Using Active Learning to Train Predictive Mutation Testing with Minimal Data

1st Miklós Borsi
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology
Karlsruhe, Germany
miklos.borsi@kit.edu

***Abstract*—Mutation testing is a powerful method of evaluating test suite adequacy. Despite growing industry attention, wide-scale application is frequently limited by the high runtime cost of mutation testing. A set of predictive models have been proposed to mitigate this cost issue, intending to replace the actual execution of a mutated program's test suite with a predicted result of the tests' outcome. These predictive models ingest static code features, dynamic execution features, or code and documentation text to produce the predictions. Feature-based models can require a large amount of training data and mutants executed by test cases to become operational. We propose active learning-based predictive mutation testing (AL-PMT) as a way to dramatically reduce the amount of training data needed for a performant model. We conduct experiments to compare AL-PMT's performance with a non-active learning model and find that AL-PMT quickly converges to improved or on-par performance compared to the baseline of the foundational PMT. AL-PMT achieves 98% of its best possible performance in over 80% of examined projects, while observing only 10% of each project's mutant set kill status. In addition to training in a fraction of the data required for previous models, AL-PMT is organized in a way that is more amenable to a potential industry application scenario. Besides not requiring the building, running and full mutation testing of several other projects or versions for training data, AL-PMT is able to identify challenging mutants and select them for execution. As such, we expand on the coverage metric provided by basic predictive mutation testing with the ability to guide the targeted execution of important mutants and guiding attention of developers to remaining survived ones. This addresses the rarely mentioned cost of human developer time on fixing the findings of mutation testing, rather than just the computational time spent producing the mutants.**

***Index Terms*—active learning, mutation testing**

I. Introduction

Software testing plays an essential role in the creation of safe, secure and maintainable software applications. A key step of the testing process is ensuring that the written tests cover the appropriate amount of the program's possible input space and behaviors. Mutation score correlates better with a test suite's fault finding ability than coverage. Mutation testing works by introducing a change to the program under test, creating a "mutant". These mutants are created through applying a set of mutation operators, for example "delete a statement" or "replace = with !=". The modified program is then executed. If a test detects an issue, the mutant is considered killed, if no tests notice the change it is considered survived, and the mutation score is the ratio of killed to total generated mutants. The test suite's ability to kill mutants can be expected to correlate with that of finding real faults [1], [2].

Mutation testing is regarded as an effective process, but also computationally expensive. Running all tests against a large number of modified versions of the initial program may add up to an unfeasible amount of runtime, with several approaches attempting to address this [3]–[5]. One of the possible ways around this issue is Predictive Mutation Testing (PMT). In this framework, a machine learning model is trained to recognize which mutants evade a test suite (survived mutants) and which ones do not (killed mutants). Using such model for prediction saves actually running the test suite on all mutants, saving considerable amounts of time and resources. Several different types of models have been proposed to predict mutant kill results, ranging from the the initial feature-based PMT [6]–[8], to language and code text-based ones [9]–[11]. Henceforth, "PMT" will refer to the specific paper of "Predictive Mutation Testing" [6], and "predictive mutation testing" will refer to the general idea of using a predictive model for mutation testing.

A less-frequently noted issue with mutation testing processes is the developer time required to fix, or write tests for, potential vulnerabilities that were uncovered by mutation testing. As such, industry applications [12]–[14] have to solve more than just computational cost, but also proper integration into Continuous Integration pipelines, developer and manager training, and changing code review practices [15]–[17].

Not all survived mutants necessarily lead to a code vulnerability. Mutants can also be non-trivially equivalent, or could end up not changing the actual behavior of the program. If one were to integrate mutation testing as an active guide to development—rather than just considering it another coverage metric, or targeting some sufficient mutation score before a release—then they should keep in mind not only the computational cost but the human time cost to address the outcomes of mutation testing [18].

Downloading, building, running, testing, and feature extracting several other projects for a large enough training set can be a large initial obstacle to initializing a PMT process. Additionally, the uncertainty inherent in a predictive model

The author gratefully acknowledges the financial support of the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00217

can be a barrier towards replacing standard coverage metrics or fully executing mutation test suites. Actual performance may differ as well due to PMT itself suffering from a highly imbalanced dataset [19] with some trivial data points [20].

