Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Cost-Effective Detection of Self-Admitted Technical Debt: A Collaborative Framework of Proxy Active Learning and Large Language Models

Zhicheng Wang
Jilin University
Tianxin Lei
Jilin University
Tie Bao
baotie@jlu.edu.cn

Jilin University

Research Article

Keywords:

Posted Date: November 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9952731/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

RESEARCH ARTICLE

Cost-Effective Detection of Self-Admitted Technical Debt: A Collaborative Framework of Proxy Active Learning and Large Language Models

Zhicheng Wang[1], Tianxin Lei[2], Tie Bao[1,2*]

[1]College of Computer Science and Technology, Jilin University, Changchun, 130012, Jilin, China.
[2]College of Software, Jilin University, Changchun, 130012, Jilin, China.

*Corresponding author(s). E-mail(s): baotie@jlu.edu.cn;
Contributing authors: zhicheng23@mails.jlu.edu.cn;
leitx23@mails.jlu.edu.cn;

Abstract

During software evolution, developers often document design compromises through code comments, a phenomenon known as Self-Admitted Technical Debt (SATD). Precise SATD identification is vital for reducing maintenance costs and enhancing reliability. Although deep learning has improved detection accuracy, its performance is heavily constrained by the extreme sparsity and class imbalance of SATD instances in real-world scenarios, making large-scale expert annotation cost-prohibitive. Existing active learning attempts struggle to balance deep semantic feature extraction with the exorbitant computational overhead of Large Language Models (LLMs).

To address these challenges, this paper proposes a deep active learning approach for SATD detection. First, we design a hybrid query strategy combining uncertainty and representativeness with domain-specific keywords to resolve cold-start and sampling redundancy issues. Second, we introduce a decoupled framework that utilizes BERT as a computationally efficient proxy query engine to localize key samples, which are then used for full-parameter training of the Flan-T5 model. Empirical results demonstrate that our method achieves performance comparable to full-supervision baselines using only 30% of the labeled data, significantly lowering the practical barrier for SATD identification.

Keywords: Self-Admitted Technical Debt, Software Maintenance, Software Quality, Active Learning, Large Language Models

1 Introduction

Technical debt is an inherent phenomenon in software development, where development teams, despite their mandate to deliver high-quality products within stringent deadlines, are often compelled by unforeseen exigencies to incorporate suboptimal code, thereby sacrificing long-term quality for short-term velocity (Cunningham (1992)). Self-Admitted Technical Debt (SATD) represents a category of technical debt that is explicitly acknowledged and intentionally introduced by developers, often necessitated by the demands for rapid delivery or accelerated functional deployment (Maldonado and Shihab (2015)). As an intentionally introduced phenomenon, SATD is typically documented within projects to facilitate future remediation, most commonly via source code annotations (Potdar and Shihab (2014)).

The imperative to study SATD arises from its ubiquity, its irreplaceability, and the adverse impacts it entails. Ubiquity refers to the pervasive nature of SATD across diverse software projects; for instance, Bavota et al. observed through their investigation of 159 Java open-source projects that SATD is prevalent in virtually every system, underscoring the necessity of researching SATD to provide robust quality assurance mechanisms for a wide range of software systems (Bavota and Russo (2016)). Its irreplaceability lies in the fact that certain defects, such as those identifiable via code metrics (de Freitas Farias et al. (2015)), are difficult to detect through traditional means. Conventional static analysis tools primarily emphasize syntactic structures and cyclomatic complexity, yet they struggle to capture the developer intent and design compromises embedded within natural language comments. Since these compromises exist solely in the form of SATD within the codebase, investigating SATD represents an efficient methodology for identifying and resolving such issues. The adverse impacts of SATD primarily manifest as diminished software quality and inflated maintenance costs. Xavier et al. demonstrated that code containing SATD often exhibits performance degradation and increased susceptibility to defects. Furthermore, such code frequently lacks rigorous design logic, serving merely as a temporary workaround. This obfuscation impedes readability and comprehension for other developers, thereby decelerating the pace of code evolution (Xavier et al. (2020)).

Recent empirical evidence indicates that at the method level, SATD correlates significantly with increased code complexity and defect introduction, validating that the precise identification and repayment of SATD are critical for reducing maintenance overhead and enhancing software reliability (Chowdhury et al. (2025)).

A comprehensive review of existing literature reveals that identifying SATD within source code comments is inherently a text classification problem (Ren et al. (2019)). The technical paradigm has evolved from rudimentary text matching to sophisticated semantic analysis based on deep learning and pre-trained models, significantly augmenting detection accuracy. Notwithstanding the commendable performance of contemporary deep learning models, their reliance on large-scale, high-quality labeled data remains a formidable barrier. Acquiring such datasets necessitates domain expertise and entails overcoming the challenges of data sparsity and extreme class imbalance, which collectively impose substantial annotation costs (Tu and Menzies (2022)).

To mitigate these annotation costs, several researchers have explored detection methodologies integrating active learning. Tu et al. proposed the DebtFree framework,

serving as a pioneering effort in evaluating unsupervised/semi-supervised learning to diminish SATD annotation costs (Tu and Menzies (2022)). Similarly, Yu et al. introduced the "human-in-the-loop" paradigm, developing a model dubbed Jitterbug based on a divide-and-conquer strategy. While this method excels in cost-effectiveness by automatically identifying easily distinguishable patterns and deferring difficult samples to human review, its classifier performance has yet to achieve the benchmarks set by state-of-the-art deep learning models (Yu et al. (2020)).

Addressing these challenges, this study follows the research trajectory of treating source code comments as natural language processing tasks, with a specific focus on SATD detection in low-resource scenarios. We first investigate the efficacy of active learning in alleviating data dependency by constructing a deep active learning baseline framework based on BERT. Subsequently, to resolve the contradiction between the limitations of discriminative models in interpreting subtle semantics and the exorbitant inference overhead of Large Language Models (LLMs) in active learning, we propose a collaborative detection method grounded in the "proxy selection" paradigm. This framework decouples sample screening from model training, thereby effectively harmonizing deep semantic comprehension with computational efficiency.

The primary contributions of this paper are summarized as follows:

(1) We propose a deep active learning approach integrating uncertainty and representativeness. Addressing the cold-start and sampling redundancy issues inherent in active learning, we design an initial training set construction strategy leveraging domain-specific keywords and TF-IDF. Furthermore, we introduce a hybrid query strategy that employs BERT to identify samples near the decision boundary and those representative of the feature space, thereby enhancing model generalization in low-resource settings.

(2) We design a collaborative detection framework based on proxy screening and LLM fine-tuning. To circumvent the prohibitive inference costs of generative large models during high-frequency active learning iterations, we introduce a paradigm that employs the discriminative model BERT as a proxy query engine for efficient sample screening, in synergy with the larger-scale Flan-T5 model for full-parameter training. This framework bypasses the semantic comprehension bottlenecks of discriminative models while simultaneously optimizing the computational overhead of data querying.

(3) We conduct extensive empirical studies and performance evaluations. Validated across ten popular Java open-source projects, our approach demonstrates effectiveness in both within-project and cross-project scenarios. Experimental results reveal that our method achieves performance comparable to, or even exceeding, full-supervision baselines, while substantially reducing manual annotation requirements (to approximately 30% of the data).

2 Background

2.1 SATD Detection

The identification of Self-Admitted Technical Debt (SATD) constitutes a pivotal research trajectory within the domain of software engineering. Fundamentally, detecting SATD within source code comments is characterized as a quintessential binary

text classification process (Ren et al. (2019)). Depending on the relationship between training and predictive datasets, this endeavor can be divided into two sub-tasks: first, within-project SATD identification (da Silva Maldonado et al. (2017)), which concentrates on leveraging historical comment or documentation data from a single software project to discern latent debt; and second, cross-project SATD identification (Huang et al. (2018)), which synthesizes heterogeneous data across diverse software projects to identify generalized debt patterns.

