Comparing TF-IDF, Sentence-BERT, and LSTM for Self-Admitted Technical Debt Classification

Rafael Santos Silva[1], **Auston Ricardo Soares Costa**[1],
Cleyton Mário de Oliveira Rodrigues[1], **Wylliams Barbosa Santos**[1]

[1]Polytechnic School of Pernambuco, University of Pernambuco (UPE)
Recife – Pernambuco – Brazil

{rss,arsc}@ecomp.poli.br, {cmor,wbs}@ecomp.poli.br

Abstract. *Self-Admitted Technical Debt (SATD) is the explicit acknowledgment of technical debt in software artifacts. This study compared TF-IDF, Word2Vec, and Sentence-BERT representations for SATD classification using five combinations on 4,071 comments. TF-IDF + linear SVM achieved the best overall performance, with a macro-F1 of 0.7444 and low computational cost. Sentence-BERT-based approaches did not outperform lexical models, while Word2Vec + LSTM showed a strong tendency toward the majority class and substantially lower macro-F1. Under the evaluated conditions, TF-IDF combined with linear models provided the best overall results.*

1. Introduction

The term Technical Debt (TD) was introduced by Cunningham [Cunningham 1992] as a metaphor to describe technical decisions aimed at accelerating software delivery in the short term at the cost of additional effort in the long term. In general, the literature highlights Self-Admitted Technical Debt (SATD) as an approach for identifying technical debt, consisting of the explicit acknowledgment of debt by software project team members through textual artifacts produced during the software development lifecycle [Potdar and Shihab 2014, da Silva Maldonado et al. 2017, Sierra et al. 2019].

SATD can be identified in several software artifacts, including source code comments, *commit* messages, *pull requests*, and *issue* tracking systems. SATD reflects the tacit knowledge of development teams regarding technical limitations, temporary decisions, and trade-offs made due to deadlines, costs, or business priorities [Potdar and Shihab 2014, Li et al. 2023]. The presence of such debt across different artifacts is associated with increased software maintenance and evolution complexity, motivating the development of automated identification mechanisms [da Silva Maldonado et al. 2017, Sierra et al. 2019].

Manual identification of SATD is a costly and poorly scalable activity, which has driven the adoption of Natural Language Processing (NLP) techniques [Huang et al. 2018]. Early studies explored traditional representations such as Bag-of-Words and TF-IDF combined with supervised classifiers [Huang et al. 2018, Gorripati et al. 2025]. More recent approaches based on distributed embeddings and neural networks are capable of capturing complex semantic relationships within textual artifacts [Ren et al. 2019, Santos et al. 2020].

Transformer-based models and Large Language Models have shown promising results for SATD identification [Gu et al. 2024, Fontana et al. 2025,

Selvanayagam et al. 2026]. However, there is still limited empirical evidence comparing lexical, distributed, and contextual representations under the same experimental protocol, particularly contextual embeddings based on Sentence-BERT combined with linear classifiers. This gap is relevant because the combination of Sentence-BERT embeddings and linear classifiers offers both low computational cost and the ability to capture semantic information from comments [Reimers and Gurevych 2019].

To investigate this gap, this study addresses the following research questions:

- **RQ1**: How do TF-IDF, Word2Vec, and Sentence-BERT representations compare in terms of performance for automatic SATD classification?
- **RQ2**: What is the trade-off between predictive performance and computational cost among different SATD vectorization and classification combinations?
- **RQ3**: Which approach is more robust in SATD classification scenarios characterized by class imbalance?

To address these questions, this work presents a comparative analysis of lexical (TF-IDF), distributed (Word2Vec), and contextual (Sentence-BERT) representations for SATD classification. The study evaluates predictive performance, computational cost, and behavior under class imbalance.

The remainder of this paper is organized as follows. Section 2 presents the fundamental concepts and related work. Section 3 describes the adopted methodology. Section 4 reports the experimental results. Section 5 discusses the findings. Section 6 presents the limitations and threats to validity. Finally, Section 7 concludes the paper and outlines future work.

