IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 January 2026, accepted 1 February 2026, date of publication 6 February 2026, date of current version 11 February 2026.

*Digital Object Identifier 10.1109/ACCESS.2026.3661069*

RESEARCH ARTICLE

# Automated Detection of Self-Admitted Technical Debt: A Systematic Literature Review

**ANAS ABDULAZIZ[1,2], RIAD SONBOL[1,2], AND MOUHIB ALNOUKARI[2]**
[1]Higher Institute for Applied Sciences and Technology (HIAST), Damascus, Syria
[2]Faculty of Informatics Engineering, Syrian Private University (SPU), Damascus, Syria
Corresponding author: Anas Abdulaziz (anas.abdulaziz@hiast.edu.sy)

**ABSTRACT** In modern software development, developers frequently introduce comments that describe temporary solutions, incomplete implementations, or deferred improvements. These comments, commonly referred to as Self-Admitted Technical Debt (SATD), may provide short-term flexibility but can accumulate into long-term maintenance challenges and quality degradation. This paper presents a systematic literature review of automated SATD detection research published between 2015 and 2025. Following established systematic review guidelines, we analyzed 46 primary studies selected from an initial pool of 1,537 publications. The review provides a structured synthesis of the field by: 1) identifying and categorizing the main SATD detection objectives; 2) analyzing representation techniques for textual, structural, and hybrid inputs; 3) comparing learning approaches ranging from traditional machine learning to transformer-based and large language models; and 4) summarizing evaluation practices and performance trends. Our findings reveal a clear methodological evolution from keyword-based and feature-engineered models toward context-aware and multi-modal architectures. While transformer-based approaches such as BERT and CodeBERT demonstrate strong potential, their adoption is constrained by limited benchmark datasets, inconsistent evaluation protocols, and a lack of explainability. To address these challenges, the review highlights key research gaps and outlines future directions, including standardized benchmarks, explainable SATD detection, and cross-project generalization. Overall, this study consolidates a decade of SATD research into a unified perspective and provides actionable guidance for developing more robust, interpretable, and scalable SATD detection techniques to support sustainable software maintenance.

**INDEX TERMS** Self-admitted technical debt, technical debt detection, systematic literature review, machine learning, deep learning, natural language processing, transformer models, software quality.

## I. INTRODUCTION

In software engineering, technical debt (TD) is a widely recognized concept describing the deliberate trade-offs developers make to accelerate short-term progress at the expense of long-term code quality and maintainability. While such decisions can expedite delivery, they often introduce hidden costs that must later be "repaid." A particularly visible manifestation of this phenomenon is Self-Admitted Technical Debt (SATD), where developers explicitly document suboptimal implementations, incomplete features, or known issues within source code comments [1], [2].

Unlike many forms of technical debt that are implicit and must be inferred through static analysis or architectural inspection, SATD is explicitly articulated by developers through natural language artifacts such as source-code comments. This explicit self-disclosure makes SATD particularly amenable to automated analysis using Natural Language Processing (NLP) techniques, as the presence of debt is directly observable without requiring complex program inference. At the same time, SATD poses unique challenges for automation: developer comments are informal, context-dependent, and often express debt implicitly without standardized terminology. As a result, effective SATD detection requires models capable of capturing semantic nuance and contextual intent rather than relying solely on surface-level keyword matching.

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

These comments often contain valuable insights into system behavior, including temporary fixes or design shortcuts adopted during implementation. However, as software systems evolve and grow in complexity, these remarks are frequently neglected, leading to an accumulation of technical debt that degrades maintainability and overall software quality.

Managing SATD poses a considerable challenge, especially in large-scale and long-lived projects where such comments are abundant. Manual identification is impractical, underscoring the importance of automated detection systems. Automated tools allow teams to continuously monitor the health of their codebases, prioritize refactoring tasks, and make informed decisions to sustain software quality.

Over the past decade, SATD detection research has advanced considerably. Early approaches, primarily rule-based methods relying on keyword matching [3], [4], were simple yet struggled to account for the diverse and context-dependent nature of developer comments. More recent studies have leveraged advances in Natural Language Processing (NLP), Machine Learning (ML), and Deep Learning (DL) to develop more robust and data-driven approaches [5], [6], [7].

These advancements have enhanced feature extraction, model design, and evaluation procedures. Nonetheless, several challenges persist. Many existing techniques still depend on surface-level text features, limiting their generalizability across projects and programming languages. Another recurring issue is the absence of standardized, high-quality datasets, which hinders reproducibility and fair comparison among studies. Our analysis shows that many works rely on manually curated datasets drawn from a limited number of projects, further emphasizing the need for common benchmarks [8], [9], [10].

A notable gap lies in the limited adoption of context- and semantics-aware methods. While transformer-based models such as BERT and CodeBERT have achieved remarkable success in other NLP domains, their application to SATD detection remains relatively unexplored [11]. Moreover, there is a growing need to integrate SATD detection seamlessly into modern development environments—such as continuous integration systems, issue trackers, and Integrated Development Environments (IDEs)—to facilitate real-time analysis and actionable feedback.

This Systematic Literature Review (SLR) seeks to address these methodological and empirical gaps by comprehensively analyzing research on SATD detection and classification. We examine 46 peer-reviewed studies published between 2015 and 2025, evaluating their objectives, datasets, experimental designs, and reported outcomes. The review aims to provide researchers, practitioners, and tool builders with a consolidated view of the current landscape and actionable insights for advancing SATD research and practice. We categorize existing SATD detection approaches into three primary groups: clustering-based methods [10], [12], supervised classification models, and hybrid frameworks that combine multiple paradigms. By examining the strengths, weaknesses, and representational strategies employed in each, this study offers a comprehensive overview of the field's evolution and emerging trends.

**The main contributions of this SLR are as follows:**

- We propose a unified taxonomy of SATD detection techniques encompassing rule-based, statistical, and deep learning methods.
- We provide a comparative analysis of data representations, feature engineering strategies, and model performance across studies.
- We highlight the lack of standardized datasets and propose directions for building publicly available benchmark corpora.
- We identify methodological gaps and outline practical recommendations for integrating SATD detection into software engineering workflows and tools.

The remainder of this paper is organized as follows. Section II describes the review methodology and paper selection process. Section III presents the results of the analysis. Section IV discusses prevailing trends, challenges, and future research directions. Finally, Section V concludes the paper and summarizes the key findings.

## II. BACKGROUND

The idea of technical debt (TD) was first introduced by Ward Cunningham in the early 1990s to describe the trade-offs developers make when prioritizing fast delivery over long-term code quality [13]. While taking shortcuts can accelerate progress in the short run, these decisions often come with hidden costs that must be ''repaid'' later through additional maintenance. Over time, the concept of TD has grown to encompass several dimensions, including design, code, and documentation debt [2].

In recent years, a specific and increasingly studied form of TD has emerged: Self-Admitted Technical Debt (SATD). This occurs when developers explicitly acknowledge technical shortcomings, workarounds, or pending improvements within their source code comments [1], [2]. Unlike other kinds of debt, SATD is openly declared by developers, serving both as a form of communication and as a record of known limitations. These comments often describe temporary fixes, known bugs, or design constraints that must be revisited. While such openness can enhance team awareness and transparency, SATD can accumulate if left unresolved, gradually making systems harder to understand, more expensive to maintain, and more prone to defects.

The first empirical investigations into SATD appeared in the mid-2010s. Potdar and Shihab's study on open-source projects revealed that SATD is not only common but also diverse in its nature, reflecting differences in project context and team practices [1]. Their findings inspired a wave of research focused on building automated tools to identify and manage SATD more systematically [2].

SATD detection has since become an important aspect of software quality assurance. Unlike traditional static analysis tools that focus on code structure and syntax, SATD detection involves interpreting natural language embedded in source code. This intersection between software engineering and Natural Language Processing (NLP) has motivated researchers to develop Machine Learning (ML) and Deep Learning (DL) techniques that can automatically recognize these self-admitted instances with higher accuracy [3], [5], [14].

Early detection methods were primarily rule-based, relying on keyword searches and handcrafted patterns. Although simple and easy to implement, they struggled to generalize across projects. Subsequent research introduced supervised classifiers and unsupervised clustering approaches capable of learning richer textual patterns and context [3], [4], [5]. The success of these methods, however, often depended on dataset quality, class balance, and the representativeness of training samples.

Recent advances in DL, particularly transformer-based architectures, have opened new opportunities for capturing the semantics and intent behind code comments [6], [11]. These models can understand subtle contextual clues that simpler algorithms often miss. Yet, challenges remain: labeled datasets are scarce, domain differences persist, and data imbalance continues to affect model robustness.

Today, SATD detection stands as a rapidly evolving research area at the crossroads of empirical software engineering and applied ML [14], [15]. Understanding how this field has evolved, where its current strengths lie, and what barriers still exist is essential for building more maintainable, reliable, and transparent software systems.

## III. RESEARCH METHODOLOGY

This study was conducted as a Systematic Literature Review (SLR), following the structured guidelines presented by Kitchenham and Charters, as well as the mapping recommendations outlined by Petersen et al. The purpose of this review is to identify, categorize, and analyze research studies published between 2015 and 2025 related to the automated detection of SATD.

The review process involved multiple phases, including planning, searching, selection, data extraction, and synthesis. During the planning phase, we formulated research questions that guided the scope and objectives of the study. These questions focused on the techniques, models, datasets, evaluation metrics, and practical challenges encountered in SATD detection.

A comprehensive search strategy was applied using digital libraries such as IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus. The search strings were constructed using combinations of relevant keywords such as ''technical debt'', ''SATD'', ''machine learning'', ''natural language processing'', ''source code comments'', and ''automated detection''.

The inclusion criteria emphasized peer-reviewed journal articles and conference papers that proposed, evaluated, or reviewed automated approaches to SATD detection. Studies focusing solely on technical debt without explicit reference to SATD were excluded. We also excluded opinion papers, editorials, and studies that were not written in English.

Data extraction was performed using a structured form that recorded metadata, methodological details, input features, classification techniques, datasets used, and performance metrics. The relevance, rigor, and replicability of each study were assessed.

The findings from this SLR provide a detailed overview of the current state of SATD detection research, offer insights into common practices and trends, and highlight critical gaps that warrant further investigation.

### A. PLANNING

During the planning phase, a review protocol was defined *a priori* in accordance with established guidelines for systematic literature reviews in software engineering. The protocol specified the research questions, search strategy, digital libraries, inclusion and exclusion criteria, screening stages, and data extraction procedure before initiating the literature search. Defining the protocol in advance ensured transparency, reduced selection and reporting bias, and improved the reproducibility of the review process.

- **RQ1:** What is the current state of the published literature on SATD identification?
- **RQ2:** What are the main categories of SATD commonly detected?
- **RQ3:** What representation techniques are used to represent textual data, source code, and combinations of both?
- **RQ4:** What are the most effective representation techniques for classifying each type of input?
- **RQ5:** What approaches and methodologies have been used to address the SATD detection problem?
- **RQ6:** What evaluation metrics are commonly used to assess the performance of SATD detection models?
- **RQ7:** What research gaps and potential directions exist for future work in the field of SATD detection?

These research questions laid the foundation for designing our search strategy, selection criteria, and data extraction process.

### B. SEARCH STRATEGY

A comprehensive search strategy was developed to identify relevant publications in several major academic databases, including IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus. The search was conducted using a Boolean query that combines key thematic concepts related to technical debt, software, detection techniques, ML, and source code analysis.

The exact search string used was

**TABLE 1. Search keyword groups utilized in the systematic literature review.**

| Group | Keywords and Synonyms |
|---|---|
| A: Topic | "technical debt" |
| B: Domain | "software" |
| C: Task/Activity | Detection, identification, prediction, detect, identify, predict |
| D: Technique | "Natural Language Processing", "machine learning", "deep learning" |
| E: Source | "source code", "comment", "pull request" |

("technical debt") AND ("software") AND (detection OR identification OR prediction OR detect OR identify OR predict) AND ("NLP" OR "Natural Language Processing" OR "machine learning" OR "deep learning") AND ("source code" OR "comment" OR "pull request")

### C. SELECTION CRITERIA

Inclusion and exclusion criteria were applied to filter out studies that do not align with the objectives and research questions of this review. These criteria helped ensure that only high-quality and relevant primary studies were selected for analysis.

#### 1) INCLUSION CRITERIA

We applied the following inclusion criteria:

- IC1: The study must be retrieved using the defined search query.
- IC2: The paper must address the detection, classification, or analysis of SATD.
- IC3: The study must be a primary research contribution published in a peer-reviewed conference or journal between 2015 and 2025.

#### 2) EXCLUSION CRITERIA

Studies were excluded based on the following criteria:

- **EC1:** The study is a secondary source, such as a literature review, survey, or meta-analysis, and therefore does not present original research findings relevant to SATD detection.
- **EC2:** The publication is not written in English, which limits its accessibility for comprehensive analysis and may hinder consistent data extraction and interpretation.
- **EC3:** The study is a duplicate publication; in such cases, only the most complete and recent version was retained to avoid redundancy in the dataset.
- **EC4:** The paper lacks sufficient methodological detail to enable systematic analysis or reproducibility, such as missing information on data sources, experimental setup, or evaluation procedure. This includes abstracts, extended abstracts, poster papers, short papers, or vision papers that do not provide a complete description of the proposed approach.
- **EC5:** The paper does not explicitly address or apply automated techniques for SATD detection, thus falling outside the scope of the study's focus.
- **EC6:** The publication is not peer-reviewed (e.g., blog posts, white papers, or unpublished manuscripts), and thus does not meet the quality threshold established for inclusion.

### D. CONDUCTING THE SEARCH

To ensure comprehensive coverage of relevant research, we conducted a systematic search across five major digital libraries: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus. The search strategy employed a carefully designed Boolean query and was executed through each platform's advanced search interface between January and July 2025, targeting publications from 2015 to 2025. Whenever possible, search results were exported for structured filtering and analysis. In total, 1,537 records were initially retrieved. After automated and manual deduplication, 1,364 unique studies remained. These records were then screened through a two-stage selection process conducted by two reviewers. During the first stage, each reviewer independently evaluated titles and abstracts to identify potentially relevant studies. In the second stage, shortlisted papers were independently assessed through full-text examination against the predefined inclusion and exclusion criteria, considering both topical relevance and methodological rigor.In cases of disagreement, borderline studies were re-evaluated during a secondary, criteria-based assessment phase that allowed partial satisfaction of inclusion criteria, after which the reviewers jointly reviewed these cases and reached a final decision through consensus.

