Check for updates

Received: 15 March 2024 | Revised: 24 May 2024 | Accepted: 7 June 2024

DOI: 10.1002/spe.3360

RESEARCH ARTICLE

WILEY

Large language model ChatGPT versus small deep learning models for self-admitted technical debt detection: Why not together?

Jun Li[1] | **Lixian Li**[2] | **Jin Liu**[1] | **Xiao Yu**[3,4] | **Xiao Liu**[5] | **Jacky Wai Keung**[6]

[1]School of Computer Science, Wuhan University, Wuhan, China

[2]China Satellite Network Exploration Co., Ltd, Chongqing, China

[3]School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan, China

[4]Wuhan University of Technology Chongqing Research Institute, Chongqing, China

[5]School of Information Technology, Deakin University, Geelong, Australia

[6]Department of Computer Science, City University of Hong Kong, Hong Kong, China

Correspondence
Jin Liu, School of Computer Science, Wuhan University, Wuhan, China, and Xiao Yu, School of Computer Science and Artifical Intelligence, Wuhan University of Technology, Wuhan, China.
Email: jinliu@whu.edu.cn, xiaoyu@whut.edu.cn

Funding information
National Natural Science Foundation of China, Grant/Award Number: 61972290; National Natural Science Foundation of Chongqing, China, Grant/Award Number: cstc2021jcyj-msxmX1115

Summary

Given the increasing complexity and volume of Self-Admitted Technical Debts (SATDs), how to efficiently detect them becomes critical in software engineering practice for improving code quality and project efficiency. Although current deep learning methods have achieved good performance in detecting SATDs in code comments, they lack explanation. Large language models such as ChatGPT are increasingly being applied to text classification tasks due to their ability to provide explanations for classification results, but it is unclear how effective ChatGPT is for SATD classification. As the first in-depth study of ChatGPT for SATD detection, we evaluate ChatGPT's effectiveness, compare it with small deep learning models, and find that ChatGPT performs better on Recall, while small models perform better on Precision. Furthermore, to enhance the performance of these approaches, we propose a novel fusion approach named FSATD which combines ChatGPT with small models for SATD detection so as to provide reliable explanations. Through extensive experiments on 62,276 comments from 10 open-source projects, we show that FSATD outperforms existing methods in performance of F1-score in cross-project scenarios. Additionally, FSATD allows for flexible adjustment of fusion strategies, adapting to different requirements of various application scenarios, and can achieve the best Precision, Recall, or F1-score.

KEYWORDS

ChatGPT, fusion, performance and interpretability, self-admitted technical debt, small deep learning models

1 | INTRODUCTION

Technical debt is a concept in the field of software engineering, introduced by Cunningham, to describe "not quite right code."[1] It is used metaphorically to represent the shortcuts taken by developers in opting for suboptimal solutions, which facilitate quicker delivery during the development process. The concept of technical debt parallels financial debt in that taking a shortcut now can lead to increased costs later, just as taking a loan incurs interest. Typical examples of technical debt, such as duplicated code blocks or functions in the codebase, deferring the refactoring of complex code, and using hard-coded values instead of configurable parameters, are often introduced intentionally or unintentionally by

 © 2024 John Wiley & Sons Ltd.

developers during the development process. These practices, while offering short-term benefits like faster delivery or simpler immediate solutions, can lead to long-term drawbacks.[2–7] They increase the maintenance burden, reduce code quality, and can make future changes more time-consuming and costly, which require interest repayment.[8–12] As time progresses, some intentionally introduced technical debts, due to lack of documentation, may be forgotten by developers and not addressed in a timely manner, thereby further compounding the difficulties in software development.[13,14] To mitigate this issue, developers often document these intentional technical debts in their code through comments. For instance, the code comment in the Apache Ant project like "// FIXME formatters are not thread-safe"* indicates an acknowledged issue in the code that needs to be addressed. This kind of documented technical debts recorded by code comments are commonly referred to as Self-Admitted Technical Debts (SATDs), a concept first introduced by Potdar and Shihab.[15]

Detecting SATDs plays a crucial role in software development and maintenance for several reasons.[10,16–18] It can help improve code quality, reduce future maintenance costs, prevent potential bugs and failures, and facilitate better project management. Due to vast quantity and diverse variety of SATDs, developing automatical methods of detecting SATDs has become a prominent subject of interest in the realm of software engineering.[19–21]

With the widespread application of deep learning technology in both artificial intelligence and software engineering fields,[22–27] many recent methods based on deep learning have also been proposed for detecting SATDs in code comments,[28–31] but these methods typically only provide classification results and lack explanations for why a particular instance is classified as SATD. In the absence of explanation, developers may not easily understand why the model marks certain comments as SATDs, which may affect their willingness to accept and trust automated detection systems. Instead, providing an explanation can help developers quickly catch the problems that technical debts imply and pay them back. With the growing popularity of Large Language Models (LLMs) like ChatGPT,† it has been extended across various domains in artificial intelligence and software engineering to provide explanations for classification results.[32–35] However, it is still unclear to what extent ChatGPT can accurately perform binary classification of SATD and whether its explanation is accurate. As the first in-depth study of ChatGPT for SATD detection, this paper explores their potential by focusing on two research questions:

- RQ1: Can ChatGPT be used for SATD detection, and how effective is ChatGPT in classification compared with small deep learning models?
- RQ2: Can we make better use of ChatGPT with existing small deep learning models?

For the first question, we explore the effectiveness of ChatGPT, respectively, equipped with BM25-based[36] few-shot[37] prompting and few-shot Chain-of-Thought (CoT) prompting[33] for SATD detection, which are two best-performing prompting strategies at the moment. In addition, we compare the effectiveness of ChatGPT with the effectiveness of small models. Experimental results show that few-shot CoT prompting strategy is superior to the other in both validity and interpretability because it can provide analysis and explanation for why a comment instance is classified as SATD, and ChatGPT performs quite better on the recall metric while small models perform better on the precision metric.

For the second question, our objective is to develop a better solution that combines the strength of ChatGPT with small deep learing models. We propose a novel approach called FSATD that fuses the results of ChatGPT and three basic small models including Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Transformer, to detect SATDs in open-source software projects and provide reasonable and reliable explanations. Specifically, we first use few-shot CoT prompts to guide ChatGPT to conduct analyze and classification for given comment texts. Then, we employ three small deep learning models that have been trained on the specific dataset to classify the identical given comment text, and obtain the full-voting result of these three classification results. Finally, we fuse the full-voting result with the classification result of ChatGPT to obtain the final classification result through a series of fusion steps. If the full-voting result of three small models exists which indicates they reach a consensus on classification, we choose to believe it as the final fusion result; Otherwise, we choose to believe ChatGPT's result. If the final fusion result is inconsistent with ChatGPT's result, we will guide ChatGPT to correct the original result and provide a reasonable explanation for the final fusion result.

Through extensive experiments on 62,276 comments from 10 open-source projects provided by Maldonado,[38] results show that, in cross-project scenario, the average values of Precision, Recall, and F1-score obtained by FSATD are

*https://github.com/maldonado/tse.satd.data.
†https://chat.openai.com.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

0.8591, 0.8697, and 0.8661, respectively. Compared with the three small models, the values of Precision, Recall, and F1-score, respectively, improve by 13.01%, 27.50%, and 19.68% on average. Compared with the state-of-the-art approaches, GGSATD[30] and HATD,[29] FSATD has an average improvement of 6.31% on Recall and 10.31% on F1-score. Although FSATD does not surpass GGSATD on Precision, it still demonstrated competitive performance. It can provide reasonable and reliable explanations for the classification of SATD. In addition, we explore the performance of different fusion strategies by changing voting mechanisms and find that different fusion strategies tend to improve the performance on different evaluation metrics, so they can be flexibly selected for different application scenarios.

The contributions of this paper are as follows:

- We are the first to explore the capabilities of ChatGPT for SATD detection and to compare it with small models. Our findings reveal that ChatGPT performs better on Recall metric, while small models perform better on Precision metric.
- We propose a novel approach, FSATD, which can not only make accurate classification but also provide reasonable and reliable explanations. Our method is the first to propose a fusion of ChatGPT with small models for SATD detection.

The remainder of the paper is organized as follows. Section 2 introduces the relevant background knowledge of small models and ChatGPT. Section 3 presents the experimental setup. Section 4 shows the results of ChatGPT and small models. Section 5 introduces our approach. Section 6 demonstrates the results of our approach. Section 7 discusses our approach. Section 8 introduces the related work. Section 9 concludes the paper.

2 | PRELIMINARY: SMALL MODELS VERSUS CHATGPT

In this section, we outline the characteristics and capabilities of small models and ChatGPT to help enhance our understanding of the distinct characteristics of various deep learning approaches.

2.1 | Small models

In essence, the distinction between large and small models in artificial intelligence is often based on the number of parameters. Large models typically have tens even hundreds of billions of parameters, while small models have 10 billion parameters or less.[39]

With the rise of the deep learning, some small deep learning models have achieved advanced results in many text classification tasks.[40,41] We explore the capabilities of three basic small models, which are CNN, LSTM, and Transformer respectively, for the main reason that the existing deep learning approaches used to detect SATDs are based on these small models[29–31,42] whose capabilities and tendency to feature extraction are different. Therefore, in our study, we select these three small models widely used in text classification tasks to detect SATD and provide an overview of their feature extraction capabilities:

- **CNN**[43] is composed of layers of convolutions, pooling, and fully connected layers. The convolution layers use filters or kernels to capture local patterns in the input data. In text classification, CNN applies convolutional layers to the text, where filters slide over word embeddings to capture and detect local features like phrases or key terms. Pooling layers then reduce the dimensionality of the data, retaining only the most significant features. These features are then passed through fully connected layers for classification.
- **LSTM**,[44] a type of recurrent neural network, designed to remember information over long periods, consists of a series of input gates, output gates, and forget gates, which control the flow of information. The gates in LSTM allow it to remember important information and forget the irrelevant, making them adept at capturing long-term dependencies in text data, crucial for understanding the overall meaning and context of sentences.
- **Transformer**[45] is built on self-attention mechanisms and feed-forward neural networks. The self-attention mechanism weighs the significance of different parts of the input data, and the model does not process data sequentially. The self-attention mechanism enables the model to consider the relevance of all words in a sentence, regardless of their position, leading to a more nuanced understanding of the text. This comprehensive approach makes Transformers particularly effective for complex text classification tasks.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Each of these models has distinct mechanisms for capturing and interpreting text features, making them suitable for various aspects of text classification. CNN is great for capturing local and positional features,[46–48] LSTM excels in understanding sequential data and long-term dependencies,[49,50] and Transformer is good at capturing dependencies between any two positions in a sequence through self-attention mechanisms and provides a comprehensive understanding of text context.[51–54] However, their interpretability is relatively poor and they can not provide sentence-level natural language analysis and explanation for text classification tasks.[42,55]

2.2 | ChatGPT

With the popularity of Transformer, pretrained large language models based on Transformer architecture emerge one after another. For example, GPT-3,[37] based on the structure of Transformer, have achieved remarkable accomplishments in the field of Natural Language Processing (NLP). Leveraging their formidable capabilities in understanding and generating natural language,[56] researchers have found that these models can perform impressively well in few-shot or even zero-shot scenarios across a series of NLP tasks.

ChatGPT is an advanced language model developed by OpenAI,[‡] based on the GPT (Generative Pretrained Transformer) architecture. It is designed to generate human-like text and engage in conversational interactions. Through its question-and-answer format, ChatGPT can assist in addressing a wide range of issues, drawing on its powerful capabilities in understanding and generating natural language. This enables it to address complex questions and provide informative, contextually relevant answers, making it a versatile tool for various applications in the field of NLP.

SATD detection is essentially a text classification task at the intersection of software engineering and NLP. It involves understanding code comments to determine whether they contain SATDs. However, there has been no research exploring the application of ChatGPT specifically to the detection of SATD.

With the popularity of ChatGPT, many researchers have focused on studying and improving ChatGPT's capabilities. Gao et al.[57] compare the performance of different retrieval-based methods including BM25,[36] SBERT,[58] UniXcoder[59] and CoCoSoDa[60] for software engineering tasks. Results demonstrate that the simple BM25 method can achieve comparable or even better performance than other methods on demonstration selection. In addition, Zhong et al.[35] investigate the synergistic potential of ChatGPT with advanced prompting strategies in a popular benchmark of text classification, GLUE,[61] and the manual few-shot CoT[33] prompting strategy yields the best performance. We present the two advanced prompting strategies:

- **Few-shot-BM25**, building upon the foundation of in-context learning, utilizes the sparse retrieval method BM25[36] to find demonstration examples with the highest relevance scores for each test sample.[62] In SATD detection, we rank samples in the source set based on their relevance scores with given target sample by utilizing BM25, which selects top-n relevant instances from the source set as demonstrations. In Figure 1A, we use BM25 to retrieve relevant instances for the test comment "// FIXME: ugly cast to ContactModel" and search out relevant comments such as "// FIXME check if the id is a correct filename!." These examples are then used to construct prompts following the template designed.
- **Few-shot-CoT**, proposed by Wei et al.,[33] involves providing manual intermediate reasoning demonstrations to guide the model through a step-by-step analysis, leading to the final output. In SATD detection, we manually select several samples from source set as demonstrations. Each demonstration involves a question about the comment, as well as an answer containing multistep analyze and result for the question. Then we concatenate these demonstrations with target comment to construct a valid prompt according to the CoT template which is illustrated in Figure 1B.

