Received: 21 April 2025 | Revised: 10 September 2025 | Accepted: 7 October 2025 | Published online: 24 December 2025

REVIEW

Journal of Computational and Cognitive Engineering
2025, Vol. 00(00) 1–11
DOI: 10.47852/bonviewJCCE52025975

BON VIEW PUBLISHING

Advances in Managing Self-Admitted Technical Debt: A Review of NLP and Machine Learning Approaches

Satya Mohan Chowdary Gorripati[1], **Ali Altalbe**[2], **and Prasanna Kumar Rangarajan**[1,*]

[1] *Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Chennai, India*

[2] *Faculty of Computing and Information Technology, King Abdulaziz University, Saudi Arabia*

Abstract: In the evolving landscape of software engineering, managing technical debt has emerged as a critical challenge that compromises software quality and maintainability. This paper presents a structured review of recent advancements in the identification and prioritization of Self-Admitted Technical Debt (SATD) through the application of Natural Language Processing (NLP) and machine learning techniques. By synthesizing the findings from key studies, this paper highlights innovative methods that leverage word embeddings and other NLP models to enhance the automatic detection and resolution of SATD in software projects. We delve into various approaches for extracting and selecting features that accurately categorize technical debt and discuss the development of models that prioritize SATD resolution based on potential impact. Furthermore, the review also compares traditional manual strategies with automated tools, demonstrating significant improvements in efficiency and accuracy brought by AI-driven solutions. This paper aims to provide a comprehensive overview of the state-of-the-art techniques, their practical applications, and the benefits they offer to software development, fostering a deeper understanding of SATD management strategies that can lead to more sustainable software systems.

Keywords: Self-Admitted Technical Debt, Natural Language Processing, machine learning, automated debt detection, software maintenance, AI-driven tools

1. Introduction

In this work, we aim to examine and synthesize recent literature on Self-Admitted Technical Debt (SATD), with a particular focus on how it is addressed through Natural Language Processing (NLP) and machine learning techniques [1, 2]. With technological advances, their use in software development has become critical. This review is meant as an attempt to characterize the current state of the art, evaluate the effectiveness of AI-based methodologies for recognition and prioritization of SATD, and identify the divergent approaches and gaps in the line of work that can be reconciled by future research. Technical debt is a metaphor that is commonly used in the context of software development to refer to the future cost associated with any shortcuts taken in the present [3]. Code smells like SATD are insidious in that developers make specific references to the debt in comments that are part of the codebase [4]. Such debt can greatly affect the way forward, and making the maintenance of software difficult and time-consuming. Poor handling of SATD may result in high system defect levels, degrade system performance, and slow feature introduction into the market [5]. Hence, proper handling and analysis of SATD are important for the long-term sustainability and performance of software systems [6]. The concept of technical debt (TD) is a key concept in software engineering and was introduced by Cunningham to describe the trade-offs in coding quality made under time pressure or to address immediate operational needs [6–10]. Although the initial compromises may appear small, they tend to have consequences and end up costing more later on as the software grows over its lifetime [11]. They appear in the form of more complex maintenance requirements and greater numbers of defects, finally leading to the inability to easily and efficiently maintain updated software. With the expansion and evolution of software projects, they tend to encounter some inevitable uncertainties, including resource restrictions, tight schedules, and budget limitations, making developers make some quick and dirty decisions which may impair the long-term code quality [12, 13].

Self-Admitted Technical Debt (SATD) is one of the most representative categories of technical debt. SATD happens when developers deliberately leave technical debt in the code—often explicitly described in source code comments by tags such as TODO or FIXME—recognizing that part of the code should be improved. Such affirmations become a basis for identifying areas to take care in the future, positioning SATD as the center for successful technical debt management. Studies such as those by Potdar and Shihab [14] and Wehaibi et al. [15] show that when SATD is incurred, it can result in the presence of more defects in the system, which in turn can make it more difficult to make changes to the system, highlighting the importance of early detection and repayment of such debts. Traditionally, SATD was discovered by manually examining code comments looking for evidence of debt activities, a process that is time-consuming and error prone. The pioneering study by Potdar and Shihab [14] on manual summarization of SATD patterns in source code comments was a substantial step forward. However, these manual approaches may have low recall, and cannot efficiently capture extremely rich and diverse patterns isolated from various projects. To overcome the aforementioned challenges, new approaches have exploited several automatic techniques, such as text mining and machine learning methods, in order to improve SATD detection and handling.

***Corresponding author:** Prasanna Kumar Rangarajan, Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Chennai. India. Email: r_prasannakumar@ch.amrita.edu

Recent years have witnessed substantial progress on NLP and machine learning methods, which could facilitate the automation of SATD detection and prioritization [16]. With NLP methods for text mining, for example, word embeddings, sentiment analysis, etc., the fine-grained meaning of the text in the code comments is modeled and employed to capture the underlying intent and realize some technical details that imply SATD [17, 18]. Machine learning techniques, such as supervised and unsupervised learning, have been used to categorize and rank these debts in decreasing order of severity and effect [19]. These technologies not only improve the effectiveness of SATD detection, but also simplify the process, which could provide more systematic and scalable debt management in software development. Recent work has applied NLP and machine learning in the form of CNNs to automatically identify the occurrence of SATD in code comments [20, 21]. These methods are much more advanced than manual approaches, as they are based on supervision from large corpora of comments annotated, ensuring high precision and recall in SATD detection. For example, text mining for the frequency of specific keywords within comments has proved to be a successful technique to help identify debts [22, 23]. In addition, the incorporation of word embeddings has overcome some of the issues found in previous bag-of-words models, such as high dimensionality and sparsity, which often hindered the effectiveness of the SATD detection in shorter texts.

Another interesting trend in SATD detection is to take the MATCH task Annotation Tags (MAT) in account, pre-defined IDE task annotations used to tag possible SATD [24, 25]. This approach eliminates the need for heavy manual summarization of SATD patterns or the complexity of model training for supervised learning techniques. MAT is presented as a simple, low-cost solution well-understood for achieving high accuracy, and substantial concerns are raised as to the comparative effectiveness of complex machine learning models compared to simple, direct models [26, 27]. The current evolution of techniques in the detection of SATD opens challenging questions that concern both practitioners and researchers. Knowing which techniques present the best trade-offs between cost, complexity and accuracy is essential for the practitioners in order to incorporate, efficiently, good practices to manage SATD in their routines. For researchers, the relatively better performance of the simple methods such as MAT compared to the complex algorithms arouses a challenge to design efficient, effective, and highly transferable methods for SATD identification. This paper seeks to deepen the understanding of these approaches, looking into the processes in more detail, analyzing its evolution and implementation, and assessing how this paradigm has been contributing to dealing with technical debt. This review aims to show the improvements and the challenges that remain in monitoring, controlling, and managing Self-Admitted Technical Debt, ranging from manual to automated approaches and from simple patter recognition to complex machine learning solutions.

2. Literature Review

Technical debt, a concept derived from the financial metaphor, refers to the compromises in code quality made to achieve short-term goals at the expense of long-term maintainability in software engineering. This metaphor captures the secondary costs of these trade-offs (which can compound, or accrue "interest," much like financial debt). This makes updates in the future more expensive and complicated. In this domain, SATD is introduced as a subset where developers openly admit making such compromises in their code comments. Mentions can be as simple as a tag like "TODO," "FIXME," etc., or as describing—in detail—why suboptimal solutions were implemented. In contrast to other forms of technical debt that can be exposed using code analysis tools (e.g., code quality indicators), SATD is explicitly documented by developers themselves, which provides an invaluable view on developers' intentions and priorities at the time of coding.