To ensure consistency, transparency, and traceability throughout the review, we used *Parsifal*—a dedicated online tool for managing systematic literature reviews. Parsifal facilitated the structured documentation of metadata such as publication year, venue, authorship, abstract, methodology, and tools used in each study, allowing efficient organization and future reproducibility of our analysis. This rigorous process ensured that the final pool of studies accurately represented the state of research on automated SATD detection within the defined period and thematic scope.

Parsifal was used strictly as a review management and documentation platform to support study selection, metadata organization, and traceability; all data extraction, classification, and analysis were performed manually by the reviewers using the predefined Data Extraction Form.

### E. DATA EXTRACTION AND CLASSIFICATION

To ensure consistency and thoroughness in synthesizing the findings, a structured Data Extraction Form (DEF) was designed. The DEF was developed to systematically capture all relevant information required to address the predefined research questions. Organized as a tabular framework, it facilitated uniform data collection across studies, promoting

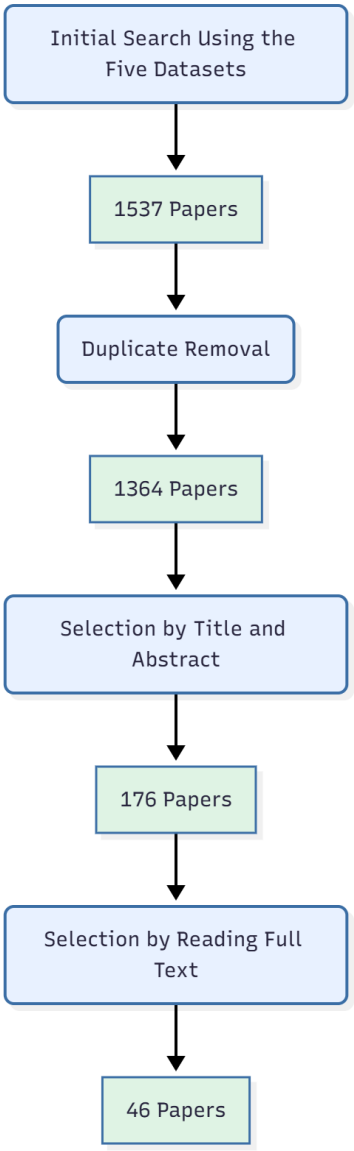



**FIGURE 1.** PRISMA flow diagram.

clarity, comparability, and reproducibility in the subsequent analysis.

- **Bibliometric Information:** authors, year of publication, type of publication, and venue.
- **SATD Type/Classification:** categories of SATD addressed in the study (e.g., design debt, requirement debt).
- **Input Representation:** techniques used to represent SATD inputs, such as textual comments, source code, or combined modalities.
- **Proposed Approach:** detection or classification methods, including rule-based heuristics, ML, or DL models.
- **Evaluation Details:** datasets used, evaluation metrics reported, and performance results.
- **Limitations and Constraints:** reported challenges, assumptions, and external validity concerns.

Each research question was explicitly addressed using a subset of the extracted DEF fields. RQ1 was informed by bibliometric information (authors, publication year, and venue) to analyze publication trends and research evolution. RQ2 relied on the extracted SATD type and classification labels. RQ3 and RQ4 were addressed using the input representation and feature encoding fields. RQ5 was supported by the proposed approach and learning methodology fields. RQ6 was analyzed based on the reported evaluation metrics. Finally, RQ7 was synthesized from the documented limitations, datasets, and experimental constraints reported in each study.

## IV. RESULTS AND DISCUSSION

This section presents a comprehensive synthesis of findings from the 46 primary studies selected for review. Each of the predefined research questions (RQ1–RQ7) is addressed systematically to highlight key trends, methodological frameworks, input representation strategies, evaluation practices, and open challenges in the field of automated SATD detection. The analysis not only captures the breadth and diversity of current research but also identifies recurring limitations and underexplored areas that warrant further investigation.

The discussion is structured to deliver a holistic view of the literature, examining the detection techniques employed, the categories of SATD targeted, the nature of input data representations, and the performance evaluation methods used. Each of the following subsections corresponds directly to one of the research questions, presenting aggregated insights, quantitative summaries, and critical interpretations drawn from the reviewed studies.

### A. RQ1: WHAT IS THE STATE OF THE PUBLISHED LITERATURE ON SATD IDENTIFICATION?

Our review identified 46 peer-reviewed studies that explicitly focus on the automated detection or classification of Self-Admitted Technical Debt (SATD). As shown in Fig. 2, research in this area has grown steadily over the past decade. The earliest studies, published between 2015 and 2017, were largely exploratory—they introduced the notion of SATD and examined how frequently developers acknowledge Technical Debt (TD) in source code comments. From 2018 onward, the field gained momentum, with a consistent rise in publications and a clear peak between 2021 and 2025.

This growing attention reflects a broader shift within software engineering research. Advances in Natural Language Processing (NLP), Machine Learning (ML), and Deep Learning (DL) have made it increasingly feasible to analyze the natural-language content of source code comments at scale. At the same time, the emergence of open datasets and shared repositories has enabled reproducible experimentation and large-scale comparisons. As a result, SATD identification has evolved from a niche research topic into an active, data-driven subfield of software maintenance and quality assurance.

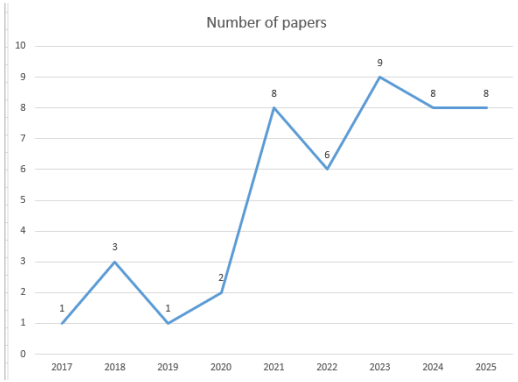



**FIGURE 2.** Annual distribution of SATD-related publications (2015–2025).

The distribution of publication venues also highlights the diversity of the research community working on SATD. Approximately half of the reviewed studies were presented at major international conferences such as SANER, MSR, ICSE, and TechDebt, emphasizing the field's strong connection to software evolution and maintenance. Around 40% appeared in indexed journals—including *IEEE Transactions on Software Engineering*, *Empirical Software Engineering*, and the *Journal of Systems and Software*—demonstrating increasing methodological rigor and academic visibility. The remaining papers were published in specialized workshops and symposia, often serving as incubators for innovative ideas and experimental tools.

Over time, the methodological landscape of SATD research has broadened considerably. Early work relied on simple keyword searches or handcrafted linguistic rules to identify potential SATD comments. While limited in scope, these initial approaches provided the foundation for subsequent advancements. In recent years, researchers have turned to neural and transformer-based models that can interpret the semantic and contextual nuances of developer comments, marking a decisive shift toward context-aware and semantics-driven detection.

Taken together, these findings indicate that SATD identification has matured into a dynamic and steadily expanding research domain. The sustained rise in publications, the diversity of publication venues, and the evolution of techniques all demonstrate a field that is both scientifically robust and practically significant. SATD is no longer viewed as a peripheral concern; it has become an integral component of the broader effort to improve software quality, maintainability, and long-term sustainability.

### B. RQ2: WHAT ARE THE MAIN CATEGORIES OF SATD COMMONLY DETECTED?

The landscape of SATD research has evolved from rudimentary identification tasks to sophisticated, context-aware frameworks that recognize the diverse ways in which debt manifests in software projects. A synthesis of the forty-six reviewed studies reveals five recurring categories of detection objectives: **Binary Identification**, **SATD Type Classification**, **Priority and Severity Estimation**, **Relationship and Evolution Analysis**, and **Specialized Contexts**. These categories delineate the maturity of the field—from recognizing debt existence to understanding its semantics, impact, and dynamics over time.

#### 1) BINARY IDENTIFICATION

The earliest and most pervasive task conceptualizes SATD detection as a *binary classification* problem—determining whether a developer comment or artifact contains debt. This foundational framing established the methodological bedrock for later research. Initial studies relied on conventional **ML** classifiers such as Support Vector Machines (SVM), Naïve Bayes, and Random Forests, trained on handcrafted lexical and syntactic features (e.g., TF–IDF, n-grams, part-of-speech tags). These models captured surface-level textual cues such as "TODO," "fix later," or "temporary solution." With the expansion of data availability, researchers adopted **DL** architectures such as Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) networks, capable of modeling semantic dependencies and developer intent. More recently, transformer-based **NLP** encoders—including BERT, RoBERTa, and CodeBERT—have become the dominant paradigm. They contextualize tokens within broader linguistic and code structures, substantially improving recall for implicit forms of SATD that lack explicit debt markers.

Binary identification remains crucial, not as an end in itself, but as a prerequisite filter for subsequent subtype classification and prioritization modules within modern SATD pipelines.

#### 2) SATD TYPE CLASSIFICATION

As the field matured, attention shifted from identifying *whether* debt exists to determining *what kind* of debt it represents. Studies in this category distinguish among well-established SATD subtypes—*Design Debt*, *Requirement Debt*, *Code Debt*, *Defect Debt*, and *Documentation Debt*. Type classification provides actionable granularity: design-related debts signal architectural degradation, requirement debts highlight evolving functionality, and documentation debts reveal knowledge erosion.

Approaches here blend structural and linguistic evidence. Classical **ML** pipelines exploit measurable code attributes (e.g., cyclomatic complexity, lines of code, coupling, and churn), whereas **DL** and transformer-based **NLP** models leverage contextual embeddings that capture developer rationale and sentiment. Several works operationalize "code-smell" patterns—such as *Long Parameter List* or *Complex Method*—as design-oriented SATD indicators. Others fuse code metrics with textual embeddings to form hybrid representations that bridge structural and semantic spaces.

Empirically, this category demonstrates the field's pivot from surface lexical matching to multimodal learning, where SATD is interpreted as an interplay of language, structure, and intent.

### 3) PRIORITY AND SEVERITY ESTIMATION

Not all debts exert equal technical or managerial impact. Consequently, a subset of studies frames SATD detection as an *impact-assessment* task, classifying or ranking debt according to its severity or urgency (e.g., high vs. low, short-term vs. long-term risk). These works integrate temporal and process-aware features—such as modification frequency, commit density, or code ownership—with textual and structural signals. By combining project-history analytics with **NLP**-derived embeddings, models can estimate the likelihood that a debt item will persist or cause failure. Ensemble **ML** methods and attention-based **DL** networks are common, allowing the models to focus on critical linguistic indicators (e.g., frustration, uncertainty) that often correlate with severe debt. This line of inquiry moves SATD research toward predictive maintainability, supporting resource allocation and technical risk management rather than mere classification.

### 4) RELATIONSHIP AND EVOLUTION ANALYSIS

Beyond static classification, several studies adopt a longitudinal lens, investigating the *relationships* among SATD instances and their *evolution* across software releases. Typical outputs include relationship labels (e.g., duplication, refinement, dependency) and lifecycle states (introduced, postponed, repaid). This research leverages cross-artifact **NLP**—linking comments, issue trackers, and commit messages—to trace how debts propagate and how repayment actions manifest in subsequent code revisions. Graph-based models, sequence learning, and clustering algorithms capture the temporal dynamics of debt interactions. Such analyses illuminate the socio-technical behavior of debt: developers frequently reopen, merge, or rephrase SATD items; some debts decay naturally as features evolve, while others persist due to organizational inertia. By modeling these transitions, relationship-oriented studies shift SATD detection from a snapshot problem to a continuous process analysis, aligning the field with contemporary research on software evolution and technical risk forecasting.

### 5) SPECIALIZED CONTEXTS

A final and rapidly expanding category applies SATD-style detection frameworks to *domain-specific technical debt*. Examples include identifying defect-related debt, security vulnerabilities framed as debt, machine-learning technical debt (MLTD) within data pipelines, and build-system or configuration-related debt. Although the target artifacts vary—from Python notebooks to build scripts—the methodological backbone remains consistent: text mining and semantic modeling via **NLP**, supported by **ML/DL** classifiers.

These studies demonstrate that SATD principles generalize beyond source comments to any artifact expressing self-acknowledged deficiency. They also underscore the convergence between traditional software-engineering analytics and modern data-centric disciplines, where debt emerges as both a linguistic construct and a system-design liability.

**TABLE 2. Reviewed studies by SATD detection category.**

| Detection Category | All Studies |
|---|---|
| Binary Identification | [3], [4], [6], [8], [9], [11], [16]–[21], [21]–[23], [23]–[39] |
| SATD Type Classification | [9], [17], [19], [23], [34], [40], [41] |
| Priority/Severity Estimation | [42] |
| Relationship & Evolution Analysis | [8], [31] |
| Specialized Contexts | [43] |

A comparative examination of these detection categories reveals a clear imbalance in how SATD research efforts are distributed. The majority of existing studies focus on binary identification, reflecting the field's early emphasis on establishing the feasibility of automated SATD detection at scale. In contrast, SATD type classification, while increasingly explored, remains less prevalent and is typically investigated using richer representations that combine textual and structural information. More advanced objectives—such as priority or severity estimation and relationship or evolution analysis—are addressed by only a limited number of studies. These tasks often require longitudinal project data, historical context, and hybrid feature sets, which increases modeling complexity and restricts dataset availability. Overall, this distribution suggests a gradual maturation of the field: early research prioritizes broad detection coverage, whereas more recent work shifts toward interpretability, impact awareness, and long-term maintenance insights, albeit at the cost of scalability.