3 | EXPERIMENTAL SETUP

In this section, we introduce our experimental setup from four aspects: dataset description, evaluation metrics, parameters, and experimental process in detail.

[‡]https://openai.com.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

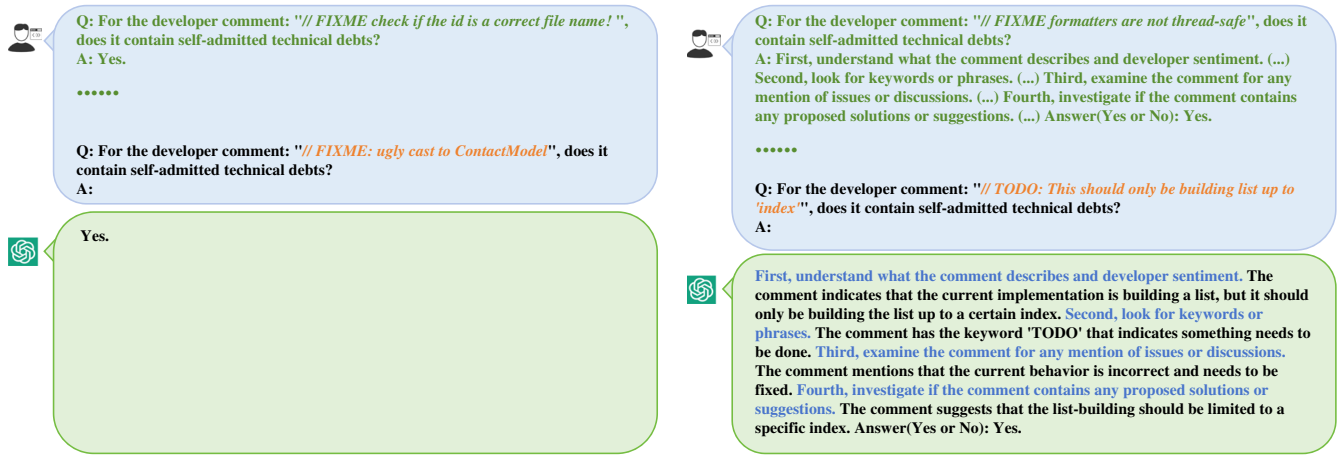

FIGURE 1 The examples of Few-shot-BM25[37] prompting strategy and Few-shot-CoT[33] prompting strategy. (A) Few-shot-BM25, (B) Few-shot-CoT.

TABLE 1 The statistics of these 10 projects.

Project	Description	Release	Contributions	Comments	SATD	%of SATD
Apache Ant	Java library and command-line tool	1.7.0	74	4098	131	3.19
ArgoUML	UML modeling tool	0.34	87	9542	1413	14.81
Columba	E-mail client	1.4	9	6468	204	3.15
EMF	Eclipse model-driven architecture	2.4.1	30	4390	104	2.37
Hibernate	ORM framework	3.3.2	226	2968	472	15.90
JEdit	Text editor	4.2	57	10, 322	256	2.48
JFreeChart	Char library	1.0.19	19	4408	209	4.74
JMeter	Performance tester	2.10	33	8057	374	4.64
JRuby	Ruby interpreter	1.4.0	328	4897	622	12.70
SQuirrel	SQL client	3.0.3	46	7215	286	3.96
Average			91	6228	407	6.54

Abbreviation: SATD, self-admitted technical debts.

3.1 | Dataset description

To conduct our experiments, we utilize the dataset provided by Maldonado.[38] The steps to build this dataset are as follows: First, they use an Eclipse plug-in to extract developer comments from 10 open-source projects. Second, they use five heuristic filtering methods to remove comments that have little to do with SATD classification. Subsequently, they remove the duplicate comments, resulting in 62,276 comments that need to be manually annotated. Finally, they manually analyze each comment and label it SATD or non-SATD.

The detailed descriptions related to the dataset are shown in Table 1. This dataset contains categorized comments from 10 open-source Java projects, including Apache Ant, ArgoUML, Columba, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and SQuirrel. These 10 projects come from different application fields, and they have varying numbers of contributors and comments. The number and proportion of comments identified as SATD also differ among them. We can observe that in each project, only a small ratio of the comments are identified as SATD.

Due to the dataset's extensive project coverage, the richness of its comment types, and the diversity of its SATD samples, it has been widely used in researches for SATD detection.[29,30,38,42,63]

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

3.2 | Evaluation metrics

To evaluate the performance of different approaches, we employ three commonly used evaluation metrics following the previous studies,[29,30,42] namely Precision, Recall, and F1-score.

Precison represents the proportion of comments that are correctly classified as SATD among all comments classified as SATD.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}. \tag{1}$$

Recall represents the proportion of comments that are correctly classified as SATD among all SATD comments.

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}. \tag{2}$$

F1-score represents a harmonic mean of precision and recall, providing a balance between the two, which is especially useful when the ratio of SATD and non-SATD samples in the dataset is uneven.

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3}$$

Here, TP (true positive) represents the number of SATD comments that are classified as SATD; FP (false positive) represents the number of non-SATD comments that are classified as SATD; and FN (false negative) represents the number of SATD comments that are classified as non-SATD.

3.3 | Parameters

Regarding the use of the ChatGPT model, we choose the "gpt-3.5-turbo" version provided by OpenAI. There are some reasons why we do not adopt GPT-4. Recent reports and research[64–67] suggest that GPT-4 manifests indications of inconsistency and imprecision, possibly attributable to its radical redesign. In addition, using GPT-4 will greatly increase the cost of the experiment. In contrast, the GPT-3.5 series has consistently shown dependable outcomes, leading to its broad adoption. In this configuration, we set the "temperature" parameter to 0 to ensure complete determinism and consistency in the generated results. All other parameters are maintained at their default values.

As for these small models, as suggested by Ren et al.,[42] we set the sizes of the convolutional kernels as (2,3,4,5) and the number of kernels as 128 in the CNN model. Based on some related studied,[68,69] we use a Bi-LSTM with two network layers and 256 units in the hidden layer for the LSTM model and implement the Transformer model with the hidden layer of 1024, two encoders and five multihead attentions. The learning rate is set as 0.001.

Considering the unbalanced distribution of SATD and non-SATD comments, we introduce a weighted cross-entropy loss as the loss function for small models training. During the experiment, we calculate the weights based on the number of two categories in the training data. For example, assuming there are n SATD comments and m non-SATD comments in the training data, the weight for SATD is $n/(n+m)$ and the weight for non-SATD is $m/(n+m)$. These weights can then be used as parameters for the cross-entropy loss function.

We execute small-model training and inference procedures on a Linux operating system (Ubuntu 16.04 LTS), which is configured with 64GB RAM and a RTX2080Ti GPU. Our code implementation is based on the Pytorch[§] framework.

3.4 | Experimental process

We employ the cross-project scenario for SATD detection. In this scenario, we use nine projects as the source set and the other one project as the target set each time. In our work, we do not explore performance in within-project scenario for several reasons as follows: First, in the work of Ren et al.,[42] they focus on explore the performance under the cross-project

[§]https://pytorch.org/.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

scenario instead of within-project scenario. Second, in real-world applications, it is difficult and costly to label the existing comments of a project as the training set and it is not practical to constantly train the model every time we detect SATDs for an unknown project. Most importantly, improving the generalization performance in cross-project scenarios can improve the performance in within-project scenarios to a certain extent. Therefore, we focus on the study on cross-project scenario, using existing labeled data sets as training sets to improve the generalization performance of our approach for other unknown projects.

In cross-project scenario, we conduct our experiments in ten rounds, with one project tested in each round. For small models, we select 10% of the source set as the validation set every time, and the remainder as the training set; for ChatGPT, studies have show that more demonstration examples in the prompt generally lead to better performance.[57] Considering the limitations on ChatGPT's input tokens, we choose the maximum number of demonstrations from the source set to construct Few-shot-CoT prompt for the target comment from target set. We calculate the values of Precision, Recall, and F1-score across the 10 project experiments to evaluate the performance of different approaches.

The Wilcoxon signed-rank test[70] with Benjamini-Hochberg[71] (BH) correction and Cliff's δ[72] are utilized to evaluate the significance of performance difference between two different approaches. Both of them are nonparametric statistical methods used to assess the degree of difference between two sets of data. If BH-corrected p-value < 0.05 and $\delta > 0.147$,[70,73] it indicates a significant performance difference between the two approaches.

4 | RESULTS OF CHATGPT AND SMALL MODELS

To answer the first research question, we further the following two subresearch questions to evaluate the performance of ChatGPT and three small models, respectively.

- RQ1.1: Can ChatGPT be used for SATD detection and which prompting strategy is more effective?
- RQ1.2: How effective is ChatGPT in classification compared with small deep learning models and what are their respective advantages?

4.1 | Effectiveness of ChatGPT

4.1.1 | Motivations

To explore whether ChatGPT can be used for SATD detection, we utilize ChatGPT with with Few-shot-BM25 and Few-shot-CoT prompting strategies and compare them to choose the one with better performance. To better highlight the advantages of the advanced prompt strategy, we introduce Zero-shot prompts as a control group. Considering the time and economic costs of using model API provided by OpenAI, we randomly select 500 samples from each project, ensuring that the ratios of SATD and non-SATD samples remain the same as the original distribution, to minimize the impact that differing ratios of positive and negative samples may have on the results. It is worth stating that the number 500 is greater than the number calculated statistically at the 95% confidence level and 5% confidence interval.

4.1.2 | Methods

We analyze the performance results of ChatGPT on these three prompting strategies in cross-project scenario with 10 projects. Table 2 presents the detailed values of Precision, Recall, and F1-score metrics on each project. The best results are highlighted in bold and the worst results are underlined.

4.1.3 | Results

The results show that the performance of these two prompting strategies in the Precision metric is quite poor, and the gap between different projects is significant. However, they perform quite well on the Recall metric with best values of 1.0000

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 The classification results of ChatGPT on these prompting strategies.

	Zero-shot			Few-shot-BM25			Few-shot-CoT		
Project	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
Apache Ant	0.0850	0.8125	0.1538	0.1290	**1.0000**	0.2286	0.2586	0.9375	0.4054
ArgoUML	**0.5036**	0.9333	**0.6541**	**0.5441**	0.9868	**0.7014**	**0.6066**	0.9867	**0.7513**
Columba	0.1414	0.8750	0.2435	0.2133	**1.0000**	0.3516	0.2745	0.8750	0.4179
EMF	0.0991	0.9167	0.1787	0.1250	0.9167	0.2200	0.2895	0.9167	0.4400
Hibernate	0.3333	0.9000	0.4865	0.3596	0.9125	0.5159	0.5515	0.9375	0.6944
JEdit	0.1447	0.8462	0.2472	0.2000	0.9231	0.3288	0.5000	0.8462	0.6286
JFreeChart	0.2556	0.9593	0.4035	0.3333	0.9583	0.4956	0.3117	**1.0000**	0.4752
JMeter	0.1774	0.9167	0.2973	0.2759	**1.0000**	0.4324	0.3898	0.9583	0.5542
JRuby	0.3735	**0.9686**	0.5391	0.4354	**1.0000**	0.6066	0.5833	0.9844	0.7326
SQuirrel	0.1224	0.9000	0.2156	0.1638	0.9500	0.2794	0.2817	**1.0000**	0.4396
Average	0.2236	0.9028	0.3419	0.2779	0.9647	0.4160	0.4047	0.9442	0.5539

and an average value of 0.9647 and 0.9442, respectively. Compared to the results from the Zero-shot prompting strategy, the performance of these two advanced strategies has improved to some extent, with the performance of Few-shot-CoT being more notable. Comparing these two prompting strategies, we find that Few-shot-CoT strategy performs better than Few-shot-BM25 strategy on Precision and F1-score metrics, and there is little performance difference on Recall metric between the two strategies. Significantly, they both tend to predict the instances as SATD instead of non-SATD to different degrees.