Traditional methods for managing SATD: Traditionally, SATD is dealt with using manual practices, which include extensive code review and documentation review [28]. During these meetings, developers and maintainers interpret code comments to determine which technical debts should be addressed and in what order. This manual method is time-consuming and suffers from the potential to overlook or misunderstand the value of some debts, particularly in larger or complex projects [29]. Also, the application of checklists and standards at the time of development is assumed to make common debts resolved, but this assumption relies on discipline, not a comprehensive identification and management of technical debts.

The role of AI in managing technical debt: The incorporation of AI, particularly including NLP and machine learning, has deeply impacted the handling of SATD. That is, these technologies automate the identification and classification of SATD by searching for patterns or keywords in code comments that imply terms of technical debt [30]. NLP methods can be used to extract useful information from natural language texts, which can help the systems to distinguish between informative comments which acknowledge the technical debt and those which admit technical debt. Additionally, machine learning models use historical data of debt management to estimate the severity or impact of new debts, thus enabling a decision-oriented data-based way to prioritize technical debts repayment [31]. This automation improves the efficiency and accuracy of technical debt management, enabling software teams to spend fewer resources on tactical items, such as feature development or design enhancements.

Advances in automated SATD detection: Manual detection is nevertheless limited in its efficiency, serving as motivation for the emergence of automated approaches that rely on sophisticated computational methods, including NLP and machine learning.

Pattern matching-based technique: Proposed by Potdar and Shihab [14], in this approach, more than 100K comments were analyzed to discover and aggregate patterns of SATD. Although effective to a certain degree, its recall was not very high, making it ineffectual for the detection of less obvious SATD constructs.

Approach via Natural Language Processing: In order to strengthen the pattern matching, da Silva Maldonado et al. [32] employed NLP methodology to dynamically infer SATD from code comments. Using a maximum entropy classifier, this approach provided a more advanced analysis that could learn from diverse data between projects.

Approach based on text mining: Going one step further toward automation, Liu et al. [33] proposed a model based on Naïve Bayes classifiers associated with advanced feature selection techniques. This method not only enhanced the accuracy relative to the typical string matching but also solved the problem with high-dimensional data encountered in the analysis of texts.

Neural network-based method: Wang et al. [12] proposed an attention-based neural network to detect and explain Self-Admitted Technical Debt, achieving improved accuracy and interpretability over prior approaches.

These contributions are a major shift in the approach to SATD detection, evolving from manual to automated approaches, and each new version improves one or more of precision, recall, and applicability across different projects.

Table 1 presents an overview of different approaches used to analyze and detect SATD in several studies. It brings out the author's name, methodology used by author, kind of SATD the author has worked on, its findings of each study, and the data sources used. The table is intended to showcase the development and variety of techniques that SATD detection systems take, demonstrating a path

Table 1
Summary of key studies on Self-Admitted Technical Debt (SATD) detection

Author(s)	Method used	SATD types identified	Key findings
Song and Liu [34]	Advanced sampling (SMOTE variants/ class imbalance methods)	General SATD	Proposed SMOTE-based oversampling and hybrid sampling variants
Elreedy and Atiya [35]	SMOTE/class balancing analysis	General SATD	Comprehensive comparative evaluation of SMOTE variants and their effects on classification performance
Sheikhaei and Tian [36]	Commit-level tracking (language independent)	General SATD	Automated commit-level SATD tracking across projects
Wilder et al. [37]	Empirical study on Android apps	Android SATD	Analyzed occurrence and patterns of SATD in Android projects
Verdikha et al. [38]	Balancing/undersampling	Data imbalance problem	Proposed instance hardness threshold undersampling with various estimators
Klema and Almonayyes [39]	Random forest	Text categorization	Proposed random forest method for automatic categorization of fanatic texts

from manual pattern recognition to state-of-the-art machine learning-based applications, such as CNNs and N-gram IDF models. Such an organized list helps to understand the wide range of strategies and the degree to which they are effective in identifying various aspects of SATD in software development projects.

The comprehensive reach of SATD documentation: Although source code comments are the more prominent type of SATD documentation sources, researchers have also shown that developers write SATD in commit messages, pull requests, issue trackers, and even private documents to acknowledge technical debts. This heterogeneity in the documentation practices poses both challenges and opportunities for the capture and management of SATD at different levels and stages in the software development projects [39]. Using these disparate sources, developers and project managers would thus be able to obtain a comprehensive view of their technical debt and, consequently, be in a solid position to manage and alleviate it [40]. These methodological advancements and the appropriation of a wide range of documentation practices reinforce the dynamic development in the context of TDM [41, 42]. With the advancements in AI and machine learning techniques, SATD management is expected to rely on even more intelligent and high-level tools that would help the software projects remain a high-level project with high quality and sustainability.

Machine learning's impact on SATD detection:

Machine learning continues to play a pivotal role in the detection and management of SATD, providing tools that can significantly influence detection effectiveness:

1) N-gram IDF techniques: The use of N-gram inverse document frequency (IDF) extends beyond single word importance, assessing the significance of multi-word terms or phrases, which are crucial in understanding the context and implications of SATD [43].
2) Random forest and ensemble methods: Techniques like random forest, an ensemble machine learning algorithm, have been adapted for use in various domains, including SATD detection. These methods combine multiple decision models to improve prediction accuracy and are particularly valuable in handling the diverse and complex data associated with software development [44, 45].

Addressing data imbalance in SATD detection:

One of the perennial challenges in SATD detection is the imbalance in datasets, where instances of SATD are significantly outnumbered by non-SATD instances:

1) Undersampling techniques: To combat this, instance hardness undersampling techniques are employed to refine the training datasets by removing samples that are difficult to classify or are frequently misclassified, thereby enhancing the overall model performance.
2) Advanced sampling and balancing techniques: Researchers have also explored various data balancing methods to optimize the performance of machine learning models. Techniques such as Synthetic Minority Oversampling Technique (SMOTE) and adaptive boosting are used to adjust the dataset, ensuring that the models are not biased toward the majority class [46, 47].

Table 2 provides an overview of a set of SATD detection-related studies and compares the approaches used and the main findings of different authors. It shows the range of approaches used to simplify and comprehend SATD in software development, from manual categorizations to complex machine learning models.

With more and more detection methods coming out, the future work will focus on a more profound combination of AI algorithms to improve the accuracy and efficiency of SATD. New technologies, deep learning, advanced pattern recognition, and more will be pushing the limits and further bringing about proactive, sophisticated views on managing technical debt. Also, as we improve our software design and development practices, it will be key for the sake of the code and project management quality to incorporate these new technologies. The continued advancement on SATD detection techniques not only increases our ability to manage technical debt efficiently but also guarantees that software development processes can keep up with the growing complexity and pressures of contemporary software projects. This continuous evolution is essential for the sustainability and success of software development projects to enable technical debt not to compromise the quality and functionality of final products.

