Fake it till you make it: exploring the usefulness of synthetic self-admitted technical debt datasets

MIROSLAV ATANASOV, University of Twente, The Netherlands

This research investigates whether synthetic data augmentation improves machine learning models' performance in detecting and classifying Self-Admitted Technical Debt (SATD) from code comments. We evaluate the DebtHunter and PILOT models using both Maldonado et al.'s dataset as well as SATDAUG, an augmented dataset based on it. Ultimately, we show that this approach yields significant results by effectively addressing class imbalance issues that has previously hindered accurate detection and classification.

Additional Key Words and Phrases: self-admitted technical debt, augmented data, neural networks, artificial intelligence, machine learning, code comments, empirical study

1 INTRODUCTION

Technical debt (TD) is a metaphor that describes suboptimal solutions that provide immediate benefits at the cost of increased maintenance costs and complexity in the future. The term was officially coined in 1992 by Cunningham [3], and researchers have been investigating how they could identify it, manage it, repay it, and resolve it since then [1], [9]. These studies have facilitated the development of metrics, tools, and methodologies that aim to make technical debt more tangible and actionable for developers faced with limited resources and competing priorities.

More recently, a particular kind of technical debt, self-admitted technical debt (SATD), has emerged as an interesting area of research. It can fall into any of the categories mentioned above for TD, but the key difference is that developers explicitly acknowledge it within the code itself. The term was first defined by Potdar and Shihab [12] who conducted an initial investigation into this phenomenon.

Following these steps, Maldonado and Shihab [10] identified 62 distinct SATD patterns in code comments in various software projects, establishing a taxonomy that has facilitated automated detection and analysis of this specific form of technical debt.

The findings of those studies [12], [10] have been built up upon by Bavota et al. [2] who conducted a large-scale empirical study on self-admitted technical debt that covered more than 600K commits and 2 billion comments that were extracted from 159 open-source Java projects that were under the management of the Apache Software Foundations and from the list of GitHub repositories managed by the Eclipse Foundation.

However, a common issue between the so far mentioned studies is the performance of the classification and identification due to the use of human-summarized patterns to match SATD instances. Vocabulary diversity, semantic variations, project uniqueness, and length make this approach less ideal [13].

The recent advancements in Artificial Intelligence have also had an impact in this field of research. Recent studies employ various natural language processing (NLP) and machine learning (ML) techniques to improve the accuracy in SATD categorization and recognition [7], [5], [11]. For example, DebtHunter [14] is an ML tool that automatically detects and classifies SATD in source code comments into five debt types (defect, design, documentation, requirement, and test). They implemented a two-step classification process: first a binary classifier to identify SATD apart from non-SATD comments, then a multi-class classifier to categorize SATD comments by debt type, using NLP preprocessing, feature selection, and data sampling techniques.

The work on DebtHunter was followed by the paper of Di Salle et al. [5] which proposes a framework, Processing and machIne Learning tO detect self-admitted Technical debt (PILOT), that combines NLP processing and neural network architecture to detect and categorize SATD in code comments. They also created a prototype which they trained and compared its performance with DebtHunter, showing that the neural approach outperforms traditional ML approaches.

However, the use of an AI model requires training on a dataset prior to use. Most investigations suffer from imbalanced sets where some types of SATD are greatly underrepresented compared to others. Several articles have noted this issue as a threat to validity [14], [5]. For example, the dataset provided by Maldonado et al. [11], used by almost all studies on this topic, contains 2,703 Code\Design Debt instances within code comments and only 54 Documentation Debt instances and 85 Test debt instances.

In order to address this shortcoming of existing datasets, Sutoyo et al. [16] proposed the SATDAUG dataset where they augmented the dataset used by Li et al. [8] using AugGPT [4] to synthetically equalize the number of SATD instances of each category.

The purpose of this study is to investigate the performance benefits of using data augmentation to enrich existing datasets that suffer from class imbalance when training ML and neural network models. As a basis of the investigation, we will train and evaluate the performance of DebtHunter and PILOT models on a non-augmented dataset and we will compare it to their performance after training on the SATDAUG dataset.

To guide the research, we have formulated the following research questions.

- **RQ1**: Does the use of an augmented dataset for training improve the performance of the DebtHunter model for identification and classification of self-admitted technical debt?
- **RQ2**: Does the use of an augmented dataset for training improve the performance of the PILOT model for identification and classification of self-admitted technical debt?

TScIT 43, July 4, 2025, Enschede, The Netherlands

After covering the necessary background knowledge in Section 2, we will go over the experimental setup in Section 3. Subsequently, the results of the experiment will be presented in Section 4. Finally, Section 5 provides an interpretation of the results and a comparison with the state-of-the-art.

