Noname manuscript No.
(will be inserted by the editor)

DebtFree: Minimizing Labeling Cost in Self-Admitted Technical Debt Identification using Semi-Supervised Learning

Huy Tu · Tim Menzies

Received: date / Accepted: date

Abstract Keeping track of and managing Self-Admitted Technical Debts (SATDs) is important for maintaining a healthy software project. Current active-learning SATD recognition tool involves manual inspection of 24% of the test comments on average to reach 90% of the recall. Among all the test comments, about 5% are SATDs. The human experts are then required to read almost a quintuple of the SATD comments which indicates the inefficiency of the tool. Plus, human experts are still prone to error: 95% of the false-positive labels from previous work were actually true positives.

To solve the above problems, we propose DebtFree, a two-mode framework based on unsupervised learning for identifying SATDs. In mode1, when the existing training data is unlabeled, DebtFree starts with an unsupervised learner to automatically pseudo-label the programming comments in the training data. In contrasts, in mode2 where labels are available with the corresponding training data, DebtFree starts with a pre-processor that identifies the highly prone SATDs from the test dataset. Then, our machine learning model is employed to assist human experts in manually identifying the remaining SATDs. Our experiments on 10 software projects show that both models yield statistically significant improvement in effectiveness over the state-of-the-art automated and semi-automated models. Specifically, DebtFree can reduce the labeling effort by 99% in mode1 (unlabeled training data), and up to 63% in mode2 (labeled training data) while improving the current active learner's F1 relatively to almost 100%.

Keywords Technical Debt · Semi-Supervised Learning · Unsupervised Learning · Labeling Effort

H. Tu and T. Menzies
Department of Computer Science,
North Carolina State University,
Raleigh, USA
E-mail: hqtu@ncsu.edu and timm@ieee.org

1 Introduction

When developers rush out code, that code often contains *technical debt* (TD), i.e. decisions that must later be repaid with further work. As the initial step towards understanding and resolving TDs, many research [25, 55, 84] first detecting the intentionally documented (via comments) TD, i.e., self-admitted TD (SATD). SATDs are crucial to identify as they (1) are diffused in the codebase [84]; (2) can survive long-term (more than 1,000 commits) [5]; and (3) complicate maintainability of the software [22, 73] State-of-the-art (SOTA) works have identified SATDs automatically [55] or semi-automatically [84].

However, models that recognize TD must be learned from *labeled data*. Generating such labels can be extremely slow and expensive. For instance, Tu et al. [67] reported that manually labeling $22,500+$ commits required 175 person-hours, including cross-checking. Due to the labor-intensive nature of the process, researchers often reuse datasets labeled from previous studies. For instance, Lo et al., Yang et al., and Xia et al. certified their methods using data generated by Kamei et al. [31, 78, 79, 81]. While this practice allows researchers to rapidly test new methods, it leaves the possibility for any labeling mistake to propagate to other related works. In fact, before reusing Maldonaldo et al.'s data [37] to identify SATDs, Yu et al. [84] discovered that more than 98% of the false positives were actually true positives, casting doubt on related work using the original dataset. Hence, it is timely to ask:

Can we reduce the labeling effort associated with building models for technical debt?

An unsupervised learning technique that learns patterns from unlabeled data is a promising direction in SATD identification. However, without supervision, the technique alone can be ineffective. As illustrated in Figure 1, our approach is to first demonstrate that previous methods can be extended or integrated with unsupervised learning to greatly reduce the labeling effort while effectively recognizing SATDs. This proposed method, called **DebtFree**, includes the combination of three separate approaches in a novel manner:

1. **Pseudo-Labeling**: This step is required if the training data does not have any labels to start with. First, we frugally pseudo-labels the training data with unsupervised learning, i.e., identifying hidden patterns in data in order to map unlabeled examples in two groups. Intuitively, the more complex the data instance [65], the more likely that the comment is describing a SATD. CLA by Nam et al.[47] is an example of an unsupervised classifier that recognizes "complex" examples (those with many values above the median). The intuition here is well documented [44, 65, 14, 26].
2. **Filtering**: This step is optional. We identify early and remove instances from the test dataset that are likely to be SATDs.
3. **Active Learning**: This step is always required. We train on some labeled data and then guide the human experts to manually find the comments that are most likely to contain SATDs. It is critical to assess whether the labeled data is insightful enough to guide the human experts for the entire labeling process. If not, we propose Falcon, a new active learning policy to take advantage of such data while still ensuring effectiveness.

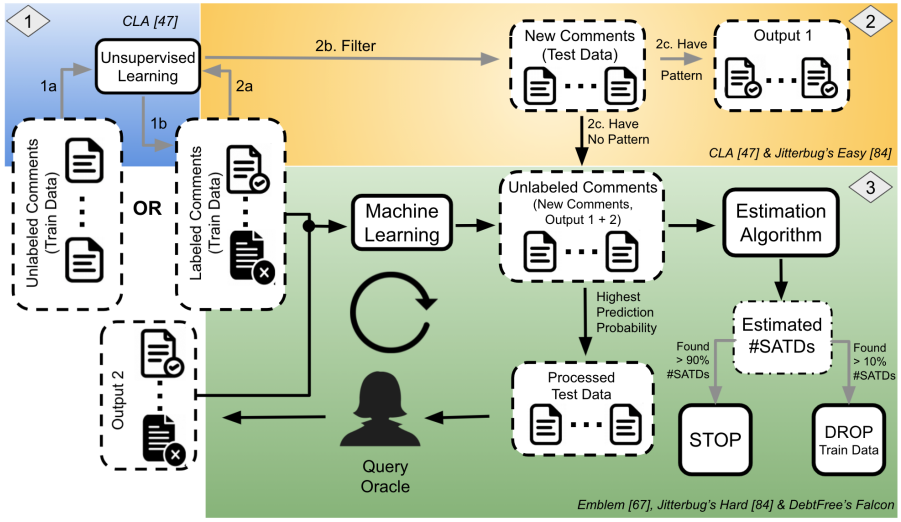


Fig. 1: Workflows of **DebtFree** = Pseudo-Labeling (via Unsupervised Learning, i.e., CLA [47]) + Filtering (via CLA [47] or Jitterbug's Easy [84]) + Active Learning (via Emblem [67], Jitterbug's Hard [84], or this study's Falcon). Step 1 is required if there is no access to the training data's labels. Step 2 is optional while Step 3 is required at all times. The gray arrows indicate different configurations of the method that will be investigated for this study.

In this work, we aim to better data generation associated with building models for SATDs identification by reducing the labeling effort come from manual method [67] and improving the labeling quality of fully automated methods [84]. Moreover, our investigation also showed that the effort-aware method we propose, **DebtFree**, also performs statistically similar or even better than two SOTA works [84, 55]. To understand and validate this end-to-end method, **DebtFree**, we investigate the following research questions:

RQ1: How well can the state-of-the-art unsupervised learning method identify SATDs? We investigate variants that stem from Nam et al.'s unsupervised learning CLA method. As these methods leverage on hidden patterns within the data, we compare them to the pattern-based SOTA for identifying SATDs, also by Yu et al. [84].

Result:
From our exploration of various unsupervised learners, the original CLA by Nam et al. performs the best. Moreover, CLA performs similarly to the SOTA pattern-based approach, Easy [84], without having access to the data's labels (100% less effort).

RQ2: How can the state-of-the-art active learning framework be combined with the state-of-the-art unsupervised learner? From RQ1, unsupervised learning methods are promising but not optimal. Hence, we study different combinations by incorporating a chosen unsupervised learner with several SOTA active learning frameworks in SE.

Result:
With the effort-aware theme, we investigate different combinations of active learners and CLA across two settings of the training data, either with having 1-*no access* and 2-*access* to the labels to propose **DebtFree**. In setting 1, or **DebtFree**(0), the best combination is Pseudo-Labeling (via CLA) with our proposed active learner, Falcon. In setting 2, or **DebtFree**(100), the best combination is Filtering (via CLA) with the SOTA active learner for SATDs identification, Hard.

RQ3: How does the proposed DebtFree perform against state-of-the-art models in identifying SATDs? After finalizing two combinations from RQ2 to propose **DebtFree**(0)/(100), it is essential to assess their usefulness by comparing against the SOTA models for SATDs identification.

Result:
When comparing against the SOTA semi-supervised learning work by Yu et al. [84] and the SOTA supervised learning work (with deep learning) by Ren et al. [55], our proposed method **DebtFree** outperforms them significantly. First, **DebtFree**(100) performs similarly to Ren et al. [55]'s work and better than Yu et al. [84]'s work while reducing the labeling cost by 2.5 times. Second, **DebtFree**(0) performs similarly to Ren et al. [55]'s work without having access to the training data's labels and outperforms Yu et al. [84]'s work while expending 99% less effort.

Our contributions to the field of software analytic are:

1. This work is the first to assess the usage of unsupervised learning to reduce the cost of labels labeling in identifying SATDs.
2. In the low-resource setting (training data with no label), our unsupervised methods outperform the prior SOTA models while requiring less knowledge (prior work used 100% labeled data while we get by with very little) [55, 84]. Counting the training data, we can reduce 99% of the number of examples that have to be labeled. This is the largest reduction ever reported in the effort required to commission a SATD identification model.
3. In the high-resource setting (training data with labels), we propose an improvement to the two-step Jitterbug technique [84] by replacing the pattern-based approach with an unsupervised learner to help reduce the commissioning effort of labeling on new data by 62.5% (5/8).
4. Our proposed active learning scheme, Falcon, outperforms both SOTA deep learning method [55] and SOTA two-step method [84] across both low-resource and high-resource settings.
5. Nearly all the prior work on unsupervised learning focus on defect prediction [21, 47, 49, 75, 76, 77, 80, 81, 87, 88]. The performance of our framework suggests that many more domains in software analytics could benefit from unsupervised learning.
6. To better support other researchers our scripts and data are on-line at https://github.com/HuyTu7/DebtFree.

The rest of this paper is structured as follows. *Section 2* discusses the motivation, background and related works. *Section 3* describes our methodology. *Section 4* focuses on our experimental design, while *section 5* analyzes the results. *Section 6* and *7* discuss our short-comings and directions for future work, respectively.

2 Motivation and Background

2.1 On the merits of studying Technical Debt and SATDs

Technical Debts (TDs) are introduced in the software when developers make decisions based on short-term benefits instead of long-term stability. TDs can accumulate interest similar to financial debts if they are not resolved in a timely manner. In 2012, after interviewing 35 software developers from diverse projects in different companies, varying both in size and type, Lim et al. [34] found developers generate technical debts due to factors like increased workload, unrealistic deadline in projects, lack of knowledge, boredom, peer-pressure among developers, unawareness or short-term business goals of stakeholders, and reuse of legacy, third-party, or open-source code. After observing five large-scale projects and companies in two studies, Wehaibi et al. [73] and Martini and Bosch [42] found that the number of technical debts in a project may be very low (only 3% on average). However, those TDs contaminate other parts of a software system and create a significant amount of defects in the future. Fixing such technical debts is more difficult than regular defects, often twice the cost if not resolving immediately [24]). The Software Improvement Group study by Nugroho et al. [50] offers a cost estimate of TD accumulation: a regular mid-level project owes $\$857,500$ in TD and resolving TD has a Return On Investment of 15% in seven years. Yet, limited success has been achieved despite a large body of research on identifying TD as part of Code Smells using static code analysis [20, 39, 40, 41, 86]. Static code analysis has a high rate of false alarms while imposing complex and heavy structures for identifying TD [2, 23, 63, 64].

Therefore, several researchers proposed to target self-admitted technical debt identification as the first step since they are often intentionally documented or "self-admitted" (via source code comments) by the developers. Some examples of SATDs within the data are shown in Table 1. In summary, identifying and resolving SATDs have several benefits:

Table 1: Examples of SATD comments.