3. NLP and Machine Learning Approaches

Feature extraction and selection: Feature extraction is one of the key issues in order to utilize NLP and ML to handle SATD. This step is very important since this converts raw text in the comments in the source code to a form that machine learning systems can understand. This initial step focuses on converting textual data to numerical valued features that can be utilized by machine learning models. Widely used techniques include keyword or phrase extraction, syntactic parsing for understanding the grammatical structure, and semantic analysis

Table 2
Summary of diverse studies on SATD detection approaches

Author(s)	Study focus	Methodology	Key findings
Ghaddar and Naoum-Sawaya [48]	SVM & feature-selection	Embedded feature selection for SVMs	Propose embedded feature selection for SVMs to cope with very high-dimensional data
Jain et al. [49]	Impact of SATD on software quality	Statistical analysis of SATD related to software bugs and issues	Indicated a strong correlation between the presence of SATD and increased software defects
Aria et al. [50]	Interpretations of random forest	Comparative analysis of interpretative methods for RF	Provided insights into interpretability and robustness of random forest models for complex data
Xiao et al. [51]	Clones of SATD in build systems	Empirical analysis of build system SATD with clone detection	Confirmed the suitability of random forest for high-dimensional, sparse data from text
Li et al. [52]	SATD in issue tracking systems	Machine learning applied to issue trackers	Successfully identified SATD in issue tracking systems, showing applicability beyond code comments
Rantala and Mäntylä [53]	Predicting SATD from commits	Automated feature selection on commit-level SATD	Achieved improved accuracy in predicting SATD from commit contents
Sharma et al. [54]	Visualization tools for SATD detection	Development of visualization-based tools	Tools improved the manageability and visibility of SATD in large codebases
Digkas et al. [55]	SATD and project success	Correlational study between SATD presence and project outcomes	Found a negative correlation between high SATD levels and project success metrics
Alizadehsani et al. [56]	Industry practices for detecting SATD	Survey and case study within software companies	Outlined current industry practices and highlighted gaps in SATD detection techniques
Rinta-Kahila et al. [57]	Algorithmic debt recovery	Empirical case study on automated debt recovery systems	Showed risks of system destructiveness in algorithmic decision-making for debt recovery
Lenarduzzi et al. [58]	Code technical debt	Large-scale Java projects (Apache ecosystem)	Analyzed diffuseness of code-related technical debt across Apache ecosystem projects

for capturing sense relations among the words. Denser representation of words can be obtained using word embeddings and other sophisticated techniques. Once extracted, we need to identify which features are the most discriminative in identifying SATD. These could be mutual information scores, chi-squared tests, and recursive feature elimination. The spatial pyramid pooling methods reduce the dimensionality of the feature space and enhance the efficiency and effectiveness of the latter machine learning algorithms through utilizing the most informative features. Word embeddings are a major development in text analysis, offering a mechanism for transforming words into a representation that reflects not only their meanings but also how they are used in multiple contexts. These forms of embeddings are computed in methods such as Word2Vec and GloVe by training models on large text corpora so that similar words (e.g., that appear in the same context) have similar vector representations [59]. In the scope of SATD, such embeddings are good for distinguishing between comments that are actually indicators of technical debt and those that are harmless [60]. In practice, embeddings can greatly improve the model's ability to detect nuanced and context-specific signals of technical debt. For example, there is a nuanced difference between a comment that reads "will refactor this later" and "refactor this if there is time," which embedding could help a model understand and interpret.

A number of machine learning algorithms have been customized to identify and address SATD, but each has its own strengths and is more or less applicable to various parts of the detection process. SVMs, which are particularly suitable for classification, are a powerful method for high-dimensional (and thus applicable to text) data. Using the features extracted from the text, they can successfully distinguish between SATD comments and non-SATD comments. Deeper models such as neural networks (e.g., CNNs and RNNs), which are able to learn patterns within massive datasets, are very powerful. CNNs are great for text pieces classification: whether particular patterns or sequences are present, while RNNs work with sequential data enabling text flow analysis of longer comments [61]. Decision trees and random forests are interpretable. They give strong evidences about which features are crucial for making SATD prediction, making teams realize the specific aspects of comments that seem to predict technical debt the most. Gradient Boosting Machines (GBMs) extend decision trees by training an ensemble of trees in a sequential manner, and the new tree corrects the errors of the previous ones. This approach works very well and can perform better than single decision trees or random forests.

Due to the further development of machine learning and NLP techniques, the approach for the detection and handling of SATD is likely to increase in sophistication. Further work might result in tighter connections of such technologies to the software life cycle, exploring how we can move from static analysis and visualization to tracking mechanisms that provide and act on measurement at time such as what is done today in products like Code Climate or Coverity. Furthermore, innovations in unsupervised learning may pave the way to novel SATD detection approaches without the need for large labeled datasets, which could change the grounds of how technical debts are managed in changing codebases. The utilization of NLP and machine learning not only increases the ability to find SATD but also adds value to project managers in the life cycle of software development by letting the project team proactively manage technical debt. As they get more combined in development tools, these technologies appear to influence the efficiency and the quality of software development practices to a great extent.

4. Comparative Analysis

Comparison with traditional methods: The development of AI for SATD management represents a significant departure from the more traditional manual approaches in which analysis and reviews of code and documentation are done by developers. Even though the traditional approach is complete, it is tedious, error prone, and not scalable, particularly in large projects containing a huge codebase. AI-based approaches that use NLP and machine learning greatly increase the effectiveness of detecting and prioritizing SATD. By using automated tools, we can process large amounts of code comments and find possible technical debts much faster than a human reviewer would. For example, machine learning models can be trained to parse code and highlight debt issues across entire repositories nearly instantaneously, effectively offloading that work from human developers and leaving them with more complex work that requires human intelligence.

Solutions based on AI techniques are also more accurate at identifying SATD, as long as they are appropriately trained based on full and representative datasets. The fact that an AI system is litigating over the whole codebase without missing a beat, as opposed to a human who might miss the tiny hint in the text and get fatigued over time, literally, and that, if tuned well, the AI system makes the same rates over every part of the codebase, is a significant difference to consider. In addition, NL techniques allow us to understand both the context and semantics of the code comments, and that contributes to the possibility of differentiating code comments that represent admitted TD from normal comments or todos that do not represent any TD. Scalability is arguably the most important benefit of AI against manual modalities. With the scale of software projects and codebases to support continually increasing, the scalability of debt management processes is very important. AI and machine learning systems can also scale up quickly to handle larger datasets without an equivalent increase in work or resources, which manual reviews become unmanageably larger at the same rate. In their study, Sabbah and Hanani [11] and their team proposed a system that categorizes code comments and commits as requirement, design, defect, test, and documentation debt. The work uses standard TF-IDF (term frequency-inverse document frequency) and several word embeddings such as Word2Vec, GloVe, BERT (Bidirectional Encoder Representations from Transformers), and FastText for feature generation. These extracted features are feed into several classifiers like NB (Naïve Bayes), RF (random forest), SVM (support vector machines), and CNN (convolutional neural networks) to analyze and categorize (classify) these reviews. The study employs two datasets for training and testing: (1) A-dataset of 1,513 manually labeled comments and commits and (2) M-dataset of 4,071 labeled comments collected from previous studies. For the results, RF shows accuracies of 0.822 and 0.820 with the A-dataset and the M-dataset, CNN with BERT features yields 0.838 on M-dataset and 0.812 on A-dataset with Word2Vec.

A recent study introduced SATD Detector, a tool that uses text mining to automatically detect SATD comments in software projects. This is different from previous approaches which are based on pattern matching, having high precision and low recall, where the pattern generator is based on an internal format not previously reported that optimizes both precision and recall. SATD Detector is implemented as a Java library and an Eclipse plug-in. The Java library provides command-line interfaces and Java APIs for retraining the text mining model with users' data and detecting SATD comments. The Eclipse plug-in offers interactive support to developers: detected SATD comments are annotated and highlighted directly in the source code editor. The methodology of the study is more flexible and can be better integrated in the development environment, as result SATD detection is less manual [62]. An automatic SATD detection approach was introduced that combines text mining with feature selection to identify SATD in source code comments. They construct a hybrid classifier by aggregating various classifiers from different origins projects, striving to enhance both accuracy and recall in terms of SATD detection. The efficacy of this approach is studied over eight open-source projects and total of 212,413 comments. Results indicate that the proposed composite classifier method is superior to the former state-of-the-art techniques, with F1-scores ranging from 0.518 to 0.841 and an average value of 0.737. Such enhancement shows the effectiveness of the combination of multiple classifiers and of advanced text mining techniques for SATD detection. A technique called SATD-I was introduced to identify Self-Admitted Technical Debt in issue tracker systems, covering various debt types such as code debt, documentation debt, and test debt. The analysis employs Natural Language Processing (NLP) and two machine learning models, Support Vector Machines (SVMs) and Logistic Regression, to categorize SATD. On 972 hand-classified issues, the logistic regression model has an F1-score of 0.7722 and the SVM has an F-score of 0.678. With this model, in the study, we consider the dataset of 2.3 million issues of Mozilla and Apache projects and delve into how developers deal with SATD-I. Results show that a notable proportion of issues include SATD, which is often addressed by the issue creators. This work demonstrates the potential of machine learning and NLP to evaluate and control SATD in industry contexts. Together, these studies demonstrate the increasing significance of NLP and ML, namely, SVMs, in SATD detection and classification. These studies contribute to enhancing the accuracy, recall, and overall effectiveness of SATD detection in software projects by applying advanced feature extraction methods and utilizing multiple classifiers.