2. Related Work

2.1. Self-Admitted Technical Debt

Self-Admitted Technical Debt (SATD) was defined by [Potdar and Shihab 2014] as technical limitations explicitly documented by developers in comments and other software artifacts. These records represent known problems whose resolution is postponed to a future time.

Early studies focused on comments containing expressions such as *TODO*, *FIXME*, and *HACK*. Subsequently, [da Silva Maldonado et al. 2017] proposed a SATD taxonomy and investigated its automatic identification using Natural Language Processing techniques. The review conducted by [Sierra et al. 2019] highlighted the evolution of the field toward the detection, classification, and management of Technical Debt.

In addition to source code comments, SATD has also been identified in *commit* messages, *pull requests*, and *issue* tracking systems [Li et al. 2023]. More recently, [Selvanayagam et al. 2026] investigated SATD in systems based on Large Language Models, revealing new categories of debt associated with the use of artificial intelligence.

2.2. Automatic SATD Classification

Early approaches to automatic SATD identification relied on lexical patterns and predefined keywords [Potdar and Shihab 2014]. However, these techniques present limitations regarding coverage and dependence on the vocabulary adopted by software developers.

With advances in machine learning, representations such as Bag-of-Words, n-grams, and TF-IDF became widely adopted. According to [Huang et al. 2018], text mining techniques outperform approaches based solely on lexical patterns, while [Gorripati et al. 2025] reinforced the competitiveness of such representations as SATD classification baselines. Subsequently, distributed embeddings and neural networks began to be explored. The study by [Ren et al. 2019] employed Word2Vec for SATD detection, whereas [Santos et al. 2020] evaluated its combination with LSTM networks. More recently, Transformer-based models have shown promising results for SATD identification and classification tasks [Gu et al. 2024, Fontana et al. 2025].

Table 1 summarizes the main studies related to automatic SATD classification. Although the literature includes comparisons among lexical representations, distributed embeddings, and Transformer-based models, there is still limited evidence regarding the use of Sentence-BERT combined with linear classifiers for automatic SATD classification. This gap motivates the investigation conducted in this work.

Table 1. Summary of related work.

Study	Text Representation	Model	Task
Huang et al.	TF-IDF	Ensemble	Binary
Ren et al.	Word2Vec	CNN	Binary
Santos et al.	Word2Vec	LSTM	Multiclass
Li et al.	fastText	MT-CNN	Multiclass
Gu et al.	BERT	Transformer	Binary
Fontana et al.	Multiple	Various	Multi/Binary
Huang et al. (2018)	TF-IDF	Ensemble	Binary
Ren et al. (2019)	Word2Vec	CNN	Binary
Santos et al. (2020)	Word2Vec	LSTM	Multiclass
Li et al. (2023)	fastText	MT-CNN	Multiclass
Gu et al. (2024)	BERT	Transformer	Binary
Fontana et al. (2025)	Multiple	Various	Multi/Binary

3. Research Methodology

This study follows a quantitative experimental approach, widely adopted in SATD automatic classification research [Huang et al. 2018, Ren et al. 2019, Fontana et al. 2025]. The objective is to compare textual representation strategies and machine learning algorithms for the identification of SATD categories. Figure 1 presents an overview of the methodological workflow adopted in this study.

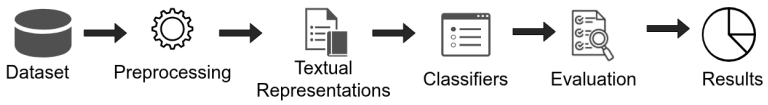


Figure 1. Methodological workflow for automatic SATD classification.