Overall, the reviewed literature reveals a clear methodological trajectory. Early rule-based and feature-engineered **ML** systems established the feasibility of automatic SATD detection. Subsequent **DL** architectures captured syntactic and semantic nuances, while recent transformer-based **NLP** models integrate contextual reasoning across code and natural language. Collectively, these five detection categories illustrate the field's progression from identifying isolated SATD comments to constructing intelligent, multimodal frameworks that interpret debt type, severity, and evolution—laying the groundwork for predictive and self-adaptive technical debt management in modern software projects.

Before presenting the individual results, it is important to clarify the conceptual distinction among RQ3, RQ4, and RQ5. RQ3 focuses on *how SATD-related artifacts are represented* prior to learning, including textual, structural, and hybrid encodings. RQ4 builds upon RQ3 by comparatively analyzing *the effectiveness of these representations* across different input types and SATD categories, based on reported empirical results. RQ5, in contrast, examines *the solving strategies themselves*, namely the learning paradigms, model architectures, and inference frameworks employed to leverage these representations. Together, these research

questions provide a layered analysis that separates input modeling (RQ3), representational effectiveness (RQ4), and learning methodology (RQ5), while preserving their natural interdependencies.

### C. RQ3: WHAT REPRESENTATION TECHNIQUES ARE USED TO ENCODE TEXTUAL DATA, SOURCE CODE, AND THEIR COMBINATION?

The performance of Self-Admitted Technical Debt (SATD) detection models is closely tied to how textual and structural information is represented prior to learning. Across the forty-six reviewed studies, representation approaches have evolved from sparse lexical encodings to semantically enriched embeddings and structure-aware models capable of integrating both text and code semantics. These representations aim to capture not only the linguistic expression of developer intent but also the syntactic and semantic context of the underlying implementation.

#### 1) REPRESENTATION TECHNIQUES FOR SATD DETECTION

##### a: TEXTUAL REPRESENTATIONS

Textual artifacts—such as source-code comments, commit messages, issue tracker reports, and pull-request discussions—remain the primary inputs for SATD detection pipelines. Early studies employed sparse lexical encodings, notably Bag-of-Words (BoW) and Term Frequency–Inverse Document Frequency (TF–IDF), in combination with classical **ML** classifiers such as Support Vector Machines (SVM), Logistic Regression, and Naïve Bayes. These methods provided reliable baselines for SATD detection but were limited in their ability to capture contextual meaning [2], [4].

Subsequent research introduced dense word embeddings—such as Word2Vec, GloVe, and FastText—which project words into continuous vector spaces that encode semantic proximity. Zhu et al. [44] and Zampetti et al. [8] used Word2Vec with convolutional and recurrent **DL** models, achieving substantial improvements in F1-score over TF–IDF. More recently, contextualized embeddings derived from pretrained transformer models (e.g., BERT, RoBERTa) have become the de facto standard. Sabbah and Hanani [6] and Skryseth et al. [11] demonstrated that transformer encoders can capture subtle linguistic cues and dependencies, yielding superior recall for implicit or indirect forms of SATD.

##### b: SOURCE CODE REPRESENTATIONS

Beyond text, several forms of SATD—especially code and design debt—manifest directly within the structure of the source code. Structure-aware representations have therefore gained prominence for modeling program semantics.

- **Abstract Syntax Trees (ASTs):** ASTs capture the hierarchical syntactic structure of source code. Sala et al. [45] and Ni et al. [46] leveraged ASTs to identify deeply nested or redundant code patterns indicative of SATD.
- **Graph-Based Models:** Control Flow Graphs (CFGs) and Program Dependency Graphs (PDGs) encode execution flow and data dependencies. These enable a more nuanced analysis of code-level and architectural debt.
- **Code Embeddings:** Pretrained transformers on code—such as CodeBERT, GraphCodeBERT, and CodeT5—produce semantically rich vector embeddings. Nanadani et al. [47] demonstrated that these models improve maintainability-related debt detection and refactoring recommendation accuracy.

##### c: MULTI-MODAL REPRESENTATIONS

Recent works integrate both textual and code-based signals, acknowledging that SATD often arises from the interaction between what developers express and how it is implemented. Representative examples include:

- Code comments paired with corresponding source code [6];
- Comments linked with commit messages [48];
- Comments combined with issue tracker text [49];
- Unified modeling of pull requests, issues, and commits [11].

In such models, textual data are typically represented using BERT or Word2Vec embeddings, while code is encoded via ASTs, graphs, or code transformers. These multi-modal embeddings enable cross-referencing between linguistic and structural information, resulting in more contextually accurate SATD classification.

##### d: EMERGING TRENDS AND HYBRID ARCHITECTURES

The ongoing evolution of SATD representation learning reflects three major research directions:

- **Semantic Transformers:** Pretrained models such as BERT, CodeBERT, and CodeT5 serve as semantic backbones for advanced SATD pipelines.
- **Graph Neural Networks (GNNs):** GNNs and Graph Convolutional Networks (GCNs) combine structural and textual features to model relationships across code entities.
- **Fusion Architectures:** Hybrid CNN–RNN–Attention networks enable multi-label classification and fine-grained SATD type prediction [47], [50].

These models embody a paradigm shift from isolated features to end-to-end learning frameworks capable of reasoning over syntax, semantics, and developer intent simultaneously.

##### e: ADDITIONAL REPRESENTATIONAL FEATURES

Some studies incorporate handcrafted features—such as comment length, entropy, revision frequency, and code churn—to complement learned embeddings [9], [51], [52]. While these features enhance interpretability and project-level analysis, they are less robust across datasets and are typically integrated into ensemble or hybrid pipelines rather than used independently.

A comparative synthesis across representation strategies highlights clear trade-offs between expressiveness,

**TABLE 3.** Summary of representation techniques used in SATD detection.

| Technique | Category | Representative Studies |
|---|---|---|
| Bag-of-Words (BoW), TF–IDF | Traditional Sparse Representations | Maipradit et al. (2020); Sridharan et al. (2021); Flisar et al. (2018) |
| Word2Vec, GloVe, FastText | Dense Word Embeddings | Zampetti et al. (2020); Zhu et al. (2021); Wattanakriengkrai et al. (2018) |
| Feature-Based Metrics (e.g., LOC, Complexity, Churn) | Statistical and Structural Features | Tsoukalas et al. (2021); Shukla & Kumar (2023) |
| Code-Smell Heuristics (e.g., Long Parameter List) | Rule-Driven Representations | Li & Zhang (2022); Nanadani et al. (2023); Sala et al. (2021) |
| AST or Graph-Based Encoding | Structure-Aware Representations | Li et al. (2023); Ghammam et al. (2025) |
| Hybrid (Text + Code) | Multi-Modal Joint Representations | Maldonado et al. (2015); Hassan et al. (2019); Russo et al. (2025) |
| Contextual Transformers (BERT, RoBERTa, CodeBERT) | Contextualized Embeddings | Sabbah & Hanani (2023); Skryseth et al. (2023); Xu et al. (2025) |
| Prompt or Instruction-Based Encoding | Instruction-Tuned Transformer Models | Yu et al. (2024); Li et al. (2024) |
| LLM-Based Multi-Task Representations | Large Language Model–Driven Representations | Russo et al. (2025); Sheikhaei et al. (2024) |
| Domain-Specific Embeddings (e.g., MLTD, Build Debt) | Specialized Context Representations | Khanvilkar et al. (2025); Ghammam et al. (2025) |

scalability, and practical applicability. Textual representations—particularly contextual embeddings derived from pretrained transformers—remain the most widely adopted due to their low preprocessing cost and strong performance on implicit SATD detection. Structure-aware code representations, such as ASTs and graph-based encodings, provide deeper semantic insight into code- and design-related debt, but their adoption is limited by higher computational overhead and the need for language-specific tooling. Multi-modal representations that combine text and code consistently achieve more robust and context-sensitive predictions, especially for fine-grained SATD analysis; however, they require richer datasets, careful feature alignment, and more complex model architectures. These patterns suggest that representation choice in SATD detection is primarily driven by a trade-off between modeling depth and deployment feasibility, with recent studies increasingly favoring hybrid approaches when sufficient data and resources are available.

Overall, representation learning in SATD research mirrors broader advances in **ML**, **DL**, and **NLP**: a transition from shallow lexical encodings toward contextual, structure-aware, and multi-modal architectures. This progression enables SATD detectors to understand not only *what* developers articulate as technical debt but also *how* such debts are embedded in the evolving codebase.

### D. RQ4: WHAT ARE THE BEST REPRESENTATION TECHNIQUES TO CLASSIFY EACH TYPE OF INPUT?

The effectiveness of SATD detection systems is highly influenced by how input artifacts are represented prior to classification. Based on reported evaluation results across 24 primary studies, this section provides a comparative analysis of representation methods used for different types of input data — textual, code, and hybrid — supported by empirical performance indicators (e.g., F1-score, precision, recall, and AUC).

#### 1) TEXTUAL REPRESENTATIONS

Textual artifacts, including code comments, commit messages, and issue tracker entries, remain the most widely used input form. Three primary categories of representation dominate the literature:

- **Sparse Representations (BoW, TF-IDF, VSM):** These methods were common in earlier works [2], [4], offering simplicity and interpretability. However, their limited semantic expressiveness often results in lower classification accuracy. For instance, studies using VSM [9], [53] did not report strong performance metrics.
- **Word Embeddings:** Techniques like Word2Vec and GloVe offer dense vector representations that encode semantic similarity. Zampetti et al. [8] and Zhu et al. [44] applied Word2Vec in combination with CNN and Bi-LSTM architectures, achieving F1 scores of 0.61 and 0.64 on code comments and code segments, respectively. Yu et al. [50] reported precision and recall above 80% using GloVe with BiLSTM and attention mechanisms.
- **Contextualized Embeddings (BERT and Variants):** Transformer-based models such as BERT have gained traction for their ability to capture bidirectional context. Sabbah et al. [6] and Skryseth et al. [11] demonstrated that BERT-based models improve classification accuracy for nuanced SATD categories. Although not all studies report metrics, those that do consistently show superior F1 and AUC values compared to older methods.

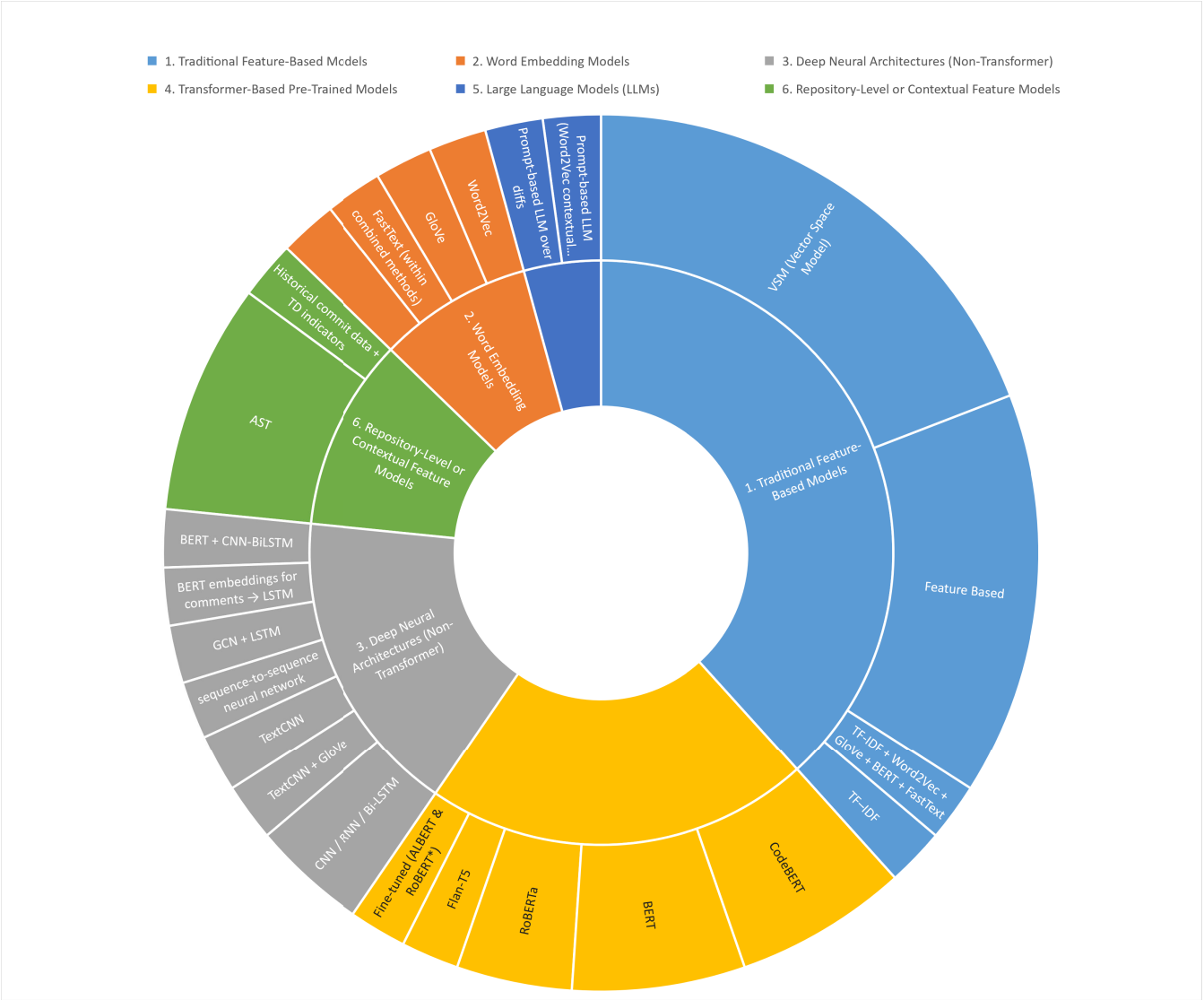



**FIGURE 3.** Distribution of representation techniques across the reviewed SATD studies.

*Conclusion:* Word2Vec paired with deep neural networks offers strong performance, while BERT remains the most semantically powerful for textual SATD when computational resources permit.