Project	**SATD comments**
Apache Ant	`// cannot remove underscores due to protected visibility >:(`
EMF	`// TODO Binary incompatibility; an old override must override putAll.`
JFreeChart	`// do we need to update the crosshair values?`
JMeter	`// Can be null (not sure why)`
SQuirrel	`// is this right???`
ArgoUML	`// Why does the next part not work?`

– Removing SATDs early reduces the maintenance cost of a software project. As reported by Wehaibi et al. [73] and Wang et al. [71], these SATDs have

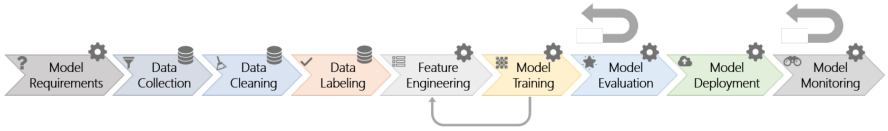


Fig. 2: Nine stages of the machine learning workflow from a case study at Microsoft by Zimmermann et al. [4]. Some stages are data-oriented (e.g., data collection, cleaning, and labeling) and others are model-oriented (e.g., model requirements, feature engineering, model training, evaluation, deployment and monitoring).

negative implications on the software development process, in particular by making it more difficult to change in the future.
- With SATDs elimination, software projects have better evolvability trajectory for accelerating new functionalities addition and integration.
- We can leverage those easily found SATDs as cheap training data for recognizing TDs [84]. SATDs are the documents of TDs that have been "admitted" by the developers, so they are not a specific type of TDs. SATDs cover different types of TDs such as code, defect, and requirement debts by Bavota et al.'s categorization [5]. In other words, as long as the document refers to some aspect of technical debt it is treated as SATD. According to the recent TDs categorization study, Fucci et al. [22] showed how SATDs are mapped across 10 categories, e.g., poor implementation choices, partially implemented, functional issues, etc.

2.2 Methods for Identification of Technical Debt

One of the goals of industrial analytics is that new conclusions can be quickly obtained from new data just by applying data mining algorithms. As shown in Figure 2, there are at least nine separate stages that must be completed before that goal be reached [4]. Each of these stages offers unique and separate challenges, each of which deserves extensive attention. Many of these steps have been extensively studied in the literature [15, 16, 17, 30, 35, 37, 38, 53, 84, 85]. However, the labeling work of step 4 has been receiving scant attention. In literature, there are several approaches for executing the labeling process:

1. Manual labeling;
2. Crowd sourcing;
3. Reuse of labels;
4. Automatic labeling;
5. Active learning (which is a special kind of semi-supervised learning)

All of these approaches have their drawbacks; e.g. they are error-prone or will not scale. In response to these shortcomings, this study will take two directions:

- First, we will try a *label-free* approach using a combination of pure *unsupervised learning* techniques to pseudo-label the data, and subsequently *active* learning, i.e., **DebtFree**(0);

– If the *label-free* approach fails, then we will try a hybrid of an *active-learning* approach, called **DebtFree**(100), which starts with the help of *unsupervised learning* to first filter out the highly technical-debt prone comments before the incrementally learning on all of the SATDs.

2.2.1 Manual labeling

In manual labeling, a team of (e.g.) graduate students assigns labels then (a) cross-checks their work via say, a Kappa statistic; then (b) use some skilled third person to resolve any labeling disagreements [36, 66, 67].

Manual labeling can be very slow. Tu et al. recently studied a corpus of 678 Github projects [67, 66]. A random selection of 10 projects from that corpus had 22,500 commits, which took 175 hours to manually label the commits *buggy, non-buggy* (time includes cross-checking). That is, in a hypothetical situation of manual labeling 500 projects (with each project has 5,000 commits) would have required 90 weeks of work.

2.2.2 Crowd Sourcing

Tu et al. [67] offers a cost estimate of what resources would be required to sub-contract that effort to dozens of crowd sourced workers via tools like Mechanical Turk (MT). Applying best practices in crowd sourcing [10], assuming (a) at least USA minimum ages [60]; and (b) our university taking a 50% overhead tax on grants; then crowd sourcing the labeling of the issues from 500 projects would require $320,000 of grant reserve.

2.2.3 Reusing Labels

Because manual labeling can be time-consuming, crowd sourcing too expensive, and micro-labeling error-prone, researchers often reuse labels from previous studies [78, 79, 81]. This approach is unsatisfactory for two reasons. One, when exploring a new domain, there may be no relevant, pre-existing labels to reuse. Two, reusing labels might propagate unsatisfactory label instances for future work. For example, the widely cited NASA datasets in defect prediction were found to have dubious quality [52, 59] in 2013 but has been utilized since then. Specifically, Yu et al. [84] were exploring self-admitted technical debt and found that their classifiers had an alarming high false positive rate. But when they manually checked the labels of their data taken from a prior study by Maldonado and Shihab [37], they found that over 98% of the reused false-positive labels were incorrect. Table 2 shows some example comments whose labels were updated in Yu et al. [84]'s study.

2.2.4 Automatic labeling

If labels cannot be generated manually or reused from other papers, using automatic labeling processes is an attractive alternative. For example, defect

Table 2: Examples of different labels from the original datasets curated by Maldonado and Shihab [37] and the updated datasets by Yu et al. [84]

Project	Comment Text	Original Label [37]	Yu et al.'s Label [84]
Apache Ant	`//TODO Test on other versions of weblogic //TODO add more attributes to the task, to take care of all jspc options //TODO Test on Unix`	no	yes
ArgoUML	`// skip backup files. This is actually a workaround for the cpp generator, which always creates backup files (it's a bug).`	no	yes
JFreeChart	`// FIXME: we've cloned the chart, but the dataset(s) aren't cloned and we should do that`	no	yes
JRuby	`// All errors to sysread should be SystemCallErrors, but on a closed stream Ruby returns an IOError. Java throws same exception for all errors so we resort to this hack...`	no	yes
Columba	`// FIXME r.setPos();`	no	yes

prediction papers [8, 28, 31, 32, 45, 48, 56] can label a commit as "bug-fixing" when the commit text contains certain keywords (e.g. "bug", "fix", "wrong", "error", "fail", etc [67]). Vasilescu et al. [68, 69] noted that these keywords are used in a somewhat ad hoc manner (researchers peek at a few results, then tinker with regular expressions that combine these keywords). Tu et al. [67] had found that these simplistic keyword approaches can introduce many errors, perhaps due to the specialization of the project nature or the ad-hoc nature of their creation [68].

Again, TDs are often "self-admitted" by developers in code comments [53] as shown in Table 1 in order to signal other developers that the corresponding code has **R-3c** TD that will be resolved for better results. In 2014, after studying four large-scale open-source software projects, Potdar and Shihab [53] concluded that developers may intentionally leave traces of TDs (i.e., SATDs) in their comments, such as "*hack, fixme, is problematic, this isn't very solid, probably a bug, hope everything will work, fix this crap*"). These comments tend to make SATDs much easier to find. Identifying and tracking SATDs have three important benefits as indicated §2.1. There are two prominent approaches to automatically identify SATDs:

1. Pattern-based approaches [15, 16, 17, 37, 53] consist of three steps: (1) manually inspect code comments and label each one as SATD or non-SATD, (2) manually analyze the labeled items and summarize patterns for SATDs, e.g., if a comment has keywords like "*hack, fixme, probably a bug*", then it has a high chance of being related to a SATD, (3) apply the summarized patterns to unlabeled comments to identify SATDs. Instead, Yu et al. [84] proposed Easy as the SOTA pattern-based method to automatically identify 20-90% of SATDs by finding patterns associated with high precision (close to 100%).

Limitation of the SOTA Pattern-based approach: this approach does need extensively labeled training data to find patterns that are associated with SATDs because it relies on precision. As mentioned above, generating that data requires intensive labor and expensive cost. Moreover, this method can still miss up to 80% of SATDs.

2. Machine Learning approaches [30, 35, 38, 85] involve models working in the *supervised learning* manner, which are trained on labeled SATD datasets to learn the underlying rules of comments admitting TDs. For example, Tan et al. [61, 62] analyzed source code comments using natural language processing to understand programming rules and documentations and indicates comment quality and inconsistency. In 2017, Maldonado et al. [38] successfully identified two types of SATD in 10 open-source projects (average 63% F1 Score) using Natural Language Processing (Max Entropy Stanford Classifier) using only 23% training data. Huang et al. [35] introduced a Multinomial Naive Bayes sub-classifier for each training dataset using information gain as feature selection then combine those sub-classifiers with boosting technique to achieve an average of 73% F1 scores [30]. A recent IDE for Eclipse was also released using this technique for identifying SATD in Java projects [35]. Recently, some studies explore different feature engineering for identifying SATDs, e.g. Wattanakriengkrai et al. [72] applied N-gram IDF as features, and Flisar and Podgorelec [19] explored how feature selection with word embedding can help the prediction. The latest progress are from Wang et al. [71]'s HATD and Ren et al. [55]'s tuned CNN utilized a deep convolutional neural network to achieve a higher F1 score than all the previous solutions. The HATD paper asserts that their algorithm defeats CNN but, after much effort, we could not reproduce that result[1]. These machine learning models can be a good indicator for which comments are more likely to be related to SATDs.
Limitation of the SOTA Machine Learning approach: deep learners often require having access to a substantial amount of labeled data which is not always available, especially in new domains (e.g., the success of open-source projects). With precision ranging from 60% to 85%, it is not reliable to fully automate the process. Human experts are then required to verify every decision the machine learning model made and thus costs a large amount of time and labor.

2.2.5 Semi-supervised Learning

Finally, another approach is to only label a representative sample of the data, build a classifier from that sample, then use that classifier to label the remaining data [74]. To find that representative example, an unsupervised learner (e.g. associations rule learner), a clustering algorithm, or an instance selection algorithm is used to find repeated patterns in the data [32]. Then a human oracle is asked to label one exemplar from each pattern. More sophisticated versions of this scheme include *active learners*, where an AI tool advances ahead of the human to fetch the most informative next sample to be labeled [33, 58]. If humans agree to first label only the most informative examples, then active learners can be used to produce better models more efficiently by reducing the number of examples that humans have to label.

[1] We found that there is no reproduction package published with HATD. We tried contacting the authors of that paper, without success.

The more general term for *active learning* is *semi-supervised learning*. Both terms mean "do what you can with a small sample of the labels" while *active learning* adds a feedback loop that checks new labels one at a time with an oracle. Moreover, semi-supervised learning relies on partially labeled data and mostly unlabeled data.

Since 2012, active learning approaches have received scarce attention in SE [33, 67, 83, 84]. Initially, active learning seems to be a promising method for addressing the cost of label checking and generating: for self-admitted technical debt, only 24% on a median of the training corpus had to be labeled [84]; using active learning, effort estimation for N projects only needed labels on 11% of those projects [33]; further, while seeking 95% of the vulnerabilities in 28,750 Mozilla Firefox C and C++ source code files, humans only had to inspect 30% of the code [83]. However, active learning still produces disappointing results. For example, it is still a daunting task to "only" label 5% to 10% of the projects in the 1,857,423 projects in RepoReapers [46] or the 9.6 million links explored by Hata et al. [27]. Although it might be justified for very mission-critical projects, consider the Firefox study [83] which required the human effort of inspecting 28,750 (total source code files) x 30% = 8,625 source code files to identify 95% of the vulnerabilities. This is beyond the resources of most analysts.

Several two-step frameworks were proposed for the active learning approach. Yu et al. [84] proposed Jitterbug to identify SATDs: (1) identify patterns for the "easy to find" SATDs (20-90% of all SATDs) with close to 100% precision and automatically classify comments with the patterns as SATDs (without human verification), (2) apply machine learning techniques to guide human experts to find the remaining "hard to find" SATDs with least number of comments read. Interestingly, Guo et al. [25] utilized a similar idea but using only four keywords (*"fixme, todo, hack, xxx"*) to identify the "easy to find" SATDs and applied supervised learning models to incrementally find the remaining "hard to find" SATDs.

Limitation of the SOTA Active Learning approach:

- Costly in real-world: both steps (pattern-based method and machine learning) require the training data to be labeled in order to proceed which can be costly in the real world, especially in a new domain. More importantly, Jitterbug's active learning strategy relies on the first step of the pattern-based method in order to reach the target recall. Thus, without the labeled training data, Jitterbug's guarantee in reaching the user-specified recall would be almost impossible.
- Difficult for Active Learning: the first step of the pattern-based approach identified up to 90% of the SATDs, but this makes it difficult for the active learning strategy to find the rest of 10% SATDs. This can increase human effort to review the labels. This will be confirmed later in §6.