Constructing a robust SATD identification model typically entails a systematic pipeline of data preprocessing and feature engineering, effectively transforming natural language text into machine-interpretable numerical vectors or word embedding features. Subsequently, data labeling is performed via manual annotation rules to denote the presence or absence of SATD. These annotated datasets are then utilized to train machine learning or deep learning classifiers—such as Convolutional Neural Networks (CNN) or Bidirectional Encoder Representations from Transformers (BERT)—to learn the complex mapping from natural language semantics to debt manifestations. Ultimately, these models empower development teams to proactively discover and manage potential debt, thereby bolstering the long-term evolvability of software systems.

2.2 Active Learning

Active learning aims to maximize model performance while minimizing manual annotation overhead by strategically selecting high-value samples for labeling. In contemporary deep learning applications, pool-based sampling stands as the predominant active learning paradigm (Zhan et al. (2022)). Within this framework, active learning operates as an iterative cycle: the system evaluates extensive samples in an unlabeled pool using predefined query strategies to identify the most informative subset. These samples are subsequently annotated by experts and integrated into the training set for model refinement, a process that continues until a prescribed annotation budget threshold is reached.

The query strategy serves as the cornerstone of any active learning system, as it directly governs the quality of the acquired knowledge.

Uncertainty-based sampling is currently the most extensively deployed strategy. Its theoretical foundation posits that samples with the lowest predictive confidence typically reside near the current decision boundary; thus, they are deemed most informative for the model. By selecting and incorporating these samples into the training pipeline, the hypothesis space can be effectively constrained. In deep classification architectures, the fundamental approach leverages the probability distribution from the Softmax output to quantify uncertainty. Common metrics include Entropy, Least Confidence, and Margin Sampling. Specifically, Entropy measures the degree of predictive disorder by calculating the information entropy of the probability distribution, where higher values indicate greater uncertainty regarding class membership (Shannon (1948)). The Least Confidence strategy focuses on the magnitude of the maximum predicted probability, where lower values signify reduced confidence (Culotta and McCallum (2005)). Conversely, Margin Sampling computes the differential between

the top two predicted probabilities; a smaller margin indicates that the model is uncertain between these categories, indicating the sample is unlikely to receive a decisive classification (Scheffer et al. (2001)).

Despite the remarkable efficiency of uncertainty sampling, it is susceptible to sampling bias, where the model may over-prioritize outliers caused by noise or labeling errors while neglecting the global structure of the data distribution. To mitigate this deficiency, representativeness-based strategies have been proposed. These strategies emphasize that selected samples should encapsulate the global distributional characteristics of the unlabeled pool to ensure training set diversity (Dasgupta (2011)).

As research has matured, single-dimensional query strategies have increasingly revealed limitations when confronting complex data distributions. Consequently, hybrid strategies that integrate multiple metrics and novel strategies based on model training dynamics have become mainstream. Hybrid strategies aim to harmonize the complementary strengths of diverse query dimensions. For instance, the BADGE algorithm computes gradient embeddings at the output layer, utilizing gradient magnitude to represent uncertainty and gradient orientation to capture feature diversity. By employing K-Means++ sampling, it simultaneously accounts for both hard-to-classify cases and overall diversity (Ash et al. (2019)). Similarly, contrastive learning sampling strategies combine similarity in the feature space with divergence in the prediction space to identify contrastive samples that are semantically similar but yield conflicting model predictions, thereby delineating the decision boundary with greater precision (Margatina et al. (2021)).

2.3 LLMs for SE Text Classification

While traditional discriminative models, such as BERT, demonstrate exceptional performance in natural language processing tasks, they encounter a "cognitive ceiling" when deciphering the subtle and intricate semantics inherent in Self-Admitted Technical Debt (SATD). In contrast, Generative Large Language Models (LLMs) empowered by Instruction Tuning offer a novel direction for surmounting these limitations. Instruction tuning fundamentally reconfigures the paradigm of textual processing by introducing natural language instructions, shifting the model from mere data distribution fitting to an advanced state of comprehending and executing task intentions (Wei et al. (2021)).

Models exemplified by Flan-T5 leverage a comprehensive encoder-decoder Transformer architecture (Raffel et al. (2020)) and undergo continuous pre-training through large-scale, multi-task instruction tuning (Chung et al. (2024)). This mechanism substantially bolsters the model's capacity for generalized semantic comprehension and cross-domain generalization. When confronted with the informal expressions, irony, or metaphors frequently encountered in source code comments, Flan-T5 capitalizes on its holistic understanding of semantic structures and pragmatic intentions to acutely capture latent cues regarding technical risks or temporary compromises. Consequently, the hidden-layer states generated by LLMs inherently encapsulate high-fidelity semantic features, offering a decisive advantage for transcending the performance bottlenecks of traditional models and precisely distinguishing elusive SATD instances.

3 The Proposed Approach

To reconcile the contradiction between the complexity of semantic comprehension in SATD identification and the prohibitive inference overhead of Large Language Models (LLMs) under a constrained annotation budget, this paper proposes a decoupled framework featuring proxy-based screening and LLM collaboration, grounded in the "proxy selection" paradigm. This framework effectively decouples sample screening from model training: it leverages the computationally efficient discriminative model (BERT) as a proxy query engine to rapidly localize key samples, which are then utilized in synergy with the larger-scale generative model (Flan-T5) for full-parameter training. The comprehensive workflow of the framework encompasses the following three core stages: Data Preparation and Cold-start Strategy, Proxy Screening Strategy, and LLM-based Collaborative Training.

3.1 Data Preparation and Cold Start

The construction of the initial training set serves as the inception of active learning, and its quality fundamentally dictates the efficiency and efficacy of subsequent iterations. Traditional active learning methodologies typically employ random sampling; however, in the highly imbalanced task of SATD identification, random sampling fails to guarantee the representativeness of selected instances and often leads to a severe paucity of positive samples within the initial training set.

To circumvent the introduction of bias from the BERT model during the cold-start phase, this study adopts the traditional TF-IDF (Term Frequency-Inverse Document Frequency) vectorization method to quantify sample representativeness. Within the feature space, the KNN algorithm is utilized to calculate representativeness scores for each sample. Simultaneously, to ensure a sufficient presence of SATD instances in the initial training set, we incorporate heuristic domain knowledge (Guo et al. (2021)). Specifically, code comments containing keywords such as "TODO," "FIXME," "XXX," and "HACK" are designated as indicative signals for positive samples. Ultimately, the system selects the top 10% of samples with the highest representativeness scores from the keyword-containing pool to constitute the initial training set. The detailed procedure of the algorithm is delineated in Algorithm 1.

First, representativeness scores for all samples in the training pool are calculated and sorted in ascending order (from least to most representative).

Subsequently, the top N samples with the highest representativeness scores are selected from the keyword-containing instances, where N is set to 10% of the total training set size. The rationale for prioritizing the highest representativeness scores is that during the initial stage, we aim to select samples that concurrently contain domain keywords (ensuring an adequate supply of positive samples) and possess significant representativeness. A high representativeness score indicates that a sample resides in a relatively dense region of the feature space, which helps ensure the selection of typical exemplars within each cluster.

Setting the initial training set size to 10% is informed by the following considerations: on one hand, the initial set must not be too small, as an insufficient size hinders the model's ability to learn effective features, causing the query strategy to degenerate

into random sampling during early iterations (Dor et al. (2020)); on the other hand, it must not be excessively large, as this would undermine the primary objective of active learning to reduce annotation overhead (Settles (2009)). A 10% ratio has been empirically validated in various studies as a robust starting point that provides sufficient initial information while preserving ample labeling budget for subsequent iterations.