3.1. Dataset

The experiments used the SATD dataset provided by [da Silva Maldonado et al. 2017], which has been widely employed in technical debt classification studies

[Huang et al. 2018, Ren et al. 2019, Gorripati et al. 2025]. The dataset contains comments extracted from software projects and labeled according to SATD categories. Records classified as `WITHOUT_CLASSIFICATION` and those containing missing values were removed. After this filtering process, the final dataset contained 4,071 comments distributed across five SATD categories: `DESIGN`, `IMPLEMENTATION`, `DEFECT`, `TEST`, and `DOCUMENTATION`.

3.2. Preprocessing

In **Experiment 01**, comment delimiters were removed, URLs, file paths, and numbers were normalized, *camelCase* and *snake_case* terms were split, text was converted to lowercase, and non-alphanumeric characters were removed. The objective was to preserve relevant technical debt markers, such as `TODO`, `FIXME`, `hack`, `workaround`, `bug`, `deprecated`, and negation terms, since these elements may serve as strong semantic indicators of comment categories [Huang et al. 2018].

In **Experiment 02**, preprocessing was refined into two distinct pipelines. For the TF-IDF representation, *stopwords* were removed while preserving negations and SATD markers, followed by lemmatization with POS tagging. For contextual embeddings, a lightweight cleaning process without lemmatization was adopted, preserving the original surface form of words under the assumption that *Transformer*-based models can better exploit the original textual structure, allowing semantically equivalent terms such as *fix*, *fixed*, and *fixing* to be represented similarly.

3.3. Evaluated Representations

Experiment 01 compared three families of textual representations: TF-IDF, Sentence-BERT, and Word2Vec + LSTM. Table 2 summarizes the main configurations adopted for each representation.

Table 2. Evaluated textual representations.

Representation	Configuration
TF-IDF	n-grams (1,3), min_df=2, sublinear_tf=True, lemmatization, and negation preservation.
Sentence-BERT	`all-MiniLM-L6-v2` model, 384-dimensional normalized embeddings.
Word2Vec + LSTM	CBOW, 100 dimensions, window=5, min_count=5, sequences of up to 100 tokens, and frozen embeddings.

In **Experiment 02**, ADASYN was employed to address class imbalance. In the TF-IDF pipeline, `TruncatedSVD` with 300 components was applied after vectorization, converting the sparse representation into dense vectors suitable for oversampling. In the Sentence-BERT pipeline, embeddings were directly used by ADASYN [He et al. 2008, He and Garcia 2009].

3.4. Evaluated Models

In **Experiment 01**, five combinations were evaluated: TF-IDF + Logistic Regression, TF-IDF + linear SVM, Sentence-BERT + Logistic Regression, Sentence-BERT + lin-

ear SVM, and Word2Vec + LSTM. This class-weighting strategy was applied to the linear classifiers but not to the Word2Vec + LSTM configuration, which should be considered when interpreting its performance under the strongly imbalanced class distribution. Logistic Regression and linear SVM were selected because they are widely used as baselines in text classification tasks [Joachims 1998] and SATD classification studies [Huang et al. 2018, Li et al. 2023, Gorripati et al. 2025].

In **Experiment 02**, the combinations TF-IDF + Logistic Regression, TF-IDF + linear SVM, Sentence-BERT + Logistic Regression, and Sentence-BERT + linear SVM were evaluated. The main difference was the introduction of ADASYN for oversampling minority classes. In the TF-IDF pipeline, the technique was applied after dimensionality reduction using `TruncatedSVD`, whereas in the Sentence-BERT pipeline it was applied directly to the dense embeddings.

3.5. Model Configuration

Table 3 presents the hyperparameters used in the evaluated models. The parameter values were kept constant across Experiments 01 and 02 to enable direct comparisons among different textual representations and class-balancing strategies. Linear classifiers were configured with balanced class weights to reduce the effects of dataset imbalance.

Table 3. Main hyperparameters used in the experiments.