Hence, in this SATDs identification work, we aim to reduce the labeling cost of both SOTA works including the SOTA semi-supervised learner from Yu et al. [84] and the SOTA supervised learner from Ren et al. [55]. In the process of developing such a method, our investigation shows our proposed

Table 3: Differences between two SOTAs of Ren et al. [55] and Yu et al. [84] against our work, DebtFree. Again, the first two steps of DebtFree are optional as they will be investigated later.

	Ren et al. [55]	**Yu et al. [84]**	**DebtFree**
Learning Type	Supervised	Semi-Supervised	Semi-Supervised
Core Process	1. Trains the deep learning model CNN on $(N-1)$ labeled datasets. 2. Uses the trained CNN model to identify SATDs on the target i dataset	1. Filtering out easy SATDs in the target i dataset with the pattern recognizer, Easy, to learn patterns with higher than 80% precision on $(N-1)$ labeled datasets. 2. Trains the RF model on $(N-1)$ labeled datasets and incrementally update the model in the active learning manner.	1. (Optional) Pseudo-labels the $(N-1)$ datasets with the unsupervised learner, CLA [47]. 2. (Optional) Filtering out easy SATDs in the target i dataset with the pattern recognizer (e.g., Easy or CLA) on $(N-1)$ pseudo-labeled/labeled datasets. 3. Trains the RF model on $(N-1)$ pseudo-labeled/labeled datasets and incrementally update the model in the active learning manner.
Labeling Effort	- Use all 100% labels from $(N-1)$ datasets. - No labeling done on the target i dataset.	- Use all 100% labels from $(N-1)$ datasets. - On average, labeling 23% on the target i dataset.	- Use 0% labels from $(N-1)$ datasets with step 1. - On average, labeling 11% on the target i dataset.

method, **DebtFree**, also performs statistically similar or even better. The differences between our approach and their are described in Table 3. The investigation explores the usefulness of *unsupervised learning* through a mix of approaches: (1) unsupervised learning in low-resource setting (unlabeled data) can frugally pseudo-label the training data, (2) an unsupervised learner acts as a preprocessor to filter out SATDs without relying on data labels in high-resource settings (labeled data), and (3) a tuned active learning strategy to specialize the learning on the target dataset by filtering out the training datasets when there is no benefit from learning on them anymore.

In order to overcome the previously documented limitations in identifying SATDs (i.e., the previous supervised learning and active learning methods are expensive), unsupervised learning is a promising direction, but not competent enough. To address this literature gap, it is an opportune time to propose the **DebtFree** framework, which is based on the integration of unsupervised learning and active learning. **DebtFree** investigates the combination of three approaches including pseudo-labeling, filtering, and active learning with different candidates for each approach. **DebtFree**'s configurations are formulated after exploring two scenarios: training data labels are known and training data labels are unknown. In the case of low-resource (training data with no labels), we will pick **DebtFree**(0). On the other hand, given resources (training data with labels), **DebtFree**(100) is employed instead.

3 Methodology

3.1 General Framework

Our proposed **DebtFree** is an end-to-end solution that labels the data, extends the data corpus, and identifies SATDs in a semi-supervised learning approach. **DebtFree** is comprised of two settings **DebtFree**(0) and **DebtFree**(100).

3.2 DebtFree(0)

When there is no access to the labels of the training data, our study shows that the filtering step is not needed here and the best combination for **DebtFree**(0) consists of two steps: unsupervised learning with CLA [47] to cheaply pseudo-labels the training data, and then our proposed active learning strategy, Falcon, to incrementally update and learn to identify the SATDs on the test data.

3.2.1 Pseudo-labeling via CLA/CLAFI

In the SOTA literature and comparative study of unsupervised models in defect prediction, CLA starts with two steps of (1) **C**lustering the instances and (2) **LA**beling those instances accordingly to the cluster. In the low resource setting with no labels available, we can label/predict all instances. CLAFI is an extension of CLA which is a full-stack framework that also include (3) **F**eatures selection and (4) **I**nstances selection. Both CLA and CLAFI were first proposed by Nam and Kim [47] in the domain of defect prediction. The intuition of such methods is based on the defect proneness tendency that is often found in defect prediction research, that is *the higher complexity is associated with the proneness of the defects* [14, 26, 44, 47, 54, 65]. Put simply, there is a tendency where the problematic instance's feature values are higher than the non-problematic ones. For instance, Hassan et al. [26] predicted defects using the entropy (or complexity) of code changes (the more complex changes to a file, the higher the chance the file will contain faults). This tendency and CLA's/CLAFI's effectiveness were confirmed via the recent literature and comparative study of 40 unsupervised models in defect prediction across 27 datasets and three types of features. They found CLA's/CLAFI's performance is superior to other unsupervised methods while similar to supervised learning approaches. Moreover, Tu et al. [65] recently applied this intuition in developing their method to further the SOTA work for static analysis and issue close time prediction. Therefore, this study investigates and finds that the hypothesized tendency is also applicable in SATDs data but only effective for the semi-automated method but not the fully automated one. CLA is preferred over CLAFI but this study examines both before choosing one over the other.

Before CLA or CLAFI, **DebtFree**(0) extracts features from each comment candidate as $L2$-normalized terms (the square root of the sum of the squared vector) terms with the TF-IDF[2] scores (after stop word removal).

Clustering:

1. Find the median of feature $F_1, F_2, ..., F_n$ $(median(F_i))$ across the whole dataset.
2. For each data instance X_i, go through each feature value of the respective data instance to count the time when the feature $F_i > median(F_i)$ as K_i.

Labeling: label the instance X_i as SATD if $K_i > median(K)$, else label it as non-TD.

Feature Selection: Calculate the violation score per feature, called metric in the original proposal of Nam et al. [47]. The process is done on both the train and the test dataset.

1. For each F_i, go through all instances of X_j, a violation happens when F_i at X_j is higher than the $median(K_i)$ but $Y_j = 1$ and vice-versa.
2. Sum all the violations per feature across the whole dataset and sort it in ascending order.
3. Select the feature with the lowest violation score, if multiple of them have the same score then pick all of them.

Instance Selection:

1. With the selected features, go through each instance X_i and check if the respective F_j values violated the proneness assumption then remove that instance X_i.
2. If the dataset does not have instances with both classes at the end then pick the next minimum violation score to select metrics.
3. This process is only done on the training dataset.

After selecting features with the minimum violation scores and removing the instances that violated the technical-debt proneness tendency, a practitioner can train any machine learners on the preprocessed training data to identify the SATDs from the unlabeled/test dataset. For this step, we picked Random Forest which is also Jitterbug's choice of learner after being compared across other ones [84].

3.2.2 Active Learning via Falcon

The data that do not share the technical-debt proneness tendency is being continuously learned through human and AI partnership, named Falcon as the proposed active learning strategy from this study. First, a classification model (for SATD or non-SATD comments) is trained on the training dataset. When reading the new comments, Falcon initially uses uncertainty sampling to quickly build the model, then switches to certainty sampling to greedily find technical debt comments. The machine learner (RF) uses this feedback

[2] For token t, its tf-idf score: $Tfidf(t) = \sum_{d \in D} Tfidf(t, d)$, in which for token t in comment or document d, $Tfidf(t, d) = w_d^t \times (\log \frac{|D|}{\sum_{d \in D} sgn(w_d^t)} + 1)$ where w_i^t is the number of times token t appears in document d.

from human to learn their models incrementally. These trained model then sorts the stream of comments such that humans read the most informative ones first (and the comments are sorted again each time a human offers a new label for a comment). After the found SATDs reach a specific threshold then Falcon drops the training data and incrementally updates it with the target data. This is similar to the separation action of the Falcon rocket to drop the thruster for boosting up the rocket's speed after reaching a required height. More specifically, Falcon executes as follows:

Step 1 Feature Extraction: Given a set of comments candidates, Falcon extracts features from each candidate as $L2$-normalized terms with the highest TF-IDF. Initialize the set of labeled data points as $L \leftarrow \emptyset$ and the set of labeled positive data points as $L_B \leftarrow \emptyset$.

Step 2 Bootstrap Learning: Falcon utilizes the labeled training dataset to train a machine learning model, i.e., Random Forest (Yu et al.'s choice of learner [84]).

Step 3 Initial Sampling: Falcon starts by randomly sampling unlabeled candidate studies until humans declare that they see $N_1 = 1$ technical-debt examples.

Step 4 Uncertainty Sampling: Then, as human assessors offer labels, one example at a time, Falcon trains and updates with weighting to control query with uncertainty sampling, until $N_2 = 10$ technical-debt examples are found. Here, different weights are assigned to each class ($W_B = 1/|L_B|$, $W_N = 1/|L_N|$).

Step 5 Certainty Sampling: Next, Falcon trains further using certainty sampling and Wallace's "aggressive undersampling" [70] that culls majority class examples closest to the decision boundary.

Step 6 Training Data Separation: Falcon drops training data after finding more than 10% of estimated SATDs then the model is retrained on the reviewed target data.

Step 7 Early Stopping: Falcon stops training when it is estimated that $N_3 = 90\%$ of the SATDs have been found.

To generate the N_4 estimate, whenever the RF model is retrained, Falcon makes temporary "guesses" about the unlabeled examples (by running those examples through the classifier). To turn these guesses into an estimate of the remaining technical-debt comments, Falcon:

1. Builds a fast and simple model, e.g., Logistic Regression, using the guesses. Faster feedback cycle to update the model and for the users to make decisions
2. Using that regression model, Hard makes new guesses on the remaining unlabeled examples.
3. Loops back to step1 until the new guesses are the same as the guesses in the previous loop.
4. Uses this logistic regression model to estimate the remaining number of positive examples in the data.

The reader will note that there are many specific engineering decisions built into the above design (e.g. the values $\{N_1 = 1, N_2 = 10, N_3 = 90\%\}$). Those decisions were initially made by Yu et al. [82] after exploring 32 different kinds of active learners. Those were later adopted by the SOTA active learning framework Jitterbug [84] for SATD identification.

3.3 DebtFree (100)

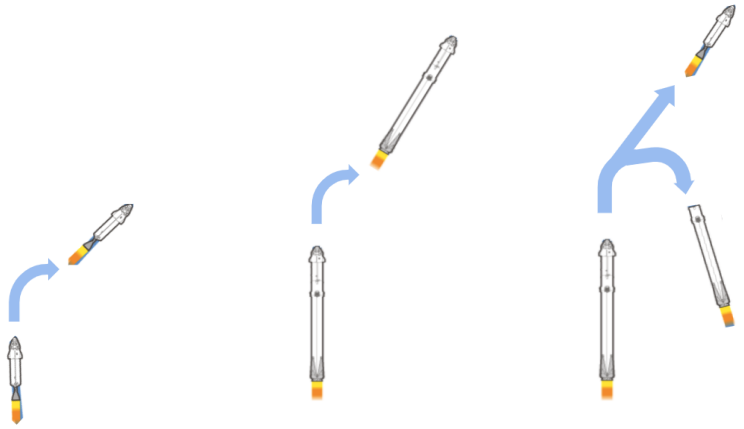
DebtFree(100) is inspired by the SOTA's Jitterbug framework in the high-resource setting, i.e., have access to labeled training data. Instead of a pattern-based approach with a active learning strategy, we propose to replace the pattern-based approach with an unsupervised learner to filter out the highly technical-debt prone comments. Then, the state-of-the-art active learning approach (i.e., Hard [84]) guides the human experts to first start with training on training data labeled by the experts and then identify all of SATD comments on new coming data.