Algorithm 1 getInitialTrainingSet - Construct Initial Training Set

Require: C_t: Training pool; n: Number of neighbors; $\Phi(.)$: Encoder
Ensure: X: Sampled code comments
1: **for** $c_p \in C_t$ **do**
2: $\{c_l^{(i)}, d(c_p, c_l^{(i)})\}, i = 1, ..., n \leftarrow \text{KNN}(\Phi(C_t), n)$ // Identify neighbors and retrieve distances
3: $s_r = \left(\frac{1}{n}\sum_{i=1}^{n} d^{(i)}\right)^{-2}$ // Calculate representativeness score
4: **end for**
5: $X = \text{argmax}(s_r \mid \text{keyword in } c_p.\text{text}), |X| = \text{len}(C_t)/10$ // Select top-tier exemplars
6: **return** X

3.2 Proxy Selection Strategy with Agent BERT

During the high-frequency iteration phase of active learning, this paper proposes a hybrid query strategy merging uncertainty and representativeness. This strategy is executed by a lightweight proxy model, BERT, to circumvent the computational bottlenecks associated with the high-frequency inference of Large Language Models (LLMs).

For the calculation of representativeness scores within the current training set (comprising the initial set and subsequently labeled samples), the K-Nearest Neighbors (KNN) algorithm is employed to identify the k most similar instances to sample c_p, denoted as $\{c_1, c_2, ..., c_k\}$. Similarity is quantified via the Euclidean distance between semantic vectors as follows:

$$d(c_p, c_i) = ||\Phi(c_p) - \Phi(c_i)||_2 \tag{1}$$

A smaller Euclidean distance signifies greater proximity in the semantic space and higher semantic similarity (see Equation 1). Regarding the choice of density estimation, while Kernel Density Estimation (KDE) is a common method for measuring representativeness, its performance is heavily contingent upon the configuration of the bandwidth parameter. Particularly within the 768-dimensional feature space extracted by BERT, selecting an optimal bandwidth is exceedingly difficult and often yields unstable estimation results. Conversely, the KNN method avoids the complexities of bandwidth selection. Theoretical research indicates that in high-density regions of high-dimensional spaces, KNN estimators exhibit lower bias and superior stability compared to KDE (Mack and Rosenblatt (1979)). Based on these considerations, this

study adopts the KNN algorithm for density estimation. Furthermore, as KNN density estimation in high-dimensional spaces is relatively insensitive to the specific value of k, we set the number of neighbors to 10. This configuration effectively mitigates the impact of individual noise samples while preserving sensitivity to local density variations—a robust empirical setting validated by multiple studies (Mack and Rosenblatt (1979)).

Upon identifying the nearest neighbors for each sample, the average distance to these neighbors is computed. For sample c_p, the mean distance to its k nearest neighbors is calculated consistently with the method described in Section 3.2. The final representativeness score is derived based on this average distance.

After obtaining the set of representativeness scores S_r for all samples in the unlabeled pool, we apply Min-Max Normalization (Equation 2) to eliminate dimensional discrepancies and ensure effective fusion with uncertainty scores on a uniform scale. The formula is defined as:

$$\hat{s}_r(c_p) = \frac{s_r(c_p) - \min(S_r)}{\max(S_r) - \min(S_r)} \tag{2}$$

where $\min(S_r)$ and $\max(S_r)$ denote the minimum and maximum representativeness scores within the unlabeled pool for the current iteration, respectively. This procedure linearly maps all scores to the $[0, 1]$ interval, ensuring numerical stability for subsequent comprehensive scoring.

For the uncertainty score calculation, we adopt a metric grounded in the Least Confidence principle. Upon inputting a sample into the proxy model, the maximum value p_{max} from the model's probability distribution is retrieved, and the uncertainty score is defined as $s_u = 1 - p_{max}$. A higher value indicates that the sample is positioned closer to the decision boundary.

In designing the multi-strategy fusion mechanism, and departing from traditional weighted-sum approaches, this study references the classical methodology proposed by Zhu et al. (Zhu et al. (2008)). We eschew weighted summation, which requires manual hyperparameter tuning, in favor of a direct multiplicative fusion strategy to construct the final sample valuation function.

The comprehensive procedure for the query strategy is delineated in 2.

3.3 Collaborative Training with Target Flan-T5

This study selects Flan-T5-Large, renowned for its robust reasoning capabilities, as the target learner. To better adapt to the requirements of discriminative tasks while preserving the pre-trained model's profound semantic comprehension, we completely remove the text generation layer of Flan-T5 and replace it with a classification head comprising a Dropout layer and a Linear layer. This newly integrated classification head directly receives input from the Last Hidden State of the decoder, with parameters updated via backpropagation using a cross-entropy loss function.

Within this decoupled framework, the high-frequency screening and iterative training phases are executed exclusively by the proxy model. The Flan-T5 model is engaged only when the active learning iterations reach a predefined stopping threshold, at

Algorithm 2 getSample - Single Iterative Step

Require: C_t: Training pool; C_u: Unlabeled code comments; a: Number of samples to acquire; M: Model; n: Number of neighbors; $\Phi(.)$: Encoder

Ensure: X: Sampled code comments

$S_{data} \leftarrow \emptyset$ // Temporary storage for candidate data

for $c_p \in C_u$ **do**

  $\{c_l^{(i)}, d(c_p, c_l^{(i)})\}, i = 1, ..., n \leftarrow \text{KNN}(\Phi(C_t), n)$ // Identify neighbors and retrieve distances

  $s_r = \left(\frac{1}{n}\sum_{i=1}^{n} d^{(i)}\right)^{-2}$ // Calculate representativeness score

  $p(y \mid c_p) \leftarrow M(c_p)$

  $s_u = 1 - \max(p(y \mid c_p))$ // Calculate uncertainty score

  $S_{data} \leftarrow S_{data} \cup \{(c_p, s_r, s_u)\}$

end for

// Collect representativeness scores for normalization

$S_R \leftarrow \{item.s_r \mid item \in S_{data}\}$

$S_{final} \leftarrow \emptyset$

for $item \in S_{data}$ **do**

  $\hat{s}_r = \frac{item.s_r - \min(S_R)}{\max(S_R) - \min(S_R)}$

  $s_{cp} = item.s_u \times \hat{s}_r$

  $S_{final} \leftarrow S_{final} \cup \{(item.c_p, s_{cp})\}$

end for

$X = \underset{c_p \in C_u}{\text{argmax}}\, s_{cp}$ with $|X| = a$ // Select a batch of high-value samples

return X

which point it undergoes full-parameter fine-tuning on the accumulated high-quality dataset. The complete collaborative workflow is delineated in Algorithm 3.

The execution of the process commences with the data preparation and cold-start phase. Following the preprocessing of the raw training and testing sets, the system initializes an empty labeled set, which is subsequently populated via the initial training set construction. Leveraging this initial data, we perform the first round of supervised training on both the BERT proxy model and the Flan-T5 target model. This stage is designed to endow the proxy model with preliminary feature recognition capabilities—enabling effective uncertainty estimation for subsequent unlabeled samples—while simultaneously establishing a foundational discriminative baseline for the target model.

Following the establishment of initial models, the system enters a cyclic iteration process until the volume of labeled data satisfies the predefined budget ratio. At the inception of each iteration, we utilize the BERT proxy model to perform inference on all instances within the unlabeled pool. The algorithm identifies samples that the current model deems most difficult to distinguish or most informative based on uncertainty metrics. These selected samples are then submitted for manual annotation and transferred from the unlabeled pool to the labeled dataset.

Upon the completion of dataset updates, the process transitions to the model collaborative update phase. To ensure the efficacy of the proxy screening strategy and

the synchronized enhancement of the target model's performance, both models are trained on the most recent, complete labeled dataset. This step adopts a full retraining strategy, wherein both models are retrained based on the entire accumulated dataset at each update interval, rather than performing incremental updates on parameters from previous rounds. This approach effectively mitigates the risk of the models forgetting features from earlier data during continuous fine-tuning, ensuring the models fully fit the current data distribution.

After multiple cycles, the training process concludes when the stopping criterion is met. At this juncture, the finalized Flan-T5 target model is output and evaluated on the test set to validate the effectiveness of the data selected via active learning.