An overview of the AL-PMT process can be found in Fig.1, and a detailed explanation in II-B. Our solution to the runtime burden synthesizes the idea of predictive models saving work, with choosing only the most informative samples from a massive set of data points, unlabeled until executed and tested. We do so by applying *Active Learning* [21] (AL) to the problem at hand. This provides ongoing suggestions on the most relevant mutant at a time, rather than a probabilistic overview of the global mutation score. We believe that the easier setup and clearer management of the accuracy/speed tradeoff in a predictive model will promote industry use.

AL is a subfield of machine learning where the guiding assumption is that the features of data points are known but the labels are not. The features for all data points are typically known at the start in entirety, called the *pool-based* scenario, and this is the one we will be considering for this paper.

The core of an AL method is the *query strategy*, i.e., a way to request the labels of unlabeled data points from the oracle. The query strategy is designed to maximize model performance while minimizing costs, for instance by selecting data points closest to the decision boundary or with the highest expected information gain. During a training step the active selection strategy computes a metric about the remaining unknown-label data points and chooses the next one through optimizing for said metric. It requests a label from the oracle, and gives the labeled data point to the model for training. This is usually done one point at a time, though depending on the time for acquiring a label versus training, it can be done in batches.

There is a set of common challenges faced by practical AL implementations. [22] provides a thorough overview of these. Some simplifying assumptions could turn out to be sufficiently untrue in some field of application. Consider speech tagging; a query strategy that chooses the longest recordings would be efficient in number of data points but not necessarily in total labeling time and effort. As another example, [23] presents difficulty in dealing with unknown labeling costs, and finding that even knowing the true cost may not improve cost-efficiency over an inaccurate estimate. This means that in practice, applying AL methodologies to real-world scenarios presents additional challenges. Our methodology is designed to overcome such challenges, hinting at potential applications to real-world use cases. We address the impact of these common AL issues in Section V and argue that the use case of predictive mutation testing avoids or strongly mitigates these issues.

In this paper we propose Active Learning-based Predictive Mutation Testing (AL-PMT), a new method for predictive mutation testing. This new method is much better suited to being in an active development workflow. It features a highly streamlined training process that uses a fraction of the data and training steps than the baseline PMT paper. The training process also has a method for finding the most informative examples to train the model on called the query strategy. This can be left as one of the existing methods in active learning literature, or customized to fit the needs of the development team, for example by integrating their feedback on which mutants in the output are found useful by developers. We trial AL-PMT method directly on the original PMT dataset [6] to provide as close a comparison of performance as possible. We plot the performance of AL-PMT at various degrees of training, give estimates for its performance on an arbitrary project, and find suitable points for stopping training along performance gain/time ratios.

In short, we make the following contributions:

- **Approach:** We propose AL-PMT, a new approach to predictive mutation testing literature, which drastically reduces training data needs. To our knowledge, we are the first to apply active learning techniques in this research area.
- **Evaluation:** We extensively compare the performance of Active Learning-based PMT to that of the original pre-trained PMT method. We find that AL-PMT achieves superior or on-par performance in the majority of cases while using the same underlying random forest model. We also analyze model performance over the whole course of training and suggest early stopping points.
- **Variance:** We look at the effects of random initialisation, and show that the query strategy converges on informative points regardless of the initial starting data.
- **Reproducibility:** We facilitate comparisons by working on a well-known and shared dataset. Additionally, all of our experiment code, experiment data, and evaluation code is made publicly available as a baseline for further studies.

II. Methodology

In this section we explain in detail the idea behind AL-PMT and how it accomplishes the previously stated goals.

A. Dataset and baseline

We use the dataset published by [6] at their repository.[1] We compare the performance of AL-PMT to PMT through F1 and ROC-AUC scores. PMT was evaluated in multiple different scenarios, and on different sets of train/test projects. In all cases they measured PMT's ability to predict mutants' killed/survived status from a set of execution, infection and propagation features in a binary classification task. These features measure how much the mutant interacts with the program and test set, what type of mutant it is, and how far the mutation's effect can spread throughout the program, all described in more detail in PMT's paper. On base projects where PMT was trained on 8 out of the 9 base projects and evaluated on the one left out, we call the "cross-project scenario". On Github projects where PMT was trained on all 9 base projects and evaluated on the Github projects we also call "cross-project". On base projects where PMT was trained