Model	Configuration
Logistic Regression	$C = 1.0$, max_iter=3000, class_weight=balanced
Linear SVM	$C = 1.0$, class_weight=balanced
TF-IDF	ngram_range=(1,3), min_df=2, sublinear_tf=True
TruncatedSVD (Exp. 02)	n_components=300, random_state=42
ADASYN (Exp. 02)	n_neighbors=3, random_state=42
Word2Vec + LSTM	128/64 units, dropout=0.3, batch_size=32, epochs=50, validation_split=0.10, Adam, EarlyStopping(patience=5), ReduceLROnPlateau(factor=0.2)

3.6. Metrics

The models evaluated in this study were assessed using accuracy, precision, recall, and F1-score. Given the strong class imbalance in the dataset, macro-F1 was considered the most relevant metric for overall comparison, as it assigns equal weight to all classes regardless of their support. Weighted F1, which is more sensitive to the actual class distribution, and macro-recall, particularly useful for analyzing model behavior on minority classes, were also reported.

3.7. Protocol

In both experiments, the dataset was split into training and test sets using an 80/20 ratio, with class stratification and a fixed random seed of 42, resulting in 3,256 training instances and 815 test instances. The same split was used across all approaches to ensure result comparability. The experimental artifacts and replication package are available at `https://doi.org/10.5281/zenodo.20763448`.

Experiment 01 evaluated linear models with class-weight balancing and a neural architecture based on Word2Vec + LSTM. **Experiment 02** evaluated a class-balancing

pipeline based on ADASYN, combined with specialized preprocessing and, for TF-IDF, dimensionality reduction using `TruncatedSVD`.

To prevent information leakage, the dataset was first split into training and test sets. All subsequent fitting and resampling operations were performed exclusively on the training partition. In **Experiment 01**, the `TfidfVectorizer` was fitted using only the training data. In **Experiment 02**, the `TfidfVectorizer`, `TruncatedSVD`, and ADASYN were encapsulated in a single *pipeline* and fitted exclusively on the training set. The test set remained untouched until transformation and final evaluation.

4. Results

4.1. Dataset Distribution

The class distribution reveals a strongly imbalanced dataset. The `DESIGN` class accounts for 66.40% of the data, followed by `IMPLEMENTATION` (18.59%), `DEFECT` (11.59%), `TEST` (2.09%), and `DOCUMENTATION` (1.33%).

4.2. Performance

In **Experiment 01**, the best overall performance was achieved by the TF-IDF + linear SVM model, reaching an accuracy of 0.8110, a macro-F1 score of 0.7444, and a weighted F1 score of 0.8083. TF-IDF + Logistic Regression also achieved competitive performance, with a macro-F1 score of 0.7057. The results of all evaluated models are presented in Table 4.

Table 4. Overall performance in Experiment 01.

Representation	Model	Acc.	Macro-F1	Macro Prec.	Macro Rec.	Weighted F1
TF-IDF	Linear SVM	**0.8110**	0.7444	0.8017	0.7053	0.8083
TF-IDF	Logistic Regression	0.7669	0.7057	0.7135	0.7089	**0.7728**
Sentence-BERT	Linear SVM	0.7141	0.6213	0.5964	0.6617	**0.7178**
Sentence-BERT	Logistic Regression	0.6098	0.5577	0.5102	**0.7053**	0.6334
Word2Vec	LSTM	0.6638	0.1596	0.1328	0.2000	0.5297

In **Experiment 02**, with ADASYN, the best result was achieved by the linear SVM model using Sentence-BERT embeddings, reaching an accuracy of 0.6515, a macro-F1 score of 0.6004, and a weighted F1 score of 0.6713. The performance results are presented in Table 5.

Table 5. Overall performance in Experiment 02 with ADASYN.