### 2) SOURCE CODE REPRESENTATIONS

Code-level SATD such as design debt and architectural debt necessitates representations that preserve structural and semantic properties. The following methods are prominent:

- **Abstract Syntax Trees (AST):** ASTs represent code syntax hierarchically and are frequently used to model structural debt. Sala et al. [45] and Kaur et al. [51] employed AST-based methods with F1-scores reaching 0.96 and precision/recall near 0.97, making this approach one of the top-performing for code-based inputs.
- **Pretrained Code Embeddings (CodeBERT):** Recent studies such as Nanadani et al. [47] and Li et al. [41] used CodeBERT to encode source code with contextual semantics. Although CodeBERT is used less frequently than AST, it provides better alignment with textual embeddings in hybrid models and shows strong maintainability classification performance.
- **Feature-Based and Historical Data:** A few studies used handcrafted metrics (e.g., method length, complexity) or commit history. These approaches, including [12], reported meaningful results using regression models, but generally lag behind AST and CodeBERT in classification tasks.

*Conclusion:* ASTs provide strong, explainable performance on code debt, while CodeBERT offers deeper semantic insight suitable for integration into end-to-end learning models.

#### 3) COMBINED REPRESENTATIONS (TEXT + CODE)

SATD often spans both code and natural language artifacts. Several studies developed multimodal pipelines to jointly process textual and source code data:

- **Sabbah et al.** [6] used BERT for comments and AST for code structure, enhancing detection of SATD types that connect narrative and implementation (e.g., design debt).
- **Li et al.** [41] extended this approach by integrating BERT and CodeBERT, forming a dual-stream encoder that significantly improved classification accuracy in multi-source scenarios.

These hybrid systems exhibit superior generalizability, particularly when used to detect SATD across repositories or in cross-project settings.

#### 4) PRACTICAL OBSERVATIONS

- **Computational Resources:** Models such as BERT and CodeBERT demand significant memory and compute power, whereas TF-IDF and VSM remain viable for lightweight applications.
- **Debt Type Sensitivity:** Structural methods (AST, CodeBERT) are best suited for detecting code/design/architecture debt. Semantic embeddings (Word2Vec, BERT) are optimal for requirement and defect debt.
- **Metric-Based Insights:** AST-based studies report the highest evaluation metrics (F1, precision, recall), followed closely by hybrid transformer models. Sparse representations consistently underperform across datasets.

### E. RQ5: WHAT SOLVING APPROACHES ARE USED TO ADDRESS SATD DETECTION?

The methodological landscape of Self-Admitted Technical Debt (SATD) detection has evolved remarkably over the last decade. Early studies emphasized surface-level textual cues and handcrafted features, while recent works increasingly exploit contextual representations, semantic embeddings, and large language models (LLMs). Based on our systematic review of forty-six primary studies, six dominant methodological families emerge: **Traditional Machine Learning (ML)**, **Deep Neural Networks (non-transformer)**, **Transformer-Based Models**, **Large Language Models (LLMs)**, **Ensemble and Fusion Approaches**, and **Specialized or Unsupervised Frameworks**. Together, they trace a clear trajectory—from shallow lexical recognition toward deep semantic understanding of developer intent.

#### *a: TRADITIONAL MACHINE LEARNING APPROACHES*

Traditional ML remains the methodological backbone of SATD research, representing roughly one-third of the reviewed corpus. Typical pipelines employ text vectorization techniques such as term frequency–inverse document frequency (TF–IDF), bag-of-words, or n-gram models, combined with classifiers including Support Vector Machines (SVM), Naïve Bayes (NB), Random Forests (RF), Logistic Regression (LR), and Decision Trees. Maldonado et al. [3] pioneered TF–IDF + SVM for detecting SATD in source-code comments, establishing a benchmark for subsequent studies. Bavota et al. [54] extended this idea through large-scale empirical analysis, while Maipradit et al. [9] and Sridharan et al. refined the feature space by incorporating syntactic cues. Ensemble-based ML baselines such as RF and XGBoost were later explored by Wattanakriengkrai, Chowdary, and others to improve recall and robustness across projects. These approaches are computationally efficient and interpretable but limited in representing nuanced semantics—an inherent weakness that motivated the transition to deep models.

#### *b: DEEP NEURAL NETWORKS (NON-TRANSFORMER)*

The advent of Deep Learning (DL) brought a paradigm shift from handcrafted features to learned representations. Around 2018–2021, several works introduced Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Bidirectional Long Short-Term Memory (BiLSTM) architectures for SATD detection. Zhu et al. and Yu et al. demonstrated that CNN–BiLSTM hybrids can capture both local patterns and sequential dependencies in comments, outperforming TF–IDF baselines in macro-F1 and recall. Other variants integrated attention mechanisms to highlight contextually important tokens. These architectures improved semantic expressiveness and generalized better to unseen repositories, though they required substantial data and careful regularization to avoid overfitting.

#### *c: TRANSFORMER-BASED MODELS*

The field's next major leap came with transformer architectures that model bidirectional context through self-attention. Roughly nine studies in our dataset rely on such models, establishing new state-of-the-art baselines. Early experiments with BERT and ALBERT demonstrated clear gains in precision and recall, while CodeBERT [55] and GraphCodeBERT [56] extended the paradigm to code-aware textual modeling. Zampetti et al. [8] applied transformer encoders to learn SATD removal patterns, whereas Sabbah et al. leveraged contextual embeddings to improve cross-project generalization. Compared with CNN- or RNN-based models, transformers capture richer developer intent and enable fine-tuning on limited data. Their scalability, however, comes with higher computational cost and interpretability challenges.

#### *d: LARGE LANGUAGE MODELS (LLMS)*

Recent studies in 2024–2025 mark a new frontier by employing general-purpose LLMs such as GPT-4 and specialized multi-task frameworks. The FRoM model, for example, integrates SATD identification and classification into a unified byte-level fine-tuning pipeline, achieving state-of-the-art F1-scores while addressing out-of-vocabulary issues. Other works explored prompt-based SATD detection in

issue trackers and repositories, demonstrating that LLMs can effectively reason about developer intent when guided by structured prompts. Although still experimental, these approaches signal a shift toward more explainable, adaptable, and zero-shot detection strategies.

#### e: ENSEMBLE AND FUSION APPROACHES

About six papers adopt hybrid or ensemble methods that combine lexical and semantic representations. Sabbah et al. and Liu et al. fused TF–IDF, Word2Vec, and BERT embeddings within voting or stacking frameworks involving SVM, RF, and CNN classifiers, achieving higher macro-F1 through complementary strengths. Other works utilized feature-level fusion, concatenating sparse and dense vectors to capture both surface patterns and contextual semantics. Such designs offer a balanced compromise between interpretability and expressiveness, bridging traditional analytics with modern DL.

#### f: SPECIALIZED OR UNSUPERVISED FRAMEWORKS

A smaller but influential subset of studies proposed custom or unsupervised solutions. Tsoukalas et al. employed clustering-based regression to discover latent topics in comment embeddings and predict SATD presence without explicit labels. Zampetti et al. [8] introduced a multi-stage pattern-learning system for automated SATD removal, while Maipradit et al. [9] designed soft-classification pipelines tailored to ''on-hold'' debt items. These domain-specific frameworks highlight the potential of integrating SATD detection directly into continuous-integration workflows.

#### g: TRENDS AND OBSERVATIONS

The chronological progression of methods reflects both conceptual and technological maturity:

- **2015–2018:** Dominance of SVM, RF, and NB models relying on lexical or keyword features [3], [54].
- **2019–2021:** Rise of CNN, BiLSTM, and attention-based networks as DL became accessible and datasets expanded.
- **2021–2023:** Transformer architectures (BERT, CodeBERT [55], GraphCodeBERT [56]) established new performance baselines.
- **2024–2025:** Emergence of LLM-driven and hybrid frameworks integrating explainability, multi-task learning, and prompt-engineering.

A comparative analysis of these methodological families reveals that their effectiveness is closely tied to the nature of the SATD detection task and the availability of data and computational resources. Traditional ML approaches remain well suited for large-scale screening scenarios where interpretability, efficiency, and rapid deployment are prioritized, but they struggle to capture implicit or context-dependent expressions of technical debt. Deep neural networks and transformer-based models address this limitation by learning richer semantic representations, making them particularly effective for nuanced SATD identification and fine-grained classification; however, their reliance on labeled data and computational cost constrains their applicability in resource-limited settings. Ensemble and fusion strategies occupy an intermediate position, leveraging complementary strengths to improve robustness and cross-project stability without fully sacrificing transparency. More recently, LLM-driven approaches have begun to redefine the problem space by enabling prompt-based reasoning, multi-task learning, and partial zero-shot detection, shifting SATD analysis from pattern recognition toward intent-aware inference. Collectively, these trends indicate that SATD-solving approaches have evolved from static classifiers into adaptive reasoning systems, with method selection increasingly driven by trade-offs between semantic depth, scalability, and practical deployment constraints.

#### h: SUMMARY

The evolution of SATD-solving approaches mirrors the broader transformation of NLP and software-analytics research. Traditional ML models remain valued for their interpretability and efficiency, yet DL and transformer architectures now dominate empirical performance. Hybrid ensembles further enhance generalization, while early LLM explorations reveal strong potential for transfer learning and explainable detection. Future research should focus on integrating **LLM-based contextual reasoning**, **explainable AI (XAI)** techniques, and **cross-project transferability** to produce scalable, trustworthy SATD detection systems. Overall, the trajectory from TF–IDF + SVM to CodeBERT- and GPT-based frameworks illustrates the community's gradual shift toward adaptive, context-aware models capable of interpreting the subtle language of technical debt.

### F. RQ6: WHAT EVALUATION METRICS ARE COMMONLY USED TO ASSESS THE PERFORMANCE OF SATD DETECTION MODELS?

Reliable evaluation is fundamental to measuring the effectiveness, fairness, and generalizability of Self-Admitted Technical Debt (SATD) detection models. Metrics determine how accurately a model captures debt-related intent and whether it generalizes across projects and repositories. Because SATD datasets are often small, noisy, and imbalanced, the choice of metric directly influences how results are interpreted and compared. From our analysis of forty-six primary studies, we observe a strong convergence around a core group of classification metrics—**F1-score**, **Precision**, and **Recall**—supplemented by selective use of **Accuracy**, **AUC (Area Under the ROC Curve)**, **MCC (Matthews Correlation Coefficient)**, and several regression-based indicators such as **MAE** and **MAPE** for forecasting tasks. Table 4 summarizes the main metrics and representative studies.

#### a: CORE METRICS FOR CLASSIFICATION TASKS

The **F1-score** remains the dominant evaluation metric in SATD research, appearing in more than three-quarters

of all reviewed studies. It offers a harmonic balance between precision and recall, mitigating the effect of skewed class distributions common in SATD datasets, where non-debt comments vastly outnumber debt-bearing ones. Maldonado et al. [3] first formalized the use of F1-score for SATD detection, setting a standard subsequently adopted by most researchers. Later works—including Wattanakriengkrai et al. [57], Maipradit et al. [9], Sridharan et al. [53], Yu et al. [50], Zhu et al. [44], and Skryseth et al. [11]—reaffirmed its role as the primary metric for both binary and multi-label SATD classification. Alongside F1, almost all studies report **Precision** and **Recall** to capture complementary aspects of model behavior. Precision measures the reliability of predicted SATD instances, while recall reflects coverage of all true debt cases. High precision but low recall indicates conservative detection; the reverse reflects overgeneralization. This triad—F1, Precision, and Recall—has become the de facto standard for evaluating SATD classifiers.

#### b: SUPPLEMENTARY CLASSIFICATION METRICS

Although **Accuracy** remains a conventional indicator in machine learning, its interpretability in SATD contexts is limited due to class imbalance. Several studies—such as Zhu et al. [44], Chen et al. [58], and Kaur and Kaur [51]—reported accuracy primarily for completeness or to compare with non-specialized models. More advanced measures like **AUC (Area Under the ROC Curve)** and **MCC (Matthews Correlation Coefficient)** have appeared in recent research, particularly in transformer-based or multi-task settings. These metrics account for class imbalance more effectively: AUC provides a threshold-independent measure of discriminative power, while MCC balances true and false positives and negatives. Patcas [59] and Russo et al. [60] both leveraged AUC and MCC to evaluate class-balanced performance across diverse SATD corpora. Additionally, a few studies—such as Russo et al. [60]—reported nonparametric significance tests (e.g., Friedman test) to confirm the statistical reliability of performance differences across models, marking a methodological maturity rarely seen in earlier SATD work.

#### c: REGRESSION AND FORECASTING METRICS

A smaller subset of research addresses SATD prediction, prioritization, or cross-project forecasting—scenarios that demand continuous rather than binary evaluation. Here, regression-based measures such as **MAE (Mean Absolute Error)** and **MAPE (Mean Absolute Percentage Error)** are employed to quantify predictive accuracy. Tsoukalas et al. [10] used MAE to assess how well clustering-based regression models forecast debt accumulation across projects, while Alhefdhi et al. [61] applied MAPE to evaluate prioritization models predicting debt severity. These metrics extend SATD research beyond binary detection, connecting it with broader software-quality and risk-prediction analytics.

#### d: TRENDS AND METHODOLOGICAL REFLECTIONS

A longitudinal view of SATD research reveals a clear evolution in evaluation rigor:

- **F1-score, Precision, and Recall** have remained dominant since 2017, providing balanced assessment under class imbalance.
- **Accuracy** usage has steadily declined as studies recognized its limitations in skewed datasets.
- **AUC and MCC** gained prominence in 2023–2025, coinciding with the adoption of transformer-based and multi-task architectures [11], [60], [62].
- **Regression-oriented metrics** (*MAE*, *MAPE*) are emerging in cross-project and forecasting work [10], [61], highlighting a methodological expansion beyond static classification.
- Statistical tests such as the *Friedman test* appear in only a few studies but signal increasing attention to reproducibility and significance.