3.3.1 Filtering via CLA

Here, SATDs from the test data can be early filtered out with CLA. However, instead of picking $threshold_K = median(K)$ to label the training and test data, we iterate $threshold_K = percentile(all_K, i)$ for i from 50% to 95% percentile. Then, the best $threshold_K$ is the one with the highest precision to identify the SATDs within the training set. The test set with $K_i > best\_threshold_K$ are labeled as SATDs. Then, instances that meet the tendency in both test and training datasets are removed. The intuition is similar to Yu et al. [84] is that the "easier" SATDs can first be easily identified where easier here means they share the tendency of *the higher complexity is associated with the proneness of the technical-debts.*

Table 4: Differences between the active learning strategies for this study: EMBLEM [67], HARD [84], and our proposed approach, i.e., FALCON.

	Emblem[67]	**Hard[84]**	**Falcon**
start with learning on existing training datasets.	✗	✓	✓
drop the training data and retrain the model on the reviewed test data after attaining a certain threshold of SATDs	✗	✗	✓



(a) Emblem (b) Hard (c) Falcon

Fig. 3: The visual analogy of rocket launching for the comparison of Emblem, Hard, and Falcon processes.

3.3.2 Active Learning via Hard

The active learning strategy here is Hard from Yu et al. [84]'s Jitterbug which is similar to Falcon (as discussed previously in §3.2.2) except it does not filter out training data at any point in the learning process. Another variation for comparison also includes Tu et al.'s Emblem [67]. Emblem only uses one hundred randomly sampled labeled test data to start the active learning strategy. For Emblem, we suspected that due to the randomness in sampling, in the first one hundred instances the performance will be unstable. In summary, the differences and similarities are documented in Table 4 and the performance of all three methods will be compared in RQ2.1. For a more intuitive comparison, Figure 3 shows the rocket launching process in each method. Emblem is cheap with no need for the booster (labeled training data) but difficult to reach a substantial height (acquiring SATDs). Hard is effective but expensive to carry the whole booster all the way till the end. Falcon is a hybrid that uses the booster upon launch to boost the rocket's speed but is discarded once the rocket reaches a certain height.

4 Experimental Design

4.1 Data

To validate this study's hypothesis and the proposed methods' effectiveness, we use the dataset from Maldonado et al. [37] which has been corrected by Yu et al. [84]. The dataset includes ten open-source projects collected from Github: Apache-Ant-1.7.0, Apache-Jmeter-2.10, Hibernate-Distribution3.3.2.GA, Ar-

Table 5: Dataset Details

Project	**Release / Year**	**Domain**	**Comments**	**Original SATDs [37]**	**Corrected SATDs [84]**
Apache Ant	1.7.0 / 2006	Automating Build	4098	131 (3.2%)	135 (3.3%)
JMeter	2.10 / 2013	Testing	8057	374 (4.6%)	416 (5.2%)
ArgoUML	-	UML Diagram	9452	1413 (15%)	1630 (17.3%)
Columba	1.4 / 2007	Email Client	6468	204 (3.2%)	220 (3.4%)
EMF	2.4.1 / 2008	Model Framework	4390	104 (2.4%)	119 (2.7%)
Hibernate	3.3.2 / 2009	Object Mapping Tool	2968	472 (16%)	493 (17%)
JEdit	4.2 / 2004	Java Text Editor	10322	256 (2.5%)	259 (2.5%)
JFreeChart	1.0.19 / 2014	Java Framework	4408	209 (4.7%)	247 (5.6%)
JRuby	1.4.0 / 2009	Ruby for Java	4897	622 (12.7%)	665 (13.4%)
SQuirrel	-	Database	7215	286 (4%)	313 (4.3%)
SUM			**62275**	**4071 (6.5%)**	**4497 (7.2%)**
MEDIAN			**5683**	**271 (4.8%)**	**286 (5%)**

goUML, Columba-1.4-src, EMF-2.4.1, jEdit-4.2, jFreeChart-1.0.19, jRuby-1.4.0, SQL12. The provided dataset contains project names, classification type (if any) with actual comments.

Table 5 lists the varying statistics of the comments found in these 10 projects. As only a small ratio of the source code comments describe SATDs, it would be time-consuming to label all comments manually. Thus, Maldonado and Shihab developed 5 filtering heuristics to eliminate comments that are unlikely to be classified as comments [37]: (1) remove license comments/auto-generated comments, (2) remove commented source code, (3) remove Javadoc commented source comments, (4) group multiple single-line comments together, and (5) removing duplicate comments. In the end, the number of comments that require manual annotation was significantly reduced from 259,229 to 62,275 (reducing the data by 75%).

Furthermore in their work [37], two humans then manually classified each comment according to the six different types of SATD mentioned by Alves et al. [3] if they contained any SATD at all, else marked them $WITHOUT\_CLASSIFICATION$. Note that, we do not use the fine-grained SATD categories proposed by Maldonado et al. [38], rather we focus on a binary problem of instances being a SATD or not. Our study is concerned specifically with identifying if a problem is SATD (e.g., $DEFECT$, $IMPLEMENTATION$, $DESIGN$, etc) or not ($WITHOUT\_CLASSIFICATION$). Simply, as long as the code comment refers to some aspect of technical debt it is treated as SATD. Similarly to previous work [84, 55], we have changed the final label into a binary problem by defining $WITHOUT\_CLASSIFICATION$ as *no* and

the rest of the categories as *yes*. Stratified sampling of the dataset is applied to check personal bias, revealing with a 99% confidence interval of 5%. A third human verified the agreement between the two using stratified sampling and reported a high level of agreement (Cohen's Kapp [12] coefficient of +0.81). The significantly high level of agreement indicates that the dataset is unbiased and reliable.

On the contrary, Yu et al. [84] inspected this dataset for biases, and found that 98% of the false positives were wrongly labeled. The differences between the original and corrected SATDs count are reported in the last two columns of the Table 5. Specifically, Maldonado et al. [37] missed more than 10% of the total SATDs. This discrepancy highlights the importance of validating prior research's conclusions, data, and methodologies should one employ them in their work, a process that might add significant overhead to the amount of required effort. Fortunately, active learning offers feasible remedial venues. Our proposed method DebtFree also established new state of the art that outshined the state-of-the-art active learning method for technical debt, Jitterbug.

4.2 Data Miners

There are several data miner options for the active learning part of the **DebtFree**(100) or supervised learning part of the **DebtFree**(0). For this study, we only test simple and fast learners since the active learning model is updated/retrained frequently while practitioners appreciate quick feedback loop to improve software,code, and technical-debts. Such learners include:

Logistic Regression: a statistical method that uses a logistic function to model a binary dependent variable [29]. A standard logistic function is a common "S" shape with Equation 1:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{1}$$

where $p(x) \in [0, 1]$ for all t. Fitting on the training data, logistic regression looks for the best parameter β to classify input data x into two target classes $\{0, 1\}$. LR is used for estimating SATDs in **DebtFree**(100)'s active learning.

Random Forest: an ensemble learning method that constructs a multitude of decision trees, each time with different subsets of the data rows R and columns C[3]. Each decision tree is recursively built to find the features that yields the most reduction in *entropy* (higher entropy indicates lower ability to draw conclusions in the paritioned data) [7]. Test data is then passed across all N trees. Conclusions are determined by a majority vote across all the trees [6]. Holistically, RF is based on bagging (bootstrap aggregation) which averages the results over multiple trees from sub-samples (reducing variance). Both methods are popular in Machine Learning, and are implemented in the open-source toolkit Scikit-learn [51]. Random Forest is used for classifying SATD comments in **DebtFree**(0), **DebtFree**(100), and Yu et al.'s Jitterbug [84].

[3] Specifically, using $\log_2 C$ of the columns, selected at random.

4.3 Experimental Process

Experiments are conducted on the SATD dataset with 10 projects described in §4.1. Each time, one project is selected as a target project (with labels unknown) and the rest 9 datasets are treated as source projects. This study addresses the following research questions:

RQ1: How well can the state-of-the-art unsupervised learning method identify SATDs?

RQ2: How can the state-of-the-art active learning framework be combined with the state-of-the-art unsupervised learner?

RQ3: How does the proposed DebtFree perform against state-of-the-art models in identifying SATDs?

Recall our study investigates the combination of three approaches including Pseudo-labeling, Filtering, and Active Learning. In this big data era, there are available datasets easily curated from the web (e.g., Github) with no labels and we can harvest valuable insights from them without labeling the data. The first RQ demonstrates that unsupervised learning by itself is not effective but it is useful for understanding some hidden patterns and early filtering out SATDs. In order to extend both the unsupervised learning approach and the active learning approach, the second RQ experiments different ways of integrating both frameworks in two settings with *RQ2.1* where there is no access to the labels of the training data and *RQ2.2* where there is access to the labels. For *RQ2.1*, we test different candidates for all three steps whereas in *RQ2.2* we only focus on candidates for the filtering step and the active learning step. With RQ2.1, we can simulate a real world situation of having data with no labels by hiding the labels of the training data. During the active learning step across the experiments, the oracles are queried for the target project, the ground truth labels are applied to label the queried comments to simulate the human-in-the-loop labeling process.

From RQ2, we finalize the best combination for two settings from RQ2 as **DebtFree**(0) and **DebtFree**(100). In RQ3, we assess **DebtFree**(0)/(100)'s usefulness by comparing them against the SOTA semi-supervised learning work by Yu et al. [84] (i.e., Jitterbug) and the SOTA supervised learning work by Ren et al. [55] (i.e., CNN). We recycle the available implementations of their approaches instead of re-implementing them ourselves to generate the results. Therefore, we are more confident that our conclusions or insights would align with the original work.

4.4 Statistical Testing

We employ Cohen'd effect size test to determine which results are similar by calculating $medium\_step2$. We take guidance from Sawilowsky [57] at al.'s work to determine the value of d to be used. That paper asserts that "small" and "medium" effects can be measured using $d = 0.2$ and $d = 0.5$, respectively. We analyze this data looking for differences larger than $d = (0.5 + 0.2)/2 = 0.35$. This d is higher than the Jitterbug's $d = 0.2$, ensuring the differences in

the results are statistically significant, and consequently higher confidence in the effectiveness of the proposed methods:

$$Medium_{step2} = 0.35 \cdot StdDev(\text{All results}) \tag{2}$$

5 Results

This section will provide details on the experiments and results for answering the research questions listed above.

RQ1: How well can the state-of-the-art Unsupervised Learning method identify SATDs?

In this experiment, we compare the performance of the following five treatments:

- **CLA:** The unsupervised learner described in §3.2.1, which clusters the test data instances based on the features' median by marking the ones with features higher than the median as SATDs and vice-versa.
- **CLAFI+RF:** A combination of CLAFI (§3.2.1) and the data miner RF (§4.2). It started with pseudo-labeling the unlabeled training data with CLA. Then the data is preprocessed by removing the features (for nine train and one target datasets) and instances (only on the nine training datasets) that violated the assumption from CLA. Then we pick Random Forest as the learner choice from SOTA Yu et al.'s Jitterbug work's data miner of choice for supervised learning.
- **CLA+RF:** This is similar to the previous step except no features and instances preprocessing step.
- **Easy:** The pattern-based approach, first step of Jitterbug. For each pattern p, find $score = Prec(p)^4 \cdot P(p) = TP(p)^4/P(p)^3$ where $P(p)$ is the number of comments p (positives) and $TP(p)$ is the number of SATD comments containing p (true positives). We iteratively select the pattern with the highest score, then removes comments containing that pattern from both train and test data until the selected pattern has lower than 80% precision on the training data. Then, stop and evaluate.
- **Easy+CLA:** First, the pattern-based approach Easy filters out test data comments with patterns that have more than 80% precision on training data and then passed to CLA for identified the rest of SATDs.

These pattern-based approaches can be seen as two groups: unsupervised learners (i.e., CLA variants) and supervised learners (i.e., Easy and Easy+CLA). In term of the effectiveness of these methods, it is recommended to evaluate multiple metrics so we employ Recall and APFD.

Recall measures the ability to identify the SATDs, $Recall = TP/(TP + FN)$. TP is the number of true positives (SATD comments predicted as SATDs), TN is the number of true negatives (non-SATD comments predicted as non-SATDs), FP is the number of false positives (non-SATD comments predicted as SATDs), and FN is the number of false negatives (SATD comments predicted as non-SATDs).