Representation	Model	Acc.	Macro-F1	Macro Prec.	Macro Rec.	Weighted F1
Sentence-BERT	Linear SVM + ADASYN	0.6515	0.6004	0.5550	**0.6913**	0.6713
TF-IDF	Logistic Regression + ADASYN	0.6405	0.5921	0.5698	**0.6815**	0.6622
TF-IDF	Linear SVM + ADASYN	0.6466	0.5884	0.5697	0.6673	**0.6693**
Sentence-BERT	Logistic Regression + ADASYN	0.6282	0.5764	0.5266	0.6984	**0.6502**

When comparing both experiments on this dataset, TF-IDF + linear SVM with balanced class weights achieved higher overall performance than the configurations evaluated in Experiment 02. The Experiment 02 pipelines increased recall for some minority classes but also produced more false positives, reducing precision, accuracy, and weighted F1. Since `TruncatedSVD` and ADASYN were jointly introduced in the TF-IDF pipeline, their individual contributions to these changes cannot be isolated.

4.3. Experimental Environment

The experiments were conducted in a Google Colab environment using Google Compute Engine infrastructure running Ubuntu 22.04.5 LTS and Python 3.12.13. The environment provided 12.7 GB of RAM and 107.7 GB of disk storage. The implementations relied on the Scikit-learn, TensorFlow 2.19.0, and PyTorch 2.10.0 libraries.

Three main metrics were collected: (i) total training time; (ii) total inference time on the test set; and (iii) average inference time per sample. For Sentence-BERT-based models, the contextual embedding generation time was also measured separately, as well as the classifier inference time after embedding generation when available.

Table 6. Computational cost of the evaluated models.

Exp.	Model	Total Training (s)	Total Inference (s)	Inference/Sample (ms)	Macro-F1
01	**TF-IDF + Linear SVM**	**13.211**	2.906	3.566	**0.7444**
01	TF-IDF + Logistic Regression	17.991	3.047	3.738	0.7057
01	Sentence-BERT + Linear SVM	53.081	14.073	17.267	0.6213
01	Sentence-BERT + Logistic Regression	52.484	14.075	17.269	0.5577
01	Word2Vec + LSTM	135.350	2.354	2.888	0.1596
02	TF-IDF + SVD + ADASYN + Linear SVM	15.909	0.163	0.201	0.5884
02	TF-IDF + SVD + ADASYN + Logistic Regression	16.072	**0.095**	**0.117**	0.5921
02	Sentence-BERT + ADASYN + Linear SVM	45.209	9.219	11.311	0.6004
02	Sentence-BERT + ADASYN + Logistic Regression	43.406	9.224	11.317	0.5764

In **Experiment 01**, the TF-IDF + linear SVM model achieved the best balance between predictive performance and computational cost, obtaining the highest macro-F1 score (0.7444), the lowest training time (13.211 s), and an inference time of 3.566 ms per sample (Table 6).

The Sentence-BERT + linear SVM and Sentence-BERT + Logistic Regression models required approximately 17.27 ms per sample, about 4.84 times the inference cost of TF-IDF + linear SVM, mainly due to contextual embedding generation. Classifier inference itself remained below 0.005 ms per sample. The Word2Vec + LSTM architecture exhibited the highest training time (135.350 s), approximately 10.25 times higher than TF-IDF + linear SVM. Despite an inference time of 2.888 ms per sample, it achieved a macro-F1 score of only 0.1596, providing no performance gains proportional to its computational complexity.

In **Experiment 02**, the TF-IDF + SVD + ADASYN pipelines maintained training times close to those observed in Experiment 01 (15.909 s for linear SVM and 16.072 s for Logistic Regression), with inference times ranging from 0.117 ms to 0.201 ms per sample. The Sentence-BERT + ADASYN models required 45.209 s and 43.406 s for training, respectively, in addition to 6.089 s for encoder loading and approximately 11.31 ms per sample during inference. Although these models improved recall for minority classes, they reduced both macro-F1 and weighted F1. The Word2Vec + LSTM configuration achieved a macro-F1 score of 0.1596 and showed a strong tendency to predict the majority `DESIGN` class. This behavior indicates that the evaluated configuration was strongly affected by class imbalance. Therefore, this result should be interpreted as a limitation of the evaluated configuration rather than as evidence against sequential architectures for SATD classification.