Additionally, the Few-shot-CoT prompting strategy offers natural language analysis and explanation for the detection of technical debt, which is an advantage over the BM25-based few-shot prompting strategy. For instance, in Figure 2, in the first example, with the true label of "Yes," the outputs of ChatGPT with Few-shot-BM25 prompting or Few-shot-CoT prompting are both Yes, but the latter provides a correct explanation. In the second example, with the true label of "No," the output of ChatGPT with Few-shot-BM25 prompting is "Yes" which is incorrect, while the output of ChatGPT with Few-shot-CoT prompting is "No" which is correct and also provide correct analysis and explanation. In terms of interpretability, the latter strategy has more advantages than the former strategy. Therefore, in the experiment section below, we use ChatGPT with Few-shot-CoT prompting strategy.

In addition to the mentioned studies, we conduct an experiment to explore the effectiveness of ChatGPT's explanations for SATD instances. For this, we randomly select 500 true positive samples, 50 from each project, and then we invite five experienced developers engaged in software development to discuss and evaluate the correctness of the explanations provided by ChatGPT. Their average age is 28 years, including two students and three professional developers, with an average of 7 years of experience. It is worth stating that the number 500 is greater than the number calculated statistically at the 95% confidence level and 5% confidence interval. The outcome of this experiment is quite revealing. The findings indicate that all the explanations provided by ChatGPT are well founded and logical. This unanimous agreement among experienced developers suggests that ChatGPT's explanations for true positive SATD instances are highly effective and reliable.

Answer to RQ1.1: The Few-shot-CoT prompting is superior to the Few-shot BM25 in both validity and interpretability. Both of them perform very well on recall metric, but very poorly on precision metric. Therefore, the challenge lies in how to improve Precision of ChatGPT approach.

4.2 | Effectiveness of small models

4.2.1 | Motivations

To explore the effectiveness of small models and the characteristic of them, we implement these three small models (CNN, LSTM, and Transformer) and compare the results with results of ChatGPT.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

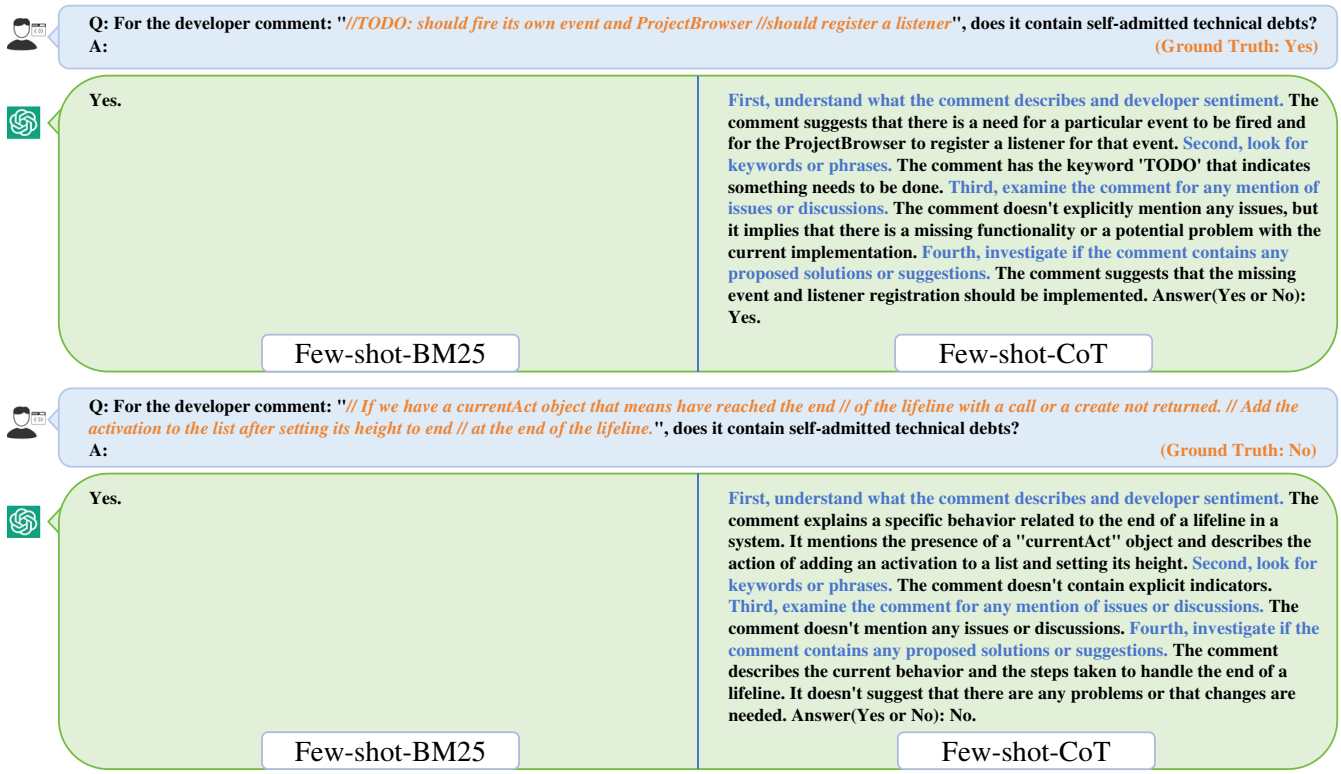

FIGURE 2 The Illustrations of different outputs of ChatGPT with Few-shot-BM25 prompting or Few-shot-CoT prompting.

TABLE 3 The classification results of three small models in cross-project scenario.

	CNN			LSTM			Transformer		
Project	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
Apache Ant	0.5354	0.5191	0.5271	0.4793	0.6183	0.5400	0.4810	0.5802	0.5260
ArgoUML	0.8696	**0.9038**	**0.8705**	0.8290	**0.9130**	**0.8690**	0.8491	0.8882	0.8682
Columba	0.7820	0.8088	0.7952	0.8017	0.9118	0.8532	**0.9282**	0.8873	**0.9073**
EMF	0.7761	0.5000	0.6082	0.6600	0.6346	0.6471	0.7067	0.5096	0.5921
Hibernate	0.8753	0.7881	0.8294	**0.8540**	0.8178	0.8355	0.9060	0.7966	0.8478
JEdit	0.6352	0.6055	0.6200	0.5709	0.6133	0.5913	0.7600	0.3711	0.4987
JFreeChart	0.8374	0.4928	0.6205	0.7192	0.5024	0.5915	0.6687	0.5215	0.5860
JMeter	**0.9038**	0.7032	0.7910	0.7840	0.7861	0.7850	0.8057	0.7540	0.7790
JRuby	0.7879	0.8778	0.8304	0.8376	0.8875	0.8618	0.8968	**0.9084**	0.9026
SQuirrel	0.7946	0.6224	0.6980	0.6851	0.7378	0.7104	0.8267	0.6503	0.7280
Average	0.7767	0.6822	0.7190	0.7221	**0.7423**	**0.7285**	**0.7829**	0.6867	0.7236

Abbreviations: CNN, convolutional neural network; LSTM, long-short term memory.

4.2.2 | Methods

We analyze the performance results of these three small models in cross-project scenario with 10 projects. Table 3 presents the results of these three small approaches for SATD detection. The best results are highlighted in bold and the worst results are underlined.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

4.2.3 | Results

We can find that the three models show similar performance for SATD detection, and LSTM achieves the best performance with the Precision, Recall, F1-score metrics, respectively, reaching 0.7221, 0.7423, and 0.7285. The average values of these three approaches also achieve impressive performance, respectively reaching 0.7606, 0.7037, and 0.7237 on three metrics. Compared with the performance of ChatGPT with Few-shot-CoT prompting strategy with the results of 0.4047, 0.9442, and 0.5539, respectively, on these metrics, small deep learning approaches are significantly superior in terms of Precision, while somewhat less effective on Recall.

This comparison highlights the varying strengths of different approaches in the SATD detection domain, with small deep learning models excelling in achieving higher Precision but lower Recall, whereas the ChatGPT with few-shot-CoT strategy shows a strong capacity in achieving a higher Recall but a lower Precision.

Therefore, it is conceivable to leverage the strengths of small models to enhance the Precision of the ChatGPT approach. By integrating the high Precision capabilities of small deep learning models with the comprehensive recall ability of the ChatGPT with Few-shot-CoT prompting strategy, we could potentially create a more balanced and effective approach for SATD detection. This synergy could lead to an approach that not only maintain high Recall but also improves Precision, offering a more robust solution in the field. Such an idea motivates us to propose the fusion approach introduced in the next section.

Answer to RQ1.2: Small models perform better on Precision metric, but relatively poorer on Recall metric compared with ChatGPT. Therefore, the challenge lies in how small deep learning models can be utilized to enhance the Precision of ChatGPT.

5 | OUR APPROACH

We propose a fusion approach combining ChatGPT with several small deep learning models, allowing the voting result of small models to be fused with the result of ChatGPT to improve performance on SATD detection task.

Algorithm 1 lists the detailed fusion steps of our approach FSATD. Firstly (Lines 1–4), given a target comment text x_t from test data set S_t, we follow the template of the few-shot-CoT prompting strategy to form a final valid prompt p_t and invoke ChatGPT model to generate the corresponding interpretation A_G and prediction result r_G (Yes or No, respectively representing SATD or non-SATD). Secondly (Lines 5–15), we input this comment text x_t into trained three small models to deduce the prediction results r_C, r_L, r_T. These results will then be combined through a full-voting ticket mechanism and lead to a voting result. Specifically, if the prediction results of the three small models are all SATDs (non-SATDs), the full-voting result r_v is SATD (non-SATD); otherwise, the full-voting result r_v is None (not valid). Most importantly (Lines 16-21), after obtaining the prediction result of ChatGPT r_G and the full-voting result r_v, we fuse the voting result r_v with the result of ChatGPT r_G to get the final fusion result. Due to the different capabilities and tendency to feature extraction of these three small models, we believe in the full-voting result of them when they reach a consensus about SATD classification to a great extent. Figure 3 illustrates the detailed fusion steps, which are as follows:

- If the **full-voting** result r_v of three small models exists, we choose to believe it as the final fusion result r_f.
- Otherwise, we choose to believe ChatGPT's result r_G as the final fusion result r_f.

If the final fusion result is inconsistent with ChatGPT's result, we will guide ChatGPT to correct the original result and provide a reasonable explanation for the fusion result.

6 | RESULTS OF OUR APPROACH

In this section, we further the following two sub research questions to measure the performance of our approach and answer RQ2.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Algorithm 1. FSATD

Input : A comment text x_t from test dataset S_t
Output: A fusion result r_f

Construct the valid prompt p_t for x_t using few-shot-CoT prompt template:
p_t = constructPropmpt(x_t)
Input p_t into ChatGPT to generate analyze A_G and prediction result r_G:
A_G, r_G = invoke("gpt-3.5-turbo," p_t)
Input x_t into trained CNN, LSTM, Transformer to, respectively, get prediction results r_C, r_L, r_T:
r_C = CNN(x_t)
r_L = LSTM(x_t)
r_T = Transformer(x_t)
Get the full-voting result r_v from r_C, r_L, r_T:
if r_C *and* r_L *and* r_T **then**
    r_v = 1 #representing SATD
if *!*(r_C *or* r_L *or* r_T) **then**
    r_v = 0#representing non-SATD
else
    r_v = None
Fuse r_G with r_v to get the fusion result r_f:
if r_v==*0 or* r_v==*1* **then**
    r_f=r_v
else
    r_f=r_G
return the fusion result r_f

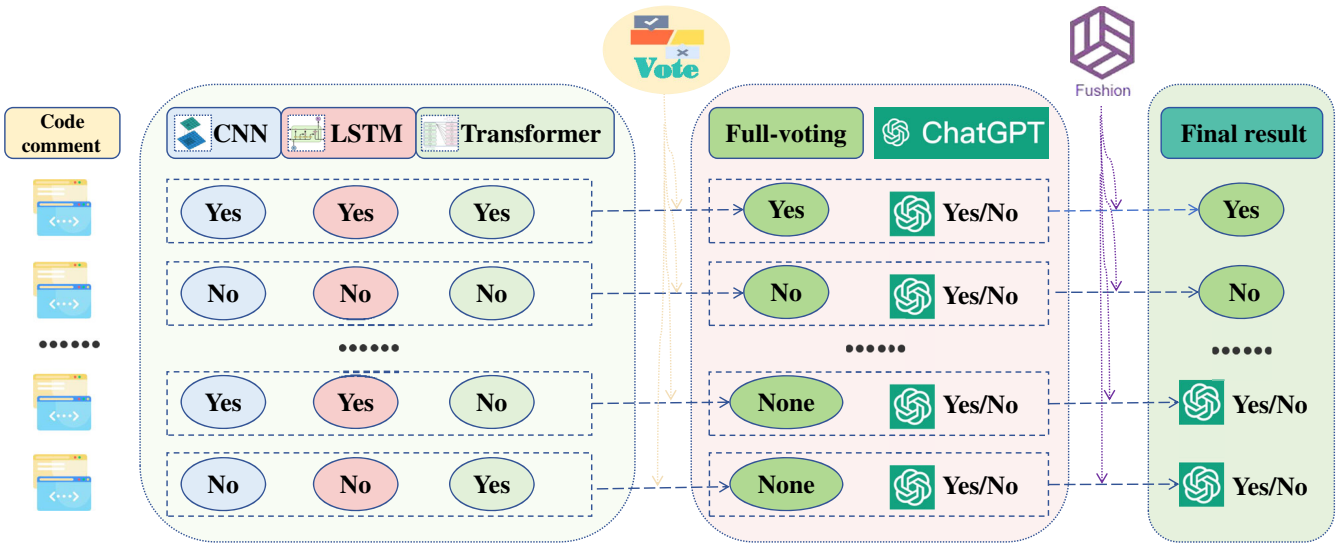

FIGURE 3 The fusion steps of our approach for self-admitted technical debts (SATD) detection. Given a target comment text, we obtain the final fusion result by fusing the result of ChatGPT with the full-voting result of three small models.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

- RQ2.1: Does our proposed approach outperform other existing methods for SATD detection?
- RQ2.2: How effective are the different fusion strategies?