Table 6: Comparison between **CLA**, **CLAFI+RF**, **CLA+RF**, **Easy**, and **Easy+CLA** are made in terms of Recall and APFD (the higher the better). Medians and IQRs (delta between 75th percentile and 25th percentile, lower the better) are calculated for easy comparisons. **CLA**, **CLA+RF**, and **CLAFI+RF** do not have access to training data's labels while **Easy** and **Easy+CLA** do. Here, the darker cells show top rank while the lighter cells show rank two where the difference between two ranks is statistically significant by being higher than M reported in the left most column.

	Treatment	**SQuirrel**	**JMeter**	**EMF**	**Apache Ant**	**ArgoUML**	**Hibernate**	**JEdit**	**JFreeChart**	**Columba**	**JRuby**	**Median**	**IQR**
Recall ($M = 8\%$)	CLA	65	72	55	67	67	62	79	55	64	65	65	5
	CLAFI+RF	53	31	30	46	47	42	39	25	23	23	35	17
	CLA+RF	81	75	77	91	85	84	76	78	61	82	80	8
	Easy	58	77	41	27	90	75	22	55	88	90	67	47
	Easy+CLA	93	96	82	91	95	95	98	71	98	98	95	7
APFD ($M = 6\%$)	CLA	73	73	61	78	72	65	91	61	70	72	72	8
	CLAFI+RF	74	63	58	64	70	57	76	57	67	68	66	10
	CLA+RF	74	74	64	73	53	62	80	64	64	68	66	9
	Easy	57	76	41	27	83	71	22	54	87	85	64	42
	Easy+CLA	92	94	81	82	96	93	93	80	98	98	93	14

In order to take effort into consideration for the performance of the model, we also employ APFD. APFD calculates the area under the curve of the recall-cost curve whereas other metrics (e.g. precision, recall, F1, G1) only evaluate a single point of the curve. APFD ranges from 0.0 to 1.0, with higher values indicating that higher recall could be achieved at a lower cost, and thus, more advantageous. An APFD value of 0.5 can be achieved by randomly select the next item each time.

- $Recall = \frac{|\{\text{SATDs}\} \cap \{\text{human verified comments}\}|}{|\{\text{SATDs}\}|}$
- $Cost = \frac{|\{\text{human verified comments}\}|}{|\{\text{comments}\}|}$

Cautionary note: Menzies et al. [43] warned that precision can be misleading for imbalanced data sets like those studied here (e.g. Table 5 reports that the median of target class is 5%). Hence, we will not report precision results and will not place much weight on F1. Table 6 shows the results of the experiment. The Cohen'd effect size test is applied to determine which results are similar by calculating $medium\_step2$ or M across Recall and APFD.

On each target project in Table 6, the darker cells show top rank while the lighter cells show rank two. Different colors indicate the statistically significant differences at least equal or larger than the best result(s) subtracts the $medium\_step2$ (M). Observations from Table 6 include:

- Surprisingly, there was no improvements from the data preprocessing (metrics and instances selection) and supervised learning (RF) step after CLA.
- CLA performs similarly to Easy in the recall, betters in APFD. In another word, CLA performs better than Easy without access to the train labels.

- CLA+RF performs better than Easy in recall and APFD. CLA+RF performs better than Easy without access to the training data's labels.
- Notably, CLA+RF performs similarly to CLA, wins in Recall but loses in APFD so CLA's performance is more balanced.
- Easy+CLA performs the best. This indicates additional effectiveness of CLA as in bettering the performance of Easy. It is notable that Easy relies on training data's labels (i.e., supervised learning) in order to identify SATDs when variants of CLA do not.

The results confirm the effectiveness of unsupervised learning (1) by itself or with supervised learning in the case of no labels available, and (2) as a post-processor after the SOTA pattern-based approach. There is a technical-debt proneness tendency within the data to identify SATDs. Moreover, this positive effect also comes with the benefit of not needing access to the labels.

The negative results from preprocessing the data (metrics and instances selection) then supervised learning show that the selected metrics and instances are not fully representative or relevant to identify SATDs on new data. Simply, the technical-debt proneness tendencies that exist inside the training datasets cannot be transferred completely to the target test dataset. Therefore, it might not be effective to learn from the training data at all.

At the same time, in case there are resources available for labeling the training data, Easy is still a useful pattern-based approach to automatically identify "easy to find" SATDs. Then, CLA can be employed to identify the rest of SATDs. The combination of Easy+CLA should be the state-of-the-art automatic method for identifying SATDs. It is simple without any deployment of a machine learning model for the traditional supervised learning approach.

The findings of this RQ is that:

Result:
From our exploration of various unsupervised learners, the original CLA by Nam et al. performs the best. Moreover, CLA performs similarly to the SOTA pattern-based approach, Easy [84], without having access to the data's labels (100% less effort).

RQ2: How can the state-of-the-art active learning framework be combined with the state-of-the-art unsupervised learner?

In the situation where business users want to reach a specific amount of SATDs, our previously discussed automatic methods would not be able to guarantee such recall. For this demand, an active learning approach is more suitable [84]. However, the SOTA active learner [84] requires the training data to be labeled which is very expensive why the RQ1's unsupervised learning method is efficient but not effective enough. Therefore, this RQ explores how to integrate both methods in order to minimize the cost of labeling while improving the effectiveness in identifying SATDs in both settings of training data labels unknown versus known.

RQ2.1: How to find SATDs with no access to labeled training data?

In the low-resource setting (unlabeled training data), unsupervised learning can help guess or pseudo-label the training data with no cost or the experts can label the first 100 instances or 1% of the test data before applying the active learning strategy. The following methods are tested in this experiment:

- **Emblem**: without utilizing the training data, we start with random 100 test data instances to incrementally update the model to guide the developers to find SATDs.
- **Pseudo-labeling** (via CLA) + **Filtering** (via CLA) + **Hard**: First, apply unsupervised learning to pseudo-label the training data that highly fits the hypothesized technical-debts proneness tendency with zero human effort. Second, automatically identify the SATDs in test data via the same unsupervised learner. Then, a active learning strategy (as explained in §3.2.2) with the data miner RF (referenced in §4.2) to incrementally acquire information and update the model in identifying the SATDs that do not fit such a tendency.
- **Pseudo-labeling** (via CLA) + **Hard**: This is synonymous with the previous one except skipping the filtering via CLA step and simply apply Hard to incrementally learn and identify SATDs on both frugally pseudo-labeled data and new data.
- **Pseudo-labeling** (via CLA) + **Filtering** (via CLA) + **Falcon**: First, apply unsupervised learning to pseudo-label the training data based on the hypothesized technical-debts proneness tendency. Then, the same method is employed to filter out the highly technical-debt prone SATDs in the test data. RF model is applied to continuously train on the pseudo-labeled data and the rest of the test/target data until the found SATDs are 10% of the estimated SATDs. Finally, the model will discard the cheaply labeled training data while retraining the model on the reviewed test data so far and continuously until the found SATDs are more than the user-specified threshold (i.e. 90%).
- **Pseudo-labeling** (via CLA) + **Falcon**: This is synonymous to the previous one except no early filtering via CLA.

Beside APFD, we will also evaluate the methods in F1, G1, and the labeling cost. It is recommended to evaluate with metrics that aggregate multiple metrics like F1 and G1. F1 is a harmonic mean of recall and precision, $Precision = TP/(TP+FP)$. G1 is a harmonic mean of recall and false-alarm rate, $FAR = FP/(TP+TN)$:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{3}$$

$$G1 = \frac{2 \cdot Recall \cdot (1 - FAR)}{Recall + (1 - FAR)} \tag{4}$$

Table 7 reports the comparison between all the previously discussed methods across four metrics: F1, G1, APFD, and the labeling Cost. F1, G1, and the labeling cost are measured for all methods aiming to find 90% of the rest SATDs. Except for the labeling cost, the higher the values of the other metrics the better. The labeling cost is for labeling the test data labels. Pseudo-labeling and Filtering are abbreviated to P and F. Similarly to **RQ1**, we also employ

Table 7: Comparison between **Emblem** [67], **P+Hard**, **P+F+Hard**, **P+Falcon**, and **P+F+Falcon** where **P** is Pseudo-Labeling and **F** is Filtering. **P** and **F** are done via CLA. The comparison is made in terms of F1 score, G-score, APFD, and cost for identifying SATDs in the low-resource labeled data setting. For F1 score, G-score, and reviewing cost, both methods target at finding 90% of the SATDs with its estimator. Medians and IQRs (delta between 75th and 25th percentile, lower the better) are calculated for easy comparisons. Here, the light red cells show best performing methods where the difference between them are higher than M reported in the left most column.

	Treatment	**SQuirrel**	**JMeter**	**EMF**	**Apache Ant**	**ArgoUML**	**Hibernate**	**JEdit**	**JFreeChart**	**Columba**	**JRuby**	**Median**	**IQR**
F1 ($M = 8\%$)	Emblem[67]	61	49	9	8	93	75	7	30	66	81	55	66
	P+Hard	28	40	12	17	70	47	32	21	36	52	34	26
	P+F+Hard	47	59	19	26	88	79	38	39	53	86	50	41
	P+Falcon	50	68	22	29	88	73	41	46	63	83	57	32
	P+F+Falcon	30	49	17	22	76	64	35	39	41	73	40	29
G1 ($M = 6\%$)	Emblem[67]	83	88	20	14	95	82	75	40	82	87	82	47
	P+Hard	82	87	75	77	87	77	87	52	88	79	81	10
	P+F+Hard	83	83	87	83	91	86	74	78	91	95	85	8
	P+Falcon	89	90	77	80	94	88	83	74	94	94	89	14
	P+F+Falcon	79	86	75	76	88	85	72	78	84	92	82	10
APFD ($M = 3\%$)	Emblem[67]	92	92	77	82	90	84	88	87	95	90	89	6
	P+Hard	77	81	66	78	81	67	90	63	81	77	78	14
	P+F+Hard	86	91	79	86	89	77	95	75	90	85	86	11
	P+Falcon	90	93	81	84	88	83	93	69	93	89	89	10
	P+F+Falcon	91	92	83	86	92	87	93	69	92	91	91	6
Cost ($M = 4\%$)	**Emblem** [67]	6	13	4	3	17	15	65	4	4	13	10	11
	P+Hard	20	17	32	25	25	38	10	15	13	24	22	10
	P+F+Hard	5	3	19	12	11	12	3	10	3	11	11	9
	P+Falcon	10	7	14	14	19	31	6	11	7	17	13	10
	P+F+Falcon	10	9	13	11	13	20	4	12	9	16	12	4

Cohen'd medium effect size test [57] to determine the best treatment(s) in each target project. From the results, we can see:

- **F1**: P+Falcon performs the best (9/10) while P+F+Hard performs the second-best (7/10).
- **G1**: P+Falcon outperforms the rest (9/10) while P+F+Hard's performance is placed at second.
- **APFD**: P+F+Falcon's performance is the best (9/10) while P+Falcon and Emblem perform similarly as the second best (7/10).
- **Cost**: Emblem's and P+F+Hard's processes are the most efficient (7/10).

P+F+Hard always performs better than P+Hard across all four metrics. However, the same effect is not observed in P+Falcon and P+F+Falcon due to Falcon losing some insights of frugally pseudo-labeled data after dropping the training data at a certain threshold and the filtering step further reduce the insights (i.e, highly technical-debt prone comments) for Falcon to learn. However, P+Falcon outperforms all methods, except slightly lost to P+F+Falcon in G1, and makes the system experts read 3% more on average. Falcon's ef-

fectiveness overwrites the necessity of filtering. Moreover, P+Falcon's performance is better than P+Hard indicating the effectiveness of Falcon over Hard.

Overall, in the situation of having no access to the labeled training data, the effectiveness of Pseudo-labeling (via CLA) + Falcon demonstrates that:

- Unsupervised learning can cheaply and quickly guess the labels of the training data to provide insights for bootstrapping the active learning strategy. However, those frugally pseudo-labeled data do not have enough insights to help Hard guide the experts efficiently to identify the SATDs the whole way up to 90%.
- Emblem alone is cost-efficient but it does not effectively identify the SATDs.
- A hybrid of both is more preferable as Falcon would drop training data with guessed labels after the found SATDs are at least 10% of the estimated SATD% so there are enough insights to help Falcon guide the experts to efficiently identify the rest of SATDs.
- The best configuration for a "label-free" process including a labeler to utilize the unlabeled data for bootstrapping and the tuned active learning strategy is labeling via CLA and Falcon. Hence, **DebtFree**(0) = **Pseudo-labeling** (via CLA) + **Falcon** as shown in Figure 4.