A closer examination of metric usage across studies reveals that evaluation practices in SATD research are closely coupled with the intended analytical objective. For binary and multi-label SATD classification, the combined use of precision, recall, and F1-score provides a balanced view of detection reliability and coverage, making these metrics essential in the presence of severe class imbalance. Metrics such as accuracy, while occasionally reported, offer limited insight when debt instances are sparse and can lead to overly optimistic interpretations. In more advanced settings—such as cross-project validation or multi-task learning—AUC and MCC offer complementary perspectives by accounting for decision thresholds and class distributions more robustly. Regression-oriented metrics, including MAE and MAPE, are most appropriate when SATD is treated as a continuous or predictive phenomenon, such as debt growth or prioritization. These patterns suggest that meaningful evaluation in SATD research depends not only on metric selection but also on aligning metrics with task formulation and dataset characteristics, underscoring the need for greater standardization and transparency in future empirical studies.

#### e: SUMMARY

Overall, the SATD research community has transitioned from reporting simplistic accuracy-based measures to adopting robust, context-aware evaluation frameworks. F1-score, precision, and recall have become foundational, while AUC, MCC, and regression-based metrics represent the next phase of methodological sophistication. Future studies should complement these metrics with effect-size reporting, confidence intervals, and standardized benchmarks to ensure fair comparison across models and datasets. This progression—from binary accuracy to multifaceted, statistically grounded evaluation—mirrors the field's broader evolution toward

transparent, replicable, and contextually meaningful SATD analysis.

### G. RQ7: WHAT GAPS AND POTENTIAL FUTURE DIRECTIONS EXIST IN SATD DETECTION RESEARCH?

Despite steady progress, our review of forty-six primary studies (2015–2025) reveals that Self-Admitted Technical Debt (SATD) detection remains an evolving research frontier with several recurring limitations. These gaps reflect both methodological and practical barriers that constrain the maturity, generalizability, and real-world adoption of existing approaches.

#### 1) LIMITED CROSS-PROJECT GENERALIZABILITY

Most SATD models exhibit strong within-project performance but deteriorate when applied to unseen repositories. Early studies such as Maldonado et al. [3] and Maipradit et al. [9] trained and evaluated models on individual datasets, demonstrating high precision but limited transferability. Later works attempted to mitigate this through multi-project experiments [49], [53], [57], yet vocabulary, style, and domain differences continue to hinder cross-project generalization. While transformer-based models such as CodeBERT [55] and RoBERTa fine-tuning [63] show improved transfer performance, systematic evaluations of domain adaptation, fine-tuning strategies, and zero-shot transfer remain scarce.

#### 2) SCARCITY OF LARGE, DIVERSE, AND PUBLICLY AVAILABLE DATASETS

SATD detection research suffers from a persistent shortage of high-quality, labeled data. Most studies rely on a few canonical datasets, including those by Maldonado et al. [3], Zampetti et al. [8], and Maipradit et al. [9]. This narrow data pool constrains reproducibility and limits the training potential of data-hungry deep learning models such as CNN [44] and BiLSTM [50]. Recent efforts to create larger or domain-specific datasets, such as Sutoyo and Capiluppi's SATDAUG corpus [64], represent an important step but still fall short of the scale needed for robust benchmarking. The absence of multilingual and multi-domain data further limits the field's inclusivity and cross-context applicability.

#### 3) TOWARD A UNIFIED SATD BENCHMARK DATASET

To address the lack of standardized benchmarks in SATD research, we advocate for a unified, context-aware SATD dataset structure grounded in real project artifacts. Building upon insights derived from widely used datasets such as that of Maldonado et al., we outline an extended representation in which each SATD instance is explicitly linked to its surrounding source code within the original projects. Each record may include the project identifier, raw comment, line of code, surrounding code context, enclosing code element (e.g., class, method), and fine-grained SATD type (e.g., implementation, documentation). Such a structure enables multi-modal analysis that combines textual intent with structural program context, supporting more robust evaluation of modern transformer-based and hybrid SATD detection models. This standardized representation can serve as a practical blueprint for future community-driven SATD benchmarks.

#### 4) CLASS IMBALANCE AND UNDERREPRESENTED CATEGORIES

Nearly all reviewed studies report severe class imbalance, where SATD comments represent less than 10% of available data [9], [49], [53]. While techniques such as oversampling, cost-sensitive learning, and balanced loss functions have been introduced, most works continue to rely on random undersampling or threshold tuning. The imbalance not only reduces recall for minority classes but also affects the detection of nuanced SATD types such as requirement, design, or test debt [6], [52]. Addressing this imbalance through semi-supervised learning, data augmentation, or adaptive sampling remains a critical need.

#### 5) INTERPRETABILITY AND EXPLAINABILITY LIMITATIONS

As deep learning models become dominant [44], [60], [62], understanding their internal decision processes becomes increasingly important. Most DL and transformer-based studies—such as those using CNN-BiLSTM [44] or BERT [11]—achieve high predictive accuracy but provide little insight into which linguistic or semantic cues drive predictions. This "opaque model" nature limits their trustworthiness and practical deployment in industrial contexts. Only a handful of recent works, such as Tsoukalas et al. [65], have explicitly explored explainable AI (XAI) mechanisms for TD identification, indicating an underdeveloped yet promising research direction.

#### 6) LACK OF LONGITUDINAL AND EVOLUTIONARY ANALYSES

Another underexplored area concerns the temporal evolution of SATD. Most existing research focuses on static detection rather than studying how debt-related comments evolve, propagate, or are repaid over time [8] [42]. Only a few studies have examined longitudinal behavior [66], despite the potential of such analyses for debt prioritization and lifecycle management. Understanding SATD evolution could lead to predictive frameworks for debt accumulation and repayment trends—currently a major blind spot in the field.

#### 7) INCONSISTENT EVALUATION PROTOCOLS AND REPRODUCIBILITY

Evaluation practices across SATD studies remain fragmented. Many papers use distinct train–test splits, sampling ratios, or evaluation metrics, making it difficult to compare results across approaches [3], [58], [60]. The lack of standard benchmarks and unified protocols has led to inflated performance claims that are often non-reproducible. Although efforts like SATDAUG [64] and the adoption of F1, MCC, and

**TABLE 4.** Evaluation metrics reported across SATD detection studies (2014–2025).

| Metric Category | Metric | Representative Studies |
|---|---|---|
| **Core Classification Metrics** | F1-score | Maldonado et al. [3], Wattanakriengkrai et al. [50], Maipradit et al. [9], Sridharan et al. [51], Yu et al. [52], Zhu et al. [44], Skryseth et al. [11], Russo et al. [55], Xu et al. [57] |
| | Precision | Maldonado et al. [3], Liu et al. [4], Yu et al. [52], Chowdary et al. [58], Li et al. [59] |
| | Recall | Wattanakriengkrai et al. [50], Maipradit et al. [9], Yu et al. [52], Sharma et al. [46], Skryseth et al. [11] |
| **Extended Classification Metrics** | Accuracy | Zhu et al. [44], Chen et al. [53], Kaur and Kaur [45], Li et al. [59] |
| | AUC (ROC) | Patcas [54], Russo et al. [55], Skryseth et al. [11] |
| | MCC | Patcas [54], Russo et al. [55] |
| | Statistical Significance (Friedman Test) | Russo et al. [55] |
| **Regression / Forecasting Metrics** | MAE (Mean Absolute Error) | Tsoukalas et al. [10], Alhefdhi et al. [56] |
| | MAPE (Mean Absolute Percentage Error) | Alhefdhi et al. [56], Tsoukalas et al. [10] |
| **Specialized or Generative Metrics** | BLEU, CrystalBLEU, LEMOD, EM (Exact Match) | Russo et al. [55], Xu et al. [57] |

AUC metrics have improved reporting consistency, broader community consensus on experimental design, data sharing, and statistical validation remains urgently needed.

### 8) EMERGING RESEARCH DIRECTIONS

Building on the identified gaps, several promising directions emerge for advancing SATD detection research toward greater maturity and practical relevance:

- **Cross-project and transfer learning.** Future studies should systematically investigate transfer and domain adaptation strategies—such as parameter-efficient fine-tuning of pretrained models (e.g., CodeBERT [55] and RoBERTa [63])—to improve robustness and generalization across heterogeneous software projects and domains.
- **Creation of large, multilingual, and multi-domain benchmarks.** The development of open, standardized, and multilingual SATD datasets—extending beyond English and single-project settings—is essential for reproducible experimentation and fair comparison of emerging approaches [9], [64].
- **Explainable and transparent SATD detection.** Integrating explainable AI (XAI) techniques, such as SHAP or LIME [65], can enhance model transparency by exposing the linguistic and structural cues driving predictions, thereby increasing developer trust and industrial applicability.
- **Systematic handling of class imbalance.** Addressing severe label skew through adaptive sampling, synthetic data generation, or self-training mechanisms remains critical for improving recall and stability across minority SATD categories [49], [53].
- **Longitudinal SATD lifecycle analysis.** Extending prior work on SATD evolution [8], [66] to model how debt is introduced, transformed, and repaid over time could enable predictive maintenance and proactive debt management.
- **Hybrid and multi-modal detection frameworks.** Combining rule-based filtering with ML/DL pipelines [14], [45] and jointly modeling textual and structural signals from code and comments offers a balanced trade-off between interpretability and predictive performance.
- **Leveraging large language models (LLMs).** Exploring general-purpose and open LLMs (e.g., ChatGPT [62], Qwen, LLaMA) for few-shot or zero-shot SATD detection presents a promising direction, particularly for low-resource or rapidly evolving projects.
- **Standardized evaluation protocols.** Establishing community-wide benchmarks with unified dataset splits and agreed-upon metrics (e.g., F1, AUC, MCC) is necessary to ensure meaningful cross-study comparison and reproducibility [59], [60].

Taken together, these directions indicate that many persistent challenges in SATD detection stem not solely from algorithmic limitations, but from the fragmented treatment of data, evaluation protocols, and deployment contexts across studies. Much of the existing literature optimizes models within isolated experimental settings—often constrained to a single project or dataset—without fully accounting for the socio-technical diversity of real-world software systems. Consequently, advances in modeling capacity have frequently outpaced progress in generalizability, interpretability, and reproducibility. Addressing these shortcomings requires a shift from model-centric innovation toward an

ecosystem-oriented perspective, where shared benchmarks, transparent evaluation practices, and explainable decision mechanisms are treated as first-class research objectives. Such a shift would enable SATD detection to evolve from a laboratory-scale classification task into a trustworthy and practical component of modern software maintenance workflows.

#### 9) SUMMARY

In summary, although the past decade has witnessed substantial methodological progress—from early feature-based classifiers [3] to transformer- and LLM-driven architectures [11], [60], [62]—SATD research continues to face challenges related to generalization, interpretability, and reproducibility. The next phase of research should prioritize transparency, scalability, and open science through explainable models, shared datasets, and standardized evaluation frameworks. By addressing these dimensions collectively, future work can move beyond isolated performance improvements toward an integrated, reliable, and industry-ready paradigm for technical debt management.

## V. MAIN FINDINGS

This systematic literature review consolidated evidence from forty-six primary studies published between 2014 and 2025, covering diverse methodologies for detecting, classifying, and managing Self-Admitted Technical Debt (SATD). The studies reveal how the field has evolved from early heuristic detection of debt-related comments toward semantically grounded, data-driven approaches supported by advanced language models. The key findings from this review are summarized below.

### A. PREDOMINANCE OF TEXTUAL REPRESENTATIONS WITH GRADUAL INTEGRATION OF CODE CONTEXT

The overwhelming majority of SATD detection models continue to rely on **textual representations of code comments** as their primary input modality. Early studies—such as Potdar and Shihab [1], Maldonado et al. [3], and Liu et al. [4]—demonstrated that lexical and syntactic features extracted from developer comments can effectively identify debt-indicating language. These works typically employed vector space representations such as TF–IDF and bag-of-words, sometimes enhanced with token filtering or n-gram weighting schemes.

As research progressed, distributed representations such as Word2Vec, GloVe, and FastText emerged to capture richer semantic relationships [5], [57]. More recently, transformer-based embeddings (e.g., BERT, CodeBERT [55], and RoBERTa) have been widely adopted [11], [62], [63]. These models have proven particularly effective at representing context-dependent SATD expressions, such as ''temporary fix'' or ''should refactor later,'' which often require semantic nuance beyond keyword detection.

However, an important trend since 2021 is the gradual incorporation of **structural and contextual information** from the surrounding code. Studies such as Zhu et al. [44], Ren et al. [14], and Ghammam et al. [67] have experimented with combining textual embeddings with structural representations derived from Abstract Syntax Trees (ASTs), dependency graphs, or code tokens. This multimodal direction suggests a shift toward viewing SATD not merely as a linguistic artifact but as a contextual signal embedded in both code and commentary.

### B. DIVERSE SOLVING APPROACHES WITH A CLEAR SHIFT TOWARD DEEP AND HYBRID MODELS

SATD detection has followed a methodological evolution similar to that of natural language processing (NLP) and software analytics more broadly. Earlier studies relied heavily on **traditional machine learning** techniques—SVM, Random Forest, Naïve Bayes, Logistic Regression, and XGBoost—to classify comments using engineered textual features [3], [9], [49], [53]. These models remain appealing for their interpretability and computational efficiency.

However, the past five years have witnessed a decisive transition toward **deep learning (DL)** architectures. CNN and BiLSTM networks [7], [44], [50], [52] have been widely applied to model sequential and contextual dependencies in comment text, outperforming traditional classifiers in recall and F1-score. Transformer-based approaches—including BERT, ALBERT, and DeBERTa variants [11], [60], [62]—represent the most recent frontier, offering contextual understanding through large-scale pretraining.

Between these extremes, **hybrid and ensemble models** have emerged as an effective compromise between interpretability and semantic expressiveness. Sabbah and Hanani [6], Liu et al. [4], and Li et al. [68] combined traditional features (TF–IDF, bag-of-words) with embedding-based features (Word2Vec, BERT) and ensemble strategies such as voting and stacking. These approaches deliver improved robustness, especially under data imbalance or cross-project variability. In contrast, clustering and regression-based methods [10], [65] provide complementary perspectives by focusing on pattern discovery and cross-project forecasting rather than pure classification.