6.1 | Answer to RQ2.1: performance comparison with other existing approaches for SATD detection

6.1.1 | Motivations

We propose the fusion approach FSATD, which fuses the result of ChatGPT with the full-voting result of three small models to get the final fusion result. To verify the effectiveness of FSATD, we compare it with seven existing approaches, which are introduced as follows:

- **GGSATD** is proposed by Yu et al.,[30] which utilizes the gated graph neural network to iteratively update node representation and employs the representation layer incorporating multilayer perceptrons and pooling mechanisms to obtain the graph level representation to detect SATDs.
- **HATD** is proposed by Wang et al.,[29] which combines a two-layer Bi-LSTM with single-head attention to form a single-head attention encoder, and uses positional encoding plus multihead attention for a multihead attention encoder. Then representations of two encoders are merged to detect SATD. GGSATD and HATD are the current state-of-the-art approaches.
- **Majority_voting** (M_v), a decision-making method, is often employed in collective decision-making or classification tasks. In the context of SATD detection, three small models classify the same data point, and the final category is determined through majority voting.
- **CNN**, **LSTM**, **Transformer**, and **Few-shot-CoT** are already introduced in Section 2.

6.1.2 | Methods

We analyze the performance results of FSATD and other existing methods in cross-project scenario with 10 projects and the significance of differences between FSATD and other methods. Table 4 presents the Precision, Recall, and F1-score values from several different methods in cross-project scenario, respectively, as well as p-value after BH correction and Cliff's δ between FSATD and other methods. The best results are highlighted in bold. Additionally, by repeating experiments multiple times, we calculate and summarize the standard deviation, denoted as σ, of FSATD on three evaluation metrics.

Results: From Table 4, the average values of the ten projects of our approach FSATD on three metrics reach 0.8591 for Precision, 0.8697 for Recall, and 0.8661 for F1-score. Compared with the performance of ChatGPT, which demonstrates high Recall but low Precision, our approach achieves a very balanced performance. In addition, regarding the results of our approach FSATD on each individual project, we find that the gap between the values of different projects is narrow, which demonstrates the excellent stability of FSATD across various projects. This balance and the consistency in achieving relatively uniform results across different projects are crucial in the context of software engineering, especially for SATD detection, as it ensures that our approach not only accurately identifies relevant instances (high Precision) but also covers a broad range of applicable cases (high Recall) in different projects, leading to a comprehensive and reliable detection capability and demonstrating the effectiveness of our approach across diverse software projects.

From Table 4a, our approach FSATD achieves the highest Precision value in two projects, with an overall average value of 0.8591 in 10 projects which is slightly lower than GGSATD of 0.879. Compared to other approaches such as HATD, CNN, LSTM, Transformer, and Majority_voting, our approach shows an improvement of 19.49%, 10.61%, 18.97%, 9.73%, and 0.02%, respectively. When compared specifically to the small model components including CNN, LSTM, and Transformer, our approach shows an average improvement of 13.01%. Against the ChatGPT with Few-shot-CoT prompting strategy, there is a substantial improvement of 114.50%. The integration of small model components has significantly enhances the Precision of the ChatGPT with Few-shot-CoT prompting strategy. This substantial improvement in Precision vividly demonstrates the effectiveness of FSATD. By leveraging the strengths of both the small models and the advanced prompting capabilities of ChatGPT, our approach effectively addresses one of the key limitations—low Precision—previously

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 4 The evaluation results of our approach and other existing approaches in cross-project scenario.

(A) Precision(σ)

Project	FSATD	GGSATD	HATD	CNN	LSTM	Transformer	M_v	Few-shot-CoT
Apache Ant	0.7161(0.0161)	**0.806**	0.657	0.5354	0.4793	0.4810	0.7426	0.2546
ArgoUML	0.8644(0.0223)	**0.909**	0.818	0.8396	0.8290	0.8491	0.8564	0.6001
Columba	0.9000(0.0195)	**0.936**	0.794	0.7820	0.8017	0.9282	0.8916	0.3019
EMF	0.8316(0.0334)	0.823	0.664	0.7761	0.6600	0.7067	**0.8710**	0.2578
Hibernate	0.9231(0.0265)	**0.927**	0.756	0.8753	0.8540	0.9060	0.9126	0.6046
JEdit	**0.9420**(0.0272)	0.864	0.699	0.6352	0.5709	0.7600	0.7512	0.4442
JFreeChart	0.7608(0.0235)	0.831	0.678	0.8374	0.7192	0.6687	**0.8750**	0.3169
JMeter	0.8528(0.0116)	0.905	0.701	0.9038	0.7840	0.8057	**0.9276**	0.3784
JRuby	0.8987(0.0254)	**0.938**	0.758	0.7879	0.8376	0.8968	0.8835	0.5652
SQuirrel	**0.9018**(0.0134)	0.851	0.671	0.7946	0.6851	0.8267	0.8610	0.2814
Average	0.8591(0.0219)	**0.879**	0.719	0.7767	0.7221	0.7829	0.8573	0.4005
p-Value	—	0.3760	0.0137	0.0684	0.0137	0.0239	1.0000	0.0137
δ	—	−0.12	0.88	0.46	0.72	0.40	0.06	1.00

(B) Recall(σ)

Project	FSATD	GGSATD	HATD	CNN	LSTM	Transformer	M_v	Few-shot-CoT
Apache Ant	0.8473(0.0182)	0.818	0.780	0.5191	0.6183	0.5802	0.5725	**0.9542**
ArgoUML	0.9837(0.0271)	0.946	0.925	0.9038	0.9130	0.8882	0.9200	**0.9887**
Columba	**0.9706**(0.0253)	0.942	0.956	0.8088	0.9118	0.8873	0.8873	0.9412
EMF	0.7596(0.0255)	0.786	0.785	0.5000	0.6346	0.5096	0.5192	**0.8750**
Hibernate	0.8321(0.0146)	0.894	0.839	0.7881	0.8178	0.7966	0.7966	**0.9428**
JEdit	0.7617(0.0126)	0.720	0.713	0.6055	0.6133	0.3711	0.5898	**0.9023**
JFreeChart	0.7608(0.0254)	0.753	0.748	0.4928	0.5024	0.5215	0.4689	**0.9856**
JMeter	0.8984(0.0192)	0.899	0.871	0.7032	0.7861	0.7540	0.7540	**0.9278**
JRuby	**0.9839**(0.0195)	0.914	0.913	0.8778	0.8875	0.9084	0.9019	0.9759
SQuirrel	0.8986(0.0169)	0.820	0.851	0.6224	0.7378	0.6503	0.6713	**0.9720**
Average	0.8967(0.0204)	0.849	0.838	0.6822	0.7426	0.6867	0.7082	**0.9466**
P-value	-	0.1309	0.0273	0.0137	0.0137	0.0137	0.0137	0.0273
δ	-	0.16	0.18	0.64	0.50	0.60	0.56	-0.52

(C) F1-score(σ)

Project	FSATD	GGSATD	HATD	CNN	LSTM	Transformer	M_v	Few-shot-CoT
Apache Ant	0.7762(0.0161)	**0.811**	0.713	0.5271	0.5400	0.5260	0.6465	0.4019
ArgoUML	0.9202(0.0182)	**0.926**	0.855	0.8705	0.8690	0.8682	0.8871	0.7469
Columba	0.9340(0.0229)	**0.939**	0.848	0.7952	0.8532	0.9073	0.8994	0.4571
EMF	0.7940(0.0295)	**0.802**	0.669	0.6082	0.6471	0.5921	0.6506	0.3982
Hibernate	**0.9191**(0.0224)	0.909	0.776	0.8294	0.8355	0.8478	0.8507	0.7368
JEdit	**0.8423**(0.0176)	0.782	0.671	0.6200	0.5913	0.4987	0.6608	0.5954
JFreeChart	0.7608(0.0246)	**0.786**	0.703	0.6205	0.5915	0.5860	0.6106	0.4796
JMeter	0.8750(0.0117)	**0.902**	0.747	0.7910	0.7850	0.7790	0.8319	0.5376
JRuby	**0.9394**(0.0225)	0.926	0.798	0.8304	0.8618	0.9026	0.8926	0.7158
SQuirrel	**0.9002**(0.0157)	0.834	0.707	0.6980	0.7104	0.7280	0.7544	0.4364
Average	**0.8661**(0.0201)	0.862	0.749	0.7190	0.7285	0.7236	0.7685	0.5506
P-value	-	1.0000	0.0137	0.0137	0.0137	0.0137	0.0137	0.0137
δ	-	0.00	0.76	0.70	0.66	0.56	0.56	1.01

Abbreviations: CNN, convolution neural network; LSTM, long short-term memory.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

observed in the ChatGPT with Few-shot-CoT strategy. In addition, from p-value after BH correction and δ, it indicates that FSATD has significant performance differences with HATD, LSTM, Transformer and Few-shot-CoT approaches on Precision metric. Although FSATD is slightly lower than GGSATD on Precision, it still achieves excellent performance and does not have significant differences with GGSATD.

From Table 4b, our approach FSATD achieves the best Recall in two projects, with an overall average value of 0.8967 in 10 projects, which presents a slight decrease compared to the ChatGPT with Few-shot-CoT prompting strategy. However, when compared to the state-of-the-art approaches, GGSATD and HATD, our approach shows an increase of 5.62% and 7.00%, respectively. Relative to other approaches such as CNN, LSTM, Transformer, and Majority_voting, our approach shows an increase of 31.44%, 20.75%, 30.58%, and 26.62%, respectively. When specifically compared to small model components like CNN, LSTM, and Transformer, our approach shows an average improvement of 27.5%. These results indicate that while our approach slightly trails the ChatGPT with Few-shot-CoT prompting strategy in terms of Recall, it significantly outperforms other current approaches, including the state-of-the-art approaches, GGSATD and HATD. The substantial improvement highlights the strength of our approach in maintaining a high level of Recall, which is essential for ensuring comprehensive coverage in SATD detection. This balance of high Recall with the previously noted improvement in Precision demonstrates the overall effectiveness and robustness of our approach, making it a valuable contribution to the field. In addition, from p-value after BH correction and δ, it indicates that FSATD has significant performance differences with all the other approaches on Recall metric.

From Table 4c, our approach FSATD achieves the best F1-score in four projects, with an overall average value of 0.8967 in 10 projects, surpassing all existing approaches. Compared to the state-of-the-art approaches, GGSATD and HATD, our approach shows an increase of 0.5% and 15.63%, respectively. Relative to other approaches such as CNN, LSTM, Transformer, and Majority_voting, our approach demonstrates an improvement of 20.46%, 18.89%, 19.69%, and 12.70%, respectively. When compared specifically to small model components such as CNN, LSTM, and Transformer, there is an average improvement of 19.68%. Against the ChatGPT with Few-shot-CoT prompting strategy, our approach shows a substantial increase of 57.06%. These results highlight the significant impact of our approach in improving the overall F1-score, a crucial metric that combines Precision and Recall, indicating a well-balanced and effective approach. Achieving the best F1-score in multiple projects and outperforming both small models and ChatGPT with advanced Few-shot-CoT prompting strategy confirms the superiority of our approach in providing a comprehensive solution for SATD detection. This high level of performance across diverse projects showcases the robustness and applicability of our approach in various software engineering contexts. In addition, from p-value after BH correction and δ, it indicates that FSATD has significant performance differences with other approaches apart from GGSATD on F1-score metric.

From the overall results, the performance of FSATD on some projects, such as Apache Ant and JFreeChart, is not ideal. This could be due to several possible reasons: On one hand, the language features of the comments in these projects are not commonly seen in other projects. After training on other projects, the small models may struggle to capture the linguistic characteristics of these comments, potentially leading to entirely incorrect predictions by the smaller models and subsequently erroneous results when fused with the result of ChatGPT. On the other hand, ChatGPT tends to predict non-SATD instances as SATD, resulting in extremely low precision and thus suboptimal fusion results.