2 BACKGROUND KNOWLEDGE

2.1 DebtHunter

DebtHunter is a framework that utilizes a supervised machine learning approach for automated detection and classification of SATD in source code comments. The paper by Sala et al. [14] proposes a hierarchical two-phase classification strategy: the initial one decides between SATD and non-SATD instances, followed by a multiclass classifier that categorizes the identified SATD comments into five technical debt categories (defect, design, documentation, requirement, and test debt).

The preprocessing pipeline incorporates standard natural language processing techniques, including tokenization, stopword elimination, morphological normalization via Lovins stemming, and feature vectorization through term frequency-inverse document frequency (TF-IDF) weighting [14].

To mitigate the class imbalance inherent in natural SATD datasets, the researchers used Spread Subsample for undersampling majority classes and Synthetic Minority Oversampling Technique (SMOTE) for augmenting minority class instances [14]. The classification engine utilizes Sequential Minimal Optimization (SMO) algorithms, with hyperparameter optimization conducted via grid search and model validation performed using stratified 10-fold cross-validation. The DebtHunter tool was implemented using Weka, a Java library that provides machine learning algorithms and data mining tools [6].

2.2 PILOT

Processing and machIne Learning tO detect self-admitted Technical debt (PILOT) proposes a deep learning approach to the more commonly used at the time machine learning approaches. It constitutes a framework that leverages neural network architectures for enhanced SATD detection and classification.

To address the limitations of ML techniques, Di Salle et al. [5] employ feedforward neural networks as the primary classification mechanism, with architectural extensibility to accommodate convolutional neural networks (CNN) and graph neural networks (GNN).

In terms of natural language processing, the prototype proposed in the paper uses a comprehensive preprocessing pipeline encompassing text standardization, word-level tokenization, stopword removal, Porter algorithm-based stemming, and lemmatization using NLTK WordNetLemmatizer for morphological analysis. Although the current implementation utilizes TF-IDF vectorization for feature extraction, it is possible to use more sophisticated techniques such as Word2Vec or GloVe.

The researchers compared a prototype of the PILOT framework and compared it with DebtHunter where the performance superiority of a neural network approach became clear.

2.3 The Maldonado dataset

In a 2017 paper, Maldonado et al. [11] decided to address the shortcomings of the current state-of-the-art, which relied on rather outdated and, as it turned out, less efficient techniques based on pattern matching previously found patterns [10] in text that suggest SATD. Additionally, a lot of manual work was required when it came to classifying code comments in terms of SATD. In their study, they propose an NLP classifier to replace the manual work.

As a result of their study, they came up with a dataset of 62,566 code comments each of which is classified as either **Design**, **Implementation**, **Defect**, **Test** and **Documentation** debt or as not having technical debt. The comments were extracted from 10 open source Java projects (Ant, ArgoUML, Columba, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and SQuirrel SQL).

The researchers removed the irrelevant comments by ignoring source code and licensing comments, while preserving the ones that contained task annotations, i.e. TODO, FIXME, XXX, etc. Additionally, they grouped sequential single line comments by making use of regular expressions, excluded Javadoc unless it contained task annotations, and finally auto-generated IDE comments were filtered out. This process narrowed down the initial 259,229 comments to 62,566 relevant ones. Table 1 presents an overview of the dataset.

Finally, Maldonado and his colleagues manually classified each of the remaining comments. They are one of the first ones to come with a rather large dataset for training AI and ML models for SATD and their work has been used by a lot of other practitioners.

Table 1. Comments per class in the Maldonado dataset

Classification	**Count**
Without Classification	58,204
Design	2,703
Implementation	757
Defect	472
Test	85
Documentation	54
Total	**62,275**

2.4 SATDAUG

The dataset proposed by Maldonado et al. [11] is a vivid example of one of the main issues that researchers have had in the creation of SATD classification engines, class imbalance.

Motivated by this problem, Sutoyo et al. [15] decided to investigate whether augmented datasets would improve the performance of popular ML and AI models. They used AugGPT, a method proposed by Dai et al. [4] that relies on ChatGPT to enhance natural language in the context of self-admitted technical debt to perform data augmentation. The researchers identified two strategies for increasing the number of instances of minority classes, namely single-turn and multi-turn dialogues. Sutoyo et al. [15] opted for the "Multi-turn dialogue prompt" approach, which refers to sequential conversational exchange of messages. They incorporated context and persona to their prompts along with the comment to be augmented and the number of comments that must be generated.

Subsequently, they trained a number of popular AI and ML models on enhanced and non-enhanced data. Their results suggest that this approach could be beneficial in trying to improve the classification and identification of SATD instances [15].