Within the reported Google Colab execution, TF-IDF combined with linear models presented the best relationship between predictive performance and computational

cost. Sentence-BERT introduced additional computational overhead due to embedding generation, whereas Word2Vec + LSTM combined high training cost with a low macro-F1 score. Since the computational measurements were obtained from single runs, these values should be interpreted as descriptive estimates rather than hardware-independent benchmarks.

4.4. Per-Class Analysis

Table 7 presents the per-class performance of the best models from Experiments 01 and 02. The TF-IDF + linear SVM model achieved the best results across all categories, particularly for `DESIGN` (0.8757) and `TEST` (0.8750). Although Sentence-BERT + linear SVM + ADASYN improved recall for minority classes, this gain came at the cost of lower precision and F1-score, resulting in a lower macro-F1 score (0.6004).

Table 7. Per-class performance of the best models from Experiments 01 and 02.

Class	Exp. 01	Exp. 02
	TF-IDF + SVM	SBERT + SVM + ADASYN
DEFECT	0.6114	0.4693
DESIGN	0.8757	0.7453
DOCUMENTATION	0.6667	0.5185
IMPLEMENTATION	0.6931	0.5373
TEST	0.8750	0.7317
Macro-F1	0.7444	0.6004

The most discriminative terms for each category were identified from the highest positive coefficients of the TF-IDF + Logistic Regression classifier. For `DEFECT`, the most relevant terms included `bug` and `fixme`; for `DOCUMENTATION`, `document` and `javadoc`; and for `TEST`, `test`, `untested`, `assert`, and `fail`. These terms exhibit semantic coherence with their respective SATD categories.

4.5. Comparison with Related Work

Table 8 compares the results obtained in this study with previous work on automatic SATD classification. Although differences in datasets, experimental protocols, and task definitions limit direct comparisons, the analysis helps contextualize the obtained results.

Table 8. Comparison between this study and related work on SATD classification.

Study	Approach	Task	Metric	Result
Huang et al. (2018)	TF-IDF + Ensemble	Binary	F1	0.737
Ren et al. (2019)	CNN + Embeddings	Binary	F1	0.752
Santos et al. (2020)	Word2Vec + LSTM	Design SATD	F1	Reported Improvement
This Study	TF-IDF + SVM	Multiclass	Macro-F1	**0.7444**

The results reported in Table 8 provide contextual references to previous SATD classification studies but should not be interpreted as direct numerical comparisons, since the studies differ in datasets, task definitions, evaluation metrics, and experimental protocols. Huang et al. [Huang et al. 2018] reported an F1-score of 0.737 using text mining techniques and traditional classifiers, whereas Ren et al. [Ren et al. 2019] achieved an F1-score of 0.752 using convolutional neural networks and distributed embeddings. In

particular, these studies primarily evaluated binary classification settings, whereas this work investigated a five-class classification problem using macro-F1.

Furthermore, TF-IDF + linear SVM outperformed the Sentence-BERT- and Word2Vec + LSTM-based approaches evaluated in Experiments 01 and 02, indicating that explicit lexical markers retain strong discriminative power in the analyzed dataset. These findings reinforce evidence from the literature that simple and interpretable models remain effective alternatives for automatic SATD classification, particularly in scenarios with limited labeled data and imbalanced classes.

5. Discussion

Under the evaluated experimental conditions, the lexical TF-IDF representation achieved better results than the distributed and contextual configurations considered in this study.

In **Experiment 01**, the best performance was achieved by the TF-IDF + linear SVM model with balanced class weights, reaching a macro-F1 score of 0.7444. This result is consistent with the frequent occurrence of explicit lexical markers in SATD comments, such as `TODO`, `FIXME`, `bug`, `test`, `javadoc`, and `workaround`, which are effectively captured by TF-IDF representations with n-grams.