[1] https://github.com/sei-pku/PredictiveMutationTesting

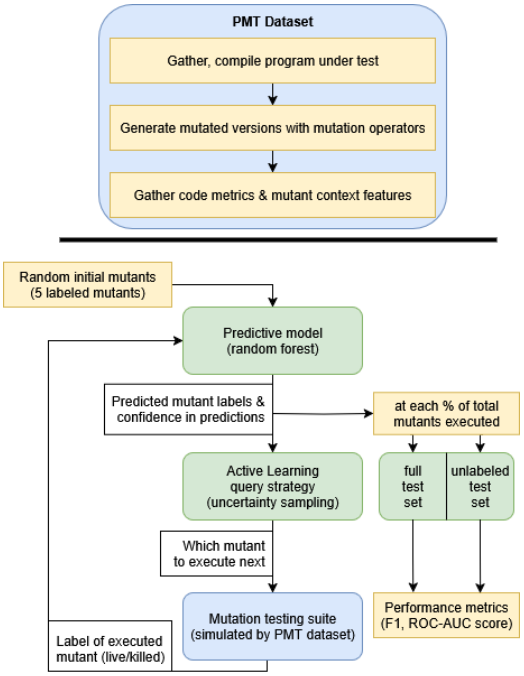

Fig. 1. Diagram of the AL-PMT process and components simulated through the PMT dataset

on version N and evaluated on version N+1 for each particular project for each possible N, we call the "cross-single-version" scenario.

For AL-PMT's training data we extract the features and labels from their .arff data files, using latest-version for the 9 base projects. The cross-single-version comparison on the 9 base projects have a variable number of experiments (2 to 6) available on each project; when using these figures we take the mean for each project when comparing to the single-shot AL-PMT process on said project. For the cross-project scenario we use the singular numbers as they are.

B. Our approach

An overview of the AL-PMT process can be seen in Fig.1. We consider the predictive mutation testing task as a binary classification exercise on a singular project at a time. We train a small-scale model for each software project in the PMT dataset separately through pool-based active learning. The model is the *random forest* algorithm, found to be the best performing in the PMT scenario. The model is initialized with a randomly selected set of 5 data points and their labels. During training the model has information of all features of all existing data points (mutants), but no knowledge of their associated labels (killed/survived status). At each step of the training process, the model selects a data point to request the label of. This new data point is chosen in some way as to maximize the useful information the model can get. In AL terms: it makes a query to the oracle according to its query strategy. In other words, this simulates choosing a mutant, then executing the test suite on it, and noting the killed/survived result. The model is then given the label, and it performs a standard training step with it. This then repeats until some stopping criterion, in our case, labeling 50% of the mutants in the project. We measure F1 score and ROC-AUC score as performance metrics.

For this paper we work with the common simplification of fixed equal cost per label. This is a reasonable approximation, though further work could look into the difference on mutations likely to cause a timeout failure versus those causing a crash. As such, the query strategy only takes mutant informativeness into account, and not some runtime cost/informativeness tradeoff.

Showing the improvement and learning process takes the form of active learning curves—plotting some accuracy metric (e.g. F1 score) against ratio of labels requested from the full label set. For our case we employ F1 and ROC-AUC score. In more complex scenarios e.g. a safety-critical application, it could instead be some optimal combined metric of measurement cost [24] and misclassification penalty [25].

The small scale of the model at hand, scoping at only a single software project, offers a number of advantages. It sidesteps some of the concerns plaguing large, pre-trained machine learning models. Overfitting on the training set is less of an issue when said training set is guaranteed to be the exact same distribution as the test set. This way the model does not have to learn the differences between projects and can be more accurate on the exact problem at hand, without needing to become more complex. The AL model is also more resilient to shifts in the underlying distribution (e.g. new coding paradigms, languages, libraries, or test suites) or low accuracy on lesser-represented data (e.g. Rust programs as opposed to Java). In the same way as previously, the shifts would be immediately represented in the data it exclusively trains on, and the minimal training data quantity means it can be applied straight to smaller subsets of software.