As a result of their previous work, Sutoyo et al. [16] proposed the SATDAUG dataset. They applied the mentioned augmentation approach to the dataset provided by Li et al. [8]. Comments in the SATDAUG dataset, similarly to the original dataset, if identified as self-admitted technical debt, fall into one of the following categories: code/design debt (C/D), documentation debt (DOC), test debt (TES), and requirement debt (REQ) [8]. Both datasets feature text from four sources. That is, code comments (CC), commit messages (CM), pull request sections (PS), and issue tracker sections (IS); however, only the CC part of the SATDAUG dataset is within the scope of this project. The authors used the largest SATD class (C/D debt with 2,703 instances) as the target baseline for the augmentation of minority classes. Table 2 compares the number of instances of each class, found in code comments (CC), in the dataset of Li et al. [8] and its augmented version, SATDAUG.

Furthermore, it is important to note that Li et al. [8] used the dataset proposed by Maldonado et al. [11] for the code comments of their own dataset, however, they decided to omit the "Defect" debt instances, since it was not as prevalent as the other four types of SATD across all four data sources. Hence, "Defect" debt is also not present in the SATDAUG dataset.

Table 2. Comparison between the number of class instance in code comments in the Maldonado dataset and SATDAUG

SATD Class	Li et al.	SATDAUG	Factor
C/D debt	2,703	2,703	1.0×
REQ debt	757	2,271	3.0×
TES debt	85	2,635	31.0×
DOC debt	54	2,700	50.0×
Total SATD	**3,599**	**10,309**	**2.9×**
Non-SATD	58,676	58,676	1.0×
Total Dataset	**62,275**	**68,985**	**1.1×**

3 EXPERIMENTAL SETTING

The studies introducing DebtHunter [14] and PILOT [5] used the original non-augmented Maldonado dataset to train their respective model. For this reason, we will train both models on this dataset and the performance will be used as a baseline during the evaluation. Subsequently, to explore the benefits of using augmented datasets for SATD detection and classification, we will train both models on the SATDAUG dataset and the results will be compared to the baseline, followed by their interpretation.

However, before the training phase can begin, the data must be preprocessed. Both models are equipped with a preprocessing pipeline that includes stopword removal, lemmatization, and stemming methods. Feature extraction is made possible via TF-IDF for both models. Furthermore, the 10-fold cross-validation method was employed to prepare the training and testing data.

3.1 PILOT setup

In order to constrain the experiment to only evaluating the effect of augmented data, we will be using PILOT's preprocessing pipeline that was proposed by Di Salle et al. [5]. Furthermore, in their study proposing the framework, the researchers experimented with three different architectures for the neural network of the classifier featuring one (C1), two (C2), and three (C3) middle layers each with 10 neurons, respectively. Their data presented almost perfect scores for models C2 and C3, while C1 is a rather suboptimal option for data at the scale of the dataset presented by Maldonado et al. [11]. The room for improvement in the C1 configuration of the PILOT model is why it was chosen for our experiment.

However, the number of inputs of the neural network had to be changed depending on the number of features extracted. Using TF-IDF as a feature extraction method for the multiclassification problem yielded 4,773 features for the Maldonado dataset and 3,007 for the binary one. In terms of output units, we had 5 and 2 respectively.

Furthermore, when preparing the SATDAUG dataset for training, 6,779 features were extracted to classify the four different types of SATD and 4 output units were needed. Finally, 5,031 features were obtained when discriminating between non-SATD and SATD instances, and 2 output units were used.

The model is trained as previously mentioned in a 10-fold cross-validation fashion in ten rounds. Each round consists of 100 epochs [5].

3.2 DebtHunter setup

The model of Sala et al. [14] is used mainly as is; however, some adjustments were needed. Due to the fact that the classes in SATDAUG are balanced in comparison with the dataset by Maldonado et al. [11] the SMOTE approach for data augmentation used in the DebtHunter is redundant to some extent, so it is omitted when training the multi-class classification model on the SATDAUG dataset. Spread Subsampling is still included when training the binary classifier since the imbalance between SATD and non-SATD is similar in both datasets. To train and evaluate the model, a 10-fold cross-validation is used [14].

4 EVALUATION

4.1 Evaluation metrics

To access the performance of our classification approach, we evaluated a collection of comments categorized into C unique categories. For each category $c \in C$ there are four possible classification outcomes:

- True Positives (TP): representing correctly classified comments
- True Negatives (TN): instances that are correctly excluded from category c
- False Positives (FP): denotes comments erroneously assigned to category c when they belong to different categories
- False Negatives (FN): comments that genuinely belong to category c but were misclassified into alternative categories

We measure how well a classification engine is performing some of the standard metrics, i.e. precision, recall, and F1-score, defined as follows.