RQ2.2: How to find SATDs with having access to labeled training data?

For this research question, we are interested in the best configurations of **DebtFree**(100) to take advantage of an abundant amount of labeled data from previous research work [37, 84]. The comparison is made through the F1 score, G1 score, APFD, and cost metrics. The methods are similar to the previous RQ but without the labeling step:

- **Filtering** (via CLA) + **Hard**: First, apply unsupervised learning to learn the hypothesized technical-debts proneness tendency on the humanly labeled training data then automatically identify the SATDs in test data with zero human effort. Second, an active learning strategy adopted from

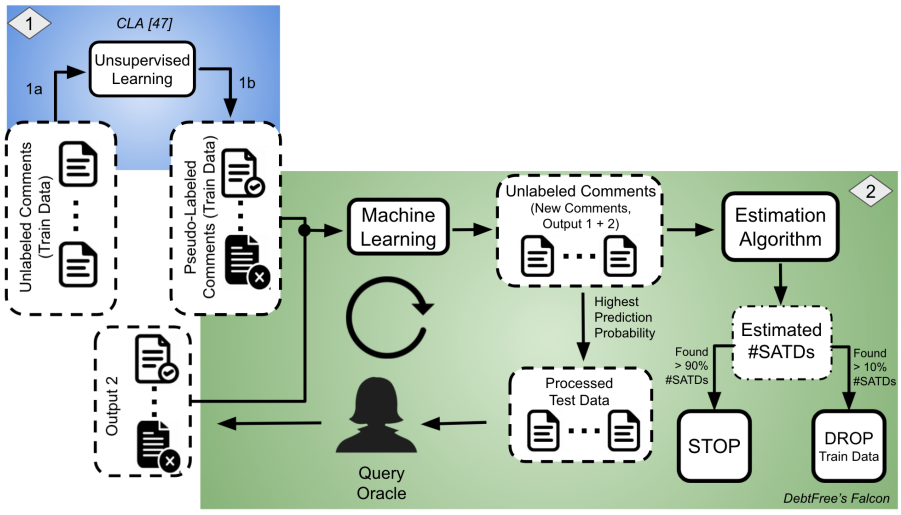


Fig. 4: Workflows of **DebtFree**(0) = Pseudo-Labeling (via Unsupervised Learning, i.e., CLA [47]) + Active Learning (via Falcon).

Table 8: Comparison between **Hard**, **F+Hard**, **Falcon**, and **F+Falcon** in terms of F1 score, G-score, APFD, and cost in identifying SATDs in the high-resource labeled data setting. Filtering-**F** are done via CLA. For F1 score, G-score, and reviewing cost, both methods target at finding 90% of the "hard to find" SATDs with its estimator. Medians and IQRs (delta between 75th and 25th percentile, lower the better) are calculated for easy comparisons. Here, the light red cells show best performing methods where the difference between them are higher than M reported in the left most column.

	Treatment	SQuirrel	JMeter	EMF	Apache Ant	ArgoUML	Hibernate	JEdit	JFreeChart	Columba	JRuby	Median	IQR
F1 ($M = 8\%$)	**Hard**	48	59	28	25	94	82	29	53	84	92	56	55
	F+Hard	34	52	23	24	82	66	44	51	73	80	52	29
	Falcon	55	70	28	44	93	73	50	48	54	82	55	25
	F+Falcon	34	45	18	24	84	66	30	34	35	70	35	36
G1 ($M = 2\%$)	**Hard**	93	92	88	87	98	89	93	82	97	96	93	8
	F+Hard	90	91	86	85	95	86	92	80	98	94	91	8
	Falcon	86	90	83	78	96	90	88	80	96	94	89	11
	F+Falcon	88	89	81	82	94	88	91	78	93	92	89	10
APFD ($M = 2\%$)	Hard	95	95	95	91	91	89	95	90	98	92	94	4
	F+Hard	98	96	97	96	97	94	96	89	98	96	96	1
	Falcon	94	95	91	90	91	89	94	83	97	92	92	4
	F+Falcon	97	97	94	92	96	92	98	76	98	96	96	5
Cost ($M = 2\%$)	Hard	13	11	14	20	18	17	14	10	4	14	14	6
	F+Hard	8	7	5	12	11	10	9	7	5	10	9	3
	Falcon	9	7	11	9	12	20	3	9	8	14	9	4
	F+Falcon	8	7	12	7	17	24	6	11	9	17	10	10

the SOTA work [84] (§3.3.2) with the data miner RF (§4.2) to incrementally acquire information and update the model in identifying the SATDs that do not fit such tendency.

- **Hard** (§3.3.2): This is synonymous with the previous one except no filtering via CLA in the beginning.
- **Filtering** (via CLA) + **Falcon**: The filtering step is the same as above. Then, an RF model is applied to continuously learn on the unfiltered training data and the rest of the unfiltered test/target data until the found SATDs are 10% of the estimated SATDs. Finally, the model will discard the labeled training data while retraining the model on the reviewed test data so far and continuously until the found SATDs are more than the user-specified threshold (i.e. 90%).
- **Falcon** (§3.2.2): This method is synonymous with the previous one except for no filtering step in the beginning.

The results for this RQ are reported in Table 8. Similarly, F1, G1, and the labeling cost are measured for all methods aiming to find 90% of the rest SATDs. Except for the labeling cost, the higher the values of the other metrics the better. The labeling cost is for labeling the test data labels. Filtering is abbreviated to F. Similarly to **RQ1**, we also employ Cohen'd medium effect size test [13, 57] to determine the best treatment to determine the best treatments in each target project. From the results, it is observed that:

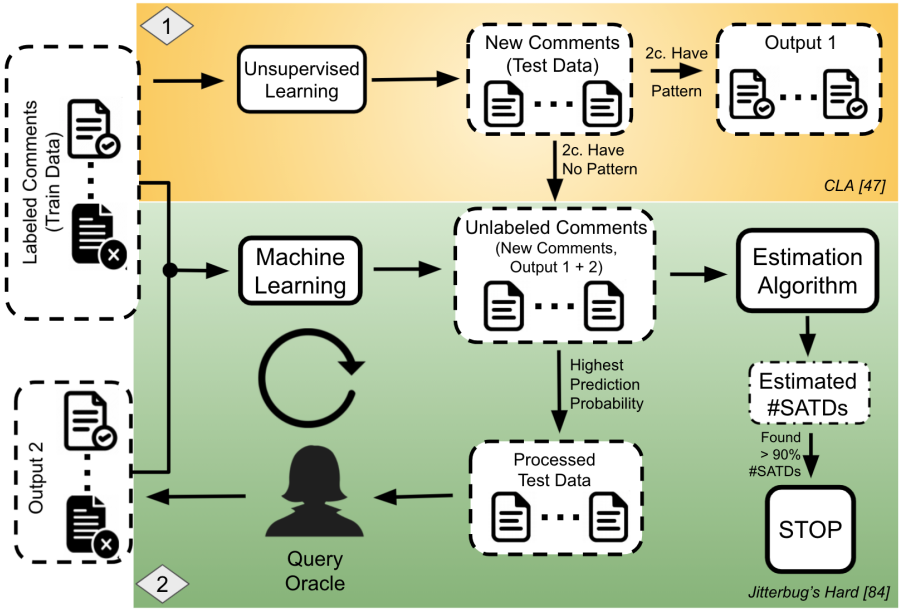


Fig. 5: Workflows of **DebtFree**(100) = Filtering (via Unsupervised Learning, i.e., CLA [47]) + Active Learning (via Hard [84]).

- **F1**: Hard's and Falcon's performance are similarly as the best ones (7/10).
- **G1**: Hard outshines the rest over all the projects while F+Hard performs as the second best (7/10).
- **APFD**: F+Hard outperforms the rest on ten out of ten projects and F+Falcon comes in as second place (7/10).
- **Cost**: F+Hard is the most efficient in eight out of ten projects and then Falcon (6/10).

Falcon still outperforms F+Falcon across F1, G1, and cost while losing on APFD. However, Hard outshines Falcon here, wins in cost, draws in F1, and loses in G1 and APFD. This indicates that when there is an abundant amount of labeled data, discarding them after reaching a certain threshold is not as effective as just active learning on the rest since there is enough insights from the labels. Hard performs similarly to F+Hard as the best ones across all methods by winning on F1 and G1 while losing on APFD and cost. However, the benefit of filtering here is it helps stabilize Hard's performance as Hard by itself has a high variance across four metrics. As mentioned previously RQ1, F1 and G1 scores are only single points of the curve while APFD calculates the area under the recall-cost curve. In the theme of effort-aware, the best configurations of **DebtFree**(100) include **Filtering** (via CLA) with **Hard** as shown in Figure 5.

From this RQ, we conclude that:

Result:
With the effort-aware theme, we investigate different combinations of active learners and CLA across two settings of the training data, either with having 1-*no access* and 2-*access* to the labels to propose **DebtFree**. In setting 1, or **DebtFree**(0), the best combination is Pseudo-Labeling (via CLA) with our proposed active learner, Falcon. In setting 2, or **DebtFree**(100), the best combination is Filtering (via CLA) with the SOTA active learner for SATDs identification, Hard.

RQ3: How does the proposed DebtFree perform against state-of-the-art models in identifying SATDs?

For this research question, we are interested in the overall performance of **DebtFree** by comparing (1) "label-free" **DebtFree**(0), (2) the latest state-of-the-art deep convolutional neural network-based approach [55], (3) two-step SOTA Jitterbug approach [84], and (4) **DebtFree**(100). The comparison is made via F1 score, G1 score, APFD, and cost metrics:

- **DebtFree(0)**: First, apply unsupervised learning to label the unlabeled training data based on the hypothesized technical-debts proneness tendency. Finally, Falcon is applied to continuously learn and identify the rest of SATDs.
- **Jitterbug [84]**: First apply pattern-based approach (i.e, Easy, described in §5's RQ1) to automatically identify the "easy to find" SATDs, then apply a active learning strategy (i.e., Hard, described in §3.3.2) with RF as the learner of choice to guide humans in identifying the "hard to find" SATDs.
- **CNN**: deep learning solution that is based on a convolutional neural network structure with word2vec features and hyperparameter tuning to classify each comment into SATD or non-SATD [55].
- **DebtFree**(100): First, apply CLA as an unsupervised learner to filter out the test data that are most likely to be technical debts. Then, the SOTA active learning strategy of Hard is applied to incrementally identify the rest SATDs.

The results for this RQ are reported in Table 9. Similarly, F1, G1, and the labeling cost are measured for all methods aiming to find 90% of the rest SATDs. Except for the labeling cost, the higher the values of the other metrics the better. The labeling cost is for labeling the test data labels. Similar to **RQ1**, we also employ Cohen'd medium effect size test [13] to determine the best treatment to determine the best treatments in each target project. From the results, it is observed that:

- **F1**: CNN outperforms the rest of the methods in eight out of ten projects and then **DebtFree**(0)'s performance is placed at second.
- **G1**: **DebtFree**(100) outshines the rest in nine projects and then Jitterbug comes in second place.
- **APFD**: **DebtFree**(100) performs similarly as Jitterbug as the best ones in nine out of ten projects.

Table 9: Comparison between **DebtFree**(0), **CNN**, **Jitterbug**, and **DebtFree**(100) in terms of F1 score, G-score, APFD, and cost. For F1 score, G-score, and reviewing cost, both methods target at finding 90% of the "hard to find" SATDs with its estimator. Medians and IQRs (delta between 75th and 25th percentile, lower the better) are calculated for easy comparisons. Here, the darker cells show top rank while the lighter cells show rank two where the difference between two ranks is statistically significant by being higher than M reported in the left most column.