The error could potentially come from two aspects: the training and inference process of small models, and the inference process of the large model. In the training process of the small model, we initialize it with a fixed random seed and save the best model weights from 100 epochs for inference. After repeating experiments multiple times, the inference results remain stable and consistent, thus not contributing significantly to the error. However, the inference process of the large model has greater randomness, though we set the "temperature" parameter to 0 to ensure as much determinism and stability in the inference results as possible. Therefore, we repeat inference experiments of ChatGPT three times, calculate the average values, and present the SD σ. From the values of σ, we can see that results of all projects are not very volatile.

Answer to RQ2.1: Our approach can achieve a balanced performance with the best F1-score compared with existing approaches.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

6.2 | Answer to RQ2.2: effectiveness of the different fusion strategies

6.2.1 | Motivations

To explore the effectiveness of different fusion strategies, we set up four different fusion strategies as follows, which cover all combinations of voting mechanisms. The detailed fusion steps are presented in Figure 4 and described as follows:

- **Strategy 1**: utilized in our approach and presented in Figure 3.
- **Strategy 2**: If the **majority-voting** result of three small models is **No**, we choose to believe it as the fusion result; if the **full-voting** result of three small models is **Yes**, we choose to believe it as the fusion result; otherwise, we choose to believe ChatGPT's result as the fusion result.
- **Strategy 3**: If the **full-voting** result of small models is **No**, we choose to believe it as the fusion result; if the **majority-voting** result of three small models is **Yes**, we choose to believe it as the fusion result; otherwise, we choose to believe ChatGPT's result as the fusion result.
- **Strategy 4**: We always believe in the **majority-voting** result of three small models as the final fusion result.

6.2.2 | Methods

We analyze the performance results of four different fusion strategies in cross-project scenario with 10 projects and the significance of differences between Strategy 1 and other strategies. Table 5 presents the Precision, Recall, and F1-score values from four different fusion strategies in cross-project scenario as well as p-value after BH correction and Cliff's δ between Strategy 1 and the other strategies. The best results are highlighted in bold.

6.2.3 | Results

From Table 5a, Strategy 2 achieves the highest Precision with 10 projects, with an overall average of 0.9795; this is nearly 12 percentage points higher than our employed Strategy 1. From Table 5b, it is evident that Strategy 3 attains the best

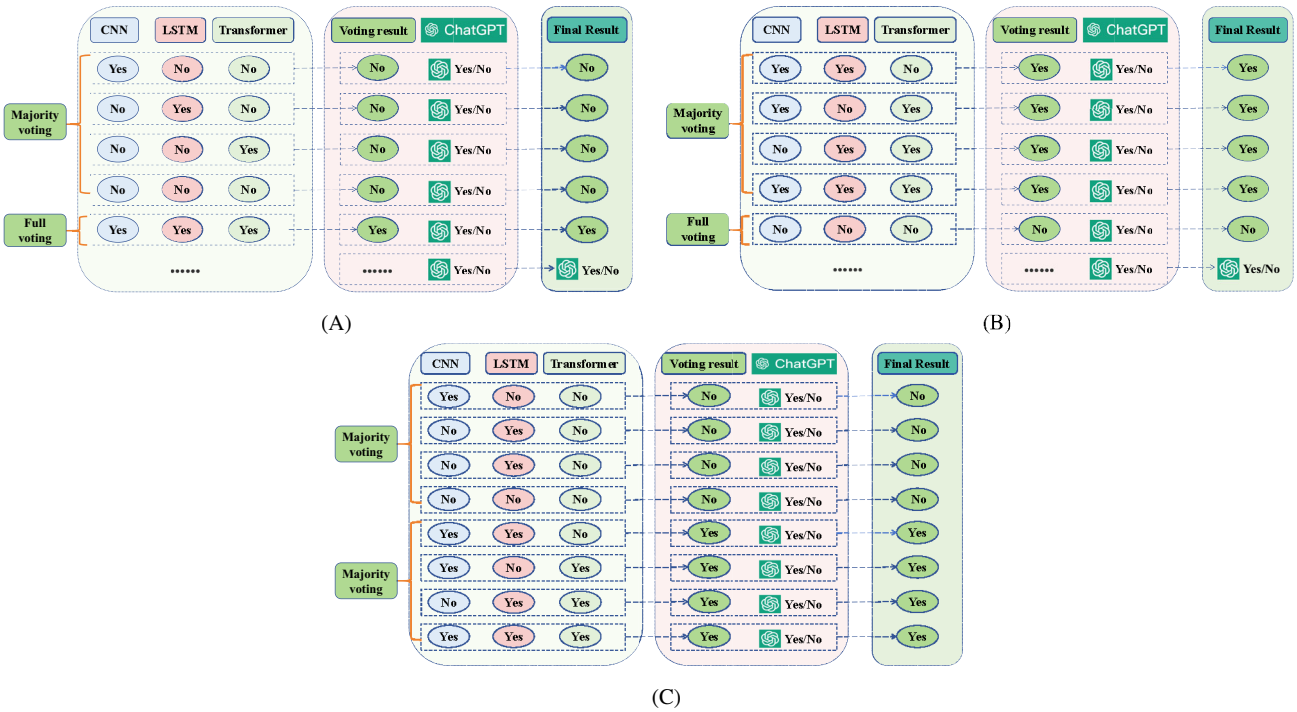

FIGURE 4 The fusion steps of the other three fusion strategies. (A) Strategy 2, (B) Strategy 3, (C) Strategy 4.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 5 The evaluation results of four different fusion strategies.

(a) Precision				
Project	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Strategy 4**
Apache Ant	0.7161	**0.9733**	0.6243	0.7426
ArgoUML	0.8644	**0.9707**	0.7791	0.8564
Columba	0.9000	**0.9835**	0.8299	0.8916
EMF	0.8316	**0.9808**	0.7810	0.8710
Hibernate	0.9231	**0.9917**	0.8673	0.9126
JEdit	0.9420	**0.9797**	0.7731	0.7512
JFreeChart	0.7608	**0.9703**	0.7228	0.8750
JMeter	0.8528	**0.9891**	0.8184	0.9276
JRuby	0.8987	**0.9772**	0.8260	0.8835
SQuirrel	0.9018	**0.9791**	0.8265	0.8610
Average	0.8591	**0.9795**	0.7848	0.8573
p-Value	—	0.0059	0.0059	1.0000
δ	—	−1.0	0.6	0.06
(b) Recall				
Project	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Strategy 4**
Apache Ant	0.8473	0.5573	**0.8626**	0.5725
ArgoUML	0.9837	0.9130	**0.9908**	0.9200
Columba	0.9706	0.8775	**0.9804**	0.8873
EMF	0.7596	0.4904	**0.7885**	0.5192
Hibernate	0.8321	0.7564	**0.9555**	0.7966
JEdit	0.7617	0.5664	**0.7852**	0.5898
JFreeChart	**0.7608**	0.4689	**0.7608**	0.4689
JMeter	0.8984	0.7246	**0.9278**	0.7540
JRuby	0.9839	0.8939	**0.9920**	0.9019
SQuirrel	0.8986	0.6538	**0.9161**	0.6713
Average	0.8697	0.6902	**0.8960**	0.7082
p-Value	-	0.0059	0.0077	0.0059
δ	-	0.66	-0.23	0.56
(c) F1-score				
Project	**Strategy 1**	**Strategy 2**	**Strategy 3**	**Strategy 4**
Apache Ant	**0.7762**	0.7087	0.7244	0.6465
ArgoUML	0.9202	**0.9410**	0.8723	0.8871
Columba	**0.9340**	0.9275	0.8989	0.8994
EMF	**0.7940**	0.6538	0.7847	0.6506
Hibernate	**0.9191**	0.8582	0.9093	0.8507
JEdit	**0.8423**	0.7178	0.7791	0.6608
JFreeChart	**0.7608**	0.6323	0.7413	0.6106
JMeter	**0.8750**	0.8364	0.8697	0.8319
JRuby	**0.9394**	0.9337	0.9014	0.8926
SQuirrel	**0.9002**	0.7841	0.8690	0.7544
Average	**0.8661**	0.7994	0.8350	0.7685
p-Value	-	0.0098	0.0020	0.0020
δ	-	0.30	0.34	0.56

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Recall in 10 projects, with an overall average of 0.8960; this is slightly over 2.6 percentage points higher than Strategy 1. Finally, from Table 5c, we find that Strategy 1 achieves the best F1-score in nine projects, with an overall average of 0.8661. From p-value after BH correction and δ, it indicates that Strategy 1 has significant performance differences with other strategies on Recall and F1-score metrics, while Strategy 1 has significant performance differences with other strategies apart from Strategy 4 on Precision metric.

These results highlight the differing impacts of various fusion strategies on the performance metrics in a cross-project scenario. Strategy 2's significant lead in Precision suggests its effectiveness in accurately identifying relevant cases, while Strategy 3's superior performance in Recall indicates its strength in covering a wide range of applicable cases. Our chosen Strategy 1, while not leading in individual metrics of Precision or Recall, demonstrates a well-balanced approach, as evidenced by its leading performance in F1-score across the majority of projects. This balance is critical in practical applications where false alarm rates (Precision) and comprehensiveness (Recall) are important. The high F1-score of Strategy 1 underscores its effectiveness as a comprehensive and reliable approach for SATD detection in varied project contexts. Actually in different application scenarios, facing varying requirements, we can indeed adopt different fusion strategies. This flexibility is crucial as it allows for the tailoring of approaches to meet specific objectives or constraints of each scenario.

Answer to RQ2.2: Strategy 1 achieves the best F1-score; Strategy 2 achieves the best Precision; Strategy 3 achieves the best Recall. In summary, different fusion strategies indeed possess distinct advantages and the choice of fusion strategy can be flexibly made based on the specific requirements of different application scenarios.

7 | DISCUSSION

7.1 | Why does our approach work?

The full-voting results of the small model are reliable and the advantages of ChatGPT and small models can be effectively combined. To explore the synergy of predictions from small models and to understand the rationale behind fusing the voting results of these models, we create Venn diagrams depicting the sets of correctly predicted results by the three small models for samples with ground truth labels of "non-SATD" and "SATD," respectively. Each Venn diagram contains seven sections, representing that (1) only CNN, (2) only LSTM, (3) only Transformer, (4) both CNN and LSTM, (5) both CNN and Transformer, (6) both LSTM and Transformer, and (7) all three models correctly predict instances in the dataset to be SATDs or non-SATDs. The diagrams display the number of instances in each of these seven sections.

Figure 5 presents the Venn diagrams of the sets of correctly predicted results by CNN, LSTM, and Transformer with groundtruth labels of "non-SATD." It is evident that the section where three models reach a full-voting consensus occupies the majority of the Venn diagram, which indicates that when the groundtruth label is "non-SATD," the three small models tend to predict "non-SATD" simultaneously, demonstrating a high accuracy rate for this category of labels. In addition, from the two numbers inside the parentheses in Figure 5, the number of non-SATD predicted by the three small models is almost equal to the number of non-SATD in the dataset, indicating that the small models rarely miss identifying non-SATD samples. For example, in dataset Apache Ant, Columba, EMF, Hibernate, and JFreeChart, these three small models even do not miss identifying any non-SATD instances. This observation suggests a strong consensus among the small models and a high coverage for identification of non-SATD instances in dataset, implying that their full-voting results are particularly reliable for identifying non-SATD instances which can help enhance the Precision for SATD classification.