Precision measures how accurate positive predictions are. It is the ratio between correctly classified instances to a category c to the total number of instances classified in that category. Precision is defined by the following formula:

$$P_c = \frac{TP_c}{TP_c + FP_c}$$

Recall, on the other hand, focuses on how well the model classifies into category c. The true positive rate (TPR), as Recall is also known, is the ratio between correctly classified comments of type c to the total number of instances of category c.

$$R_c = TPR_c = \frac{TP_c}{TP_c + FN_c}$$

Finally, the F1-score or simply F1, measures how balanced the performance of a classification engine is by calculating the harmonic mean between the precision and the recall. This metric is an indicator of how accurate and reliable the predictions of a model are. The formula is as follows:

$$F1_c = \frac{2 \cdot P_c \cdot R_c}{P_c + R_c}$$

4.2 Results

The results of both DebtHunter and PILOT on the Maldonado dataset will be used as a baseline to assess the usefulness of augmented data. This baseline will be compared with the performance of the models when trained on the SATDAUG dataset.

4.2.1 DebtHunter. Table 3 measures the baseline performance of DebtHunter against its enhanced version. In terms of binary classification, the model improved when it came to SATD identification, with scores for precision, recall, and F1 increased from 0.913, 0.635, and 0.749 to 0.977, 0.901, and 0.938, respectively. The improvement of around 53% on recall indicates that after training on the augmented dataset, DebtHunter became better at accurately spotting SATD instances. The improvement is further supported by the distribution of the predictions in Figs. 1a and 2a. The baseline version of DebtHunter correctly identified 2,583 of 4,071 SATD instances, while the enhanced one successfully classified 9,293 of 10,310 SATD instances. This was also reflected in the overall performance of the model, represented by the F1. Both versions of DebtHunter perform similarly when classifying NON-SATD comments. This can also be observed in Figs. 1a and 2a, which present the confusion matrices of DebtHunter's binary classifier on both datasets.

Furthermore, improved performance can also be observed on the classification problem as a result of the use of the SATDAUG dataset. Fig 2b shows that balancing the SATD classes resulted in a clear distinction between the four different types of SATD debt. In contrast, when trained on the Maldonado dataset, DebtHunter performed poorly in distinguishing the underrepresented classes. Most of the "Implementation" and "Defect" debt instances were classified as "Design", while barely any success was achieved when categorizing "Test" and "Documentation" debt 1b. This is further supported by the results in Table 3 where the improvement in F1-scores for all classes is present when DebtHunter was trained on the SATDAUG dataset.

Furthermore, the improvements in "Documentation" and "Test" in terms of F1-score reflects the increase in recall and precision.

In addition, the baseline for precision on "REQ (S) + IMP(M)" outperforms the enhanced version of the model, but the extremely low score on recall indicates that the baseline is overly conservative.

Finally, both datasets yield different results in the "Design/Code" category. DebtHunter on the SATDAUG dataset was slightly more conservative (higher precision than recall, 0.948 and 0.859 respectively), suggesting that the model prioritized prediction confidence over correctly identifying into "Design/Code". However, on the Maldonado dataset, it performed more liberally (lower precision than recall, 0.752 and 0.995 respectively), indicating the model favored capturing more actual Design/Code cases at the expense of prediction accuracy.

Table 3. Comparison of DebtHunter's performance on SATDAUG (S) and Maldonado (M) Datasets

Category	Precision		Recall		F1 Score	
	S	M	S	M	S	M
Binary Classification						
SATD	**0.977**	0.913	**0.901**	0.635	**0.938**	0.749
NON-SATD	0.983	**0.975**	0.996	0.996	**0.989**	0.985
Multi-class Classification						
DESIGN/CODE	**0.948**	0.752	0.859	**0.995**	**0.901**	0.857
REQ (S) + IMP(M)	0.806	**0.978**	**0.937**	0.238	**0.867**	0.383
DEFECT	-	0.975	-	0.411	-	0.578
DOCUMENTATION	**0.993**	0.944	**0.950**	0.630	**0.971**	0.756
TEST	**0.992**	0.882	**0.993**	0.788	**0.993**	0.832

4.2.2 PILOT. Table 4 measures the baseline performance of PILOT against its enhanced version. In terms of binary classification, the model improved when it came to SATD identification, with the scores for precision, recall, and F1 increased from 0.84, 0.83, 0.83 to 0.94, 0.88, and 0.91, respectively. The improvement of around 6% on recall and 12% on precision indicates that after training on the augmented dataset, PILOT became better at accurately spotting SATD instances while maintaining confidence. This is also supported by the distribution of predictions in Figs. 3a and 3a. The baseline version of PILOT correctly identified 3,380 of 4,071 SATD instances, while its enhanced version successfully classified 9,073 of 10,310 SATD instances. This is also present in the overall F1-score improvement from 0.83 to 0.91. Both versions of PILOT perform similarly when classifying NON-SATD comments, with the enhanced version showing a rather unsignificant decrease in precision (0.99 to 0.98) while preserving almost perfect recall. Those results can be seen in Table 4.