Alongside these advantages evaluating the resulting model's performance does become more challenging than usual. Throughout evaluation we keep in mind that AL-PMT is intended to circumvent having to execute each mutant in a project, and that acquiring a training data point's hidden label is the simulation of testing a particular mutant. The intent of the query strategy step is to select the most informative data point to add to the model, and to minimize the total number of labels requested. Our goal is to train a model by only using a fraction of the available data points—if there is a notable informativeness difference in the dataset, the model might end up selecting a near-matching subset of the best points every time, unless the informative points were all randomly excluded by a fold. Ultimately, for an end user of AL-PMT the test set that is most pertinent is the mutants that have *not* been executed the tests on yet. We discuss decisions taken on creating appropriate comparisons to PMT in Section IV and go into more detail on the evaluation's potential weaknesses in Section V.

III. Experimental setup

In this section we detail the experimental setup for testing our approach against the pre-trained PMT random forest model. We wish to match the performance of PMT but use a heavily reduced training set. Although the performance metrics and dataset is shared as closely as possible, its separation into particular training and test sets is different. PMT was evaluated in several different train/test variations—as explained in more detail below. We list several ways of comparing AL-PMT and PMT's performance metrics and argue which of these is most appropriate when considering a potential industry application. Finally, we analyze early stopping tradeoff points for even further savings on training data usage.

A. Research questions

- **RQ1:** How does AL-PMT's performance compare to that of the pre-trained PMT random forest model?
- **RQ2:** How does the choice of AL-PMT's test set affect model performance?
- **RQ3:** How does the choice of initial data points affect model performance?
- **RQ4:** How many labeled mutants are required for a good time/performance tradeoff?

B. Implementation

The random forest algorithm we use is the sklearn [26] python library's implementation. We use skactiveml's [27] implementation of uncertainty sampling. We use 1 as batch size, i.e. one mutant is chosen by the query strategy, then the random forest is fit on all labeled points including the newly chosen one, then the query strategy chooses another mutant. AL-PMT's random forest is initialized with 5 randomly selected mutants before training. Supporting data processing uses numpy, pandas, matplotlib, and underlying Python version 3.11.9. We make the results available online[2].

C. Evaluation

We train the AL-PMT on each project separately. The models are trained to up to 50% of the available mutants for a project. We take measurements of the model's performance after each 1% of data points ingested, stopping at 50% (exclusive). We save the full set of model predictions at each % to aid in reproducibility. Measured performance metrics are F1 score and ROC-AUC score on two test sets: performance on points that the model has not yet queried (we refer to this as "unlabeled") and performance on all points in the projects (referred to as "full").

We answer **RQ1** and **RQ2** by looking at F1 and ROC-AUC score under different conditions. For **RQ3** we look at the range of possible F1 and ROC-AUC scores over multiple runs, as well as the ratios of shared mutants, informative mutants that are selected for labeling among multiple differently initialised runs. For **RQ4** we look at training data percentages in two ways: first, how much training data is needed for surpassing PMT's F1 and ROC-AUC scores on the 9 base projects. Second, we measure early stop thresholds on all, base and Github, projects. This is given by computing the difference between AL-PMT's lowest and highest F1 pr ROC-AUC scores for all projects, then checking in how many projects AL-PMT achieves a higher F1/ROC-AUC score than $L + ((H - L) * X)$ at Y% of training data used, where L, H are the lowest and highest scores for that project, and X, Y are arbitrary thresholds.

AL-PMT is intended to be run on a single project in practice and to sample informative points from the project in the process. This does influence the distribution that we compute performance metrics on. As a result, the standard technique of k-fold cross validation is far less useful. To alleviate this as best as possible we present performance data in two ways, wrt. to the test set used. When labeled mutants are taken out of the test set ("unlabeled" case) the performance more closely matches to how well AL-PMT would predict mutants it had not requested test set execution on. Though, this does come with removing the most informative (and oftentimes difficult) examples from the test set for labeling, so we expect the performance to plateau less over training. Alternatively, when labeled mutants are not taken out ("full" case) we can more accurately measure how quickly AL-PMT converges to the underlying distribution of a project's mutants. This does come at a cost of some train and test mutants being shared.

IV. Results

To evaluate the outcomes of an active learning model, we employ active learning curves. An example of such (on project "joda") is shown in Fig. 2. The core question of this paper is how little training data we can utilize while producing an accurate model. In most cases, continuing to train a model with more data will keep increasing its performance, typically at a decreasing pace, bar overfitting. We limit the train set to a maximum of 50% of a project's mutants as beyond that any savings would be marginal. In an applied case if executing tests on x mutants is acceptable 2x should be too, though not necessarily 5x, 10x, 20x or more. Additionally, at 100% of the training set any sufficiently large neural network could perfectly learn the set with just step functions. Ideally though, we are looking for a suitable earlier stopping point.