### C. EVALUATION PRACTICES: DOMINANCE OF F1-SCORE AND GRADUAL METHODOLOGICAL MATURITY

Evaluation of SATD models has matured alongside methodological sophistication. Nearly all reviewed studies rely on **F1-score** as the primary performance measure, often accompanied by **Precision** and **Recall** [3], [9], [11], [50], [53], [57]. The F1-score effectively mitigates the distortions caused by extreme class imbalance, which remains a defining challenge in SATD datasets.

While early studies occasionally reported **Accuracy**, later works increasingly adopt metrics that provide more balanced evaluation. **AUC (Area Under the Curve)** and **MCC (Matthews Correlation Coefficient)** have been introduced in recent transformer-based or multi-task frameworks [59],

[60], signaling a methodological turn toward class-balanced, threshold-independent evaluation. A few studies even report statistical significance testing (e.g., the Friedman test) to validate comparative improvements [60].

In forecasting and prioritization contexts, regression-oriented measures such as **MAE** and **MAPE** have been used to assess numerical prediction quality [10], [61]. Overall, the community's gradual move toward comprehensive, multi-metric evaluation indicates a growing concern for reliability and reproducibility.

### D. DATASET LIMITATIONS, CLASS IMBALANCE, AND GENERALIZABILITY

Consistent with the gaps identified in RQ7, three interrelated issues—limited datasets, strong class imbalance, and weak generalization—remain the most persistent barriers. Most studies depend on small, manually annotated datasets derived from open-source Java projects [3], [8], [9]. Although larger datasets such as SATDAUG [64] have begun to appear, they remain relatively modest in size and diversity compared to general NLP benchmarks.

Class imbalance further compounds this limitation, with SATD comments often representing less than 10% of samples [49], [53]. Several works introduced oversampling or reweighting techniques [7], [52], yet large-scale comparative evaluations of imbalance mitigation remain scarce. The combination of limited data and imbalanced labels undermines model robustness, particularly in cross-project or multilingual scenarios. Without large, diverse, and balanced benchmarks, reproducibility and fair model comparison remain difficult to achieve.

### E. EXPLAINABILITY, PRACTICAL DEPLOYMENT, AND INDUSTRY READINESS

Although deep and transformer-based models have consistently improved performance, their **opaque model nature** limits interpretability and developer trust. Most studies report accuracy gains without explaining why specific comments are classified as debt. Tsoukalas et al. [65] initiated efforts toward explainable SATD detection, demonstrating how interpretable models can help practitioners understand the rationale behind predictions. Nonetheless, explainability remains an exception rather than the norm.

Equally underexplored is the issue of **practical deployment**. Only a handful of studies—such as Sala et al. [45] and Zampetti et al. [8]—proposed end-to-end pipelines suitable for integration into Continuous Integration (CI) environments. Scalability, runtime cost, and maintenance of SATD detection models are rarely addressed, despite their importance for real-world adoption. As SATD research matures, bridging the gap between academic experimentation and industrial applicability will be essential.

### F. SYNTHESIS AND OUTLOOK

In summary, SATD detection research has progressed from handcrafted feature extraction to semantically rich, learning-based models capable of nuanced understanding. Traditional ML methods offered interpretability and simplicity but were limited in context-awareness; deep and transformer-based models brought semantic depth at the expense of transparency and data efficiency. Meanwhile, the field continues to grapple with dataset scarcity, imbalance, and
reproducibility.

Collectively, the reviewed studies highlight a clear research trajectory: from surface-level text analysis toward holistic modeling that incorporates code structure, semantics, and developer intent. Future work should focus on developing large, multilingual, and multi-domain benchmarks; integrating explainable and transparent model designs; and exploring lightweight, transferable architectures that enable real-world deployment. Only by addressing these challenges can SATD detection evolve into a reliable, interpretable, and actionable component of modern software-quality management.

## VI. FUTURE DIRECTIONS

This section builds on the gaps identified in RQ7 by outlining concrete architectural, methodological, and deployment-oriented research directions beyond the scope of existing studies.

Although significant progress has been achieved in automating Self-Admitted Technical Debt (SATD) detection, our review reveals that current methods remain largely text-centric—focused predominantly on the linguistic surface of comments while neglecting the contextual and structural cues encoded in the surrounding code. Future research must move beyond isolated text classification to context-aware modeling that jointly interprets comments, source code, and development history. Such integration can enhance detection precision, interpretability, and practical utility in modern software pipelines.

### A. LEVERAGING SOURCE CODE CONTEXT FOR DISAMBIGUATION

One of the most persistent limitations in current SATD models is their inability to disambiguate comments that are ambiguous in isolation. Phrases such as "*temporary fix*" or "*needs refactoring*" may or may not represent technical debt depending on the complexity or criticality of the associated code. Without structural or semantic context, purely textual models risk misclassification.

Future research should therefore explore **joint modeling of comments and their associated code regions**. Transformer-based architectures such as CodeBERT [55], GraphCodeBERT [56], and CodeT5 can serve as a foundation for learning joint embeddings that capture both linguistic and programmatic semantics. By integrating information from function- and class-level embeddings, models could distinguish between genuine SATD and benign documentation, thereby reducing false positives. This multi-view understanding aligns with recent progress in code summarization and

bug localization, where contextual integration has proven transformative.

### B. EXPLOITING SURROUNDING COMMENTS FOR CONTEXTUAL CLARITY

SATD comments rarely exist in isolation; they are embedded within conversational threads of developer intent. Adjacent comments often clarify the purpose or scope of an identified issue—for example, a TODO note followed by an explanation or linked issue identifier. Future SATD detection pipelines should therefore incorporate **surrounding comment windows** (e.g., the previous and subsequent few comments) as part of the model input. This design would allow systems to learn local narrative patterns and discourse cues, improving their ability to differentiate between transient reminders, genuine self-admitted debt, and simple documentation. Such contextual modeling has already shown success in defect-tracking and commit-message analysis, and its adaptation to SATD detection represents a natural next step toward deeper semantic understanding.

### C. MODELING CODE COMMENT RELATIONSHIPS

To effectively link developer intent with implementation realities, SATD research must evolve toward explicit **code–comment relationship modeling**. Promising directions include:

- Extracting **Abstract Syntax Tree (AST)** representations of relevant code to capture syntactic and structural relationships.
- Embedding comments together with their associated code blocks using dual-encoder or cross-attention architectures.
- Employing **graph neural networks (GNNs)** to model dependencies between comments, code entities, and files, enabling the identification of multi-level debt propagation.

This relational modeling would enable SATD detection systems to reason about how comments relate to implementation complexity, refactoring opportunities, and software quality hotspots. Such approaches echo emerging trends in software representation learning, where graph- and code-aware embeddings significantly improve performance in bug detection, code completion, and vulnerability analysis.

### D. HARNESSING PRE-TRAINED LANGUAGE MODELS FOR CODE-AWARE SATD DETECTION

Recent transformer-based models pre-trained on code corpora—such as CodeBERT [55], GraphCodeBERT [56], CodeT5, and StarCoder—offer unprecedented representational power for capturing both syntactic and semantic characteristics of source code. Future research should investigate how these models can be effectively **fine-tuned jointly on code and comment pairs** to align the linguistic signals of SATD with their structural counterparts.

Potential directions include:

- Integrating **code summarization outputs as auxiliary signals** to provide interpretive explanations for SATD predictions.
- Leveraging **contrastive pretraining** on large-scale unlabeled code–comment pairs to enhance generalization in low-resource settings.
- Exploring **multi-task learning setups** where SATD detection is trained alongside tasks such as defect prediction, bug localization, or code smell detection.

Such research could unify the fragmented approaches seen in current SATD literature into a single, transferable modeling framework that captures the interplay between comment semantics and code intent.

### E. ENSURING EFFICIENCY, SCALABILITY, AND PRACTICAL DEPLOYMENT

While contextual integration promises higher accuracy, it also introduces substantial computational overhead. For SATD detection to become a practical component of industrial software analytics, **efficiency and scalability** must remain at the forefront of design. Future models should therefore aim to:

- Apply **contrastive and self-supervised learning** to pretrain lightweight representations, minimizing the need for extensive fine-tuning.
- Develop **hybrid detection frameworks** that combine rule-based preprocessing with learned models to filter out trivial or irrelevant comments efficiently.
- Implement **incremental embedding updates** for CI/CD environments, caching static representations to avoid redundant computation across code revisions.

Such optimizations would not only improve runtime efficiency but also facilitate integration with continuous integration pipelines and modern DevOps workflows, a gap still largely unaddressed in the reviewed literature.

### F. TOWARD EXPLAINABLE AND TRUSTWORTHY SATD DETECTION

As SATD detection models continue to grow in sophistication, interpretability will become increasingly essential for their practical adoption. Developers are unlikely to rely on systems whose decisions they cannot understand or validate. Building on early explainability efforts by Tsoukalas et al. [65], future research should integrate **explainable AI (XAI)** techniques—such as SHAP, saliency maps, or attention-weight visualization—to clarify the rationale behind model predictions.

Making the reasoning process visible would allow practitioners to trace which textual cues or contextual signals influenced a classification, enabling both trust and accountability. Ultimately, progress in SATD detection will not depend on accuracy alone, but on the ability to provide transparent, interpretable, and developer-aligned explanations that support informed decision-making in real-world development environments.

**TABLE 5. Summary of the reviewed papers.**

| ID | Authors | Year | Title | Representation | Approach | Input Data | Metric |
|---|---|---|---|---|---|---|---|
| 1 | Zampetti, Fiorella and Serebrenik, Alexander and Di Penta, Massimiliano | 2020 | Automatically learning patterns for self-admitted technical debt | Word2Vec, CNN, RNN | softmax layer | Code comments, Source code | F1, AUC, Precision, Recall, MCC |
| 2 | Maipradit, Rungroj and Lin, Bin and Nagy, Csaba and Bavota, Gabriele | 2020 | Automated identification of on-hold self-admitted technical debt | VSM | softmax layer | Code comments | Precision, Recall, F1 |
| 3 | Sridharan, Murali and Mantyla, Mika and Rantala, Leevi and Claes, Maëlick | 2021 | Data balancing improves self-admitted technical debt detection | VSM | softmax layer | Code comments | F1 |
| 4 | Zhu, Kuiyu and Yin, Ming and Li, Yizhen | 2021 | Detecting and classifying SATD using CNN-BiLSTM | Word2Vec, CNN-BiLSTM | Deep learning (softmax) | Code comments | Accuracy, F1 |
| 5 | Tsoukalas, Dimitrios and Mathioudaki, Maria and Siavvas, Miltiadis | 2021 | Cross-project technical debt forecasting | Feature based | Clustering + regression | Source code | MAPE |
| 6 | Yu, Dongjin and Wang, Lin and Chen, Xin and Chen, Jie | 2021 | BiLSTM with attention for SATD detection | GloVe | Bi-LSTM | Code comments | Precision, Recall, F1 |
| 7 | Kaur, Inderpreet and Kaur, Arvinder | 2021 | Ensemble feature selection for SATD identification | AST features | Random forest | Source code | F1 |
| 8 | Sala, Irene and Tommasel, Antonela and Arcelli Fontana, Francesca | 2021 | DebtHunter: A ML-based approach for detecting SATD | AST + text | SMO (SVM) | Code comments | F1, Precision, Recall |
| 9 | Aversano, Lerina and Bernardi, Mario Luca and Cimitile, Marta | 2021 | SATD prediction through temporal convolutional network | Historical commit data | Regression | Source code history | MAE |
| 10 | Tsoukalas, Dimitrios and Mittas, Nikolaos | 2021 | Machine learning for technical debt identification | Feature based | RF, SVM | Source code metrics | F1 |
| 11 | Yu, Zhe and Fahid, Fahmid Morshed and Tu, Huy and Menzies, Tim | 2022 | Jitterbug: identifying self-admitted technical debts | VSM | Rule-based + ML | Code comments | F1, Recall, Precision |
| 12 | Xiao, Tao and Wang, Dong and McIntosh, Shane and Hata, Hideaki | 2022 | Characterizing SATD in build systems | TF-IDF | SVM | Build script comments | F1 |
| 13 | Sharma, Rishab and Shahbazi, Ramin and Fard, Fatemeh H. | 2022 | SATD in R: detection and causes | ALBERT, RoBERTa | CNN + transformer | Code comments | F1 |
| 14 | Li, Yichen and Zhang, Xiaofang | 2022 | Multi-label code smell detection with GCN-LSTM | GCN + LSTM | Fusion model | Source code | F1 |
| 15 | Alhefdhi, Abdulaziz and Dam, Hoa Khanh and Nugroho, Yusuf S. | 2022 | Conditional statement technical debt detection | VSM | LSTM | Code segments | F1 |
| 16 | Chen, Xin and Yu, Dongjin and Fan, Xulin and Wang, Lin | 2022 | Multiclass SATD classification with XGBoost | VSM | XGBoost | Code comments | Macro F1 |
| 17 | Qu, Yubin and Bao, Tie and Yuan, Meng | 2023 | Deep learning-based SATD detection using TextCNN | TextCNN + GloVe | MaxEnt classifier | Code comments | F1 |
| 18 | Sabbah, Ahmed F. and Hanani, Abualsoud A. | 2023 | Ensemble SATD classifiers | TF-IDF, Word2Vec, BERT | RF, SVM, SL-CNN | Code comments, commits | F1 |
| 19 | Ni, Chao and Yang, Kaiwen and Zhu, Yan and Chen, Xiang | 2023 | Unifying defect prediction and SATD classification | AST + metadata | Multi-classifier | Source code + issue data | F1 |
| 20 | Chowdary, Satya Mohan and Kumar, Prasanna | 2023 | Prioritizing SATD via SVM | VSM | SVM ranking | Comments | Accuracy, F1 |
| 21 | Skryseth, Daniel and Shivashankar, Karthik and Pilán, Ildikó | 2023 | Technical debt classification in issue trackers using BERT | BERT | DeBERTaV3 | Issue tracker comments | F1 |
| 22 | Li, Yikun and Soliman, Mohamed and Avgeriou, Paris | 2023 | TextCNN for SATD in Python repositories | TextCNN | Random classifier | Pull requests, issues | F1 |

### G. SUMMARY

The next decade of SATD research will likely see a paradigm shift—from surface-level text classification toward holistic, context-enriched, and explainable analysis that unites comment semantics, code structure, and developer intent. By integrating contextual understanding with scalable, transparent, and code-aware modeling, the field can move closer to its ultimate goal: building intelligent systems that not only identify self-admitted technical debt but also assist developers in managing, prioritizing, and repaying it effectively within the lifecycle of modern software projects.