Furthermore, major improvements can be observed in the multiclass classification problem as a result of using the SATDAUG dataset. Fig. 3b shows that balancing the SATD classes resulted in a clear distinction between the four SATD debt types. Initially, when trained on the Maldonado dataset, PILOT performed poorly in distinguishing underrepresented classes. Fig. 3b reveals that the majority

of "Implementation" instances (480 out of 510 misclassified cases, or 94%) were incorrectly labeled as "Design" debt, while "Documentation" and "Test" debt achieved minimal success with F1-scores of only 0.48 and 0.57, respectively. This bias toward the "Design" class was almost completely resolved with the augmented dataset, based on the clear diagonal pattern that is present in Figure 4b.

Improvements in the distribution of predictions are also present in the evaluation metrics. Identifying "Documentation" debt improved from an F1-score of 0.48 to 0.97, driven by noticeable gains in both precision (0.76 to 0.99) and recall (0.35 to 0.96). Similarly, PILOT showed improvement when classifying "Test" debt, from 0.57 to 0.99 in terms of F1-score, with recall increasing from 0.46 to 0.99. The gains in the "REQ (S) + IMP(M)" category are comparable. The values for precision, recall, and F1-score improved from 0.46, 0.33, and 0.38 to 0.75, 0.86, and 0.80, respectively.

Table 4. Comparison of PILOT's performance on SATDAUG (S) and Maldonado (M) Datasets

Category	Precision		Recall		F1 Score	
	S	M	S	M	S	M
Binary Classification						
SATD	**0.94**	0.84	**0.88**	0.83	**0.91**	0.83
NON-SATD	0.98	**0.99**	0.99	0.99	0.98	**0.99**
Multi-class Classification						
DESIGN/CODE	**0.87**	0.75	0.78	**0.87**	**0.82**	0.80
REQ (S) + IMP(M)	**0.75**	0.46	**0.86**	0.33	**0.80**	0.38
DEFECT	-	0.47	-	0.29	-	0.36
DOCUMENTATION	**0.99**	0.76	**0.96**	0.35	**0.97**	0.48
TEST	**0.99**	0.74	**0.99**	0.46	**0.99**	0.57

5 DISCUSSION

5.1 Result Interpretation

Overall, both models improved compared to the baseline after being trained on the SATDAUG dataset, except for the performance on identifying non-SATD comments; the results there are comparable. We believe that this is due to the unchanged number of instances before and after the augmentation Table 2. An overview of the change in F1 scores for each category is provided in Table 5.

Between the two models, PILOT benefited more from the use of augmented data. The performance in categorizing minority classes, "REQ (S) + IMP(M)", "Documentation" and "Test" improved by 42%, 49%, and 42%, respectively. However, DebtHunter's performance on the same classes improved by 48.4%, 21.5%, and 16.1% respectively. This is not as great as PILOT's improvement, since DebtHunter performed rather well initially when it came to the "Documentation" and "Test" categories, which is most likely due to the SMOTE technique for data augmentation that DebtHunter utilizes.

Furthermore, both baseline models had a very difficult time identifying instances of "Implementation" debt. Fig 1b shows that DebtHunter misclassified 578 out of 757 "Implementation" debt instances of which 570 were classified as "Design" debt. PILOT performed slightly better in that category; 510 of the 757 comments labeled "Implementation" were not classified as such, with 480 of them falling under the "Design" debt category. In addition, most False Negative categorizations across the other classes are also thought to be "Design" debt, which suggests that due to the imbalance in the dataset, the models are not able to effectively learn to distinguish the classes apart.

However, the described issues were resolved once both models were trained on the SATDAUG dataset, Figures 2b and 4 present the confusion matrices obtained after training and testing.

5.2 Comparison with State-of-the-art

During the initial exploration of AugGPT, Sutoyo et al. [15] concluded that the use of data augmentation can help to improve the performance of classification engines when addressing the identification and classification of self-admitted technical debt. The researchers concluded that augmenting the dataset, provided by Li et al. [8], with the help of AugGPT to train BiLSTM and BERT, yields better results than using the dataset as is. They managed to improve BiLSTM's F1-scores with 16% on average when identifying SATD text and BERT's with on average 42% when categorizing different types of SATD.