Figure 6 presents the Venn diagrams of the sets of correctly predicted results by CNN, LSTM, and Transformer with groundtruth labels of "SATD." From this diagram, we observe that the synergy among the three small models is not as pronounced, which suggests that their synergetic effectiveness in accurately identifying SATD instances is not as high and they tend to miss identifying SATD instances in datasets. In these cases where the small models do not reach a full-voting consensus, we choose to believe ChatGPT's results. From Section 4, we find that ChatGPT has the advantages of achieving a very high Recall which indicates that it has very strong ability to correctly identify SATD instances. These advantages of ChatGPT can compensate for the low consistency and high underreporting rate of small models for identifying SATD samples, which reinforces the reliability of our approach to fuse ChatGPT with small models.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

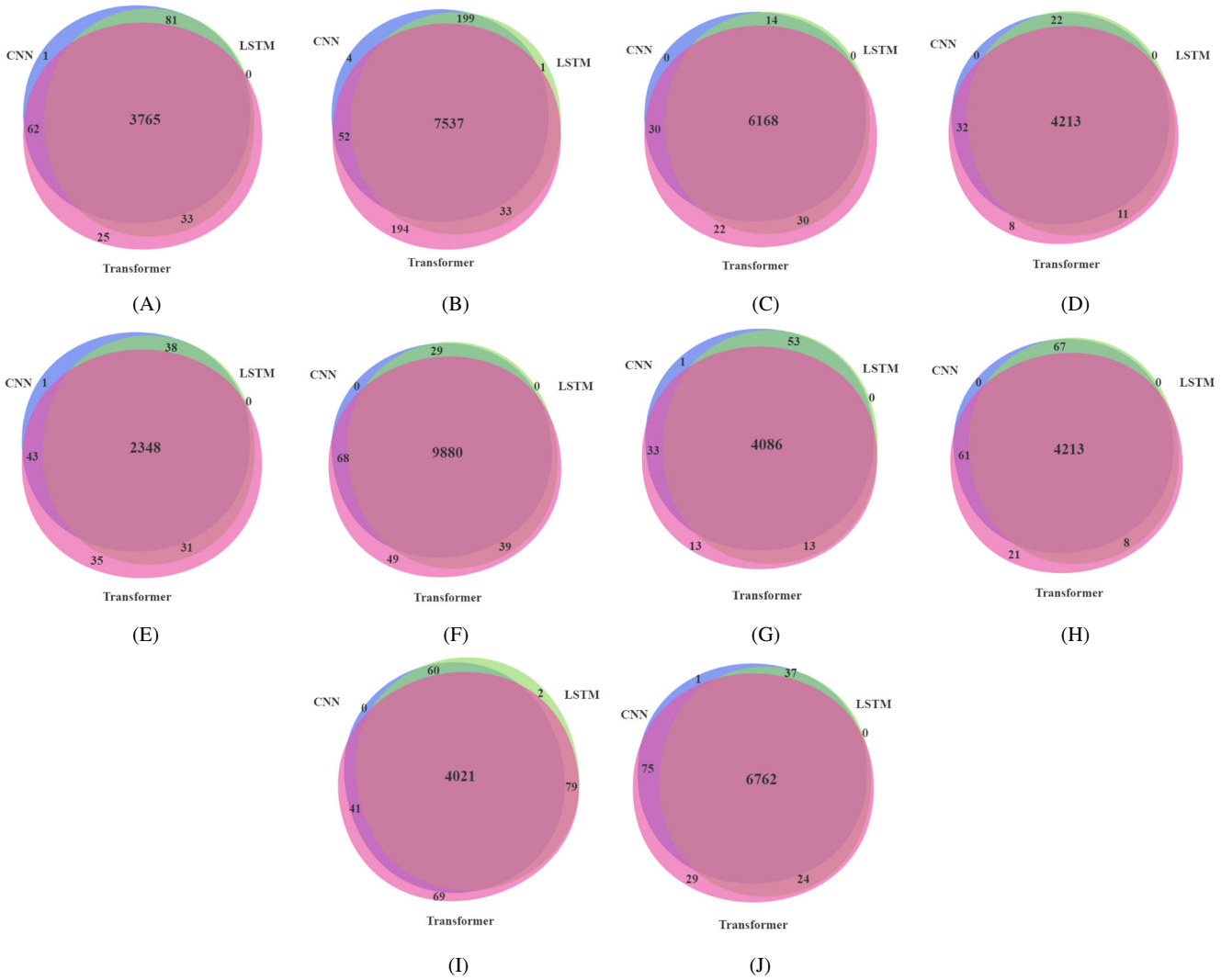

FIGURE 5 The Venn diagrams of the sets of correctly predicted results by convolutional neural network (CNN), long short-term memory (LSTM), and Transformer with the ground truth of "non-SATD." Blue represents "only CNN," yellow-green represents "only LSTM," pink represents "only Transformer," and so forth. The two numbers in parentheses respectively indicate the number of non-SATD samples correctly predicted by small models and the number of non-SATD samples included in the dataset. (A) Apache Ant (3967 / 3967), (B) ArgoUML (8020 / 8039), (C) Columba (6264 / 6264), (D) EMF (4286 / 4286), (E) Hibernate (2496 / 2496), (F) JEdit (10065 / 10066), (G) JFreeChart (4199 / 4199), (H) JMeter (7682 / 7683), (I) JRuby (4272 / 4275), (J) SQuirrel (6828 / 6929).

Instances of correct fusion are much more numerous than instances of false fusion, and occupy a much larger proportion. To quantitatively analyze the effectiveness of our fusion strategy, we compile data on the number of successes and failures for two different cases of Strategy 1. Table 6 presents four possible fusion scenarios. Why are there these four fusion scenarios? On one hand, there are two cases in which ChatGPT's result is inconsistent with the full-voting result of small models. On the other hand, there are two different groundtruths. By permutation and combination, we can get these four fusion scenarios. Table 7 presents the number of instances of four fusion scenarios in 10 projects. By analyzing the number of instances, we can intuitively see a significant difference between correct fusion and false fusion.

From Table 7, we observe that in the 10 projects, the number of correct fusion instances in Scenario 1 and Scenario 2 achieved by our strategy is 38 and 4229, which is significantly higher than the number of false fusion instances in Scenario 3 and Scenario 4, respectively 188 and 8, totaling 196. Significantly, it is evident that instances of correct fusion occupy a large proportion of all instances in Scenario 2, which can greatly enhance the Precision in SATD classification. By contrast, instances of false fusion in Scenario 3 are rare, which slightly reduce the Recall in SATD classification. In addition, instances in Scenarios 1 and 4 are rarer, which have little impact on the performance of our approach.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

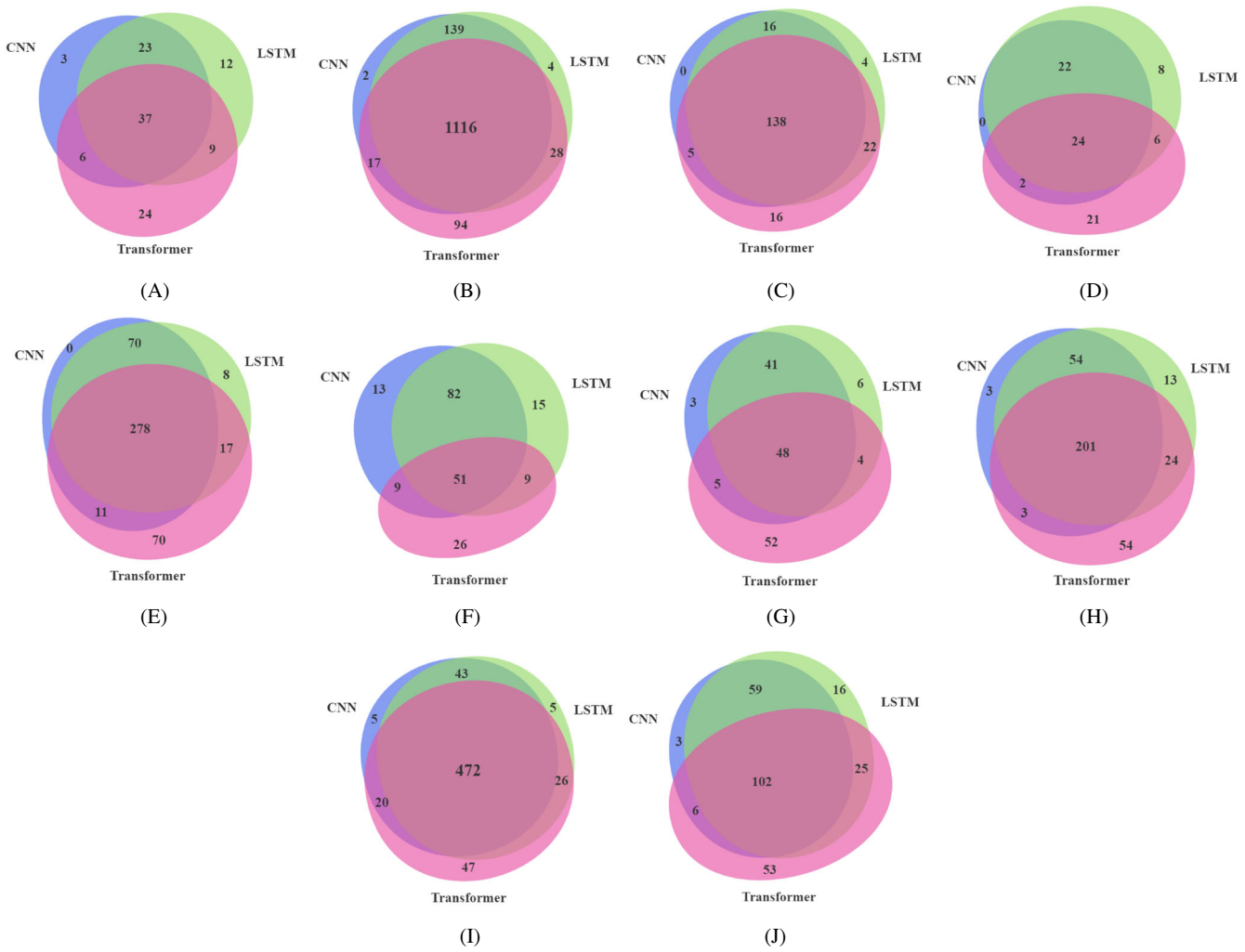

FIGURE 6 The Venn diagrams of the sets of correctly predicted results by convolutional neural network (CNN), long short-term memory (LSTM), and Transformer with ground truth of self-admitted technical debts ("SATD"). Blue represents "only CNN," yellow-green represents "only LSTM," pink represents "only Transformer," and so forth. The two numbers in parentheses, respectively, indicate the number of SATD samples correctly predicted by small models and the number of SATD samples included in the dataset. (A) Apache Ant (114/131), (b) ArgoUML (1400/1413), (C) Columba (201/204), (D) EMF (83/104), (E) Hibernate (454/472), (F) JEdit (205/256), (G) JFreeChart (159/209), (H) JMeter (352/374), (I) JRuby (618/622), (J) SQuirrel (264/286).

TABLE 6 Four possible fusion scenarios in Strategy 1.

		Prediction				
	Groundtruth	**ChatGPT**	**CNN**	**LSTM**	**Transformer**	**Fusion result**
Scenario (1)	Yes	No	Yes	Yes	Yes	Yes (✓)
Scenario (2)	No	Yes	No	No	No	No (✓)
Scenario (3)	Yes	Yes	No	No	No	No (×)
Scenario (4)	No	No	Yes	Yes	Yes	Yes (×)

Notes: "Yes" represents "SATD" and "No" represents "non-SATD." (✓) represents correct fusion in this scenario, and (×) represents false fusion in this scenario.
Abbreviations: CNN, convolution neural network; LSTM, long short-term memory.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 7 The number of instances of four fusion scenarios in 10 projects.

	Scenario			
Project	**(1) (✓)**	**(2) (✓)**	**(3) (×)**	**(4) (×)**
Apache Ant	0 (6)	322 (366)	14	0
ArgoUML	5 (16)	720 (931)	12	7
Columba	8 (12)	422 (444)	2	0
EMF	5 (13)	246 (262)	17	0
Hibernate	1 (27)	255 (291)	14	0
JEdit	4 (25)	277 (289)	40	0
JFreeChart	0 (3)	394 (444)	47	0
JMeter	5 (27)	512 (570)	16	0
JRuby	9 (15)	398 (467)	4	0
SQuirrel	1 (8)	683 (710)	22	1
Total	38 (152)	4229 (4774)	188	8

Notes: (✓) represents correct fusion in this scenario, and (×) represents false fusion in this scenario. The numbers in parentheses represent the number of instances that meet the GroundTruth and ChatGPT predictions in this scenario.

This significant difference between correct and false fusion instances highlights several key points:

- High effectiveness of the fusion strategy: The dominance of correct fusions indicates that the strategy is highly effective for most instances, successfully leveraging the strengths of the combined models.
- Reliability across different projects: The consistency of this trend across 10 different projects suggests that the strategy is robust and reliable, capable of performing well in diverse scenarios.
- Validation of the fusion: The overwhelming success of the fusion strategy validates the concept of fusing models to enhance performance, especially in complex tasks like SATD detection.

7.2 | Some inaccurate labeled data

During the course of our experiments, we identify some instances that appear to be mislabeled. Figure 7 shows two comment examples containing the keywords “Fixme” and “TODO,” and both of two explanations provided by ChatGPT accurately refer to these key words that indicate SATD. Both comment instances are predicted as Yes by the ChatGPT with few-shot-CoT prompting strategy, and the majority-voting results from the small models also indicate Yes. In addition, five invited experienced developers have a secondary review of the analysis and explanation provided by ChatGPT and find that ChatGPT’s explanations are logical and well-founded. Therefore, it is possible that these instances are incorrectly labeled in the dataset. This finding indicates that ChatGPT can help improve the reliability of data labels, as human reviewers can re-evaluate and verify labels using the explanations provided by AI models.