Algorithm 3 Collaborative Active Learning with Proxy Selection - Decoupled Collaborative Workflow

Require: C_t: Training pool; C_e: Test set; μ: Stopping criterion threshold; a: Number of samples to acquire; M_{proxy}: Proxy model; M_{target}: Target model; n: Number of neighbors; $\Phi(.)$: Encoder

Ensure: R: Final prediction results

preprocess(C_t, C_e)
$C_l \leftarrow \emptyset$ // Initialize the labeled dataset
$C_i \leftarrow$ getInitialTrainingSet$(C_t, n, \Phi(.))$ // Acquire initial cold-start data
tagComments(C_i) // Simulate manual annotation
$C_l \leftarrow C_l \cup C_i$
$C_u \leftarrow C_t \setminus C_l$ // Initialize the unlabeled pool
$M_{proxy} \leftarrow$ trainModel(M_{proxy}, C_l)
while len(C_l)/len$(C_t) < \mu$ **do**
    // Screen samples using only the proxy model based on uncertainty and representativeness
    $X \leftarrow$ getSample$(C_t, C_u, a, M_{proxy}, n, \Phi(.))$
    tagComments(X) // Acquire manual annotations
    $C_l \leftarrow C_l \cup X$
    $C_u \leftarrow C_u \setminus X$
    $M_{proxy} \leftarrow$ trainModel(M_{proxy}, C_l)
end while
$M_{target} \leftarrow$ trainModel(M_{target}, C_l) // Perform one-time full-parameter fine-tuning upon reaching the threshold
$R \leftarrow$ getFinalPrediction(M_{target}, C_e) // Utilize the target model for final performance evaluation **return** R

Since the core innovation of this section focuses on the integration of uncertainty and representativeness-based query strategies, we modified and implemented the model training and human oracle workflows based on the existing open-source library small-text to enhance development efficiency and preclude redundant efforts. This library provides standardized interfaces for query strategies, facilitating seamless collaboration with various classifiers (Schröder et al. (2023)).

4 Empirical Evaluation

To conduct a comprehensive validation of the performance of the proposed SATD detection method—augmented by Large Language Model (LLM) collaboration—this section delineates a systematic experimental protocol.

4.1 Research Questions

The experiments in this study are designed to evaluate our approach across three core dimensions, which are operationalized into the following research questions:

RQ1 (Model Effectiveness): How does the collaborative framework perform compared to traditional deep learning baselines in within-project scenarios?

This question aims to verify whether the integration of a Large Language Model (LLM) as the target learner can transcend the "cognitive bottleneck" of existing BERT-based discriminative models in interpreting subtle semantics, thereby augmenting recognition precision.

RQ2 (Cost Efficiency): To what extent can the active learning strategy mitigate manual annotation costs?

By analyzing performance across varying stopping thresholds and tracking annotation budget consumption, this question evaluates the efficacy of the proxy screening mechanism in reducing the computational overhead associated with the high-frequency iterative training of LLMs.

RQ3 (Generalization): How robust is the generalization capability of the collaborative framework in cross-project scenarios?

This question investigates whether generative models, refined via large-scale instruction tuning, possess superior domain transfer and generalization capabilities when confronting code comment data from unseen domains characterized by significant distributional shifts.

4.2 Dataset

This study utilizes the publicly available dataset released by Guo et al. (Guo et al. (2021)), encompassing ten popular open-source Java projects: Dubbo, Gradle, Groovy, Maven, Hive, Storm, Poi, SpringFramework, Tomcat, and Zookeeper, totaling approximately 266,000 code comments. Following the methodology of Maldonado et al. (Maldonado and Shihab (2015)), we applied five heuristic filtering rules to the comments: (1) exclusion of license-related comments; (2) merging consecutive single-line comments into a single entry; (3) removal of source code snippets within comments that typically do not contain technical debt; (4) elimination of IDE-generated comments (e.g., from Eclipse); and (5) exclusion of Javadoc comments unless they contain at least one task-related tag (e.g., "TODO:" or "FIXME:"). These rules effectively filter out instances that are ostensibly not SATD.

As delineated in Table 1, the data distribution across these projects exhibits extreme class imbalance. The proportion of SATD samples is remarkably low across all projects, ranging from 0.23% to 2.50%, with a mean prevalence of only 1.12%. Such distributional characteristics impose stringent requirements on the model's feature

extraction capabilities and the robustness of the sampling strategy (He and Garcia (2009)).

In the data preprocessing phase, we adhere to the standard protocols of the domain (Guo et al. (2021)): all non-alphanumeric characters and specific punctuation marks are removed; word contractions are expanded or split (e.g., separating "'s" and "'ll"); and all text is converted to lowercase. Furthermore, to mitigate noise interference potentially remaining from the initial preprocessing, we implemented additional cleaning rules to discard entirely blank samples. The cleaned text is subsequently tokenized using a pre-trained BERT tokenizer with a maximum sequence length of 128 (with truncation and zero-padding as necessary) and the addition of a $[CLS]$ token to accommodate model input requirements. The mask vector length is consistent with the maximum sequence length.

Table 1 Dataset Specifics of the Ten Selected Projects

Project Name	**#Comments**	**#Filtered Comments**	**#SATD**	**SATD Prevalence**
Dubbo	5,875	1,649	85	1.45%
Gradle	15,901	3,324	321	2.02%
Groovy	14,199	4,435	249	1.75%
Hive	81,127	29,340	1,046	1.29%
Maven	5,448	1,219	136	2.50%
Poi	45,666	15,033	618	1.35%
SpringFramework	42,574	7,712	98	0.23%
Storm	12,258	3,639	92	0.75%
Tomcat	37,038	12,218	287	0.77%
Zookeeper	6,894	2,691	63	0.91%
Total	**266,980**	**81,260**	**2,995**	**1.12%**

4.3 Performance Metrics

Regarding identification performance, given the severe class imbalance inherent in the datasets, standalone Accuracy fails to objectively reflect the model's capabilities. Consequently, this study primarily employs the F1-score (Equation 5), derived from the confusion matrix, as the core metric for evaluating comprehensive performance. The F1-score is the harmonic mean of Precision (Equation 3) and Recall (Equation 4). In this binary classification task (where SATD is the positive class), these metrics are formally defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

To quantify annotation overhead, we adopt the Cost metric, defined as the cumulative number of labeled samples required for the model to achieve a specific performance threshold. A lower Cost value indicates higher efficiency in mitigating manual annotation effort.

Furthermore, to ascertain whether the performance disparities between different methodologies are statistically significant, we perform the Wilcoxon signed-rank test for significance analysis. A calculated p-value of less than the significance level of 0.05 indicates that the observed performance differences are statistically significant.

4.4 Experiment Configuration

Regarding the experimental evaluation paradigms, all within-project detection tasks are conducted using 10-fold cross-validation. Specifically, the dataset for each project is partitioned into ten equal subsets, which serve as the testing set in rotation, with the final results reported as the average over the ten iterations. For cross-project detection tasks, a Leave-One-Project-Out (LOPO) strategy is adopted. In this setting, one project is iteratively selected as the target testing set, while the remaining nine projects are integrated to form the training data pool. The baseline for Flan-T5 in both scenarios is reproduced following the methodology of Sheikhaei et al. (Sheikhaei et al. (2024)), utilizing their publicly shared codebase. The BERT baseline is reproduced according to the approach delineated by Gu et al. (Gu et al. (2024)).

Regarding the active learning hyperparameters, the initial training set size is configured at 10% of the total training pool, and the batch size for each proxy model query is set to 1%. To investigate the equilibrium between performance and cost, we establish multiple groups of stopping thresholds μ, ranging from 0.1 to 1.0 with a step size of 0.1.

In the hyperparameter configuration for the deep learning models, the maximum input sequence length for all models is truncated or padded to 128. The batch size is set to 4, the learning rate is configured at $2e-5$, and the AdamW optimizer is utilized uniformly. The training duration for the BERT proxy model is set to 10 epochs. Upon reaching the active learning stopping threshold, the Flan-T5-Large target model undergoes one-time full-parameter fine-tuning for 20 epochs.

5 Results

5.1 Within-Project Effectiveness (RQ1)

This section focuses on the performance of the Flan-T5-AL methodology in within-project scenarios, providing an in-depth analysis of the synergistic effects between Large Language Models (LLMs) and active learning by comparing the F1-scores of various models.