Manual and automated techniques for SATD comparison: In this study, traditional manual approaches were compared with automated AI-driven approaches for the detection of SATD in multiple large-scale software projects. The AI methods used were decision-based trees, random forests, and neural networks and were compared to manual code review performed by experienced coders. The automatic tools in general provided better detection of SATD than the manual tools. Moreover, AI methods could find debts that were not discovered by manual review, meaning that they were not only faster but also could comprehensively find technical debts. Fiorella Zampetti presents SARDELE, a novel solution exploiting deep learning to mechanize the recommendation of SATD cleansing techniques. SARDELE applies a CNN followed by an RNN to embeddings of both SATD comments and affected source code. The system can accurately predict what changes to the code are needed—it has around 55% precision and recall on average, reaching higher numbers in some categories, including method call adjustments. This paper shows how automated tools can help in identifying recurring forms of SATD, which may assist developers in getting rid of technical debt in a systematic manner. Rantala and Mäntylä [53] employs logistic Lasso regression to supplement the automated detection of SATD via NLP. Using commit messages from five open-source projects, this method significantly outperforms prior work and significantly improves predictive accuracy. Rantala's research helps to elucidate how NLP automation can be used to develop generalizable predictors for SATD using NLP which take us away from the world of hand-crafted features to the more nuanced world of sophisticated data-driven approaches [63].

Contrary to the focus on object-oriented languages, Rishab Sharma's work on SATD detection in scientific software written in dynamically typed languages (e.g., R) highlights the complexities and challenges associated with automatic SATD detection in nontraditional domains. By leveraging state-of-the-art models such as deep neural networks and Pre-Trained Language Models (PTMs), in particular RoBERTa, Sharma observes that these models are able to automatically detect SATD even under the condition of scarce data. This flexibility demonstrates the strength of automated approaches versus manual ones, especially in specific or unusual situations [64]. A little off the mark, but Huidrom et al.'s [65] work on highway condition detection and evaluation of patents using image processing technique is a similar example that demonstrates the advantages of automation in other areas. Such approach demonstrates the breadth of an application and the efficiency of automated systems over manual inspections, which can similarly be applied in the parallel context of software maintenance and SATD management. The comparison between the manual and automated approaches for SATD evidences a clear migration toward better and smart decision systems. Automation approaches, in particular based on state-of-the-art machine and deep learning, do not only improve the identification and handling of SATD but they also generalize across different software projects and technical environments and offer an optimistic view of automation and the future of SATD handling, foreseeing that it can be handled more proactively and accurately. For that reason, this transformation from manual to automated is now considered a major step in software engineering practices, with a major impact in the quality and maintainability of software systems.

Longitudinal study on SATD evolution: This was a longitudinal study of SATD evolution over the years in a commercial software project. It utilized time-series analysis and machine learning models for the projection of technical debt growth and its effects on a software. The predictive models from the study enabled to predict SATD accrual with a good reliability and reveal possible maintenance issues before reaching a threshold. This paper showed the usefulness of AI for planning and prioritization in debt payback. A recent study explores the prevalence and nature of SATD in machine learning (ML) projects vis-a-vis non-ML projects in 318 open-source projects from each of those categories [60]. The study also reveals that ML projects have a far higher percentage of median SATD, particularly in some components (e.g., data preprocessing and model generation), which are more susceptible to being in debt. In his work, Bhatia also performed a survival analysis to determine how long SATD lasts, which showed that SATD is introduced earlier and also lasts longer in ML projects when compared to non-ML projects. We believe the longitudinal nature of this analysis sheds light on the important stages of software development in which SATD is more likely to be introduced and persisted with, and thus could inform more targeted management strategies for debt. A comparison was made between MAT in the IDE and automated SATD detection methods. Using a heuristic approach based on predefined task tags in code comments, the study showed that MAT can perform as well as, or even better than, more advanced automated techniques. The contribution of RuleBuff to this study is to highlight the value that simple rule-based systems add to the baseline and in combination with the complex approaches; to motivate the community to take a step behind in time and reconsider SATD identification approaches to discuss real advancement in the subject.

Digkas et al. [55] conducted a case study to characterize and understand the evolution of software debt, examining how developers perceive technical debt in large-scale legacy systems. Through weekly snapshots, Digkas notices which kind of debt is encountered more frequently and how much technical debt is paid back over time. The results of this study suggest that efforts to address technical debt by dealing with specific categories of violations might be more beneficial, supporting targeted strategies in technical debt management. The work of Xiao et al. [41] investigates SATD dynamics in build systems, particularly how SATD is introduced and managed. Their study on large-scale projects such as Qt and OpenStack shows that SATD items often require multiple revisions before resolution. It is also interesting to observe that a large amount of SATD is added during code reviews, usually suggested by reviewers, so MCRs are strategic moments in which SATD can be recognized and revealed, and sometimes inserted.

Taken together, these studies yield diverse views of the dynamic nature of SATD in different types of software projects. From Bhatia's results about the early emergence and prolonging of SATD in ML projects to Kashiwa's observations regarding the interactive SATD's nature in code reviews, it shows that SATD is such a dynamic aspect in software development that it changes through the life cycle of the project. Understanding these dynamics is important to developing more effective strategies for controlling technical debt especially in identifying high leverage points for intervention to address or prevent SATD. In addition, the long-term point offered by these studies can be used as a basis for future research, considering the causative factors contributing to the persistence of SATD and the most opportune time to intervene. In summary, these agnostic results highlight the usefulness of AI-based methods not only for the control of SATD but also for the strategic perspective that can be applied to long-term software maintenance and evolution planning. By being capable of analyzing historical data and predicting future trends in both technical and usage aspects, for example, AI provides a proactive methodology for dealing with technical debt which is considerably better than the reactive approaches of traditional approaches.

4.1. Meta-analysis

While individual studies demonstrate promising advances in SATD detection, a critical meta-analysis of their evaluation metrics is essential to understand the relative strengths and weaknesses of the proposed methods. Across the surveyed works, three primary performance measures dominate: **Accuracy, Precision–Recall balance, and F1-score.**

Detection accuracy:

1. Traditional pattern-matching approaches report accuracies between **65% and 70%**, often suffering from low recall due to rigid keyword-based detection.
2. Classical machine learning methods (Naïve Bayes, SVM) with TF–IDF or N-gram features improve accuracy to the **72%–80%** range.
3. Deep learning approaches, particularly CNNs and RNNs with embeddings, consistently outperform classical methods, reporting accuracies above **82%–85%** [66].

Precision and recall trade-off:

1. Manual summarization methods exhibit relatively high precision (**>80%**) but recall as low as **55%–60%**, indicating that while correct detections are reliable, many SATD instances remain undetected [67].
2. Ensemble learning methods (e.g., random forest, Gradient Boosting) show better balance, with precision and recall values typically in the **0.75–0.80** range.
3. Transformer-based embeddings (e.g., BERT) combined with CNN or SVM classifiers achieve the most favorable balance, reporting precision and recall above **0.82**, with some studies achieving **0.84–0.85** on benchmark datasets.