The evaluated Sentence-BERT configuration achieved lower performance than TF-IDF. Since `all-MiniLM-L6-v2` is a general-purpose sentence encoder and was not specifically pretrained on source code, this result should not be generalized to domain-specific or fine-tuned contextual models. Furthermore, SATD comments are typically short, fragmented, and often combine natural language with source code identifiers.

The Word2Vec + LSTM configuration yielded the lowest macro-F1 score (0.1596) and concentrated its predictions on the majority `DESIGN` class. The corresponding macro-recall (0.2000) and macro-precision (0.1328) are consistent with a strong majority-class prediction tendency. Therefore, this outcome is treated as a limitation of the evaluated training configuration rather than evidence of inferior performance of sequential neural architectures in general.

Experiment 02 showed increased recall for some minority classes but lower overall performance than the best configuration from Experiment 01. However, because `TruncatedSVD` and ADASYN were jointly introduced in the TF-IDF pipeline, the observed changes cannot be attributed exclusively to ADASYN. Although some underrepresented categories achieved higher recall, the best overall result in Experiment 02 was a macro-F1 score of 0.6004, which remained below the performance of TF-IDF + linear SVM in Experiment 01.

Overall, within the evaluated configurations and experimental protocol, the best trade-off among predictive performance, computational cost, and interpretability was achieved by **TF-IDF + linear SVM**, which obtained the highest macro-F1 score (0.7444).

6. Limitations and Threats to Validity

Although the study was conducted rigorously, some limitations and threats to validity may affect the results:

1. The dataset is dominated by the `DESIGN` class, whereas classes such as `TEST` and

DOCUMENTATION contain relatively few instances. This imbalance may have influenced model training and favored specific classes.
2. The experiments were conducted using a single SATD dataset. Although this dataset is widely adopted in technical debt research, the results may vary when applied to other datasets.
3. The dataset consists of comments extracted from projects written in a single programming language. Therefore, the results cannot be automatically generalized to other programming languages.
4. Experiments 01 and 02 relied on a single stratified 80/20 train-test split. Consequently, the observed differences among models should be interpreted as indicative rather than conclusive, since no statistical significance tests were performed.
5. The contextual representation was limited to the general-purpose all-MiniLM-L6-v2 Sentence-BERT model without domain-specific fine-tuning. Therefore, the results should not be generalized to code-aware or fine-tuned contextual encoders.
6. In the TF-IDF pipeline of Experiment 02, TruncatedSVD and ADASYN were introduced jointly. Consequently, their individual effects on model performance cannot be isolated from the current experimental design.
7. The Word2Vec + LSTM configuration showed a strong tendency toward the majority DESIGN class and did not use the same class-weighting strategy adopted for the linear classifiers. Its result should therefore be interpreted as a limitation of the evaluated configuration rather than as a general comparison between sequential and linear architectures.
8. Computational cost measurements were obtained from single runs in a shared Google Colab environment and should therefore be interpreted as descriptive estimates rather than hardware-independent benchmarks.

7. Conclusion

To contribute to the automation of SATD identification, this study was motivated by the need to evaluate the impact of different textual representation strategies on the automatic classification of software comments.

Regarding **RQ1**, the results indicate that the **TF-IDF + linear SVM** combination achieved the best performance for SATD classification, reaching an accuracy of 0.8110, a macro-F1 score of 0.7444, and a weighted F1 score of 0.8083. These findings suggest that lexical markers such as TODO, FIXME, bug, test, and javadoc have strong discriminative power in the analyzed dataset. The evaluated Sentence-BERT configuration achieved lower performance than TF-IDF under the adopted experimental protocol. The Word2Vec + LSTM configuration obtained a macro-F1 score of 0.1596 and showed a strong tendency toward the majority DESIGN class. Therefore, this result is interpreted as a limitation of the evaluated LSTM configuration rather than as evidence against sequential architectures in general.