7.3 | Threats of validity

7.3.1 | Unbalanced data

An unbalanced dataset may bring a significant risk to our approach. To deal with the problem, we utilize a weights cross-entropy loss as the loss function for small models training. It can improve the recognition ability of the model on small-class instances, thereby improving the overall performance and fairness of the model. Therefore, our results are still reliable.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

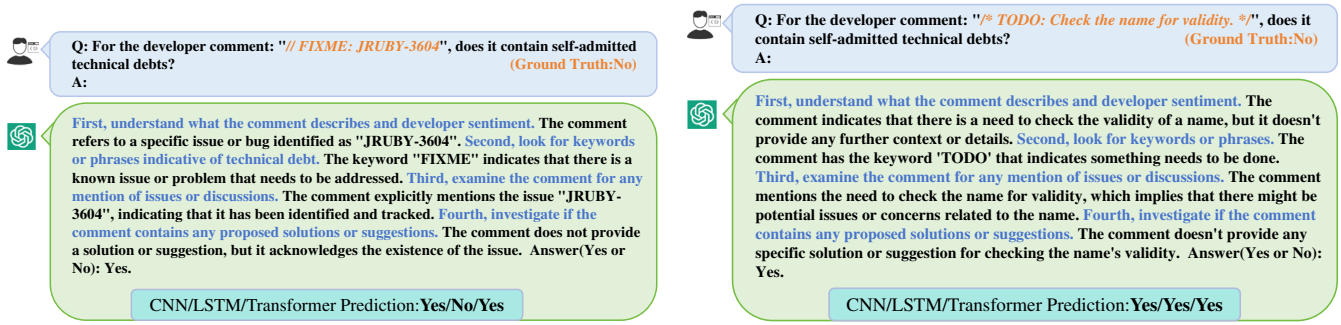

FIGURE 7 Two incorrectly labeled instances containing keywords indicative of self-admitted technical debts (SATD). (A) Example containing the keyword of "Fixme," (B) Example containing the keyword of "TODO."

7.3.2 | Small models

We focus on selecting basic small models with wide applications rather than state-of-the-art models which are designed elaborately to solve specific problems and may lead to overfitting. The small models we selected have distinct mechanisms for capturing text features, such as capturing positional features, understanding long-term dependencies and providing a comprehensive understanding of context, making them suitable for various aspects of text classification. Therefore, we believe that our approach can generalize to more extensive text classification tasks.

7.3.3 | Data leakage

Data leakage is a potential concern in our approach, due to utilizing ChatGPT, a large language model pretrained on a vast corpus. Data leakage refers to the possibility that ChatGPT, due to its training on extensive and diverse datasets, might have been exposed to specific content or patterns that it could later recognize during its application in SATD detection. This recognition is not due to the model's learning and generalizing capabilities but rather its prior exposure to similar data. However, ChatGPT's effectiveness in zero-shot setting is disastrous through preliminary experiment in Section 4, which indicates ChatGPT fails to directly memorize these instances of the dataset. Therefore, our results are still convincing.

7.3.4 | Evaluation

The threats to validity also stem from the evaluation metrics system adopted in the experiment. In our work, we utilize three widely used evaluation metrics, namely Precision, Recall, and F1-score. Additionally, we employ statistical methods, the Wilcoxon signed-rank test[70] with Benjamini–Hochberg[71] (BH) correction and Cliff's δ, to assess the significance of the differences between our approach and others, providing a comprehensive evaluation of our approach's performance.

7.3.5 | The language of open projects

Our experiment is conducted on a dataset provided by Maldonado,[38] which includes 10 open-source projects developed in the Java programming language. However, whatever the develop language of projects is, the developer comments are in natural language form. SATD detection essentially involves classifying comments provided by developers, which is a task of natural language understanding and classification. Therefore, the results of our approach to classifying natural language comments provided by developers remains convincing, which can be generalized to projects developed with other languages.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

8 | RELATED WORK

8.1 | SATD detection

SATD, as a specific form of technical debt, has garnered considerable attention from researchers.[21,74–76] In recent years, researchers have made considerable efforts to solve the problem of SATD detection in the field of software engineering.[38,77–79] Various methods have been proposed to detect SATD, which can be broadly categorized into two main types: pattern-based methods and machine learning-based methods.[30]

Potdar and Shihab conducted a manual analysis of Java project comments and pinpointed 62 distinct patterns for SATD detection.[15] Huang et al.[80] developed a method based on text mining to identify SATDs. They employed feature selection techniques to pinpoint useful characteristics for training classifiers. Furthermore, they amalgamated multiple classifiers, each derived from different source projects, to create a composite classifier. This composite classifier was then utilized for the detection of SATD annotations within target projects. Yan et al.[81] concentrated on detecting SATDs at the change-level by utilizing features of software changes. They first identified technical debt from the source code comments found in all versions of the source files. Then, they marked the changes that first introduceed SATD annotations as the changes introducing technical debt. Following this, they constructed a determination model by extracting 25 features from software changes. Flisar et al.[74] employed pretrained word embedding models to refine and enhance the original features in SATD identification to improve the identification of SATDs. Although these methods achieve good results in specific tasks, they have a common disadvantage, requiring a lot of manual operation.

Some SATD detection methods based on machine/deep learning are gradually proposed. Wattanakriengkrai et al.[82] combined N-gram IDF and auto-sklearn machine learning to construct an automated model, which was then utilized for the identification of SATDs. Ren et al.[42] proposed a method based on CNNs for SATD detection, introducing a weighted loss function to address data imbalance issues. Wang et al.[29] proposed a hybrid attention-based SATD detection method, featuring flexible switching word embedding techniques based on project uniqueness and comment features. Yu et al.[30] developed a SATD detection method using Gated Graph Neural Networks, representing the state-of-the-art in this field. Although these deep learning methods are good at identifying and classifying SATD features, their interpretability is really poor. They cannot provide sentence-level explanations. In addition, these methods are single deep learning models that do not involve the fusion of large and small models and they do not use large models to improve interpretability.

8.2 | LLMs for software engineering

Large Language Models (LLMs), such as GPT,[37] have gradually been applied to the field of software engineering due to its strong capabilities of natural language generation and understanding. However, most of these studies utilize ChatGPT for generation tasks, for example, code completion, program repair, test assertion generation, and so on. For example, Surameery et al.[83] explored the roles and limitations of ChatGPT in dealing with programming problems, and they pointed that combining the benefits of ChatGPT with other debugging tools can better identify and correct bugs. Rahmaniar et al.[84] discussed ChatGPT's capabilities for code relief in the field of software engineering, and they pointed out a careful approach to integrating ChatGPT into software development paradigms. Furthermore, building upon standard few-shot prompting techniques, Nashid et al.[62] proposed an approach for creating prompts based on embeddings or frequency-based few-shot retrieval, and conducted empirical studies with ChatGPT in test assertion generation and program repair tasks in the field of software engineering. Their results indicated that their method for prompt construction achieved better performance than fine-tuned models and specific-task models. Few studies have applied ChatGPT to classification tasks in software engineering. Fu et al.[85] utilized ChatGPT to study tasks such as vulnerability prediction and vulnerability classification, but found that the performance was so poor. A possible reason is that the analysis of code vulnerabilities is difficult for ChatGPT, whereas it excels at text analysis. Mastropaolo et al.[9] utilized LLMs to automatically pay back SATDs, but their work did not refer to SATD detction. Notably, neither of them combines large language models and small models to apply to software engineering tasks. We are the first to propose the fusion of ChatGPT with small deep learning models to identify and explain SATDs in the field of software engineering.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

8.3 | LLMs for text classification

LLMs have also showed significant abilities across considerable natural language understanding tasks, especially text classification tasks. Leveraging their robust generative abilities, they can provide detailed explanations for these tasks. Brown et al.[37] proposed an approach, In-Context Learning (ICL), which involved concatenating a few demonstrations with the target input and providing context for the model to learn from. To address the low performance of LLMs in reasoning tasks, Wei et al.[33] proposed a novel method, few-shot CoT, which represented a sequence of intermediate reasoning steps in natural language, leading to the desired output. CoT is essentially an extension of the ICL method. Afterwards, Kojima et al.[32] provided a method called zero-shot CoT, successfully generating a plausible reasoning path in a zero-shot manner and reached the correct answer in a problem where the standard zero-shot approach failed, by adding a simple prompt, "Let's think step by step," to facilitate step-by-step thinking before answering each question. These methods have been widely applied and studied in text classification tasks. For example, in a series of studies, Zhong et al.[35] found that ChatGPT exhibits performance comparable to BERT in a series of text classification tasks. They investigated the synergistic potential of ChatGPT with advanced prompting strategies, such as standard few-shot prompts[37] (also known as ICL), few-Shot CoT prompts,[33] and zero-Shot CoT prompts,[32] with the few-Shot CoT yielding the best performance in a popular benchmark, GLUE.[61] In addition, Xu et al.[34] proposed a method, SuperICL, which embeded the prediction results and confidence levels of fine-tuned models into prompts. Then the prompts were fed into LLMs to boost their performance. Experiments on benchmark GLUE showed that this method could improve performance beyond fine-tuned models. SATD detection is essentially a text classification task in the field of software engineering. Although these methods have made good progress in text classification, there has been no research on fusing LLMs with small models to detect SATDs. We are the first to propose the fusion of ChatGPT with small deep learning models for SATD detection.

9 | CONCLUSION

This paper presents a pioneering effort in making full use of the capabilities of ChatGPT for the detection of SATD, setting a precedent in the comparative study with small models. Through our novel FSATD approach, we have successfully demonstrated the feasibility of fusing the strengths of ChatGPT with small deep learning models to not only accurately recognize SATDs in open-source software projects but also to provide coherent and reliable explanations for these classifications. Our extensive experiments conducted on a comprehensive dataset reveal that while FSATD shows comparable performance to state-of-the-art methods in terms of Precision, Recall, and F1-score, it notably excels in offering valuable insights through explanations, a feature largely absent in existing models. Furthermore, the exploration of various fusion strategies underscores the adaptability of our approach to different application contexts, highlighting its potential as a versatile tool in the realm of software engineering. This work not only contributes to the advancement of SATD detection methodologies but also opens avenues for future research in effectively combining large language models with specialized smaller models for enhanced performance across various domains in software engineering.

AUTHOR CONTRIBUTIONS

Jun Li: Writing - original draft, Methodology, Data curation. **Lixian Li:** Methodology and Software. **Jin Liu:** Conceptualization and Supervision. **Xiao Yu:** Conceptualization, Supervision and Writing - review & editing. **Xiao Liu:** Writing - review & editing. **Jacky Wai Keung:** Conceptualization and Supervision.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grants (Number 61972290) and the National Natural Science Foundation of Chongqing, China (cstc2021jcyj-msxmX1115).

CONFLICT OF INTEREST STATEMENT

The authors have no conflicts of interest to declare that are relevant to the content of this article.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available in tse.satd.data at https://github.com/maldonado. These data were derived from the following resources available in the public domain: https://github.com/maldonado/tse.satd.data.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

ORCID

Jun Li https://orcid.org/0009-0004-3671-5478
Jin Liu https://orcid.org/0000-0003-0359-0248
Xiao Yu https://orcid.org/0000-0002-4473-3068
Xiao Liu https://orcid.org/0000-0002-4151-8522
Jacky Wai Keung https://orcid.org/0000-0002-3803-9600