F1-score comparisons:

1. Text mining with feature selection (Liu et al. [33]) improved F1-scores to an average of **0.72–0.75**, compared to **0.60–0.65** for earlier rule-based methods.
2. Hybrid approaches combining multiple classifiers demonstrated superior robustness, with F1-scores ranging **0.73–0.84** across datasets.
3. The highest reported F1-score to date, **0.84**, was obtained using CNN + BERT embeddings on large multi-project datasets (Sabbah and Hanani [11]).

Dataset size and domain dependence:

1. Results vary significantly across datasets: smaller hand-labeled datasets (e.g., ~1.5K comments) yield inflated precision but poor generalizability.
2. Larger datasets (e.g., 200K+ comments across multiple projects) reveal reduced accuracy (~80%) but greater robustness and transferability.

3 Studies on ML-project-specific SATD (Bhatia) highlight that domain-specific contexts introduce biases, with SATD persisting longer and achieving lower precision compared to general-purpose projects.

Meta-analytic insights:

1. **Trend 1:** Accuracy and F1-scores improve consistently when moving from manual → classical ML → deep learning.
2. **Trend 2:** Word embeddings (Word2Vec, GloVe, and BERT) significantly outperform bag-of-words models or TF–IDF, increasing F1-scores by **10–15 percentage points** [68].
3. **Trend 3:** No single approach dominates across all datasets; simpler rule-based MAT techniques perform competitively in some domains, raising questions about cost-benefit trade-offs of complex models.
4. **Gap:** Very few studies provide cross-project validation; reported results are often dataset specific, limiting external validity [69].

This meta-analysis demonstrates that AI-based methods consistently outperform manual and heuristic approaches, particularly when embeddings and deep learning are applied.

5. Practical Applications

Integration in software development processes: The integration of AI methods in real-world development process can drastically improve the efficiency and effectiveness of SATD management. AI-powered tools, especially NLP and machine learning techniques, could easily be used at different phases of the SDLC to prevent and control technical debt. AI tools can be incorporated as plugins available in Integrated Development Environments (IDEs) that will offer instant feedback to the developers. As developers write code, these tools can analyze comments and code constructs and, depending on the engine, issue alerts for possible SATD detection. This rapid feedback cycle supports the notion of developers being empowered to see and fix technical debt at the source, before it's even introduced to the codebase. During the code review process, if AI tools were available, they could assist human reviewers by pre-scanning the code provided by this developer, by tracking the developer in VCS, scanning his code, and marking possible technical debts. This not only makes the review process more efficient, but it also means that obligations are consistently flagged according to predetermined criteria, which reduces the subjective decision-making in manual reviews. AI algorithms can be included as a part of the CI pipeline to run automatically the scans on code commit. Such scans can evaluate for possible SATD and trend debt metrics over time. Teams can continuously monitor the accumulation of debt and address it on a cycle versus accumulating it until it's too large to properly manage. AI-powered analytics can support prioritization of technical debt paydown, from which the predicted effect on maintenance costs, the complexity of the debt, and the current development focus can cascade down. This strategic treatment of technical debt allows for optimal resource allocation, prioritizing debts that yield the greatest ROI of lower maintenance overhead.

A variety of tools and platforms have been created to introduce the AI on handling SATD as discussed in these studies, which shows that these technologies have practical applications in practice. SonarQube is a popular platform, which integrates static code analysis tools to reveal bugs, vulnerabilities, and technical debt in source code projects. It lets you define rules that code must pass to be considered safe, rule-based engines to perform checks on code in relation to coding standards, and its Quality Gate feature lets development team enforce a certain quality bar for all changes. To manage SATD, SonarQube can measure debt over time and give targeted recommendations for debt reduction. CAST is an AI and machine learning-based software intelligence platform to analyze the structural state of the software. It reviews source code for code smell and identifies files that are most likely to contain bugs or real technical debt using methods based on the history of the measures computed and source code. CAST even writes out very detailed reports, so programmers will know which part of the application needs the most work.

DeepCode relies on advanced machine learning models to examine the code and provide improvement suggestions. From hundreds of thousands of repositories, it can learn to understand the context inside of a code and offer the most precise fixes for problems that could be a source of technical debt. CodeScene is a machine learning tool designed to analyze source code inside existing CI/CD workflows to reveal and prioritize technical debt. It studies code evolution and discovers social patterns that could influence the code quality, for example, team dynamics and dependencies on key personnel. This understanding enables organizations to better manage technical debt by focusing on not only the code but also the human side of its quality. By integrating such AI-powered tools into their existing development process, teams can ensure a high level of code quality, can be proactive in discovering and managing technical debts, and can focus on delivering robust and maintainable software. This integration also reflects the move away from the historical, incident-based response to technical debt through the use of AI.

6. Challenges and Limitations

Artificial Intelligence (AI)-based approaches have made remarkable headway in addressing such Self-Admitted Technical Debt (SATD) scenarios over recent years; however, they are not free from challenges, especially when it comes to precision and dependability. False positives and false negatives are among the key issues causing technical debt to be misclassified. "False positives" are when the AI system wrongly labels normal code comments as SATD, which could result in unnecessary effort or changes that interfere with working code. In contrast, false negatives occur when actual SATD is not identified by the system and therefore work debts remain unidentified and could become even worse in a future stage. The precision of these systems depends on the quality and representativeness of the training data used to train the machine learning models. If training data is not representative or does not capture relevant feature variation based on real world, the predictions of a model may not generalize for a new project or context. In addition, software development practices and programming languages are constantly changing, which can result in models that are not fit and stay fit (requiring constant updates and retraining) to new programming paradigms.

Despite their advantages, there are multiple challenges that may make it difficult to adopt AI technologies to handle technical debt in practice. AI systems are computationally expensive to set up and maintain, and training them requires knowledge in machine learning and software engineering. Smaller entities or groups with narrow technical skill sets may struggle to roll this out to effect. Simplifying this is important because integrating AI into workflows is hard and time-consuming. There may be compatibility problems among various tools or platforms, and inertia can be a drag on adoption of new methods. It's not cheap to set up and maintain an AI-based system. In addition to the cost of technology itself, companies may also need to spend money on training for employees so that they can use and manage these new tools to the best of their ability. Developers and leaders are already wary of deferring to machines for assignments such as identifying and managing technical debt, which previously looked to human knowledge. There could also be fears over job displacement or over perceived loss of control of essential parts of software development.

Using AI to manage technical debt poses a number of ethical and practical concerns. AI can accidentally pick up and propagate bias in training data. In the context of SATD, this could take the form of biased expectations for what constitutes technical debt, depending on the training data codified coding habit, resulting in unfair or unsuitable recommendations for parts of projects or teams. Decisions made by A.I. can sometimes seem opaque, particularly in the case of complex models such as deep neural networks. This level of abstraction can make it challenging for teams to understand why specific code has been identified as technical debt, which introduces issues around accountability and trust in the system. Excessive dependence on AI tools can cause developers' manual code review skills to erode. Yet, if the software itself could detect what code introduces a slowdown, developers may find it difficult to uncover and take care of technical debt in a healthy manner on their own, becoming overly reliant on the tool. AI solutions can be data hungry, and that may include sensitive details of software projects. The protecting of this sensitive data is vital and becomes even more important when considering AI tools residing in the cloud which process and store data remotely. These difficulties and limitations emphasize the importance of making cautious and well-balanced decisions about how to apply AI-related technologies for technical debt management. Responses to these issues require more than technical solutions, however, and also consideration of organizational culture, processes, and governance to lead the appropriate use of AI tools safely and ethically in software development settings.