	Treatment	**SQuirrel**	**JMeter**	**EMF**	**Apache Ant**	**ArgoUML**	**Hibernate**	**JEdit**	**JFreeChart**	**Columba**	**JRuby**	**Median**	**IQR**
F1 ($M = 8\%$)	DebtFree(0)	50	68	22	29	88	73	41	46	63	83	57	32
	CNN [55]	60	62	51	45	82	78	52	51	76	87	61	27
	Jitterbug [84]	24	55	61	17	21	34	23	75	47	23	29	32
	DebtFree(100)	34	52	23	24	82	66	44	51	73	80	52	29
G1 ($M = 2\%$)	DebtFree(0)	89	90	77	80	94	88	83	74	94	94	89	14
	CNN [55]	87	91	83	84	92	90	79	76	98	95	89	33
	Jitterbug [84]	91	85	88	85	93	88	91	93	90	88	89	3
	DebtFree(100)	90	91	86	85	95	86	92	80	98	94	90	8
APFD ($M = 4\%$)	DebtFree(0)	90	93	81	84	88	83	93	69	93	89	89	10
	CNN [55]	69	83	58	66	87	79	64	81	89	88	80	21
	Jitterbug [84]	95	99	95	93	92	94	100	99	95	94	95	5
	DebtFree(100)	98	96	97	96	97	94	96	89	98	96	96	1
Cost ($M = 4\%$)	DebtFree(0)	10	7	14	14	19	31	6	11	7	17	13	10
	CNN [55]	0	0	0	0	0	0	0	0	0	0	0	0
	Jitterbug [84]	18	35	28	24	12	25	17	19	23	27	24	8
	DebtFree(100)	8	7	5	12	11	10	9	7	5	10	9	3

– **Cost**: CNN is the most efficient since it is modeled as the traditional supervised approach (i.e., not touching the test/target data).

DebtFree(100) performs similarly to the SOTA supervised learning approach of CNN [55], wins in APFD, draws in G1, and loses in F1 and cost. However, **DebtFree**(100)'s performance exceeds the SOTA two-step Jitterbug approach [84] across three metrics (F1, G1, and cost) while performing similarly in APFD. In particular, **DebtFree**(100) saves the labeling cost 250% from Jitterbug on average (except in *ArgoUML* projects).

DebtFree(0) outperforms the SOTA two-step Jitterbug approach [84] while performing similarly to the SOTA supervised learning approach of CNN [55]. **DebtFree**(0) wins CNN in APFD, draws in G1, and loses in F1 and cost. **DebtFree**(0) wins Jitterbug in F1 and cost, draws in G1, and loses in APFD. Specifically, **DebtFree**(0) saves the labeling cost almost double than Jitterbug on average (except in *ArgoUML* and *Hibernate* project). However, **DebtFree**(0) loses to **DebtFree**(100) across four metrics. In general, both of these SOTA work's and **DebtFree**(100)'s effectiveness rely on the labeled training data, without them, they might not exist. On the other hand, **DebtFree**(0) does not require the training data to be labeled, and requiring no labels from the training data is already a win in itself. Specifically, out of the total ten datasets, there are nine labeled training datasets (90%) so CNN needs

90% labeled data while Jitterbug needs 92% on average while ***DebtFree****(0) only need 1% of the data to be labeled (99% cheaper).*

Notably, Falcon having access to labeled training data actually surpasses CNN on three metrics (F1, G1, and APFD) while losing in cost. Moreover, Falcon performs better than Jitterbug in F1 and cost, draws in G1, and loses in APFD. Specifically, Falcon saves the labeling cost 240% from Jitterbug on average (except in *ArgoUML* project). Hence, **Falcon** is more balanced than **Filtering** (via CLA) and **Hard** in term of effectiveness over both methods.

On a side note, active learning without the training data, Emblem, performs similarly to Jitterbug (with access to labeled training data and a pattern-based approach) as Emblem wins in F1 and Cost but loses in G1 and APFD. Moreover, Emblem loses to Jitterbug by a small margin for G1 (median at 82 versus 89) versus APFD (median at 89 versus 95) but wins over by a larger margin for F1 (median at 55 versus 29) and Cost (median at 13 versus 24). This restates how labeled training data and insights from them are not completely transferred to the learning of the continuous strategy which contradicts the previous SOTA's conclusion.

Altogether, **DebtFree**'s effectiveness can be concluded that:

Result:
When comparing against the SOTA semi-supervised learning work by Yu et al. [84] and the SOTA supervised learning work (with deep learning) by Ren et al. [55], our proposed method **DebtFree** outperforms them significantly. First, **DebtFree**(100) performs similarly to Ren et al. [55]'s work and better than Yu et al. [84]'s work while reducing the labeling cost by 2.5 times. Second, **DebtFree**(0) performs similarly to Ren et al. [55]'s work without having access to the training data's labels and outperforms Yu et al. [84]'s work while being 99% less effort.

6 Threats of Validity

There are several validity threats [18] to the design of this study. Any conclusion made from this work must be considered with the following issues in mind:

Conclusion validity focuses on the significance of the treatment. To enhance conclusion validity, we ran experiments on 10 different target projects and found that our proposed method always performed better than the state-of-the-art approaches. More importantly, we applied a wider step ($d = 0.35$) for the statistical testing of Cohen'd (described in **RQ1** of §5) than the state-of-the-art work [84] ($d = 0.2$) so the observed effects are validated with more confidence. In addition, we have considered generalization issues of single evaluation metrics (e.g., recall and precision) and instead evaluate our methods on metrics that aggregate multiple metrics like F1 and G1 while being more effort-aware (APFD and cost). As future work, we plan to test the proposed

Table 10: The left subtable showed TP/P rate after Easy and CLA while the right subtable are the median ratios of estimated SATDs over actual SATDs per iteration of the active learning strategies of Emblem, **DebtFree**(0)'s-D(0)'s Falcon, **DebtFree**(100)'s-D(100)'s Hard, and Jitterbug's-J's Hard [84]. The median and IQR are also reported for ease of comparison.

(a) Percentage of SATDs being identified by the Easy and CLA in each project.

Datasets	Easy	CLA
SQuirrel	58	47
JMeter	77	22
EMF	41	25
Apache Ant	27	42
ArgoUML	90	45
Hibernate	76	31
JEdit	22	35
JFreeChart	55	13
Columba	88	18
JRuby	90	27
Median	67	29
IQR	47	13

(b) Median ratio of estimated SATDs over actual SATDs across all iterations per project by each active learning method. The higher the number than 100%, the more the approach overestimate and vice-versa.

Datasets	Emblem	D(0)'s Falcon	D(100)'s Hard	J's Hard
SQuirrel	81	88	69	153
JMeter	94	104	83	429
EMF	13	101	75	76
Apache Ant	7	60	100	23
ArgoUML	125	93	58	1195
Hibernate	74	97	85	362
JEdit	122	71	85	148
JFreeChart	34	60	81	128
Columba	60	98	86	592
JRuby	72	103	84	826
Median	73	95	84	258
IQR	60	30	4	464

methods with additional analyses that are endorsed within SE literature (e.g., P-opt20 [67]) or general ML literature (e.g., MCC [11]).

Finally, in order to understand why our proposed method outshines the others, we offer a deeper analysis via Table 10. The SOTA active learning Jitterbug starts with the filtering step (Easy) before active learning (Hard) on the rest. Recall §2.2.5, one of our hypothesized Jitterbug's shortcomings is Jitterbug's Easy can find up to 90% of the SATDs can make it difficult for Hard to follow up and find the rest of 10% SATDs. Hence, Table 10.a reported the percentage of SATDs being found via the filtering step (Easy [84] or CLA [47]) and Table 9.b reported the median ratio of estimated SATDs 0over actual SATDs across all iterations per project for Jitterbug's Hard [84] (after Easy), **DebtFree**(0)'s Falcon, **DebtFree**(100)'s Hard (after CLA), and Emblem [67]. The hypothesis here is that the higher the SATDs being discovered by the early filtering step, the harder it is to find the rest of SATDs, and the higher the overestimation is done by the consequent active learning strategy. This forces the human experts to review more comments than necessary, which results in more cost and effort. For instance, in *JRuby*, Easy finds 90% of the SATDs so Jitterbug's Hard has a difficult time finding the rest 10% and ends up overestimating the rest SATDs on average by 8.3 times. Jitterbug's Hard always overestimates (up to 12 times than the actual SATDs) than the rest of the methods except in the case of *Apache Ant* and *EMF*. Meanwhile, CLA only finds up to 29% of the SATDs on average so the estimation of **DebtFree**(100)'s Hard is only 16% away of the actual SATDs and more balanced (i.e, lowest in IQR). With no prior filtering step, Emblem tends to underestimate by 27%

but **DebtFree**(0)'s Falcon has the closest estimation to the actual SATDs , i.e., 95%.

Internal validity focuses on how sure we can be that the treatment caused the outcome. To enhance internal validity, we heavily constrained our experiments to the same dataset, with the same settings, except for the treatments being compared.

Construct validity focuses on the relation between the theory behind the experiment and the observation. To enhance construct validity, we compared solutions with and without our strategies in Table 7 and 8 while showing that both components (unsupervised learning with **CLA** and active learning of **Falcon/Hard**) and in both settings (low-resource labels versus labels abundant) to improve the overall performance. However, we only showed that with our setting of featurization and default parameters of random forest learners. The performance can get even better by tuning the parameters, variety of scalers, and different data preprocessors (e.g., synthetic minority over-sampling or SMOTE that is known to help with unbalanced datasets [1, 9]). We plan to explore these in our future work.

External validity concerns how widely our conclusions can be applied. In order to test the generalizability of our approach, we always kept a project as the holdout test set and never used any information from it in training.

7 Conclusion and Future Work

Managing Self-Admitted Technical Debts are important to maintaining a healthy software project. The current automated solutions do not have satisfactory precision and recall in identifying SATDs to fully automate the process. Moreover, the learning requires the training data to be labeled, which is not always available because of high cost and labor as the case discussed in §2.1. We showed that there is a "technical-debt proneness tendency" in the data where SATDs are associated with higher complexity of the data. In order to reduce the label famine and human effort, a half-automated two-mode framework was proposed, called **DebtFree**. If there is a lack of labeled data, **DebtFree**(0) first pseudo-labels the training data's labels using an unsupervised learning method that is based on "technical-debt proneness tendency". When there are abundant labeled training data, **DebtFree**(100) applies the same unsupervised learner to find the best tendency on the training data to filter out the *highly prone* SATDs from the test data. Then, an active learning model iteratively trains and update on both historically labeled data and new human-labeled ones while guiding the human experts to target the most likely SATDs according to the model's ranking. Our proposed active learning method (i.e, Falcon) is the best one for **DebtFree**(0) while Yu et al.'s Hard [84] is the best one for **DebtFree**(100). The process can be repeated till the estimated recall from the model reaches the predefined target recall. Our findings include:

1. When combining the previous SOTA's pattern-based approach (i.e., Easy) and a simple unsupervised learning (i.e., CLA), the performance was higher

than Easy alone without any human effort. Therefore, Easy+CLA should be considered as the new simple baseline method for identifying SATDs.
2. In case of low-resource data, a combination method of CLA to pseudo-label the training data and a novel active learning strategy (i.e., Falcon) surpassed both the SOTA semi-automated method Jitterbug [84] and the SOTA deep learning automated method CNN [55]. This serves as a proof-of-concept of how unsupervised learning can cheaply label the training data to bootstrap the active learning of a machine learning model to identify SATDs on the test data.
3. In case of having access to the labeled data, a combination method of CLA to filter out the highly prone SATDs and Hard performed similarly to Ren et al.'s CNN [55] while outperforming Yu et al. [84] (5/8 cheaper).
4. Falcon is our novel active learning method for its effectiveness in identifying SATDs by frugally pseudo-labeling data and also when having access to the training data labels (as Falcon without filtering performed better than both SOTA methods, in RQ3).
5. The success of those methods for technical debt suggests that there could be many more domains in software analytics that could benefit from unsupervised learning. As mentioned above, those benefits include the ability to commission new models, faster, with much less time consuming and less error-prone labeling of examples.
6. Overall, our proposed super learning method with **DebtFree** is the most effective and efficient in identifying SATDs.