As mentioned earlier the intended application scenario of AL-PMT makes choosing a test set more challenging. The figures showcase multiple combinations of AL-PMT and PMT scenarios in comparison, and we outline which comparison we find most appropriate in **RQ2**.

Fig. 2 also shows the effect of measuring the model's fit on the full mutant set versus doing so on the unlabeled part. The unlabeled version starts off slower but continues improving beyond that of the full fit. This is explained by the fact that the query strategies target data points that are difficult or high information value in some way. Any gains that the full-set evaluation gets in early training where some training samples are included in the test set, are later offset by the difficult

[2]https://zenodo.org/records/17259461

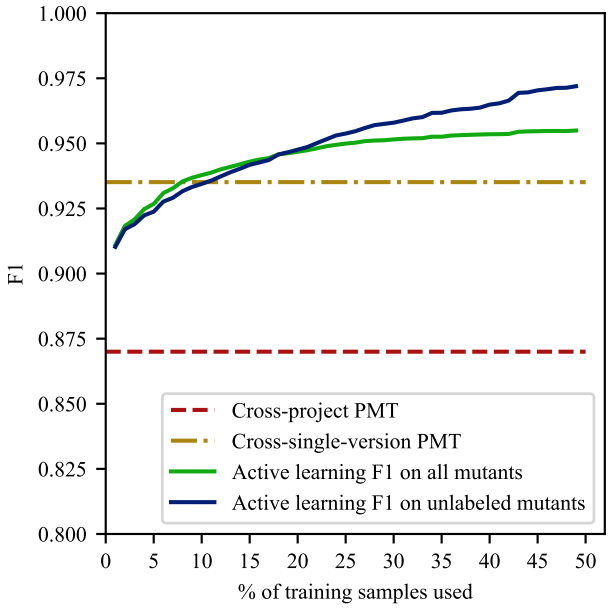

Fig. 2. Active learning curve of the joda project

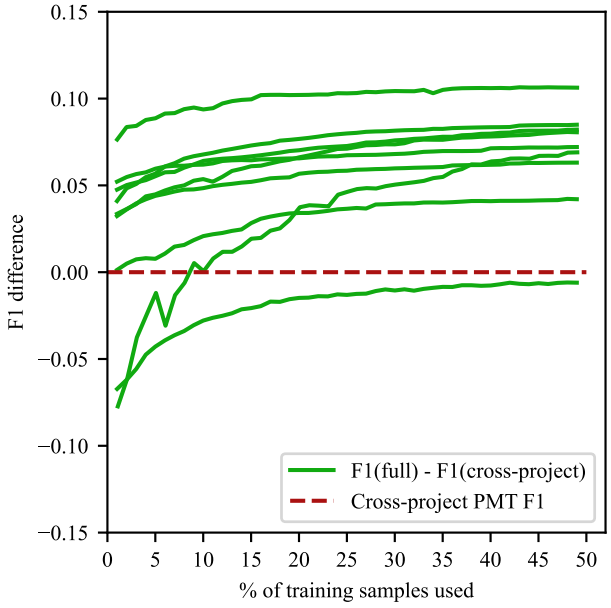

Fig. 3. AL-PMT to PMT comparison on 9 base projects, on AL-PMT's **full test set** vs. PMT's **cross-project set**, using F1 score

points being excluded from the test set later on by virtue of being selected for training.

A. *RQ1: How does AL-PMT's performance compare to that of the pre-trained PMT random forest model?*

Figs. 3-6, shows the active learning curves on all 9 of the base projects under different comparison bases. As PMT has different performance metrics on each base project, we take the difference between the metrics of AL-PMT and PMT, so that we can show both on the same axis. We list the performance achieved by AL-PMT on the 9 base projects at various training data thresholds in Tables I-IV. In these tables values are **bolded** if the AL-PMT value surpasses that of PMT in the associated comparison chart - the comparisons are made full set to cross-project, and unlabeled set to cross-single-version. Table VI lists these bolding thresholds of the previous tables in more detail: at what point does AL-PMT surpass PMT for the combination of each metric, project, scenario, and choice of AL-PMT's test set. Here we can see that AL-PMT is capable of matching or exceeding the performance of PMT on almost all base projects, answering **RQ1** in the positive. We contrast AL-PMT's performance with Table V where we show the outcome of randomly sampling labels, rather than consulting a query strategy. We observe that random sampling performs noticeably worse than AL-PMT, even more so earlier in training and in harder projects. It also does not exhibit the behavior where performance on the unlabeled set overtakes performance on the full set. This reinforces that active learning is capable of finding mostly informative and difficult samples, while random sampling is not.