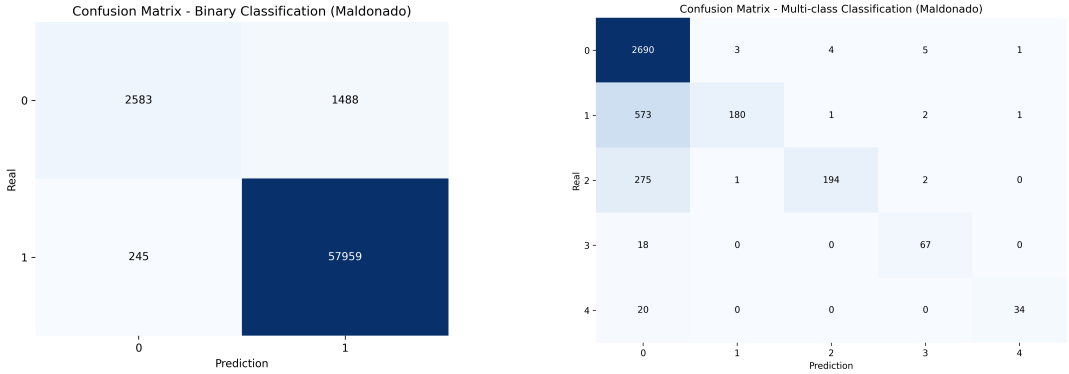
Furthermore, data augmentation was also addressed when creating DebtHunter. Sala et al. [14] experimented with different amounts of synthetic data added through SMOTE, namely 50%, 100%, and 150%. They found that doubling the number of SATD instances leads to better performance on precision, recall, and F1-score, namely +4.52%, +1.36%, and +8.10%, compared to DebtHunter's performance on the original data set. Moreover, the second-best option was adding 50% augmented comments, which also improved the baseline for all three metrics, namely precision +4.02%, recall +2.33%, and F1 +6.5%. Finally, Sala et al. [14] deemed adding 50% synthetic instances to be the best option for data augmentation due to the close scores on precision and F1 with adding 100% synthetic instances, while improving recall and decreasing complexity.

Table 5. F1 Score Comparison: Maldonado (M), SATDAUG (S), and Improvement (Δ)

Category	DebtHunter			PILOT		
	M	S	Δ	M	S	Δ
Binary Classification						
SATD	0.749	0.938	**+0.189**	0.83	0.91	**+0.08**
NON-SATD	0.985	0.989	**+0.004**	0.99	0.98	**-0.01**
Multi-class Classification						
DESIGN/CODE	0.857	0.901	**+0.044**	0.80	0.82	**+0.02**
REQ (S) + IMP(M)	0.383	0.867	**+0.484**	0.38	0.80	**+0.42**
DEFECT	0.578	-	-	0.36	-	-
DOCUMENTATION	0.756	0.971	**+0.215**	0.48	0.97	**+0.49**
TEST	0.832	0.993	**+0.161**	0.57	0.99	**+0.42**

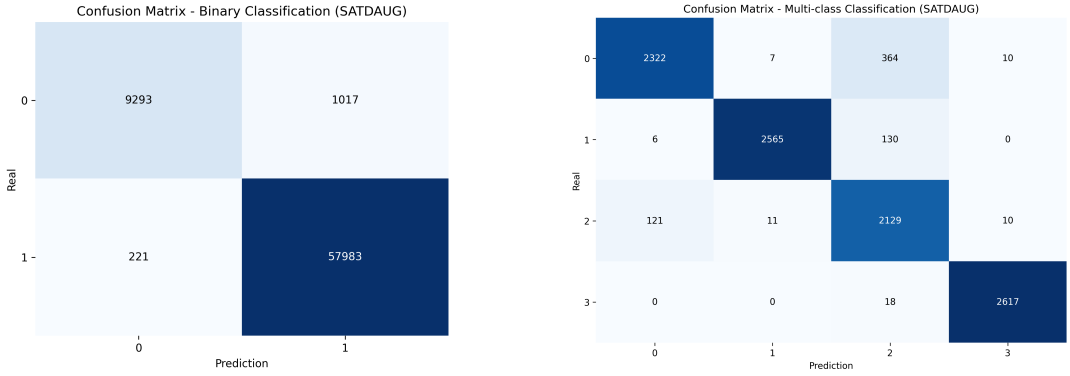
5.3 Threats to Validity

5.3.1 Construct Validity: Evaluation Metrics Misalignment. A potential threat lies between the potential misalignment of the metrics used and the real world utility of such a system potentially deeming it as more useful than it actually is. The traditional metrics used threat all misclassification equally, but in practice, developers may