REFERENCES

1. Cunningham W. The wycash portfolio management system. *ACM Sigplan Oops Messeng*. 1992;4:29-30.
2. Izurieta C, Ozkaya I, Seaman C, Snipes W. Technical debt: a research roadmap report on the eighth workshop on managing technical debt (mtd 2016). *ACM SIGSOFT Softw Eng Notes*. 2017;42:28-31.
3. Kruchten P, Nord RL, Ozkaya I. Technical debt: from metaphor to theory and practice. *IEEE Softw*. 2012;29:18-21.
4. Li Z, Avgeriou P, Liang P. A systematic mapping study on technical debt and its management. *J Syst Softw*. 2015a;101:193-220.
5. Lim E, Taksande N, Seaman C. A balancing act: what software practitioners have to say about technical debt. *IEEE Softw*. 2012;29:22-27.
6. Nord RL, Ozkaya I, Schwartz EJ, Shull F, Kazman R. Can knowledge of technical debt help identify software vulnerabilities? *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*. CSET; 2016.
7. Point AT. from Dagstuhl, R., Perspectives on managing technical debt.
8. Ampatzoglou A, Chatzigeorgiou A, Arvanitou EM, Bibi S. Sdk4ed: a platform for technical debt management. *Softw Pract Exp*. 2022;52:1879-1902.
9. Mastropaolo A, Di Penta M, Bavota G. Towards automatically addressing self-admitted technical debt: how far are we? *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE; 2023:585-597.
10. Tom E, Aurum A, Vidgen R. An exploration of technical debt. *J Syst Softw*. 2013;86:1498-1516.
11. Wehaibi S, Shihab E, Guerrouj L. Examining the impact of self-admitted technical debt on software quality. *2016 IEEE 23Rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE; 2016:179-188.
12. Zazworka N, Spínola RO, Vetró A, Shull F, Seaman C. A case study on effectively identifying technical debt. *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. ACM; 2013:42-47.
13. Alves NS, Mendes TS, De Mendonça MG, Spínola RO, Shull F, Seaman C. Identification and management of technical debt: a systematic mapping study. *Inform Softw Technol*. 2016;70:100-121.
14. Stopford B, Wallace K, Allspaw J. Technical debt: challenges and perspectives. *IEEE Softw*. 2017;34:79-81.
15. Potdar A, Shihab E. An exploratory study on self-admitted technical debt. *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE; 2014:91-100.
16. Ampatzoglou A, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P. The financial aspect of managing technical debt: a systematic literature review. *Inform Softw Technol*. 2015;64:52-73.
17. Foucault M, Blanc X, Storey MA, Falleri JR, Teyton C. Gamification: a game changer for managing technical debt? a design study. *arXiv preprint arXiv:1802.02693* 2018.
18. Xuan J, Hu Y, Jiang H. Debt-prone bugs: technical debt in software maintenance. *arXiv preprint arXiv:1704.04766* 2017.
19. Li Z, Liang P, Avgeriou P. Architectural technical debt identification based on architecture decisions and change scenarios. *2015 12th Working IEEE/IFIP Conference on Software Architecture*. IEEE; 2015b:65-74.
20. Marinescu R. Detection strategies: metrics-based rules for detecting design flaws. *20th IEEE International Conference on Software Maintenance, 2004. Proceedings*. IEEE; 2004:350-359.
21. Zampetti F, Noiseux C, Antoniol G, Khomh F, Di Penta M. Recommending when design technical debt should be self-admitted. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE; 2017:216-226.
22. Chen Y, Huang J, Mou L, Jin P, Xiong S, Zhu XX. Deep saliency smoothing hashing for drone image retrieval. *IEEE Trans Geosci Remote Sens*. 2023b;61:1-13.
23. Chen Y, Lu X, Wang S. Deep cross-modal image–voice retrieval in remote sensing. *IEEE Trans Geosci Remote Sens*. 2020;58:7049-7061.
24. Liu X, Zhu Y, Wu X. Joint user profiling with hierarchical attention networks. *Front Comp Sci*. 2023;17:173608.
25. Ma X, Keung JW, Yu X, Zou H, Zhang J, Li Y. Attsum: a deep attention-based summarization model for bug report title generation. *IEEE Trans Reliab*. 2023;72:1663-1667.
26. Qiao B, Wu Z, Ma L, Zhou Y, Sun Y. Effective ensemble learning approach for sst field prediction using attention-based predrnn. *Front Comp Sci*. 2023;17:171601.
27. Zhang F, Yu X, Keung J, et al. Improving stack overflow question title generation with copying enhanced codebert model and bi-modal information. *Inform Softw Technol*. 2022;148:106922.
28. Li H, Qu Y, Liu Y, Chen R, Ai J, Guo S. Self-admitted technical debt detection by learning its comprehensive semantics via graph neural networks. *Softw Pract Exp*. 2022a;52:2152-2176.
29. Wang X, Liu J, Li L, Chen X, Liu X, Wu H. Detecting and explaining self-admitted technical debts with attention-based neural networks. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. IEEE; 2020:871-882.
30. Yu J, Zhao K, Liu J, Liu X, Xu Z, Wang X. Exploiting gated graph neural network for detecting and explaining self-admitted technical debts. *J Syst Softw*. 2022;187:111219.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

31. Zampetti F, Serebrenik A, Di Penta M. Automatically learning patterns for self-admitted technical debt removal. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE; 2020:355-366.
32. Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y. Large language models are zero-shot reasoners. *Adv Neural Inform Process Syst*. 2022;35:22199-22213.
33. Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models. *Adv Neural Inform Process Syst*. 2022b;35:24824-24837.
34. Xu C, Xu Y, Wang S, Liu Y, Zhu C, McAuley J. Small models are valuable plug-ins for large language models. *arXiv preprint arXiv:2305.08848* 2023.
35. Zhong Q, Ding L, Liu J, Du B, Tao D. Can chatgpt understand too? a comparative study on chatgpt and fine-tuned bert. *arXiv preprint arXiv:2302.10198* 2023.
36. Robertson S, Zaragoza H, et al. The probabilistic relevance framework: Bm25 and beyond. *Found Trends Inform Retriev*. 2009;3:333-389.
37. Brown T, Mann B, Ryder N, et al. Language models are few-shot learners. *Adv Neural Inform Process Syst*. 2020;33:1877-1901.
38. da Silva Maldonado E, Shihab E, Tsantalis N. Using natural language processing to automatically detect self-admitted technical debt. *IEEE Trans Softw Eng*. 2017;43:1044-1062.
39. Kuka V. Token 1.10: Large vs small in ai: The language model size dilemma. 2023 Https://www.turingpost.com/p/largesmallmodel. Accessed November 22, 2023.
40. Kora R, Mohammed A. A comprehensive review on transformers models for text classification. *2023 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*. IEEE; 2023:1-7.
41. Minaee S, Kalchbrenner N, Cambria E, Nikzad N, Chenaghlu M, Gao J. Deep learning–based text classification: a comprehensive review. *ACM Comput Surv*. 2021;54:1-40.
42. Ren X, Xing Z, Xia X, Lo D, Wang X, Grundy J. Neural network-based detection of self-admitted technical debt: from performance to explainability. *ACM Trans Softw Eng Methodol*. 2019;28:1-45.
43. Bouvrie J. Notes on convolutional neural networks. 2006.
44. Memory LST. Long short-term memory. *Neural Comput*. 2010;9:1735-1780.
45. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. *Adv Neural Inform Process Syst*. 2017;30:6000-6010.
46. Chen Y, Dai H, Yu X, Hu W, Xie Z, Tan C. Improving ponzi scheme contract detection using multi-channel textcnn and transformer. *Sensors*. 2021;21:6417.
47. Keeling R, Chhatwal R, Huber-Fliflet N, et al. Empirical comparisons of cnn with other learning algorithms for text classification in legal document review. *2019 IEEE International Conference on Big Data (Big Data)*. IEEE; 2019:2038-2042.
48. Yang Z, Keung J, Kabir MA, et al. Acomnn: attention enhanced compound neural network for financial time-series forecasting with cross-regional features. *Appl Soft Comput*. 2021a;111:107649.
49. Jia K, Yu X, Zhang C, Hu W, Zhao D, Xiang J. Software aging prediction for cloud services using a gate recurrent unit neural network model based on time series decomposition. *IEEE Trans Emerg Top Comput*. 2023;11:580-593.
50. Luan Y, Lin S. Research on text classification based on CNN and LSTM. *2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE; 2019:352-355.
51. Viel F, Maciel RC, Seman LO, Zeferino CA, Bezerra EA, Leithardt VRQ. Hyperspectral image classification: an analysis employing cnn, lstm, transformer, and attention mechanism. *IEEE Access*. 2023;11:24835-24850.
52. Yang Z, Keung J, Yu X, et al. A multi-modal transformer-based code summarization approach for smart contracts. *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE; 2021b:1-12.
53. Yang Z, Keung JW, Yu X, Xiao Y, Jin Z, Zhang J. On the significance of category prediction for code-comment synchronization. *ACM Trans Softw Eng Methodol*. 2023;32:1-41.
54. Zhang F, Liu J, Wan Y, Yu X, Liu X, Keung J. Diverse title generation for stack overflow posts with multiple-sampling-enhanced transformer. *J Syst Softw*. 2023;200:111672.
55. Li X, Xiong H, Li X, et al. Interpretable deep learning: interpretation, interpretability, trustworthiness, and beyond. *Knowl Inform Syst*. 2022b;64:3197-3234.
56. Wei J, Tay Y, Bommasani R, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* 2022a.
57. Gao S, Wen XC, Gao C, Wang W, Zhang H, Lyu MR. What makes good in-context demonstrations for code intelligence tasks with llms? *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE; 2023:761-773.
58. Reimers N, Gurevych I. Sentence-bert: sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* 2019.
59. Guo D, Lu S, Duan N, Wang Y, Zhou M, Yin J. Unixcoder: unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850* 2022.
60. Shi E, Wang Y, Gu W, et al. Cocosoda: effective contrastive learning for code search. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE; 2023:2198-2210.
61. Wang A, Singh A, Michael J, Hill F, Levy O, Bowman SR. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* 2018.
62. Nashid N, Sintaha M, Mesbah A. Retrieval-based prompt selection for code-related few-shot learning. *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)*. IEEE; 2023.
63. Yu J, Zhou X, Liu X, Liu J, Xie Z, Zhao K. Detecting multi-type self-admitted technical debt with generative adversarial network-based neural networks. *Inform Softw Technol*. 2023;158:107190.

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

64. Barr A. Openai's gpt-4 ai model got lazier and dumber - chatgpt. 2023 https://www.businessinsider.com/openai-gpt4-ai-model-got-lazier-dumber-chatgpt-2023-7. Accessed September 1, 2023.
65. Chen L, Zaharia M, Zou J. How is chatgpt's behavior changing over time? 2023a arXiv preprint arXiv:2307.09009.
66. Madaan R. Chatgpt gpt-4 response quality and performance drop issues to be investigated. 2023 https://piunikaweb.com/2023/07/14/chatgpt-gpt-4-response-quality-performance-drop-issues/. Accessed September 1, 2023.
67. Zakori I. Gpt-4: The world's most powerful ai model - an analysis of its performance decline. 2023 https://artifcialcorner.com/gpt-4-the-worlds-most-powerful-ai-model-an-analysis-of-its-performance-decline598ad723a595. Accessed September 1, 2023.
68. Hameed Z, Garcia-Zapirain B. Sentiment classification using a single-layered bilstm model. *IEEE Access*. 2020;8:73992-74001.
69. Soyalp G, Alar A, Ozkanli K, Yildiz B. Improving text classification with transformer. *2021 6th International Conference on Computer Science and Engineering (UBMK)*. IEEE; 2021:707-712.
70. Wilcoxon F. Individual comparisons by ranking methods. *Breakthroughs in Statistics: Methodology and Distribution*. Springer; 1992:196-202.
71. Ferreira J, Zwinderman A. On the benjamini–hochberg method. 2006.
72. Cliff N. Dominance statistics: ordinal analyses to answer ordinal questions. *Psychol Bull*. 1993;114:494.
73. Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. The impact of automated parameter optimization on defect prediction models. *IEEE Trans Softw Eng*. 2018;45:683-711.
74. Flisar J, Podgorelec V. Identification of self-admitted technical debt using enhanced feature selection based on word embedding. *IEEE Access*. 2019;7:106475-106494.
75. Guo Z, Liu S, Liu J, et al. Mat: a simple yet strong baseline for identifying self-admitted technical debt. *arXiv preprint arXiv:1910.13238* 2019.
76. Iammarino M, Zampetti F, Aversano L, Di Penta M. Self-admitted technical debt removal and refactoring actions: Co-occurrence or more? *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE; 2019:186-190.
77. de Freitas Farias MA, de Mendonça Neto MG, Kalinowski M, Spínola RO. Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Inform Softw Technol*. 2020;121:106270.
78. Mensah S, Keung J, Svajlenko J, Bennin KE, Mi Q. On the value of a prioritization scheme for resolving self-admitted technical debt. *J Syst Softw*. 2018;135:37-54.
79. Yu Z, Fahid FM, Tu H, Menzies T. Identifying self-admitted technical debts with jitterbug: a two-step approach. *IEEE Trans Softw Eng*. 2020;48:1676-1691.
80. Huang Q, Shihab E, Xia X, Lo D, Li S. Identifying self-admitted technical debt in open source projects using text mining. *Empiric Softw Eng*. 2018;23:418-451.
81. Yan M, Xia X, Shihab E, Lo D, Yin J, Yang X. Automating change-level self-admitted technical debt determination. *IEEE Trans Softw Eng*. 2018;45:1211-1229.
82. Wattanakriengkrai S, Maipradit R, Hata H, Choetkiertikul M, Sunetnanta T, Matsumoto K. Identifying design and requirement self-admitted technical debt using n-gram idf. *2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE; 2018:7-12.
83. Surameery NMS, Shakor MY. Use chat gpt to solve programming bugs. *Int J Inform Technol Comput Eng*. 2023;3:17-22.
84. Rahmaniar W. Chatgpt for software development: opportunities and challenges. *Authorea Preprints*. 2023.
85. Fu M, Tantithamthavorn C, Nguyen V, Le T. Chatgpt for vulnerability detection, classification, and repair: How far are we? *arXiv preprint arXiv:2310.09810* 2023.

How to cite this article: Li J, Li L, Liu J, Yu X, Liu X, Keung JW. Large language model ChatGPT versus small deep learning models for self-admitted technical debt detection: Why not together?. *Softw: Pract Exper*. 2025;55(1):3-28. doi: 10.1002/spe.3360

1097024x, 2025, 1, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/spe.3360 by Wuhan University, Wiley Online Library on [12/01/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License