Regarding **RQ2**, TF-IDF + linear SVM provided the best trade-off between predictive performance and efficiency, achieving a macro-F1 score of 0.7444 with 13.211 s of training time and 3.566 ms of inference time per sample. Sentence-BERT-based approaches incurred higher computational costs due to contextual embedding generation,

whereas Word2Vec + LSTM exhibited the highest training time (135.350 s). In Experiment 02, TF-IDF-based pipelines showed lower inference times, while Sentence-BERT + ADASYN remained more computationally expensive. These measurements are specific to the reported Google Colab environment and should be interpreted as descriptive estimates.

Regarding **RQ3**, TF-IDF + linear SVM with balanced class weights achieved the best overall performance. Experiment 02 increased recall for some minority classes but reduced precision, accuracy, and weighted F1. Since `TruncatedSVD` and ADASYN were jointly introduced in the TF-IDF pipeline, their individual effects cannot be isolated. The best model in Experiment 02 achieved a macro-F1 score of 0.6004, below the 0.7444 obtained by TF-IDF + linear SVM in Experiment 01.

Therefore, considering predictive performance, computational efficiency, class-imbalance behavior, and interpretability, **TF-IDF + linear SVM** achieved the best overall results among the configurations evaluated under the adopted experimental protocol.

7.1. Future Work

Several research opportunities can be explored based on this study. The following directions are suggested for future work:

1. Evaluate code-aware contextual encoders, such as CodeBERT and GraphCodeBERT, and compare them with the general-purpose Sentence-BERT representation evaluated in this study.
2. Investigate alternative strategies for handling class imbalance, including cost-sensitive learning and other oversampling techniques.
3. Replicate the experiments using stratified k-fold cross-validation to assess result stability across partitions, report mean and standard deviation, and enable paired statistical significance testing.
4. Isolate the effects of dimensionality reduction and synthetic oversampling by evaluating TF-IDF + TruncatedSVD with and without ADASYN under the same experimental protocol.

References

Cunningham, W. (1992). The wycash portfolio management system. *ACM Sigplan Oops Messenger*, 4(2):29–30.

da Silva Maldonado, E., Shihab, E., and Tsantalis, N. (2017). Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, 43(11):1044–1062.

Fontana, F. A., Di Rocco, J., Di Ruscio, D., Di Salle, A., and Nguyen, P. T. (2025). Binary and multi-class classification of self-admitted technical debt: How far can we go? *Information and Software Technology*, page 107862.

Gorripati, S. M. C., Altalbe, A., and Rangarajan, P. K. (2025). Automated classification of self-admitted technical debt using advanced word embedding techniques. *Journal of Computational and Cognitive Engineering*.

Gu, H., Zhang, S., Huang, Q., Liao, Z., Liu, J., and Lo, D. (2024). Self-admitted technical debts identification: How far are we? In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 804–815. IEEE.

He, H., Bai, Y., Garcia, E. A., and Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Neural Networks*, pages 1322–1328.

He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.

Huang, Q., Shihab, E., Xia, X., Lo, D., and Li, S. (2018). Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, 23(1):418–451.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning*, pages 137–142. Springer.

Li, Y., Soliman, M., and Avgeriou, P. (2023). Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering*, 28(3):65.

Potdar, A. and Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 91–100. IEEE.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 3982–3992.

Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., and Grundy, J. (2019). Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM transactions on software engineering and methodology (TOSEM)*, 28(3):1–45.

Santos, R. M., Santos, I. M., Júnior, M. C. R., and de Mendonça Neto, M. G. (2020). Long term-short memory neural networks and word2vec for self-admitted technical debt detection. In *ICEIS (2)*, pages 157–165.

Selvanayagam, N., Ghaleb, T. A., and Abdellatif, M. (2026). Self-admitted technical debt in llm software: An empirical comparison with ml and non-ml software. *arXiv preprint arXiv:2601.06266*.

Sierra, G., Shihab, E., and Kamei, Y. (2019). A survey of self-admitted technical debt. *Journal of Systems and Software*, 152:70–82.