Table 2 illustrates the comparative F1-scores of each methodology across ten projects. Here, BERT and Flan-T5 denote the fully supervised baseline methods

Table 2 Comparative F1-scores of BERT, Flan-T5, and Flan-T5-AL across various projects

Project	Fully Supervised BERT	Fully Supervised Flan-T5	Flan-T5-AL (Ours)
Dubbo	0.711	0.745	**0.769**
Gradle	0.758	0.766	**0.771**
Groovy	0.786	0.815	**0.829**
Hive	0.764	0.803	**0.811**
Maven	0.731	0.706	**0.767**
Poi	0.868	**0.885**	0.883
SpringFramework	0.699	0.726	**0.741**
Storm	0.690	0.691	**0.735**
Tomcat	0.779	**0.794**	0.789
Zookeeper	0.601	0.604	**0.615**
Average	0.739	0.754	**0.771**

using their respective models, while Flan-T5-AL represents the proposed active learning approach integrated with a proxy screening mechanism. From an aggregate perspective, Flan-T5-AL demonstrates a pronounced performance advantage. The methodology achieves a mean F1-score of 0.771, representing a 0.014 improvement over the BERT-AL method (0.757) introduced in Section 3. This enhancement is particularly salient in projects characterized by complex semantics; for instance, in the Groovy project, Flan-T5-AL yields an F1-score of 0.829, outperforming BERT-AL's 0.805. Similarly, in the Maven project, Flan-T5-AL reaches 0.767, significantly exceeding BERT-AL's 0.745.

These empirical outcomes underscore that while BERT-AL successfully filters high-value samples via active learning, it remains constrained by the model's inherent scale and pre-training paradigm when deciphering elusive technical debt. Conversely, Flan-T5-Large, leveraging reasoning capabilities acquired through large-scale instruction tuning, more accurately captures the true semantics of hard samples situated near the decision boundary. This further augments recognition precision and validates the efficacy of the collaborative framework.

5.2 Cost Efficiency Analysis (RQ2)

To investigate the convergence speed of the models under varying labeling proportions, we plotted the evolutionary trends of F1-scores relative to the stopping threshold μ for each project, as illustrated in Figure 1. Observations of the trend curves for projects such as Dubbo, Gradle, Groovy, and Hive reveal that Flan-T5-AL rapidly ascends to a high-performance interval at extremely low labeling proportions (e.g., $\mu = 0.2$ or 0.3). Throughout the process, it consistently remains superior to or on par with both BERT-AL and the fully supervised Flan-T5. Notably, in the Maven project, the performance curve of Flan-T5-AL significantly surpasses BERT-AL in the early stages

and demonstrates a continuous upward trajectory. This suggests that the samples screened by the BERT proxy are effectively absorbed by the Flan-T5 model, indicating a high degree of consistency between the two models in assessing sample difficulty. Furthermore, the performance curve of Flan-T5-AL exhibits minimal fluctuations after reaching its peak, demonstrating robust stability and further validating the reliability of the collaborative framework in data filtering.

Figure 2 The results indicate that Flan-T5-AL demonstrates a clear cost advantage across all projects. In the Storm project, the annotation cost required to reach the fully supervised baseline performance is substantially reduced. In the Hive and SpringFramework projects, the Flan-T5-AL methodology achieves baseline performance with a significantly smaller sample size.

Table 3 Performance of Flan-T5-AL in Low-resource Settings and its Ratio Relative to Full-supervision Performance

Project	$\mu = 0.2$ **(Ratio)**	$\mu = 0.3$ **(Ratio)**	**Fully Supervised Flan-T5**
Dubbo	0.709 (95.2%)	0.696 (93.4%)	0.745
Gradle	0.693 (90.5%)	0.721 (94.1%)	0.766
Groovy	0.798 (97.9%)	**0.821 (100.7%)**	0.815
Hive	0.751 (93.5%)	0.778 (96.9%)	0.803
Maven	0.670 (94.9%)	**0.712 (100.8%)**	0.706
Poi	0.825 (93.2%)	0.849 (95.9%)	0.885
SpringFramework	0.676 (93.1%)	0.701 (96.6%)	0.726
Storm	0.648 (93.8%)	**0.692 (100.1%)**	0.691
Tomcat	0.698 (87.9%)	0.742 (93.5%)	0.794
Zookeeper	0.403 (66.7%)	0.441 (73.0%)	0.604

To determine the optimal stopping threshold for practical applications, Table 3 presents the performance of Flan-T5-AL in low-resource settings ($\mu = 0.2, 0.3$) and its ratio relative to full-supervision performance. The data reveal that when the stopping threshold $\mu = 0.3$ (i.e., labeling only 30% of the data), the performance ratio for the Groovy project reaches 100.7%, Maven reaches 100.8%, and Storm reaches 100.1%—all of which exceed the full-supervision performance. Hive and SpringFramework also achieve high levels of 96.9% and 96.6%, respectively. Collectively, at $\mu = 0.3$, the F1-scores of all projects either approach or surpass the fully supervised benchmarks. Consequently, this study suggests setting 0.3 as the optimal stopping threshold. This implies that in practical deployments, an LLM-driven, high-precision detector can be obtained by investing only 30% of the original manual annotation budget.

5.3 Cross-Project Generalization (RQ3)

Cross-project identification tasks necessitate an exceptional capacity for domain transfer to reconcile the distributional disparities between training and testing data.

