsumoto, Automatic Classifying Self-Admitted Technical Debt Using N-Gram IDF, in: Proceedings - Asia-Pacific Software Engineering Conference, APSEC, Vol. 2019-Decem, IEEE, 2019, pp. 316–322. [doi:10.1109/APSEC48747.2019.00050](https://doi.org/10.1109/APSEC48747.2019.00050).
- [69] L. Xavier, F. Ferreira, R. Brito, M. T. Valente, Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems, in: Proceedings - 2020 IEEE/ACM 17th International Conference on Mining Software Repositories, MSR 2020, 2020, pp. 137–146. [doi:10.1145/3379597.3387459](https://doi.org/10.1145/3379597.3387459).
- [70] L. Rantala, M. Mantyla, D. Lo, Prevalence, Contents and Automatic Detection of KL-SATD, in: Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, 2020, pp. 385–388. [doi:10.1109/SEAA51224.2020.00069](https://doi.org/10.1109/SEAA51224.2020.00069).
- [71] R. Maipradit, C. Treude, H. Hata, K. Matsumoto, Wait for it: identifying "On-Hold" self-admitted technical debt, Empirical Software Engineering 25 (5) (2020) 3770–3798. [doi:10.1007/s10664-020-09854-3](https://doi.org/10.1007/s10664-020-09854-3).
- [72] L. Rantala, M. Mäntylä, Predicting technical debt from commit contents: reproduction and extension with automated feature selection, Software Quality Journal 28 (4) (2020) 1551–1579. [doi:10.1007/s11219-020-09520-3](https://doi.org/10.1007/s11219-020-09520-3).
- [73] R. Maipradit, B. Lin, C. Nagy, G. Bavota, M. Lanza, H. Hata, K. Matsumoto, Automated Identification of On-hold Self-Admitted Technical Debt, Proceedings - 20th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2020 29 (6223) (2020) 54–64. [doi:10.1109/SCAM51674.2020.00011](https://doi.org/10.1109/SCAM51674.2020.00011).
- [74] V. Rajalakshmi, S. Sendhilkumar, G. S. Mahalakshmi, Classification of Technical Debts in Software Development Using Text Analytics, EAI/Springer Innovations in Communication and Computing (2021) 313–323 [doi:10.1007/978-3-030-49795-8_31](https://doi.org/10.1007/978-3-030-49795-8_31).
- [75] M. Sridharan, M. Mantyla, L. Rantala, M. Claes, Data balancing improves self-admitted technical debt detection, in: Proceedings - 2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021, IEEE, 2021, pp. 358–368. [doi:10.1109/MSR52588.2021.00048](https://doi.org/10.1109/MSR52588.2021.00048).

- [76] I. Sala, A. Tommasel, F. Arcelli Fontana, DebtHunter: A machine learning-based approach for detecting self-admitted technical debt, ACM International Conference Proceeding Series (2021) 278–283 [doi:10.1145/3463274.3464455](https://doi.org/10.1145/3463274.3464455).
- [77] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, S. Li, An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks, Empirical Software Engineering 26 (2) (2021). [doi:10.1007/s10664-020-09917-5](https://doi.org/10.1007/s10664-020-09917-5).
- [78] X. Chen, D. Yu, X. Fan, L. Wang, J. Chen, Multiclass Classification for Self-Admitted Technical Debt Based on XGBoost, IEEE Transactions on Reliability 71 (3) (2022) 1309–1324. [doi:10.1109/TR.2021.3087864](https://doi.org/10.1109/TR.2021.3087864).
- [79] Y. Li, M. Soliman, P. Avgeriou, Identifying self-admitted technical debt in issue tracking systems using machine learning, Empirical Software Engineering 27 (6) (2022). [doi:10.1007/s10664-022-10128-3](https://doi.org/10.1007/s10664-022-10128-3).
- [80] T. Xiao, D. Wang, S. McIntosh, H. Hata, R. G. Kula, T. Ishio, K. Matsumoto, Characterizing and Mitigating Self-Admitted Technical Debt in Build Systems, IEEE Transactions on Software Engineering 48 (10) (2022) 4214–4228. [doi:10.1109/TSE.2021.3115772](https://doi.org/10.1109/TSE.2021.3115772).
- [81] E. A. AlOmar, B. Christians, M. Busho, A. H. AlKhalid, A. Ouni, C. Newman, M. W. Mkaouer, SATDBailiff-mining and tracking self-admitted technical debt, Science of Computer Programming 213 (2022) 102693. [doi:10.1016/j.scico.2021.102693](https://doi.org/10.1016/j.scico.2021.102693).
- [82] A. F. Sabbah, A. A. Hanani, Self-admitted technical debt classification using natural language processing word embeddings, International Journal of Electrical and Computer Engineering 13 (2) (2023) 2142–2155. [doi:10.11591/ijece.v13i2.pp2142-2155](https://doi.org/10.11591/ijece.v13i2.pp2142-2155).
- [83] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, J. Grundy, Neural network-based detection of self-Admitted technical debt: From performance to explainability, ACM Transactions on Software Engineering and Methodology 28 (3) (2019). [doi:10.1145/3324916](https://doi.org/10.1145/3324916).
- [84] R. M. Santos, M. C. R. Junior, M. G. de Mendonça Neto, Self-Admitted Technical Debt classification using LSTM neural network, Advances in