## VII. THREATS TO VALIDITY

To ensure the rigor of this systematic literature review (SLR) on self-admitted technical debt (SATD) detection, we identify and reflect on potential threats to validity based on the guidelines of systematic reviews in software engineering. We categorize these threats into *construct validity*, *internal validity*, *external validity*, and *reliability*.

### A. CONSTRUCT VALIDITY

Construct validity concerns whether the measures and data used in the review accurately reflect the concepts under investigation. One threat relates to the **selection of search keywords and strings**, which may have excluded relevant papers that use different terminologies for SATD or debt detection. Although we designed comprehensive keyword groups (e.g., "technical debt," "self-admitted technical debt," "SATD detection," "machine learning," "deep learning," "source code comments"), there remains the possibility of missing studies using alternative phrasing.

Another potential threat arises from the **inclusion and exclusion criteria**. Papers focusing on general technical debt management without explicit SATD detection or lacking

**TABLE 5.** *(Continued.)* **Summary of the reviewed papers.**

| ID | Authors | Year | Title | Representation | Approach | Input Data | Metric |
|---|---|---|---|---|---|---|---|
| 23 | Nanadani, Himesh and Saad, Mootez and Sharma, Tushar | 2023 | CodeBERT and CodeT5 for SATD detection | CodeBERT, CodeT5 | Transfer learning | Source code | Precision, Recall, F1 |
| 24 | Li, Yikun and Soliman, Mohamed and Avgeriou, Paris | 2023 | Relation classification of SATD comments | BERT | Fine-tuned BERT | Comments, commits | Precision, Recall, F1 |
| 25 | Shukla, Suyash and Kumar, Sandeep | 2023 | Bug prediction using SATD information | Feature based | SVM | Issue text | Accuracy, F1 |
| 26 | Maldonado, Everton and Shihab, Emad and Tsantalis, Nikolaos | 2017 | NLP-based SATD detection in open-source software | VSM | MaxEnt, SVM, Bayes | Code comments | F1 |
| 27 | Liu, Zhongxin and Huang, Qiao and Xia, Xin and Shihab, Emad | 2018 | SATD Detector: a text-mining approach | VSM | NB, Voting, SVM, KNN | Code comments | F1 |
| 28 | Wattanakriengkrai, Supatsara and Maipradit, Rungroj | 2018 | Design and requirement SATD identification | VSM | Auto-sklearn, RF | Requirement comments | Precision, Recall, F1 |
| 29 | Flisar, Jernej and Podgorelec, Vili | 2018 | Word2Vec for enhanced SATD feature selection | Word2Vec | SVM | Code comments | F1 |
| 30 | Ren, Xiaoxue and Xing, Zhenchang and Xia, Xin and Lo, David | 2019 | Neural network-based SATD detection | Word embeddings + CNN | CNN | Code comments | Precision, Recall, F1 |
| 31 | H. Wang, Z. Gao, et al. | 2024 | Just-In-Time TODO-Missed Commits Detection | CodeBERT | Dual-encoder CodeBERT | Code + commits | F1, Precision, Recall |
| 32 | R.-D. Patcas | 2024 | Technical Debt Prediction | AST | Extremely Randomized Trees | Code metrics | Precision, Recall, F1 |
| 33 | D. Tsoukalas, N. Mittas, A. Chatzigeorgiou, A. Ampatzoglou | 2024 | Local and Global Explainability for TD Identification | Feature based | Binary classifiers | Code metrics | Precision, Recall |
| 34 | H. Gu, J. Liu, et al. | 2024 | How Far Are We? LLMs for SATD detection | BERT | MT-BERT-SATD | Code comments, PRs, issues | F1, Precision, Recall |
| 35 | A. Alhefdhi, et al. | 2024 | Towards automating SATD repayment | seq2seq neural network | SATD repayment automation | Code + comments | EM, BLEU, LEMOD |
| 36 | M. S. Sheikhaei, Y. Tian, S. Wang, B. Xu | 2024 | LLMs vs small DL models for SATD | Flan-T5 | Comparison (ChatGPT vs DL) | Code comments | F1 |
| 37 | J. Yu, et al. | 2024 | Detecting SATD via prompt-based method | RoBERTa | LLM prompt-based | Issue tracker text | F1, Precision, Recall |
| 38 | A. Ghammam, D. E. Rzig, M. Almukhtar, et al. | 2025 | Build Code Needs Maintenance Too | Prompt-based LLM | Rule-based | Commits, code diffs | Precision, Recall, F1 |
| 39 | B. Russo, J. Melegati, M. Mock | 2025 | Multi-task learning for SATD and vulnerability | CodeBERT | MTL-BERT | Code + comments | Precision, Recall, F1 |
| 40 | R. A. Halepmollasi, et al. | 2025 | Predicting vulnerable code changes via TD indicators | Feature based | RF, XGBoost | Commits, TD metrics | Precision, Recall, F1 |
| 41 | Li, J.; Li, L.; Liu, J.; Yu, X.; Liu, X.; Keung, J.W. | 2024 | ChatGPT vs deep learning for SATD detection | BERT, Word2Vec | Comparison (LLM vs DL fusion) | Code comments | Recall (ChatGPT higher) |
| 42 | Bagane, Pooja et al. | 2025 | Automatic detection of TD in large-scale Java | CNN + BiLSTM + attention | Multi-model ensemble | Code + comments | Accuracy, F1 |
| 43 | V. K. Njeru, J. Kuria and B. Kituku. | 2025 | Hybrid BERT, CNN-BiLSTM for SATD | BERT + CNN-BiLSTM | Hybrid DL model | Code comments | F1, Accuracy |
| 44 | O. Khanvilkar, M. W. Mkaouer, E. A. AlOmar, A. ElSaid, A. Chaaben | 2025 | Automated identification of ML technical debt | Feature based | Multi-label classifier | Code comments | F1, Precision, Recall |
| 45 | Shahzeidi, Maryam; Mollahoseini Ardakani, Mohammadreza; Javdani, ... | 2025 | Hybrid LSTM-QPSO model for SATD | BERT + LSTM | QPSO-tuned LSTM | Code comments | F1 |
| 46 | Xu, Yihang and Yu, Dongjin and Chen, Xin and Yang, Quanxin | 2025 | Fine-tuning RoBERTa with multi-task learning for SATD | RoBERTa | Fine-tuned MTL-RoBERTa | Code comments | Precision, Recall, F1 |

empirical evaluation were excluded, which may have omitted potentially relevant conceptual discussions that could inform future SATD detection approaches.

### B. INTERNAL VALIDITY

Internal validity refers to the soundness of the review process itself. To mitigate bias in paper selection, we followed a systematic selection workflow, including independent screening and validation by multiple reviewers where possible. However, there may still be subjectivity in interpreting the relevance of papers, particularly for borderline cases where the SATD detection was part of a broader technical debt management pipeline.

Additionally, while data extraction from papers (e.g., representation techniques, solving approaches, evaluation metrics) was systematically logged, there remains the possibility of **misinterpretation of terminology** due to varied phrasing across studies. For example, the use of ‘‘labels’’ and ‘‘categories’’ may differ across papers, potentially affecting the extraction of evaluation metrics and dataset usage.

### C. EXTERNAL VALIDITY

External validity concerns the generalizability of our findings to the broader research community. Our SLR primarily covers papers published in recognized journals and conferences in software engineering and empirical software engineering, which may introduce a publication bias, excluding relevant gray literature (e.g., technical reports, preprints).

Furthermore, our focus on papers using empirical SATD detection with ML and DL may limit the generalizability of

findings to approaches using alternative paradigms (e.g., pure static analysis, developer-centric methods).

### D. RELIABILITY

Reliability pertains to the repeatability of the review process. We have documented our search strategy, selection criteria, data extraction approach, and categorization systematically, which increases the reproducibility of this study. The use of structured data sheets for logging paper IDs, authors, year, techniques, and results allows other researchers to replicate our analysis under similar conditions.

However, the inherent subjectivity in categorizing ''solving approaches'' (e.g., distinguishing hybrid methods from classical ML) and interpreting evaluation metrics across varied studies may affect reliability. Additionally, paper availability and access constraints may influence replication if some papers become inaccessible over time.

### E. MITIGATION STRATEGIES

To address these threats:

- We designed **comprehensive keyword sets** and pilot searches to refine search strategies.
- We clearly defined and adhered to **inclusion and exclusion criteria** with systematic screening.
- We maintained detailed **extraction sheets** with cross-checking for consistency.
- We included a wide range of publication venues to reduce publication bias.
- We explicitly documented potential gaps and future directions in the findings to contextualize the limitations.

## VIII. CONCLUSION

This review set out to map and consolidate a decade of research on the automated detection of Self-Admitted Technical Debt (SATD)—a phenomenon that lies at the intersection of software engineering and natural language understanding. By synthesizing evidence from forty-six primary studies published between 2015 and 2025, we traced how the field has evolved from handcrafted feature extraction and keyword heuristics toward deep, contextual, and increasingly explainable models that capture developer intent and code semantics.

Our analysis reveals a discipline in rapid transition. Traditional machine learning approaches established the foundation for SATD detection, but the last five years have witnessed the rise of transformer-based architectures that unify text and code representations. At the same time, the research community has begun to confront deeper methodological issues—data scarcity, class imbalance, and limited cross-project generalization—while exploring emerging paradigms such as multi-task learning, prompt-based modeling, and large language model (LLM) adaptation.

The implications are twofold. Scientifically, SATD research now stands as a mature, data-driven subfield of empirical software engineering, with clear trajectories toward multimodal reasoning, interpretability, and continuous integration. Practically, automated SATD detection is moving from research prototypes to deployable tools capable of supporting real-time maintainability decisions in industrial codebases.

Looking ahead, the next frontier lies in **holistic, explainable, and context-aware SATD systems** that fuse linguistic, structural, and historical cues to reason about technical debt as a living entity within evolving software ecosystems. Achieving this vision will require not only algorithmic innovation but also open, shared benchmarks and transparent evaluation practices. By pursuing these directions, the community can transform SATD detection from an analytical challenge into a cornerstone of intelligent, sustainable software quality management.

Despite these identified threats, we believe that our systematic approach, transparency in methodology, and critical reflection on limitations contribute to the rigor and trustworthiness of this SLR. Future replications and extensions can build on this review by incorporating additional gray literature, expanding to non-English studies, and leveraging automated systematic mapping techniques to reduce manual biases.

## IX. FULL TABLE OF REVIEWED PAPERS

See Table 5

## REFERENCES

[1] A. Potdar and E. Shihab, ''An exploratory study on self-admitted technical debt,'' in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 91–100.

[2] E. D. S. Maldonado and E. Shihab, ''Detecting and quantifying different types of self-admitted technical debt,'' in *Proc. IEEE 7th Int. Workshop Manag. Tech. Debt (MTD)*, Oct. 2015, pp. 9–15.

[3] E. D. S. Maldonado, E. Shihab, and N. Tsantalis, ''Using natural language processing to automatically detect self-admitted technical debt,'' in *Proc. Int. Conf. Softw. Maintenance Evol. (ICSME)*, vol. 43, 2017, pp. 1044–1062.

[4] Z. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, ''SATD detector: A text-mining-based self-admitted technical debt detection tool,'' in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 9–12, doi: 10.1145/3183440.3183478.

[5] J. Flisar and V. Podgorelec, ''Enhanced feature selection using word embeddings for self-admitted technical debt identification,'' in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2018, pp. 230–233, doi: 10.1109/SEAA.2018.00045.

[6] A. F. Sabbah and A. A. Hanani, ''Self-admitted technical debt classification using natural language processing word embeddings,'' *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 13, no. 2, p. 2142, Apr. 2023.

[7] Y. Qu, T. Bao, M. Yuan, and L. Li, ''Deep learning-based self-admitted technical debt detection empirical research,'' *J. Internet Technol.*, vol. 24, no. 4, pp. 975–987, 2023.

[8] F. Zampetti, A. Serebrenik, and M. Di Penta, ''Automatically learning patterns for self-admitted technical debt removal,'' in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 355–366.

[9] R. Maipradit, B. Lin, C. Nagy, G. Bavota, M. Lanza, H. Hata, and K. Matsumoto, ''Automated identification of on-hold self-admitted technical debt,'' in *Proc. IEEE 20th Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2020, pp. 54–64.

[10] D. Tsoukalas, M. Mathioudaki, M. Siavvas, D. Kehagias, and A. Chatzigeorgiou, ''A clustering approach towards cross-project technical debt forecasting,'' *Social Netw. Comput. Sci.*, vol. 2, no. 1, p. 22, Feb. 2021, doi: 10.1007/s42979-020-00408-4.

[11] D. Skryseth, K. Shivashankar, I. Pilán, and A. Martini, "Technical debt classification in issue trackers using natural language processing based on transformers," in *Proc. ACM/IEEE Int. Conf. Tech. Debt (TechDebt)*, May 2023, pp. 92–101.

[12] L. Aversano, M. L. Bernardi, M. Cimitile, and M. Iammarino, "Technical debt predictive model through temporal convolutional network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.

[13] W. Cunningham, "The WyCash portfolio management system," in *Proc. Object-Oriented Program., Syst., Languages, Appl. (OOPSLA) Conf.*, 1992, pp. 29–30.

[14] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network-based detection of self-admitted technical debt: From performance to explainability," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 3, pp. 1–46, 2019, doi: 10.1145/3324916.

[15] D. Tsoukalas, N. Mittas, A. Chatzigeorgiou, D. D. Kehagias, A. Ampatzoglou, T. Amanatidis, and L. Angelis, "Machine learning for technical debt identification," in *Proc. Int. Conf. Quality Inf. Commun. Technol.*, 2021, pp. 1–14.