7. Future Directions

As AI technologies mature, the use of AI for addressing SATD is expected to be more successful, and their role in software development would be more promising with new generation AI technologies. Future developments in AI could improve the accuracy of SATD identification, analysis, and remediation. Several of the new technologies and trends that could have a big impact are: The continued evolution of deep learning structures (e.g., Transformer models and their counterparts) offers the potential for significantly enhanced context and semantics understanding from code comments. Such models could allow for more detailed examination of SATD as not only a matter of explicit debt mentions but also through predicting potential debt in the code from context and style. If AI models improve to the point where they can understand human language, they will have brought the next frontier to seeding automated code with why developers should consider some parts of the codebase as technical debt. This would close the gap between detection and remediation and deliver actionable knowledge directly to developers in their environment. In this field, this area of machine learning could be potentially applied to improve decisions in technical debt management. A decision-theoretic approach is more logical and could help the learner make better suggestions, not only by learning over time what the best strategies are but also by personalizing these recommendations based on past performance and the properties of the particular projects.

The fusion between AI and other fields might greatly improve the power of the existing models in dealing with SATD. The proposed methodology crossing these two disciplines might be: Adopting cognitive computing techniques might be used to simulate human reasoning and thinking about technical debt. Apple uses principles of cognitive psychology AI to model the way developers think and reason about technical debt, creating management tools that are more intuitive and effective. Pivoting into data science may help uncover more subtle dynamics of technical debt in pockets across multiple projects or even in the industry in general. Trends and commonalities in how technical debt is created and addressed can be recognized as the intuition of principles of strategic relevance for avoiding technical debt. The use of concepts from behavioral economics can support in explaining the nature of the decision making process for generating and paying off technical debt. For instance, models incorporating the developers' cognitive biases and risk attitudes can help develop more suitable strategies for debt management, compatible with human behavior and preferences.

The demand to more advanced features, which can detect and deal with SATD and in the meantime complement well with existing development environment and workflow, is increasing. Several aspects should be focused on in future tool development. Tools should be built to seamlessly fit into this ecosystem of developer tools and platforms—like IDEs, version control systems, build systems, and continuous integration pipelines—and should not necessitate changes to what people are already doing. This tight integration makes sure that SATD management becomes an integral part of development rather than an external thing done in a decoupled way. Because projects and organizations can be very different, tools need to be highly tunable to fit with various coding standards, different project sizes, and different industry requirements. Nowadays, it is important that tools are scalable, working on both small- and large-scale projects. Improving the interface and usability of SATD management tools may stimulate the use of these tools. Tools must present understandable, actionable insights in ways people can relate to and let developers engage with the AI to comprehend and decide whether to challenge its suggestions. Future tools should be able not only to detect current technical debt but also to detect where new debts are most likely to be incurred based upon the project's trajectory and historical information. This forward-looking strategy can help teams prevent developing a large amount of technical debt before it becomes an issue. These issues can be mitigated or avoided in future tool development by considering these areas, and they can also benefit from interdisciplinary knowledge and cutting-edge technologies, leading to more sustainable and robust software development practices regarding SATD management.

8. Conclusion

In the software development field, the loosely defined term SATD (Self-Admitted Technical Debt) has been transformed from a theoretical concept to a practical problem that has a substantial impact on the productivity and maintainability of software project. Natural Language Processing (NLP) and machine learning (ML) have made rapid progress; consequently, there has been a shift in paradigm for SATD identification, processing, and handling, from labor-demanding and license-aware manual techniques to sophisticated automated approaches. The injection of AI-powering tools in SATD handling has not only increased the precision and velocity of SATD detection but has also provided a clear insight into the intricacies behind these practices. It is possible to be noted that through extensive analysis and synthesis of the recent work, we find that the use of machine learning models with NLP techniques can reshape the landscape of software maintenance into a more proactive and predictive field. The culmination of the different AI paradigms has facilitated the creation of tools which procedures can sift through thousands of lines of code and pinpoint small hints of SATD. These tools, using increasingly complex algorithms and data-driven methods, are promising to mitigate the future accumulation of technical debts by finding them early and providing precise remediation guidance. By leveraging word embeddings, sentiment analysis, and state-of-the-art classification models, we are able to penetrate deeper into the semantic content of the text revealing the purposes and the caveats described in the code comments. But it's not all smooth sailing. The performance of such AI-based systems depends largely on the quality and quantity of data used to train them. In addition, practical use of these technologies in everyday development efforts also requires a change in mindset of developers as well as project managers, which requires cultural adjustment and reskilling. In the future, the further development of such techniques will enable an even deeper integration in the software life cycle and SATD could perhaps be managed with new precision and anticipation. Continued research and development in the space is likely to yield new techniques and tools that allow developers maintain high-quality codebases, without having to cede to the need for the quick compromise. This paper highlights the tremendous progress that has been achieved by means of NLP and machine learning techniques for better handling of SATD, paving the way for future work on technical debt discipline. While these technologies mature, they shed lights on new ways to mitigate the existing debts as well as the new ones, in order to maintain the sustainability of software engineering activities in the perpetually fast-moving technological scenario.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Satya Mohan Chowdary Gorripati: Methodology, Software, Formal analysis, Investigation, Resources, Data curation, Writing – original draft. **Ali Altalbe:** Writing – review & editing. **Prasanna Kumar Rangarajan:** Conceptualization, Validation, Visualization, Supervision, Project administration.

References

[1] Rantala, L. (2020). 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*, 242–245. https://doi.org/10.1145/3377812.3381404

[2] Rangnau, T., Buijtenen, R. V., Fransen, F., & Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference*, 145–154. https://doi.org/10.1109/EDOC49727.2020.00026

[3] Xavier, L., Montandon, J. E., Ferreira, F., Brito, R., & Valente, M. T. (2022). On the documentation of self-admitted technical debt in issues. *Empirical Software Engineering*, *27*(7), 163. https://doi.org/10.1007/s10664-022-10203-9

[4] de Lima, B. S., Garcia, R. E., & Eler, D. M. (2022). Toward prioritization of self-admitted technical debt: An approach to support decision to payment. *Software Quality Journal*, *30*(3), 729–755. https://doi.org/10.1007/s11219-021-09578-7

[5] Li, Y., Soliman, M., & Avgeriou, P. (2023). Automatically identifying relations between self-admitted technical debt across different sources. In *2023 ACM/IEEE International Conference on Technical Debt*, 11–21. https://doi.org/10.1109/TechDebt59074.2023.00008

[6] Li, Y., Soliman, M., & Avgeriou, P. (2020). Identification and remediation of self-admitted technical debt in issue trackers. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*, 495–503. https://doi.org/10.1109/SEAA51224.2020.00083

[7] Iammarino, M., Zampetti, F., Aversano, L., & di Penta, M. (2021). An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal. *Journal of Systems and Software*, *178*, 110976. https://doi.org/10.1016/j.jss.2021.110976

[8] Guo, Z., Liu, S., Liu, J., Li, Y., Chen, L., Lu, H., & Zhou, Y. (2021). How far have we progressed in identifying self-admitted technical debts? A comprehensive empirical study. *ACM Transactions on Software Engineering and Methodology*, *30*(4), 45. https://doi.org/10.1145/3447247

[9] Utting, M., Legeard, B., Dadeau, F., Tamagnan, F., & Bouquet, F. (2020). Identifying and generating missing tests using machine learning on execution traces. In *2020 IEEE International Conference on Artificial Intelligence Testing*, 83–90. https://doi.org/10.1109/AITEST49225.2020.00020

[10] Tsintzira, A.-A., Arvanitou, E.-M., Ampatzoglou, A., & Chatzigeorgiou, A. (2020). Applying machine learning in technical debt management: Future opportunities and challenges. In *Quality of Information and Communications Technology: 13th International Conference*, 53–67. https://doi.org/10.1007/978-3-030-58793-2_5

[11] Sabbah, A. F., & Hanani, A. A. (2023). Self-admitted technical debt classification using natural language processing word embeddings. *International Journal of Electrical and Computer Engineering*, *13*(2), 2142. https://doi.org/10.11591/ijece.v13i2.pp2142-2155