- Intelligent Systems and Computing 1134 (Itng) (2020) 679–685. doi:10.1007/978-3-030-43020-7{_}93.
- [85] F. Zampetti, A. Serebrenik, M. DI Penta, Automatically Learning Patterns for Self-Admitted Technical Debt Removal, in: SANER 2020 - Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering, IEEE, 2020, pp. 355–366. doi:10.1109/SANER48275.2020.9054868.
 - [86] X. Wang, J. Liu, L. Li, X. Chen, X. Liu, H. Wu, Detecting and Explaining Self-Admitted Technical Debts with Attention-based Neural Networks, in: Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, 2020, pp. 871–882. doi:10.1145/3324884.3416583.
 - [87] R. M. Santos, I. M. Santos, M. C. Rodrigues, M. G. de Mendonça Neto, Long term-short memory neural networks and word2vec for self-admitted technical debt detection, ICEIS 2020 - Proceedings of the 22nd International Conference on Enterprise Information Systems 2 (May) (2020) 157–165. doi:10.5220/0009796001570165.
 - [88] D. Yu, L. Wang, X. Chen, J. Chen, Using BiLSTM with attention mechanism to automatically detect self-admitted technical debt, Frontiers of Computer Science 15 (4) (2021). doi:10.1007/s11704-020-9281-z.
 - [89] R. M. Santos, I. M. Santos, M. C. Júnior, M. Mendonça, Evaluating a LSTM Neural Network and a Word2vec Model in the Classification of Self-admitted Technical Debts and Their Types in Code Comments, Lecture Notes in Business Information Processing 417 (2021) 542–559. doi:10.1007/978-3-030-75418-1{_}25.
 - [90] K. Zhu, M. Yin, Y. Li, Detecting and Classifying Self-Admitted of Technical Debt with CNN-BiLSTM, Journal of Physics: Conference Series 1955 (1) (2021). doi:10.1088/1742-6596/1955/1/012102.
 - [91] J. Yu, K. Zhao, J. Liu, X. Liu, Z. Xu, X. Wang, Exploiting gated graph neural network for detecting and explaining self-admitted technical debts, Journal of Systems and Software 187 (2022) 111219. doi:10.1016/j.jss.2022.111219.

- [92] A. Di Salle, A. Rota, P. T. Nguyen, D. Di Ruscio, F. A. Fontana, I. Sala, PILOT: Synergy between Text Processing and Neural Networks to Detect Self-Admitted Technical Debt, in: Proceedings - International Conference on Technical Debt 2022, TechDebt 2022, 2022, pp. 41–45. doi: [10.1145/3524843.3528093](https://doi.org/10.1145/3524843.3528093).
- [93] A. Alhefdhi, H. K. Dam, Y. S. Nugroho, H. Hata, T. Ishio, A. Ghose, A Framework for Conditional Statement Technical Debt Identification and Description, *Automated Software Engineering* 29 (2) (2022) 1–36. doi: [10.1007/s10515-022-00364-8](https://doi.org/10.1007/s10515-022-00364-8).
- [94] H. Li, Y. Qu, Y. Liu, R. Chen, J. Ai, S. Guo, Self-admitted technical debt detection by learning its comprehensive semantics via graph neural networks, *Software - Practice and Experience* 52 (10) (2022) 2152–2176. doi: [10.1002/spe.3117](https://doi.org/10.1002/spe.3117).
- [95] G. Zhuang, Y. Qu, L. Li, X. Dou, M. Li, An Empirical Study of Gradient-based Explainability Techniques for Self-admitted Technical Debt Detection, *Journal of Internet Technology* 23 (3) (2022) 631–641. doi: [10.53106/160792642022052303021](https://doi.org/10.53106/160792642022052303021).
- [96] M. Yin, K. Zhu, H. Xiao, D. Zhu, J. Jiang, Deep neural network ensembles for detecting self-admitted technical debt, *Journal of Intelligent and Fuzzy Systems* 43 (1) (2022) 93–105. doi: [10.3233/JIFS-211273](https://doi.org/10.3233/JIFS-211273).
- [97] K. Zhu, M. Yin, D. Zhu, X. Zhang, C. Gao, J. Jiang, SCGRU: A general approach for identifying multiple classes of self-admitted technical debt with text generation oversampling, *Journal of Systems and Software* 195 (2023) 111514. doi: [10.1016/j.jss.2022.111514](https://doi.org/10.1016/j.jss.2022.111514).
- [98] Q. Huang, E. Shihab, X. Xia, D. Lo, S. Li, Identifying self-admitted technical debt in open source projects using text mining, *Empirical Software Engineering* 23 (1) (2018) 418–451. doi: [10.1007/s10664-017-9522-4](https://doi.org/10.1007/s10664-017-9522-4).
- [99] I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT press, 2016.
- [100] H. Wang, J. Li, H. Wu, E. Hovy, Y. Sun, Pre-Trained Language Models and Their Applications, *Engineering* (2022). doi: [10.1016/j.eng.2022.04.024](https://doi.org/10.1016/j.eng.2022.04.024).

- [101] N. B. Ruparelia, Software development lifecycle models, ACM SIGSOFT Software Engineering Notes 35 (3) (2010) 8–13. [doi:10.1145/1764810.1764814](https://doi.org/10.1145/1764810.1764814).
- [102] A. Alshamrani, A. Bahattab, A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model, International Journal of Computer Science Issues (IJCSI) 12 (1) (2015) 106.