[16] M. Sridharan, M. Mäntylä, L. Rantala, and M. Claes, "Data balancing improves self-admitted technical debt detection," *Empirical Softw. Eng.*, vol. 26, no. 5, pp. 1–25, 2021.

[17] X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy, "Neural network-based detection of self-admitted technical debt: From performance to explainability," in *Proc. 26th Asia–Pacific Softw. Eng. Conf.*, 2019, pp. 218–225.

[18] J. Flisar and V. Podgorelec, "Enhanced feature selection using word embeddings for self-admitted technical debt identification," *Inf. Softw. Technol.*, vol. 102, pp. 1–13, Jun. 2018.

[19] Y. Li, M. Soliman, and P. Avgeriou, "Automatic identification of self-admitted technical debt from four different sources," *Empirical Softw. Eng.*, vol. 28, no. 3, p. 65, May 2023.

[20] J. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li, "Is using deep learning frameworks free? Characterizing technical debt in deep learning frameworks," in *Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., Softw. Eng. Soc. (ICSE-SEIS)*, Oct. 2020, pp. 1–10.

[21] Y. Li and X. Zhang, "Multiclass classification for self-admitted technical debt detection," *Appl. Sci.*, vol. 12, no. 14, p. 7075, 2022.

[22] T. Xiao, D. Wang, S. McIntosh, H. Hata, R. G. Kula, T. Ishio, and K. Matsumoto, "Characterizing and mitigating self-admitted technical debt in build systems," *IEEE Trans. Softw. Eng.*, vol. 48, no. 10, pp. 4214–4228, Oct. 2021.

[23] Y. Xu, D. Yu, X. Chen, Q. Yang, S. Wang, and W. Yan, "Leveraging multi-task learning to fine-tune roberta for self-admitted technical debt identification and classification," *J. Syst. Softw.*, vol. 231, Jan. 2026, Art. no. 112629.

[24] B. Russo, M. Camilli, and M. Mock, "WeakSATD: Detecting weak self-admitted technical debt," in *Proc. IEEE/ACM 19th Int. Conf. Mining Softw. Repositories (MSR)*, May 2022, pp. 448–453.

[25] K. Zhu, M. Yin, and Y. Li, "Deep modeling of self-admitted technical debt using CNN-bilstm," in *Proc. Int. Conf. Softw. Maintenance Evol. (ICSME)*, Jun. 2021, pp. 1–12.

[26] D. Tsoukalas, M. Mathioudaki, N. Mitsas, and A. Ampatzoglou, "A clustering approach towards cross-project technical debt classification," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, Aug. 2021, pp. 1–10.

[27] I. Sala, A. Tommasel, and F. A. Fontana, "Debthunter: A machine learning-based approach to detect self-admitted technical debt," in *Proc. Int. Conf. Mining Softw. Repositories (MSR)*, Aug. 2021, pp. 1–12.

[28] T. Xiao, D. Wang, S. McIntosh, and A. E. Hassan, "Characterizing and mitigating self-admitted technical debt in source code," in *Proc. Int. Conf. Softw. Maintenance Evol. (ICSME)*, Jul. 2022, pp. 1–11.

[29] I. Kaur and A. Kaur, "A novel four-way approach designed with ensemble learning to detect code smells and technical debt," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Jul. 2021, pp. 1–10.

[30] D. Tsoukalas, N. Mittas, A. Ampatzoglou, and I. Stamelos, "Machine learning for technical debt identification," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2021, pp. 1–10.

[31] L. Aversano, M. L. Bernardi, M. D. Penta, and V. Guardia, "Technical debt predictive model through temporal features," in *Proc. Int. Conf. Softw. Maintenance Evol. (ICSME)*, Jul. 2021, pp. 1–10.

[32] C. Ni, K. Yang, Y. Zhu, X. Chen, and Y. Z. Wang, "Unifying defect prediction, categorization, and technical debt detection using multi-view learning," *J. Syst. Softw.*, vol. 205, Mar. 2023, Art. no. 111691.

[33] Y. Li, M. Soliman, and P. Avgeriou, "Automatic identification of self-admitted technical debt from four different sources," in *Proc. Int. Conf. Tech. Debt*, 2023.

[34] J. Li, L. Li, J. Liu, X. Yu, X. Liu, and J. W. Keung, "Large language model chatgpt versus small deep learning models in self-admitted technical debt detection," *Appl. Sci.*, vol. 14, no. 6, p. 2591, 2024.

[35] M. S. Sheikhaei, Y. Tian, S. Wang, and B. Xu, "An empirical study on the effectiveness of large language models for self-admitted technical debt detection," *IEEE Access*, vol. 12, pp. 105210–105224, 2024.

[36] H. Gu and J. Liu, "Self-admitted technical debt: A deep study," in *Proc. Int. Conf. Softw. Anal., Evol. Reeng.*, Jun. 2024, pp. 1–12.

[37] H. Wang, Z. Gao, X. Hu, D. Lo, J. Grundy, and X. Wang, "Just-in-time TODO-missed commits detection," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, vol. 50, Oct. 2024, pp. 2732–2752.

[38] J. Li, L. Li, J. Liu, X. Yu, X. Liu, and J. W. Keung, "Advancing self-admitted technical debt identification with large-scale benchmarks," *Appl. Sci.*, vol. 14, no. 8, p. 3110, 2024.

[39] K. Lee et al., "Self-admitted technical debt detection in context-aware environments," in *Proc. 20th Int. Conf. Softw. Eng./Automated Softw. Eng.*, 2021.

[40] F. Zampetti, A. Serebrenik, and M. Di Penta, "Was self-admitted technical debt removal a real removal? An in-depth perspective," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories (MSR)*, May 2018, pp. 526–536.

[41] Y. Li, M. Soliman, and P. Avgeriou, "Automatically identifying relations between self-admitted technical debt across different sources," in *Proc. ACM/IEEE Int. Conf. Tech. Debt (TechDebt)*, May 2023, pp. 11–21.

[42] E. da Silva Maldonado, E. Shihab, and N. Tsantalis, "An empirical study on the removal of self-admitted technical debt," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3770–3798, 2020.

[43] P. Shukla et al., "Test self-admitted technical debt: Detecting and managing test-related SATD," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Jul. 2023, pp. 1–10.

[44] K. Zhu, M. Yin, and Y. Li, "Detecting and classifying self-admitted technical debt with CNN-BiLSTM," in *Proc. Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Jun. 2021, pp. 1–18.

[45] I. Sala, A. Tommasel, and F. Arcelli Fontana, "DebtHunter: A machine learning-based approach for detecting self-admitted technical debt," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, Jun. 2021, pp. 278–283.

[46] C. Ni, K. Yang, Y. Zhu, X. Chen, and X. Yang, "Unifying defect prediction, categorization, and repair by multi-task deep learning," in *Proc. 38th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2023, pp. 1980–1992.

[47] H. Nanadani, M. Saad, and T. Sharma, "Calibrating deep learning-based code smell detection using human feedback," in *Proc. IEEE 23rd Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Oct. 2023, pp. 37–48.

[48] Z. Yu, F. M. Fahid, H. Tu, and T. Menzies, "Identifying self-admitted technical debts with jitterbug: A two-step approach," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1676–1691, May 2022, doi: 10.1109/TSE.2020.3031401.

[49] G. S. M. Chowdary and R. P. Kumar, "Automated identification and prioritization of self-admitted technical debt using NLP word embeddings," in *Proc. Int. Conf. Self Sustain. Artif. Intell. Syst. (ICSSAS)*, Oct. 2023, pp. 963–971.

[50] D. Yu, L. Wang, X. Chen, and J. Chen, "Using BiLSTM with attention mechanism to automatically detect self-admitted technical debt," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, vol. 15, Jun. 2021, pp. 1–18.

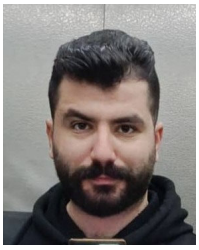
[51] I. Kaur and A. Kaur, "A novel four-way approach designed with ensemble feature selection for code smell detection," *IEEE Access*, vol. 9, pp. 8695–8707, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9358716/

[52] R. Sharma, R. Shahbazi, F. H. Fard, Z. Codabux, and M. Vidoni, "Self-admitted technical debt in R: Detection and causes," *Automated Softw. Eng.*, vol. 29, no. 2, p. 53, Nov. 2022, doi: 10.1007/s10515-022-00358-6.

[53] M. Sridharan, M. Mantyla, L. Rantala, and M. Claes, "Data balancing improves self-admitted technical debt detection," in *Proc. IEEE/ACM 18th Int. Conf. Mining Softw. Repositories (MSR)*, May 2021, pp. 358–368.

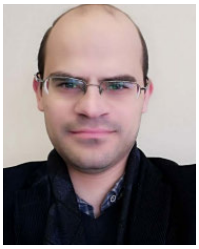
[54] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, May 2016, pp. 315–326.

[55] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Proc. Findings Assoc. Comput. Linguistics*, Jun. 2020, pp. 1536–1547.
[56] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. B. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "GraphCodeBERT: Pre-training code representations with data flow," in *Proc. Int. Conf. Learn. Represent.*, 2021.
[57] S. Wattanakriengkrai, R. Maipradit, H. Hata, M. Choetkiertikul, T. Sunetnanta, and K. Matsumoto, "Identifying design and requirement self-admitted technical debt using N-gram IDF," in *Proc. 9th Int. Workshop Empirical Softw. Eng. Pract. (IWESEP)*, Dec. 2018, pp. 7–12, doi: 10.1109/IWESEP.2018.00010.
[58] L.-P. Chen, "Network-based discriminant analysis for multiclassification," *J. Classification*, vol. 39, no. 3, pp. 410–431, Nov. 2022.
[59] R.-D. Patcas, "Technical debt prediction," in *Proc. 26th Int. Symp. Symbolic Numeric Algorithms Sci. Comput. (SYNASC)*, Sep. 2024, pp. 99–106.
[60] B. Russo, J. Melegati, and M. Mock, "Leveraging multi-task learning to improve the detection of SATD and vulnerability," in *Proc. IEEE/ACM 33rd Int. Conf. Program Comprehension (ICPC)*, Apr. 2025, pp. 1–12.
[61] A. Alhefdhi, H. K. Dam, Y. S. Nugroho, H. Hata, T. Ishio, and A. Ghose, "A framework for conditional statement technical debt identification and description," *Automated Softw. Eng.*, vol. 29, no. 2, p. 60, Nov. 2022.
[62] J. Li, L. Li, J. Liu, X. Yu, X. Liu, and J. Keung, "Large language model ChatGPT versus small deep learning models for self-admitted technical debt detection: Why not together?" in *Proc. Int. Conf. Softw. Maintenance Evol. (ICSME)*, vol. 55, 2024, pp. 3–28.
[63] Y. Xu, D. Yu, X. Chen, Q. Yang, S. Wang, and W. Yan, "Leveraging multi-task learning to fine-tune roberta for self-admitted technical debt identification and classification," in *Proc. Int. Conf. Tech. Debt (TechDebt)*, vol. 231, Jun. 2025.
[64] E. Sutoyo and A. Capiluppi, "SATDAUG—A balanced and augmented dataset for detecting self-admitted technical debt," in *Proc. 21st Int. Conf. Mining Softw. Repositories*, Apr. 2024, pp. 289–293.
[65] D. Tsoukalas, N. Mittas, E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and D. Kehagias, "Local and global explainability for technical debt identification," in *Proc. Int. Conf. Tech. Debt (TechDebt)*, vol. 50, Feb. 2024, pp. 2110–2123.
[66] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, and A. Bacchelli, "When testing meets code review: Why and how developers review tests," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. (ICSE)*, May 2018, pp. 677–687.
[67] A. Ghammam, D. E. Rzig, M. Almukhtar, R. Khalsi, F. Hassan, and M. Kessentini, "Build code needs maintenance too: A study on refactoring and technical debt in build systems," in *Proc. IEEE/ACM 22nd Int. Conf. Mining Softw. Repositories (MSR)*, Apr. 2025, pp. 616–628.
[68] Y. Li and X. Zhang, "Multi-label code smell detection with hybrid model based on deep learning," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, vol. 2022, Jul. 2022, pp. 42–47.



**ANAS ABDULAZIZ** received the B.S. degree in computer and informatics engineering from Syrian Private University (SPU), Syria, in 2022. He is currently pursuing the M.S. degree in informatics engineering with the Higher Institute for Applied Sciences and Technology (HIAST), Syria.

He is a full-time AI Team Leader in the government sector in Syria, where he leads the development of machine intelligence systems and data-driven solutions. He also serves as a part-time Assistant Teacher with SPU, where he teaches courses related to artificial intelligence and software engineering. His research interests include natural language processing, machine learning, deep learning, software quality and maintainability (with a focus on self-admitted technical debt), OCR for Arabic documents, and retrieval-augmented generation systems.



**RIAD SONBOL** received the Ph.D. degree in artificial intelligence, specializing in deep learning and natural language processing (NLP).

He is currently an Assistant Professor of artificial intelligence with the Higher Institute for Applied Sciences and Technology (HIAST), Damascus University, and Syrian Private University (SPU), Syria. He also works on AI applications in software engineering, telecommunications, cybersecurity, and healthcare signal analysis. He supervises M.Sc. and Ph.D. students, reviews for multiple Scopus-indexed AI journals, and provides consultancy in AI and machine learning for industry and governmental organizations. His research interests include NLP, semantic representation, parameter-efficient tuning of large language models, machine unlearning, and LLM-based steganography.



**MOUHIB ALNOUKARI** received the B.S. degree in informatics engineering from the Higher Institute for Applied Sciences and Technology (HIAST), Syria, in 1989, the D.E.A. degree in informatics from the University of Montpellier, France, in 1993, and the Ph.D. degree in software engineering from Damascus University, Syria. He is currently an Assistant Professor with the Department of Software Engineering and Intelligent Information Systems, Faculty of Artificial Intelligence Engineering, Syrian Private University (SPU), Syria. He has co-authored multiple academic books in computer science and has published more than 30 research articles in the fields of software engineering and intelligent systems. His research interests include software engineering, business intelligence, data mining, data warehousing, and agile methodologies.