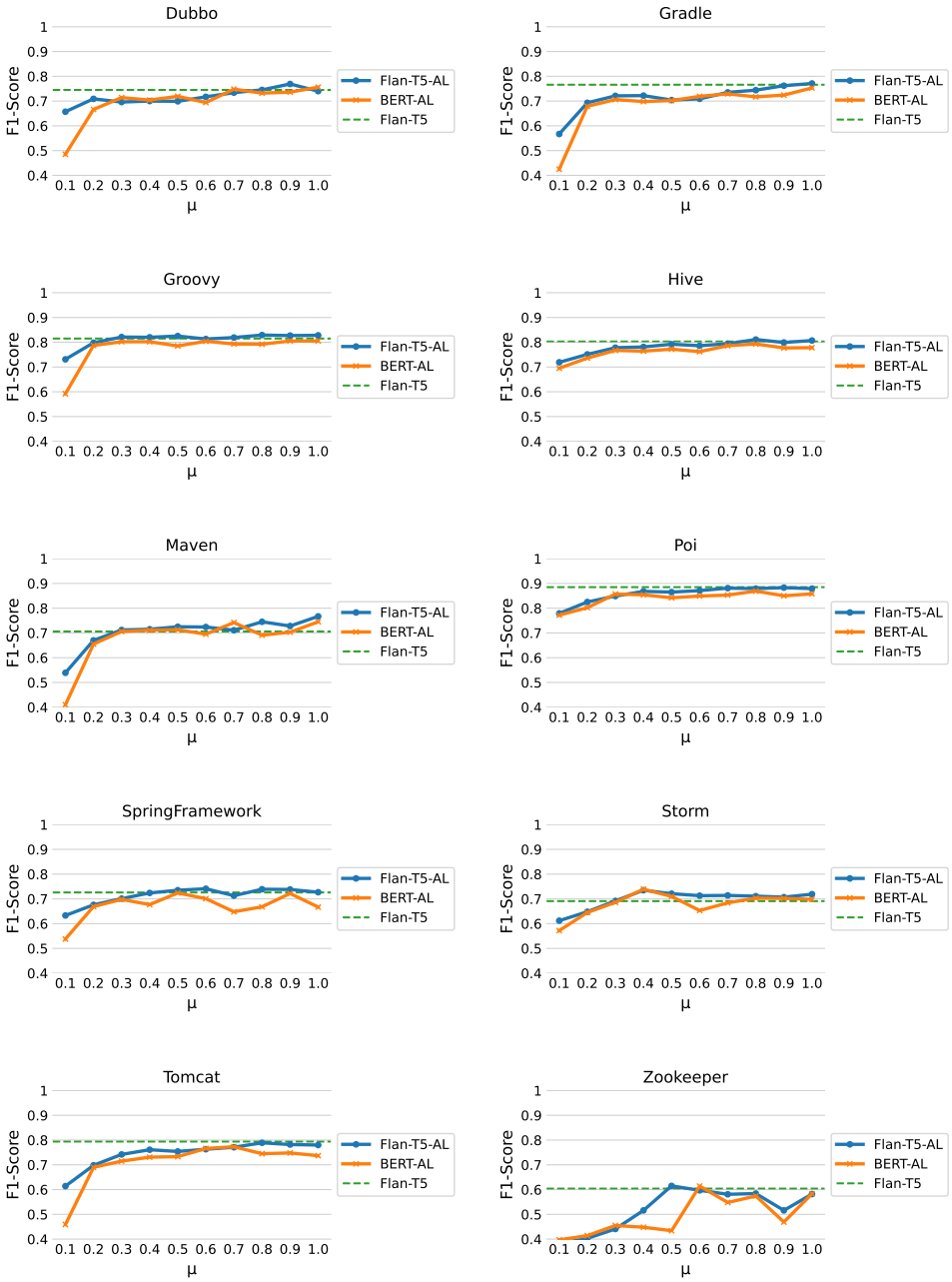


Fig. 1 Performance trends of F1-scores across ten projects under varying labeling proportions in within-project scenarios. The curves demonstrate that Flan-T5-AL rapidly converges to high-performance levels at low labeling ratios (e.g., $\mu = 0.2$ or 0.3), consistently outperforming or matching both BERT-AL and fully supervised Flan-T5 baselines.

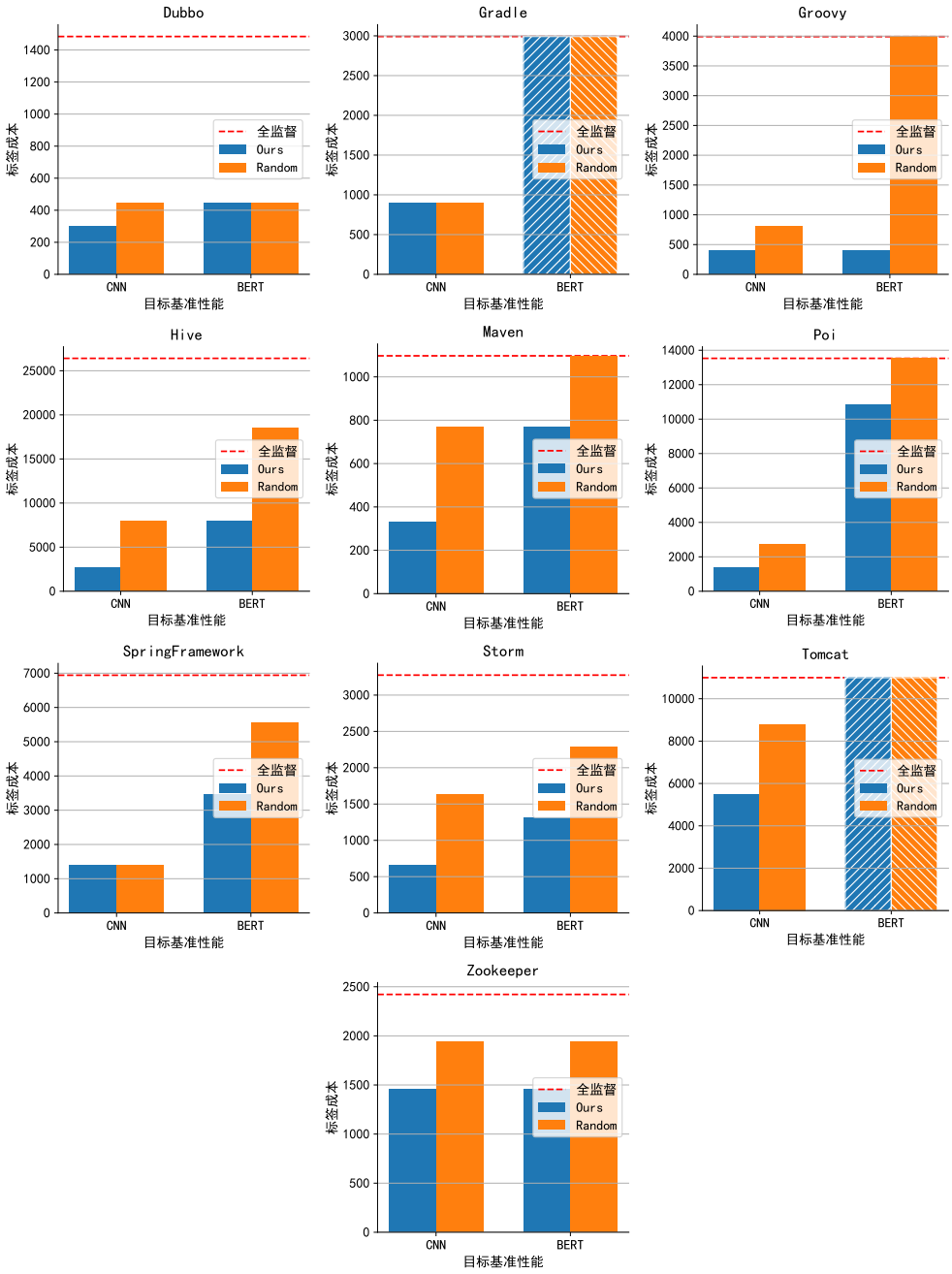


Fig. 2 Annotation cost comparison across different methods to achieve target baseline performance. Flan-T5-AL demonstrates clear cost advantages across all projects, requiring substantially fewer labeled samples than BERT-AL and fully supervised baselines to reach comparable performance levels in most projects including Storm, Hive, and SpringFramework.

Table 4 illustrates the granular results of the cross-project experiments. Initially, comparing the fully supervised BERT and Flan-T5, the latter achieves a mean F1-score of 0.761, significantly outperforming BERT's 0.736. Specifically, in the Storm project, Flan-T5 achieves 0.800 compared to BERT's 0.693; similarly, in SpringFramework, Flan-T5 (0.742) demonstrates a considerable lead over BERT (0.689). This suggests that BERT-based models are susceptible to overfitting on source projects—learning project-specific vocabulary or idiomatic expressions—which subsequently degrades generalization on target projects. Conversely, Flan-T5 benefits from instruction tuning on massive corpora, acquiring more generalized linguistic patterns and logical reasoning capabilities that offer inherent robustness for domain transfer.

Further examination of Flan-T5-AL reveals a mean F1-score of 0.776, which not only surpasses BERT by a significant margin but also exceeds the fully supervised Flan-T5 benchmark. In the Dubbo project, Flan-T5-AL achieves 0.775, outperforming Flan-T5's 0.761; in the Poi project, it reaches a high of 0.845. These outcomes unveil the unique value of active learning in cross-project contexts. By filtering based on uncertainty and representativeness, Flan-T5-AL essentially selects the most universal and representative samples from the source projects for learning. These instances typically encapsulate the essential semantic features of technical debt rather than project-specific noise. Consequently, the proposed method demonstrates superior generalization in cross-project scenarios.

Figure 3 The performance curves exhibit a steady upward trajectory as labeling data from the target project increases. In most projects, when the stopping threshold reaches approximately 0.3, the performance substantially surpasses the BERT and Flan-T5 baselines trained on the full source project data. This validates that the proposed methodology can achieve efficient domain transfer with only a minimal volume of target project data, underscoring its excellent generalization capability.