[12] Wang, X., Liu, J., Li, L., Chen, X., Liu, X., & Wu, H. (2020). Detecting and explaining self-admitted technical debts with attention-based neural networks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 871–882. https://doi.org/10.1145/3324884.3416583

[13] Zhuang, G., Qu, Y., Li, L., Dou, X., & Li, M. (2022). An empirical study of gradient-based explainability techniques for self-admitted technical debt detection. *Journal of Internet Technology*, *23*(3), 631–641. https://doi.org/10.53106/160792642022052303021

[14] Potdar, A., & Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*, 91–100. https://doi.org/10.1109/ICSME.2014.31

[15] Wehaibi, S., Shihab, E., & Guerrouj, L. (2016). Examining the impact of self-admitted technical debt on software quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, 179–188. https://doi.org/10.1109/SANER.2016.72

[16] Meng, Y., Tie, B., & Lin, D. (2023). From SATD recognition to an interpretation method based on the dataset. *International Journal of Software Engineering and Knowledge Engineering*, *33*(02), 157–180. https://doi.org/10.1142/S0218194022500693

[17] Farias, M. A. F., Neto, M. G. M., Kalinowski, M., & Spínola, R. O. (2020). Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Information and Software Technology*, *121*, 106270. https://doi.org/10.1016/j.infsof.2020.106270

[18] Lin, C., Jiang, B., Huang, Q., & Wang, Y. (2024). An empirical study on the urgent self-admitted technical debt. In *Computer Supported Cooperative Work and Social Computing: 18th CCF Conference*, 309–320. https://doi.org/10.1007/978-981-99-9640-7_23

[19] Kataria, D., Walid, A., Daneshmand, M., Dutta, A., Enright, M. A., Gu, R., ..., & Darema, F. (2022). INGR roadmap artificial intelligence and machine learning chapter. In *2022 IEEE Future Networks World Forum*, 1–70. https://doi.org/10.1109/FNWF55208.2022.00133

[20] Tsoukalas, D., Mittas, N., Chatzigeorgiou, A., Kehagias, D. D., Ampatzoglou, A., Amanatidis, T., & Angelis, L. (2022). Machine learning for technical debt identification. *IEEE Transactions on Software Engineering*, *48*(12), 4892–4906. https://doi.org/10.1109/TSE.2021.3129355

[21] Farias, M. A. F., Neto, M. G. M., da Silva, A. B., & Spinola, R. O. (2015). A Contextualized Vocabulary Model for identifying technical debt on code comments. In *2015 IEEE 7th International Workshop on Managing Technical Debt*, 25–32. https://doi.org/10.1109/MTD.2015.7332621

[22] Flisar, J., & Podgorelec, V. (2018). Enhanced feature selection using word embeddings for self-admitted technical debt identification. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications*, 230–233. https://doi.org/10.1109/SEAA.2018.00045

[23] Zampetti, F., Serebrenik, A., & Di Penta, M. (2020). Automatically learning patterns for self-admitted technical debt removal. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*, 355–366. https://doi.org/10.1109/SANER48275.2020.9054868

[24] Terdchanakul, P., Hata, H., Phannachitta, P., & Matsumoto, K. (2017). Bug or not? Bug report classification using N-gram IDF. In *2017 IEEE International Conference on Software Maintenance and Evolution*, 534–538. https://doi.org/10.1109/ICSME.2017.14

[25] Maipradit, R., Hata, H., & Matsumoto, K. (2019). Sentiment classification using N-gram inverse document frequency and automated machine learning. *IEEE Software*, *36*(5), 65–70. https://doi.org/10.1109/MS.2019.2919573

[26] Zampetti, F., Fucci, G., Serebrenik, A., & Di Penta, M. (2021). Self-admitted technical debt practices: A comparison between industry and open-source. *Empirical Software Engineering*, *26*(6), 131. https://doi.org/10.1007/s10664-021-10031-3

[27] Northrop, L., Feiler, P., Gabriel, R. P., Goodenough, J., Linger, R., Longstaff, T., ..., & Wallnau, K. (2006). Ultra-large-scale systems: The software challenge of the future (ADA610356). *Software Engineering Institute, Carnegie Mellon University*. https://apps.dtic.mil/sti/html/tr/ADA610356/

[28] Besker, T., Martini, A., & Bosch, J. (2018). Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software*, *135*, 1–16. https://doi.org/10.1016/j.jss.2017.09.025

[29] Nissen, M., Kamel, M., & Sengupta, K. (2000). Integrated analysis and design of knowledge systems and processes. *Information Resources Management Journal*, *13*(1), 24–43. https://doi.org/10.4018/irmj.2000010103

[30] Bespalov, D., Bai, B., Qi, Y., & Shokoufandeh, A. (2011). Sentiment classification based on supervised latent n-gram analysis. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 375–382. https://doi.org/10.1145/2063576.2063635

[31] Shirakawa, M., Hara, T., & Nishio, S. (2015). N-gram IDF: A global term weighting scheme based on information distance. In *Proceedings of the 24th International Conference on World Wide Web*, 960–970. https://doi.org/10.1145/2736277.2741628

[32] da Silva Maldonado, E., Shihab, E., & Tsantalis, N. (2017). Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, *43*(11), 1044–1062. https://doi.org/10.1109/TSE.2017.2654244

[33] Liu, Z., Huang, Q., Xia, X., Shihab, E., Lo, D., & Li, S. (2018). SATD detector: A text-mining-based self-admitted technical debt detection tool. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 9–12. https://doi.org/10.1145/3183440.3183478

[34] Song, J., & Liu, J. (2024). ISMOTE: A more accurate alternative for SMOTE. *Neural Processing Letters*, *56*(5), 240. https://doi.org/10.1007/s11063-024-11695-w

[35] Elreedy, D., & Atiya, A. F. (2019). A comprehensive analysis of synthetic minority oversampling technique (SMOTE) for handling class imbalance. *Information Sciences*, *505*, 32–64. https://doi.org/10.1016/j.ins.2019.07.070

[36] Sheikhaei, M. S., & Tian, Y. (2023). Automated self-admitted technical debt tracking at commit-level: A language-independent approach. In *2023 ACM/IEEE International Conference on Technical Debt*, 22–26. https://doi.org/10.1109/TechDebt59074.2023.00009

[37] Wilder, G., Miyamoto, R., Watson, S., Kazman, R., & Peruma, A. (2023). An exploratory study on the occurrence of self-admitted technical debt in android apps. In *2023 ACM/IEEE International Conference on Technical Debt*, 1–10. https://doi.org/10.1109/TechDebt59074.2023.00007

[38] Verdikha, N. A., Adji, T. B., & Permanasari, A. E. (2018). Study of undersampling method: Instance hardness threshold with various estimators for hate speech classification. *International Journal of Information Technology and Electrical Engineering*, *2*(2), 39–44. https://doi.org/10.22146/ijitee.42152

[39] Klema, J., & Almonayyes, A. (2006). Automatic categorization of fanatic texts using random forests. *Kuwait Journal of Science and Engineering*, *33*(2), 1–18.