TABLE I
AL-PMT'S F1 SCORE AT % OF TRAINING DATA THRESHOLDS, USING FULL TEST SET. **BOLDED** VALUES EXCEED CROSS-PROJECT PMT

Project name	F1 score at training data %					
	1%	*5%*	*10%*	*20%*	*30%*	*49%*
apns	0.807	0.872	**0.885**	**0.921**	**0.934**	**0.953**
assj	**0.963**	**0.971**	**0.973**	**0.977**	**0.979**	**0.983**
joda	**0.911**	**0.927**	**0.938**	**0.947**	**0.952**	**0.955**
lafj	**0.903**	**0.914**	**0.923**	**0.935**	**0.943**	**0.95**
lang	**0.928**	**0.94**	**0.944**	**0.953**	**0.956**	**0.959**
msg	**0.909**	**0.916**	**0.929**	**0.942**	**0.947**	**0.95**
uaa	0.843	0.867	0.882	0.895	0.899	**0.904**
vrapt	**0.938**	**0.945**	**0.954**	**0.96**	**0.966**	**0.972**
wire	**0.931**	**0.944**	**0.949**	**0.957**	**0.959**	**0.961**

TABLE II
AL-PMT'S F1 SCORE AT % OF TRAINING DATA THRESHOLDS, USING UNLABELED TEST SET. **BOLDED** VALUES EXCEED CROSS-SINGLE-VERSION PMT

Project name	F1 score at training data %					
	1%	*5%*	*10%*	*20%*	*30%*	*49%*
apns	0.805	0.865	0.872	0.909	0.926	**0.953**
assj	**0.963**	**0.97**	**0.972**	**0.978**	**0.98**	**0.992**
joda	0.91	0.924	0.934	**0.948**	**0.958**	**0.972**
lafj	0.902	**0.911**	**0.918**	**0.935**	**0.949**	**0.969**
lang	0.928	0.938	0.943	0.953	0.959	**0.974**
msg	0.908	0.913	0.922	0.942	**0.957**	**0.97**
uaa	0.841	0.859	0.868	0.886	0.899	0.911
vrapt	**0.937**	**0.944**	**0.952**	**0.959**	**0.965**	**0.975**
wire	0.931	**0.942**	**0.948**	**0.96**	**0.965**	**0.971**

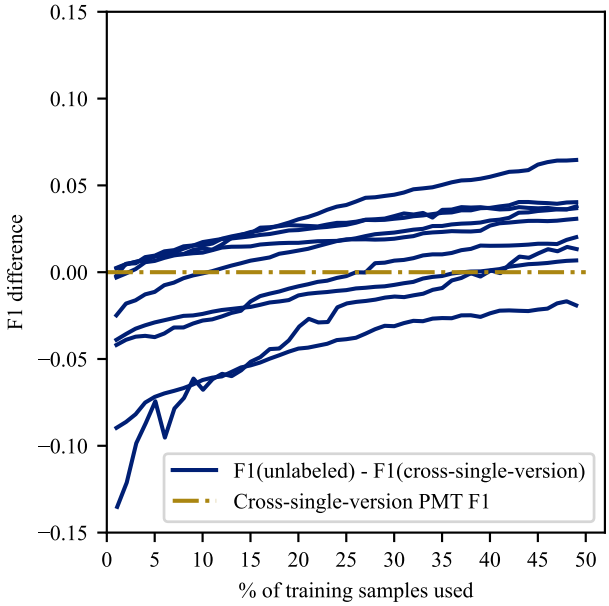

Fig. 4. AL-PMT to PMT comparison on 9 base projects, on AL-PMT's **unlabeled test set** vs. PMT's **cross-single-version set**, using F1 score