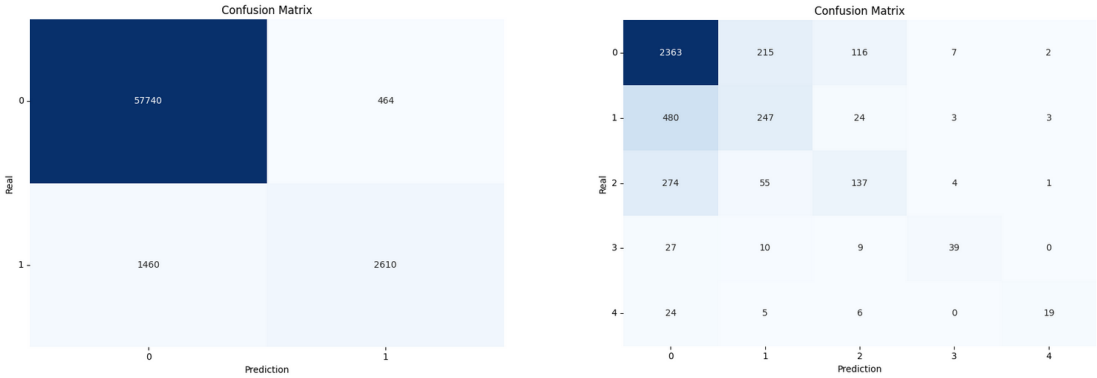
(a) Binary classification (b) Multi-class classification

Fig. 1. Confusion matrices on the Maldonado dataset and DebtHunter



(a) Binary classification (b) Multi-class classification

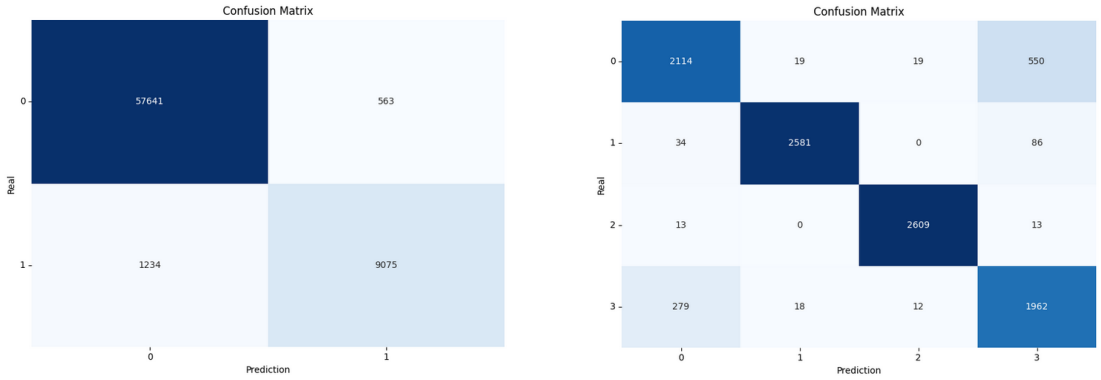
Fig. 2. Confusion matrices on the SATDAUG dataset and DebtHunter



(a) Binary classification (b) Multi-class classification

Fig. 3. Confusion matrices on the Maldonado dataset and PILOT

have asymptotic costs for different types of errors. For example, false positives that waste developer time reviewing non-SATD comments may be more costly than missing some actual SATD instances, or certain categories of technical debt (e.g., security-related) may be more critical to detect than others.



(a) Binary classification

(b) Multi-class classification

Fig. 4. Confusion matrices on the SATDAUG dataset and PILOT

5.3.2 External Validity: Limited Domain Evaluation. During this study, we only explored data augmentation of SATD datasets only focused on code comments in open-source Java projects from the Apache and Eclipse foundations. This threatens generalizability across different programming languages, development cultures, and project types, as commenting conventions and technical debt acknowledgment patterns vary significantly across ecosystems.

5.3.3 Internal Validity: Experimental Design Confounds. When designing our experiment we decided to maintain the original functionality of the DebtHunter model by retaining the SMOTE technique when training on the Maldonado dataset. Since SMOTE itself is a data augmentation technique it may have potentially undermined the true benefits from using augmented datasets.

6 CONCLUSION

6.1 RQ1: Does the use of an augmented dataset for training improve the performance of the DebtHunter model for identification and classification of self-admitted technical debt?

DebtHunter showed significant performance improvements when trained on the SATDAUG dataset. Binary classification F1-score improved from 0.749 to 0.938 (+25.2%), with particularly notable gains in minority classes: "REQ (S) + IMP(M)" debt improved from 0.383 to 0.867 (+48.4%), Documentation debt from 0.756 to 0.971 (+21.5%), and Test debt from 0.832 to 0.993 (+16.1%).

6.2 RQ2: Does the use of an augmented dataset for training improve the performance of the PILOT model for identification and classification of self-admitted technical debt?