Table 4 Comparative F1-scores across various models in cross-project scenarios

Project	**Fully Supervised BERT**	**Fully Supervised Flan-T5**	**Flan-T5-AL (Ours)**
Dubbo	0.725	0.761	**0.775**
Gradle	0.771	0.791	**0.801**
Groovy	0.793	0.821	**0.831**
Hive	0.767	0.794	**0.805**
Maven	0.766	0.764	**0.793**
Poi	0.804	0.813	**0.845**
SpringFramework	0.689	0.742	**0.759**
Storm	0.693	**0.800**	0.797
Tomcat	**0.760**	0.733	0.739
Zookeeper	0.594	0.593	**0.612**
Average	0.736	0.761	**0.776**

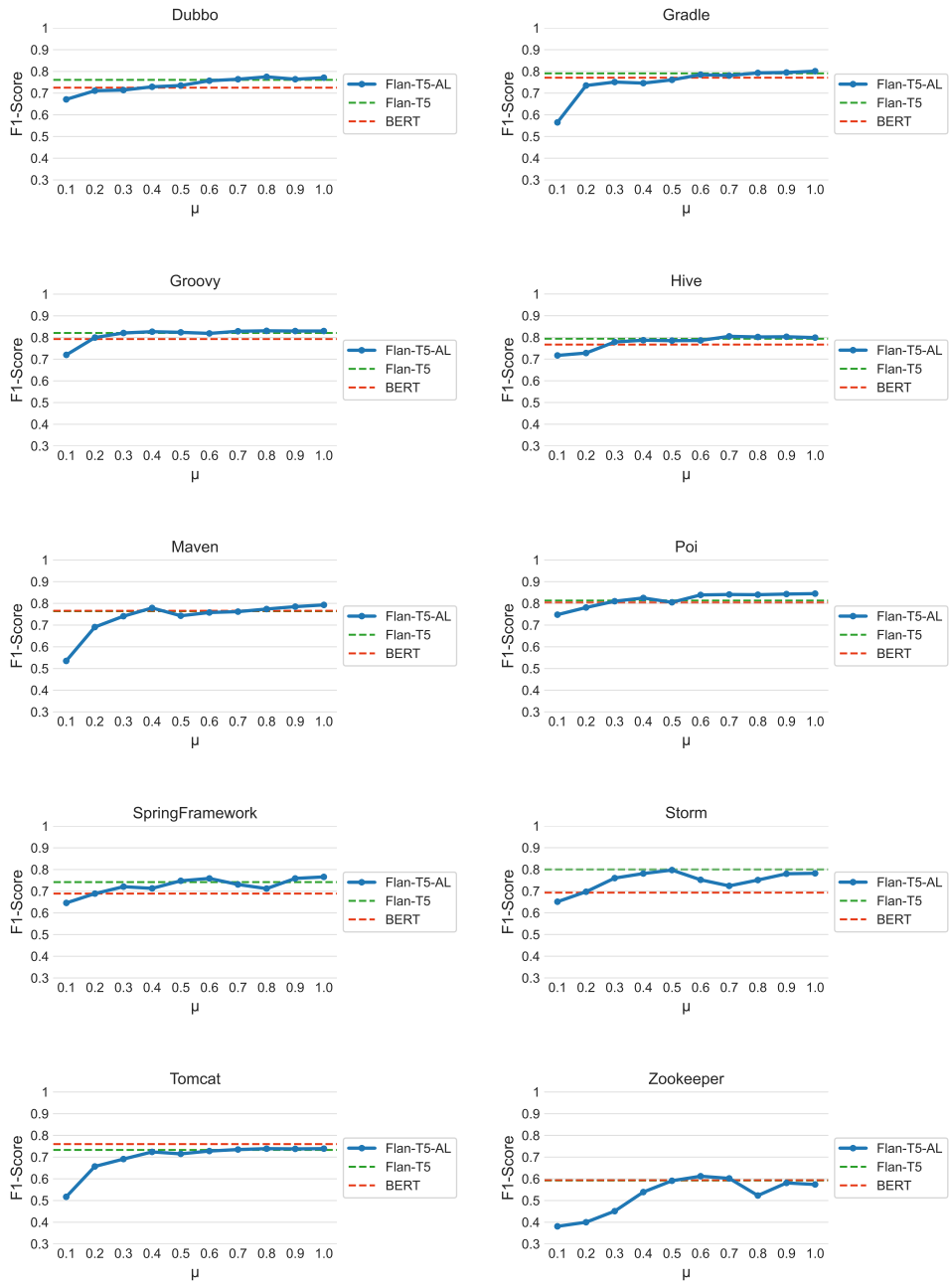


Fig. 3 Performance trends relative to the stopping threshold in cross-project scenarios. The curves show steady upward trajectories as labeling data increases, with Flan-T5-AL significantly surpassing BERT and Flan-T5 baselines when the stopping threshold reaches approximately 0.3, demonstrating superior generalization capability with minimal target project data.

6 Related Works

The identification of Self-Admitted Technical Debt (SATD) has emerged as a prominent research focus within the software engineering domain in recent years. Fundamentally, detecting SATD within source code comments is characterized as a quintessential text classification process (Ren et al. (2019)). A comprehensive review of existing literature reveals that the evolution of identification technologies has progressed through several distinct stages:

Early Text Matching and Traditional Machine Learning: Preliminary studies primarily relied on manual analysis and rudimentary Natural Language Processing (NLP) techniques. For instance, early exploratory research leveraged foundational text mining to analyze comments extracted from open-source projects (Potdar and Shihab (2014)). Other researchers proposed simple heuristic methodologies to extract SATD solely through predefined rules without reliance on training data (Guo et al. (2021)). Subsequently, automated identification models were introduced, marking the first instance of employing NLP combined with traditional classifiers, such as Maximum Entropy, for supervised recognition (da Silva Maldonado et al. (2017)).

Deep Learning-based Semantic Analysis: With the proliferation of deep learning, identification technologies transitioned into a paradigm dominated by neural networks. Initial efforts introduced Convolutional Neural Networks (CNN) to extract key phrases from comments (Ren et al. (2019)), while other scholars utilized Long Short-Term Memory (LSTM) networks to achieve fine-grained classification of debt (Santos et al. (2020)). Recently, the application of pre-trained models has further advanced performance benchmarks; for example, leveraging the BERT architecture for SATD detection has significantly augmented model efficacy (Aiken et al. (2023)).

Existing Attempts to Mitigate Annotation Costs: Despite the superior performance of deep learning models, their reliance on large-scale, high-quality labeled data remains a formidable barrier. To diminish manual annotation overhead, researchers have begun exploring detection methods for low-resource scenarios. The DebtFree framework, for example, served as a pioneering evaluation of using unsupervised and semi-supervised learning to reduce labeling costs (Tu and Menzies (2022)). Furthermore, Jitterbug introduced a "human-in-the-loop" paradigm employing a divide-and-conquer strategy, which automatically identifies easily distinguishable patterns before deferring challenging samples to human experts (Yu et al. (2020)).

Distinction of This Work: Although significant progress has been made, existing research continues to confront the "high-precision requires high-cost" dilemma. Current low-resource explorations often struggle to adequately capture complex semantic features, resulting in performance that trails behind state-of-the-art deep models. Distinct from existing work, this paper proposes a collaborative detection framework augmented by Large Language Models (LLMs) based on the "proxy selection" paradigm. This approach decouples sample screening from model training: it utilizes a lightweight discriminative model (BERT) integrated with a hybrid query strategy for high-frequency iterative screening to resolve cold-start issues and computational bottlenecks; subsequently, it collaborates with a generative LLM (Flan-T5) for one-time full-parameter fine-tuning. This design substantially mitigates manual annotation

costs while effectively transcending the performance ceiling of traditional models in understanding deep, elusive semantics.

7 Threats to Validity

The empirical findings in this study may be subject to several threats to validity, which we have endeavored to mitigate during the experimental design process:

7.1 Construct Validity

Construct validity primarily concerns potential biases in the selection of performance metrics and the configuration of model hyperparameters. Due to the extreme class imbalance inherent in the SATD datasets (with an average positive sample proportion of only 1.12%), standalone accuracy fails to objectively represent model performance. Consequently, we adopt the F1-score—the harmonic mean of precision and recall—as our core evaluation metric, and incorporate the Wilcoxon signed-rank test to ensure the statistical significance of the comparative results. Furthermore, the efficacy of deep learning models and active learning strategies is heavily dependent on hyperparameters such as batch size, learning rate, and stopping thresholds. Although we have configured these parameters based on best practices delineated in existing literature, potential biases in hyperparameter selection may still influence the optimal performance ceiling of the models.