That said, **DebtFree** still suffers from the validity threats discussed in §6. To further reduce those threats and to move forward with this research, we propose the following future work:

- Apply hyper-parameter tuning on data preprocessing and model configuration to see if our current conclusions still hold and whether tuning can further improve the performance.
- Explore more complex patterns (other than just single word patterns **Easy** has explored).
- Survey more advanced feature engineering in the active learning strategy for finding the rest of SATDs. For example, explore N-gram patterns [72] and word embeddings with deep neural networks [19].
- Explore other sampling techniques to help with unbalanced class data (one of the key characteristics for SATDs [55]).
- Test whether replacing the random forest model in **DebtFree** with a deep learning model (i.e., CNN [55]) will further improve its performance.
- Try different settings of labeling and filtering (via unsupervised learning) combine with a deep learning model (i.e., CNN [55]) to automatically identify the SATDs.
- Survey other unsupervised learning methods for frugally pseudo-labeling and filtering data.
- Extend the work to label debt in other artifacts where technical debt may be presented (e.g. issue trackers, documentation) and other software engineering domains (e.g., security, issue close times, static warning analysis,

etc) and compare it with other state-of-the-art methods which continue to appear.

Acknowledgements

This work was partially funded by an NSF CISE Grant #1931425.

References

1. A. Agrawal and T. Menzies. Is" better data" better than" better data miners"? In *ICSE*, 2018.
2. K. Ali and O. Lhoták. Application-only call graph construction. In *ECOOP*, 2012.
3. N. SR Alves, L. F Ribeiro, V. Caires, T. S Mendes, and R. O Spínola. Towards an ontology of terms on technical debt. In *TechDebt*, 2014.
4. S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *ICSE*, 2019.
5. G. Bavota and B. Russo. A large-scale empirical study on self-admitted technical debt. In *MSR*, 2016.
6. L Breiman. Random Forests. *Machine learning*, 2001.
7. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. *Cytometry*, 1987.
8. G. Catolino. Just-in-time bug prediction in mobile applications: The domain matters! In *MOBILESoft*, 2017.
9. N. V Chawla, K. W Bowyer, L. O Hall, and W P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *JAIR*, 2002.
10. D. Chen, K. T Stolee, and T. Menzies. Replication can improve prior results: A github study of pull request acceptance. In *ICPC*, 2019.
11. D. Chicco and G. Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 2020.
12. J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, 1968.
13. P. R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.
14. M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *EMSE*, 2012.
15. M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. da Silva, and R. O. Spínola. A contextualized vocabulary model for identifying technical debt on code comments. In *MTD*, 2015.
16. M. A. de Freitas Farias, J. A. Santos, M. Kalinowski, M. Mendonça, and R. O. Spínola. Investigating the identification of technical debt through code comment analysis. In *ICEIS*, 2016.

17. M. A. de Freitas Farias, M. G. de Mendonça Neto, M. Kalinowski, and R. O. Spínola. Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *IST*, 2020.
18. R. Feldt and A. Magazinius. Validity threats in empirical software engineering research-an initial survey. In *SEKE*, 2010.
19. J. Flisar and V. Podgorelec. Identification of self-admitted technical debt using enhanced feature selection based on word embedding. *IEEE Access*, 2019.
20. F. A. Fontana, V. Ferme, and S. Spinelli. Investigating the impact of code smells debt on quality code evaluation. In *MTD*, 2012.
21. W. Fu and T. Menzies. Revisiting unsupervised learning for defect prediction. In *FSE*, 2017.
22. G. Fucci, N. Cassee, F. Zampetti, N. Novielli, A. Serebrenik, and M. Di Penta. Waiting around or job half-done? sentiment in self-admitted technical debt. In *MSR*, 2021.
23. J. Graf. Speeding up context-, object-and field-sensitive sdg generation. In *SCAM*, 2010.
24. Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. QB Da Silva, A. LM Santos, and C. Siebra. Tracking technical debt—an exploratory case study. In *ICSME*, 2011.
25. Z. Guo, S. Liu, J. Liu, Y. Li, L. Chen, H. Lu, Y. Zhou, and B. Xu. Mat: A simple yet strong baseline for identifying self-admitted technical debt, 2019.
26. A. E. Hassan. Predicting faults using the complexity of code changes. In *ICSE*, 2009.
27. H. Hata, C. Treude, R. G. Kula, and T. Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *ICSE*, 2019.
28. A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: A taxonomical study of large commits. MSR, 2008.
29. D. W Hosmer Jr, S. Lemeshow, and R. X Sturdivant. *Applied logistic regression*. 2013.
30. Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li. Identifying self-admitted technical debt in open source projects using text mining. *EMSE*, 2018.
31. Y. Kamei, E. Shihab, B. Adams, A. E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi. A large-scale empirical study of just-in-time quality assurance. *TSE*, 2013.
32. S. Kim, E. J. Whitehead, Jr., and Y. Zhang. Classifying software changes: Clean or buggy? *IEEE Trans SE*, 2008.
33. E. Kocaguneli, T. Menzies, J. Keung, D. Cok, and R. Madachy. Active learning and effort estimation: Finding the essential content of software effort estimation data. *TSE*, 2012.
34. E. Lim, N. Taksande, and C. Seaman. A balancing act: What software practitioners have to say about technical debt. *IEEE Software*, 2012.
35. Z. Liu, Q. Huang, X. Xia, E. Shihab, D. Lo, and S. Li. Satd detector: a text-mining-based self-admitted technical debt detection tool. In *ICSE*, 2018.

36. R. R Lutz and I. C. Mikulski. Empirical analysis of safety-critical anomalies during operations. *TSE*, 2004.
37. E. da S Maldonado and E. Shihab. Detecting and quantifying different types of self-admitted technical debt. In *MTD*, 2015.
38. E. da S. Maldonado, E. Shihab, and N. Tsantalis. Using natural language processing to automatically detect self-admitted technical debt. *TSE*, 2017.
39. R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *ICSME*, 2004.
40. R. Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 2012.
41. R. Marinescu, G. Ganea, and I. Verebi. Incode: Continuous quality assessment and improvement. In *CSMR*, 2010.
42. A. Martini and J. Bosch. The danger of architectural technical debt: Contagious debt and vicious circles. In *12th ICSA*, 2015.
43. T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald. Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'". *TSE*, 2007.
44. T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *TSE*, Jan 2007.
45. A. Mockus and L. Votta. Identifying reasons for software changes using historic databases. In *ICPC*, 2000.
46. N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan. Curating github for engineered software projects. *EMSE*, 2017.
47. J. Nam and S. Kim. Clami: Defect prediction on unlabeled datasets. In *ASE*, 2015.
48. M. Nayrolles and A. Hamou-Lhadj. Clever: Combining code metrics with clone detection for just-in-time fault prevention and resolution in large industrial projects. In *MSR*, 2018.
49. C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu. Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. *TSE*, 2020.
50. A. Nugroho, J. Visser, and T. Kuipers. An empirical model of technical debt and interest. In *MTD*, 2011.
51. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 2011.
52. J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo. The jinx on the nasa software defect data sets. In *EASE*, 2016.
53. A. Potdar and E. Shihab. An exploratory study on self-admitted technical debt. In *ICSME*, 2014.
54. F. Rahman and P. Devanbu. How, and why, process metrics are better. In *ICSE*, 2013.
55. X. Ren, Z. Xing, X. Xia, D. Lo, X. Wang, and J. Grundy. Neural network-based detection of self-admitted technical debt: From performance to explainability. *TOSEM*, 2019.

56. C. Rosen, B. Grawi, and E. Shihab. Commit guru: Analytics and risk prediction of software commits. ESEC/FSE 2015, 2015.
57. S. Sawilowsky. New effect size rules of thumb. *JMASM*, 2009.
58. B. Settles. Active learning literature survey. 2009.
59. M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect datasets. *TSE*, 2013.
60. M S. Silberman, B. Tomlinson, R. LaPlante, J. Ross, L. Irani, and A. Zaldivar. Responsible research with crowds: pay crowdworkers at least minimum wage. *Communications of the ACM*, 2018.
61. L. Tan, D. Yuan, G. Krishna, and Y. Zhou. /* icomment: Bugs or bad comments?*. In *OSR*, 2007.
62. S. H. Tan, D. Marinov, L. Tan, and G. T Leavens. @ tcomment: Testing javadoc comments to detect comment-code inconsistencies. In *ICST*, 2012.
63. N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *JSS*, 2011.
64. N. Tsantalis, D. Mazinanian, and G. P. Krishnan. Assessing the refactorability of software clones. *TSE*, 2015.
65. H. Tu and T. Menzies. Frugal: Unlocking ssl for software analytics. In *ASE*, 2021.
66. Huy Tu, Rishabh Agrawal, and Tim Menzies. The changing nature of computational science software, 2020.
67. Huy Tu, Zhe Yu, and Tim Menzies. Better data labelling with emblem (and how that impacts defect prediction). *TSE*, 2020.
68. B. Vasilescu. Personnel communication at fse'18, 2018.
69. B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in github. In *FSE*, 2015.
70. B. C Wallace, T. A Trikalinos, J. Lau, C. Brodley, and C. H Schmid. Semi-automated screening of biomedical citations for systematic reviews. *BMC bioinformatics*, 2010.
71. X. Wang, J. Liu, L. Li, X. Chen, X. Liu, and H. Wu. Detecting and explaining self-admitted technical debts with attention-based neural networks. In *ASE*, 2020.
72. S. Wattanakriengkrai, N. Srisermphoak, S. Sintoplertchaikul, M. Choetkiertikul, C. Ragkhitwetsagul, T. Sunetnanta, H. Hata, and K. Matsumoto. Automatic classifying self-admitted technical debt using n-gram idf. In *APSEC*, 2019.
73. S. Wehaibi, E. Shihab, and L. Guerrouj. Examining the impact of self-admitted technical debt on software quality. In *SANER*, 2016.
74. I Witten, Eibe Frank, M Hall, and C Pal. : Data mining: practical machine learning tools and techniques. elsevier inc. 2017.
75. Z. Xu, L. Li, M. Yan, J. Liu, X. Luo, J. Grundy, Y. Zhang, and X. Zhang. A comprehensive comparative study of clustering-based unsupervised defect prediction models. *JSS*, 2021.
76. M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang. File-level defect prediction: Unsupervised vs. supervised models. In *ESEM*, 2017.

77. J. Yang and H. Qian. Defect prediction on unlabeled datasets by using unsupervised clustering. In *HPCC/SmartCity/DSS*, 2016.
78. X. Yang, D. Lo, X. Xia, and J. Sun. Tlel: A two-layer ensemble learning approach for just-in-time defect prediction. *IST*, 2017.
79. Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. Deep learning for just-in-time defect prediction. In *QRS*, pages 17–26. IEEE, 2015.
80. Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In *24th SIGSOFT FSE*, 2016.
81. Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. FSE, 2016.
82. Z. Yu, N. A. Kraft, and T. Menzies. Finding better active learners for faster literature reviews. *EMSE*, 2018.
83. Z. Yu, C. Theisen, L. Williams, and T. Menzies. Improving vulnerability inspection efficiency using active learning. *TSE*, 2019.
84. Z. Yu, F. M. Fahid, H. Tu, and T. Menzies. Identifying self-admitted technical debts with jitterbug: A two-step approach. *TSE*, 2020.
85. F. Zampetti, A. Serebrenik, and M. Di Penta. Automatically learning patterns for self-admitted technical debt removal. In *SANER*, 2019.
86. N. Zazworka, R. O Spínola, A. Vetro, F. Shull, and C. Seaman. A case study on effectively identifying technical debt. In *EASE*, 2013.
87. F. Zhang, Q. Zheng, Y. Zou, and A. E Hassan. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *ICSE*, 2016.
88. Y. Zhou, Y. Yang, H. Lu, L. Chen, Y. Li, Y. Zhao, J. Qian, and B. Xu. How far we have progressed in the journey? an examination of cross-project defect prediction. *TOSEM*, 2018.