[40] Mari, & Eila. (2003). The impact of maintainability on component-based software systems. In *Proceedings of the 29th Euromicro Conference*, 25–32. https://doi.org/10.1109/EURMIC.2003.1231563

[41] Xiao, T., Wang, D., McIntosh, S., Hata, H., Kula, R. G., Ishio, T., & Matsumoto, K. (2022). Characterizing and mitigating self-admitted technical debt in build systems. *IEEE Transactions on Software Engineering*, *48*(10), 4214–4228. https://doi.org/10.1109/TSE.2021.3115772

[42] Sierra, G., Shihab, E., & Kamei, Y. (2019). A survey of self-admitted technical debt. *Journal of Systems and Software*, *152*, 70–82. https://doi.org/10.1016/j.jss.2019.02.056

[43] Howard, M. J., Gupta, S., Pollock, L., & Vijay-Shanker, K. (2013). Automatically mining software-based, semantically-similar words from comment-code mappings. In *2013 10th Working Conference on Mining Software Repositories*, 377–386. https://doi.org/10.1109/MSR.2013.6624052

[44] Irshad, M., Petersen, K., & Poulding, S. (2018). A systematic literature review of software requirements reuse approaches. *Information and Software Technology*, *93*, 223–245. https://doi.org/10.1016/j.infsof.2017.09.009

[45] Pustejovsky, J., Bergler, S., & Anick, P. (1993). Lexical semantic techniques for corpus analysis. *Computational Linguistics*, *19*(2), 331–358.

[46] Hindocha, E., Yazhiny, V., Arunkumar, A., & Boobalan, P. (2019). Short-text Semantic Similarity using GloVe word embedding. *International Research Journal of Engineering and Technology*, *6*(4), 553–558.

[47] Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., ..., & Barnes, P. (2020). Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 33–44. https://doi.org/10.1145/3351095.3372873

[48] Ghaddar, B., & Naoum-Sawaya, J. (2018). High dimensional data classification and feature selection using support vector

machines. *European Journal of Operational Research*, *265*(3), 993–1004. https://doi.org/10.1016/j.ejor.2017.08.040

[49] Jain, P. K., Saravanan, V., & Pamula, R. (2021). A hybrid CNN-LSTM: A deep learning approach for consumer sentiment analysis using qualitative user-generated contents. *ACM Transactions on Asian and Low-Resource Language Information Processing*, *20*(5), 84. https://doi.org/10.1145/3457206

[50] Aria, M., Cuccurullo, C., & Gnasso, A. (2021). A comparison among interpretative proposals for Random Forests. *Machine Learning with Applications*, *6*, 100094. https://doi.org/10.1016/j.mlwa.2021.100094

[51] Xiao, T., Zeng, Z., Wang, D., Hata, H., McIntosh, S., & Matsumoto, K. (2024). Quantifying and characterizing clones of self-admitted technical debt in build systems. *Empirical Software Engineering*, *29*(2), 54. https://doi.org/10.1007/s10664-024-10449-5

[52] Li, Y., Soliman, M., & Avgeriou, P. (2022). Identifying self-admitted technical debt in issue tracking systems using machine learning. *Empirical Software Engineering*, *27*(6), 131. https://doi.org/10.1007/s10664-022-10128-3

[53] Rantala, L., & Mäntylä, M. (2020). Predicting technical debt from commit contents: Reproduction and extension with automated feature selection. *Software Quality Journal*, *28*(4), 1551–1579. https://doi.org/10.1007/s11219-020-09520-3

[54] Sharma, R., Shahbazi, R., Fard, F. H., Codabux, Z., & Vidoni, M. (2022). Self-admitted technical debt in R: Detection and causes. *Automated Software Engineering*, *29*(2), 53. https://doi.org/10.1007/s10515-022-00358-6

[55] Digkas, G., Lungu, M., Avgeriou, P., Chatzigeorgiou, A., & Ampatzoglou, A. (2018). How do developers fix issues and pay back technical debt in the Apache ecosystem? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering*, 153–163. https://doi.org/10.1109/SANER.2018.8330205

[56] Alizadehsani, Z., Gomez, E. G., Ghaemi, H., González, S. R., Jordan, J., Fernández, A., & Pérez-Lancho, B. (2022). Modern integrated development environment (IDEs). In *Sustainable Smart Cities and Territories*, 274–288. https://doi.org/10.1007/978-3-030-78901-5_24

[57] Rinta-Kahila, T., Someh, I., Gillespie, N., Indulska, M., & Gregor, S. (2022). Algorithmic decision-making and system destructiveness: A case of automatic debt recovery. *European Journal of Information Systems*, *31*(3), 313–338. https://doi.org/10.1080/0960085X.2021.1960905

[58] Lenarduzzi, V., Saarimaki, N., & Taibi, D. (2019). On the diffuseness of code technical debt in Java projects of the Apache ecosystem. In *2019 IEEE/ACM International Conference on Technical Debt*, 98–107. https://doi.org/10.1109/TechDebt.2019.00028

[59] Le, T. H. M., Chen, H., & Babar, M. A. (2021). Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Computing Surveys*, *53*(3), 62. https://doi.org/10.1145/3383458

[60] Ullah, S., Uddin, M. I., Adnan, M., Alarood, A. A., Alsulami, A., & Habibullah, S. (2025). In-depth exploration of software defects and self-admitted technical debt through cutting-edge deep learning techniques. *PLoS One*, *20*(6), e0324847. https://doi.org/10.1371/journal.pone.0324847

[61] Sotiropoulos, P., & Vassilakis, C. (2022). The additional testsuite framework: Facilitating software testing and test management. *International Journal of Web Engineering and Technology*, *17*(3), 296–334. https://doi.org/10.1504/IJWET.2022.127876

[62] Anim-Annor, A., Boafo, F., & Mensah, S. (2023). An NLP approach for identification and detection of self-admitted technical debt: A review of existing techniques. *International Journal of Computer Applications*, *185*(31), 38–44. https://doi.org/10.5120/ijca2023923074

[63] Kaur, A., & Nayyar, R. (2020). A comparative study of static code analysis tools for vulnerability detection in C/C++ and JAVA source code. *Procedia Computer Science*, *171*, 2023–2029. https://doi.org/10.1016/j.procs.2020.04.217

[64] Li, Y., Soliman, M., Avgeriou, P., & van Ittersum, M. (2023). DebtViz: A tool for identifying, measuring, visualizing, and monitoring self-admitted technical debt. In *2023 IEEE International Conference on Software Maintenance and Evolution*, 558–562. https://doi.org/10.1109/ICSME58846.2023.00072

[65] Huidrom, L., Das, L. K., & Sud, S. K. (2013). Method for automated assessment of potholes, cracks and patches from road surface video clips. *Procedia – Social and Behavioral Sciences*, *104*, 312–321. https://doi.org/10.1016/j.sbspro.2013.11.124

[66] Gurusamy, B. M., Kumar, R. P., & Altalbe, A. (2024). Whale-optimized LSTM networks for enhanced automatic text summarization. *Frontiers in Artificial Intelligence*, *7*, 1399168. https://doi.org/10.3389/frai.2024.1399168

[67] Mohan, G. B., Kumar, R. P., Parathasarathy, S., Aravind, S., Hanish, K. B., & Pavithria, G. (2023). Text summarization for big data analytics: A comprehensive review of GPT-2 and BERT approaches. In R. Sharma, G. Jeon, & Y. Zhang (Eds.), *Data analytics for internet of things infrastructure* (pp. 247–264). Springer. https://doi.org/10.1007/978-3-031-33808-3_14

[68] Chowdary, S. M., & Rangarajan, P. K. (2023, October). Automated identification and prioritization of self-admitted technical debt using NLP word embeddings. *Proceedings of the 2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)* (pp. 963–971). IEEE. https://doi.org/10.1109/ICSSAS57918.2023.10331839

[69] Sutoyo, E., & Capiluppi, A. (2024). SATDAUG—A balanced and augmented dataset for detecting self-admitted technical debt. In *Proceedings of the 21st International Conference on Mining Software Repositories*, 289–293. https://doi.org/10.1145/3643991.3644880

How to Cite: Gorripati, S. M. C., Altalbe, A., & Rangarajan, P. K. (2025). Advances in Managing Self-Admitted Technical Debt: A Review of NLP and Machine Learning Approaches. *Journal of Computational and Cognitive Engineering.* https://doi.org/10.47852/bonviewJCCE52025975