7.2 External Validity

External validity focuses on the generalizability and robustness of our findings beyond the current experimental setting. First, our evaluation is grounded in a benchmark dataset comprising ten popular open-source Java projects. While these encompass diverse application domains, variations in comment styles and terminology across different programming languages (e.g., Python, C++, JavaScript) may exist. Thus, the generalizability of the proposed methodology to other language environments necessitates further validation. Second, the data utilized in this research originates exclusively from open-source software repositories. Development paradigms, coding conventions, and code review standards in open-source communities often diverge from those in proprietary industrial software projects. This disparity, to some extent, constrains the direct extrapolation of our findings to industrial closed-source scenarios.

8 Conclusion and Future Work

8.1 Conclusion

Addressing the challenges of prohibitive manual annotation overhead, extreme sample sparsity, and the intricate semantic interpretation inherent in Self-Admitted Technical Debt (SATD) identification, this paper proposes a decoupled deep active learning framework augmented by Large Language Model (LLM) collaboration. The framework initiates with a cold-start strategy integrating domain-specific keywords and representativeness to mitigate initial sampling bias. Subsequently, a computationally efficient

discriminative model (BERT) is utilized as a proxy query engine to execute a hybrid query strategy merging uncertainty and representativeness, facilitating the rapid localization of high-value instances. Finally, the target learner—a generative LLM (Flan-T5) with robust reasoning capabilities—undergoes full-parameter fine-tuning on the filtered high-fidelity dataset. Extensive empirical investigations demonstrate that the proposed collaborative framework achieves, and in several instances exceeds, fully supervised baseline performance using only approximately 30% of the labeled data. Furthermore, the framework exhibits superior domain transfer capabilities in cross-project scenarios, significantly lowering the practical threshold for deploying high-precision SATD identification systems.

8.2 Future Work

Grounded in our current empirical findings and the evolving landscape of Large Language Model technologies, future research may be directed along several trajectories:

Fine-grained Multi-class Classification: In practical software maintenance, distinguishing specific categories of technical debt (e.g., design debt, requirement debt, or defect debt) is of paramount importance. Future work could involve designing specialized instruction-tuning tasks to empower LLMs to automatically categorize technical debt, thereby offering more granular decision support for practitioners.

Cross-language Environment Validation: We intend to further collect SATD datasets spanning multiple programming languages (e.g., Python, C++). This will allow us to evaluate the applicability and robustness of the proposed collaborative framework across diverse linguistic paradigms and coding conventions.

Application of Parameter-Efficient Fine-Tuning (PEFT): Integrating PEFT techniques, such as LoRA (Low-Rank Adaptation) or Adapters, into the collaborative training phase remains a promising direction. By updating only a minimal subset of parameters for downstream tasks, we expect to further alleviate the computational barriers of LLMs, enabling the seamless deployment of high-precision detection models within standard development environments or edge devices.

Declarations

- Funding: The authors report that there are no funding details to declare.
- Conflict of interest/Competing interests: The authors declare that they have no conflict of interest.
- Ethics approval and consent to participate: Not applicable.
- Consent for publication: Not applicable.
- Data availability: The dataset used in this study is publicly available from Guo et al. (2021).
- Materials availability: Not applicable.
- Code availability: The code will be made available upon acceptance.
- Author contribution: Zhicheng Wang: Conceptualization, Methodology, Software, Experiments, Writing – original draft; Tianxin Lei: Data Curation, Experiments; Tie Bao: Supervision, Writing – review & editing, Funding acquisition.

References

Aiken W, Mvula PK, Branco P, et al (2023) Measuring improvement of f 1-scores in detection of self-admitted technical debt. In: 2023 ACM/IEEE International Conference on Technical Debt (TechDebt), IEEE, pp 37–41

Ash JT, Zhang C, Krishnamurthy A, et al (2019) Deep batch active learning by diverse, uncertain gradient lower bounds. arXiv preprint arXiv:190603671

Bavota G, Russo B (2016) A large-scale empirical study on self-admitted technical debt. In: Proceedings of the 13th international conference on mining software repositories, pp 315–326

Chowdhury S, Kidwai H, Asaduzzaman M (2025) Evidence is all we need: Do self-admitted technical debts impact method-level maintenance? In: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), IEEE, pp 813–825

Chung HW, Hou L, Longpre S, et al (2024) Scaling instruction-finetuned language models. Journal of Machine Learning Research 25(70):1–53

Culotta A, McCallum A (2005) Reducing labeling effort for structured prediction tasks. In: AAAI, pp 746–751

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

Dasgupta S (2011) Two faces of active learning. Theoretical computer science 412(19):1767–1781

Dor LE, Halfon A, Gera A, et al (2020) Active learning for bert: an empirical study. In: Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP), pp 7949–7962

de Freitas Farias MA, de Mendonça Neto MG, da Silva AB, et al (2015) A contextualized vocabulary model for identifying technical debt on code comments. In: 2015 IEEE 7th international workshop on managing technical debt (MTD), IEEE, pp 25–32

Gu H, Zhang S, Huang Q, et al (2024) Self-admitted technical debts identification: How far are we? In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp 804–815

Guo Z, Liu S, Liu J, et al (2021) How far have we progressed in identifying self-admitted technical debts? a comprehensive empirical study. ACM Transactions on Software Engineering and Methodology (TOSEM) 30(4):1–56

He H, Garcia EA (2009) Learning from imbalanced data. IEEE Transactions on knowledge and data engineering 21(9):1263–1284

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

Mack Y, Rosenblatt M (1979) Multivariate k-nearest neighbor density estimates. Journal of Multivariate Analysis 9(1):1–15

Maldonado EdS, Shihab E (2015) Detecting and quantifying different types of self-admitted technical debt. In: 2015 IEEE 7th international workshop on managing technical debt (MTD), IEEE, pp 9–15

Margatina K, Vernikos G, Barrault L, et al (2021) Active learning by acquiring contrastive examples. arXiv preprint arXiv:210903764

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

Raffel C, Shazeer N, Roberts A, et al (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of machine learning research 21(140):1–67

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

Santos RM, Junior MCR, de Mendonça Neto MG (2020) Self-admitted technical debt classification using lstm neural network. In: 17th International Conference on Information Technology–New Generations (ITNG 2020), Springer, pp 679–685

Scheffer T, Decomain C, Wrobel S (2001) Active hidden markov models for information extraction. In: International symposium on intelligent data analysis, Springer, pp 309–318

Schröder C, Müller L, Niekler A, et al (2023) Small-text: Active learning for text classification in python. In: Proceedings of the 17th conference of the European chapter of the association for computational linguistics: system demonstrations, pp 84–95

Settles B (2009) Active learning literature survey. Tech. rep., University of Wisconsin-Madison Department of Computer Sciences

Shannon CE (1948) A mathematical theory of communication. The Bell system technical journal 27(3):379–423

Sheikhaei MS, Tian Y, Wang S, et al (2024) An empirical study on the effectiveness of large language models for satd identification and classification. Empirical Software Engineering 29(6):159

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

Tu H, Menzies T (2022) Debtfree: minimizing labeling cost in self-admitted technical debt identification using semi-supervised learning. Empirical Software Engineering 27(4):80

Wei J, Bosma M, Zhao VY, et al (2021) Finetuned language models are zero-shot learners. arXiv preprint arXiv:210901652

Xavier L, Ferreira F, Brito R, et al (2020) Beyond the code: Mining self-admitted technical debt in issue tracker systems. In: Proceedings of the 17th international conference on mining software repositories, pp 137–146

Yu Z, Fahid FM, Tu H, et al (2020) Identifying self-admitted technical debts with jitterbug: A two-step approach. IEEE Transactions on Software Engineering 48(5):1676–1691

Zhan X, Wang Q, Huang Kh, et al (2022) A comparative survey of deep active learning. arXiv preprint arXiv:220313450

Zhu J, Wang H, Yao T, et al (2008) Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In: 22nd international conference on computational linguistics, coling 2008, pp 1137–1144