Additionally, PILOT also demonstrated consistent improvements across all metrics when trained on augmented data. Binary classification F1-score increased from 0.83 to 0.91 (+9.6%). The model achieved substantial improvements in minority class detection: "REQ (S) + IMP(M)" debt improved from 0.38 to 0.80 (+42%), Documentation debt from 0.48 to 0.97 (+49%), and Test debt from 0.57 to 0.99 (+42%).

6.3 Future work

During the project, we wondered how well the models generalize, and do augmented data have an effect on it, whether it is positive or negative. However, it was pulled out of scope due to time and resource constraints. Investigating this matter could be a key towards wider adoption of this method.

Furthermore, it would be interesting to investigate the usefulness of augmented data on models utilizing various types of feature extraction and neural network architectures or machine learning algorithms to figure out if there are different classifiers that benefit more than others.

Finally, future work should investigate whether data augmentation also improves models when identifying and classifying SATD instances in different sources, such as pull request sections, commit messages, and issue tracker sections.

REFERENCES

[1] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing Technical Debt in Software Engineering. *Dagstuhl Reports* 6, 4 (2016), 110–138. https://doi.org/10.4230/DagRep.6.4.110

[2] Gabriele Bavota and Barbara Russo. 2016. A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. Association for Computing Machinery, New York, NY, USA, 315–326. https://doi.org/10.1145/2901739.2901742

[3] Ward Cunningham. 1992. The WyCash Portfolio Management System. In *Addendum to the Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*. ACM, Vancouver, British Columbia, Canada, 29–30.

[4] Haixing Dai, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Wei Liu, Ninghao Liu, Sheng Li, Dajiang Zhu, Hongmin Cai, Lichao Sun, Quanzheng Li, Dinggang Shen, Tianming Liu, and Xiang Li. 2023. AugGPT: Leveraging ChatGPT for Text Data Augmentation. arXiv:2302.13007 [cs.CL] https://arxiv.org/abs/2302.13007

[5] Amleto Di Salle, Alessandra Rota, Phuong T. Nguyen, Davide Di Ruscio, Francesca Arcelli Fontana, and Irene Sala. 2022. PILOT: synergy between text processing and neural networks to detect self-admitted technical debt. In *Proceedings of the International Conference on Technical Debt* (Pittsburgh, Pennsylvania) *(TechDebt '22)*. Association for Computing Machinery, New York, NY, USA, 41–45. https://doi.org/10.1145/3524843.3528093

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278

[7] Jun Li, Lixian Li, Jin Liu, Xiao Yu, Xiao Liu, and Jacky Wai Keung. 2025. Large language model ChatGPT versus small deep learning models for

self-admitted technical debt detection: Why not together? *Software: Practice and Experience* 55, 1 (2025), 3–28. https://doi.org/10.1002/spe.3360 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3360

[8] Yikun Li, Mohamed Soliman, and Paris Avgeriou. 2023. Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering* 28, 3 (2023), 61. https://doi.org/10.1007/s10664-023-10297-9

[9] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 101 (2015), 193–220. https://doi.org/10.1016/j.jss.2014.12.027

[10] Everton da S. Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical Debt. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. 9–15. https://doi.org/10.1109/MTD.2015.7332619

[11] Everton da Silva Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Trans. Softw. Eng.* 43, 11 (Nov. 2017), 1044–1062. https://doi.org/10.1109/TSE.2017.2654244

[12] Aniket Potdar and Emad Shihab. 2014. An Exploratory Study on Self-Admitted Technical Debt. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME '14)*. IEEE Computer Society, USA, 91–100. https://doi.org/10.1109/ICSME.2014.31

[13] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. 2019. Neural Network-based Detection of Self-Admitted Technical Debt: From Performance to Explainability. *ACM Trans. Softw. Eng. Methodol.* 28, 3, Article 15 (July 2019), 45 pages. https://doi.org/10.1145/3324916

[14] Irene Sala, Antonela Tommasel, and Francesca Arcelli Fontana. 2021. DebtHunter: A Machine Learning-based Approach for Detecting Self-Admitted Technical Debt. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '21)*. Association for Computing Machinery, New York, NY, USA, 278–283. https://doi.org/10.1145/3463274.3464455

[15] Edi Sutoyo, Paris Avgeriou, and Andrea Capiluppi. 2024. Deep Learning and Data Augmentation for Detecting Self-Admitted Technical Debt. In *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 01–10. https://doi.org/10.1109/apsec65559.2024.00022

[16] Edi Sutoyo and Andrea Capiluppi. 2024. SATDAUG - A Balanced and Augmented Dataset for Detecting Self-Admitted Technical Debt. In *Proceedings of the 21st International Conference on Mining Software Repositories* (Lisbon, Portugal) *(MSR '24)*. Association for Computing Machinery, New York, NY, USA, 289–293. https://doi.org/10.1145/3643991.3644880