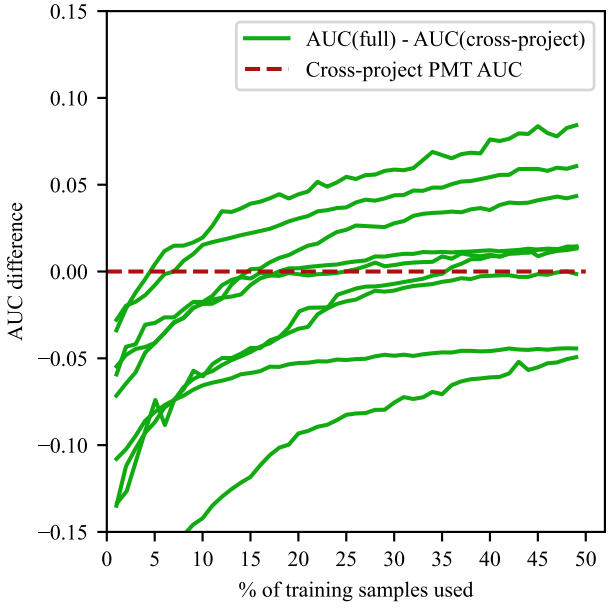

Fig. 5. AL-PMT to PMT comparison on 9 base projects, on AL-PMT's **full test set** vs. PMT's **cross-project set**, using ROC-AUC score

TABLE III
AL-PMT'S ROC-AUC SCORE AT % OF TRAINING DATA THRESHOLDS, USING FULL TEST SET. **BOLDED** VALUES EXCEED CROSS-PROJECT PMT

Project name	ROC-AUC score at training data %					
	1%	*5%*	*10%*	*20%*	*30%*	*49%*
apns	0.802	0.861	0.875	0.912	0.928	**0.949**
assj	0.805	**0.843**	**0.859**	**0.883**	**0.898**	**0.923**
joda	0.763	0.811	0.839	0.865	0.887	0.897
lafj	0.804	0.835	0.857	**0.888**	**0.904**	**0.919**
lang	0.647	0.723	0.754	0.803	0.82	0.847
msg	0.869	0.883	0.906	**0.926**	**0.933**	**0.938**
uaa	0.849	0.876	0.891	0.904	0.908	0.913
vrapt	0.854	0.874	**0.897**	**0.913**	**0.926**	**0.943**
wire	0.86	0.889	0.901	0.917	**0.923**	**0.932**

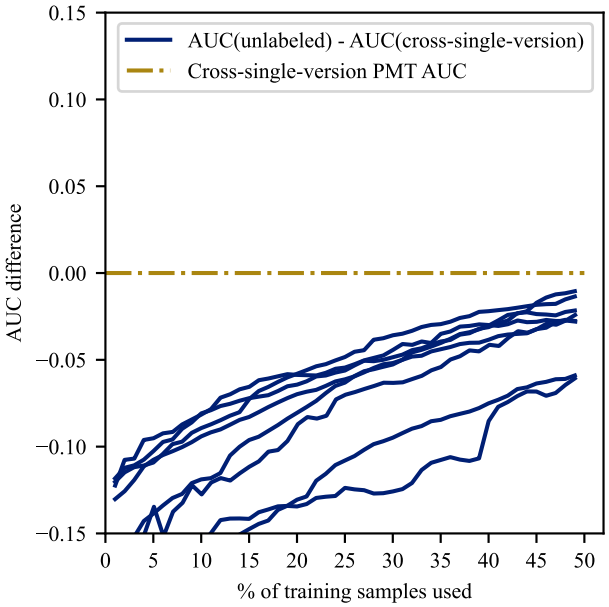

Fig. 6. AL-PMT to PMT comparison on 9 base projects, on AL-PMT's **unlabeled test set** vs. PMT's **cross-single-version set**, using ROC-AUC score

TABLE IV
AL-PMT'S ROC-AUC SCORE AT % OF TRAINING DATA THRESHOLDS, USING UNLABELED TEST SET. NO VALUES EXCEED CROSS-SINGLE-VERSION PMT

Project name	ROC-AUC score at training data %					
	1%	*5%*	*10%*	*20%*	*30%*	*49%*
apns	0.799	0.853	0.861	0.901	0.925	0.96
assj	0.8	0.826	0.833	0.855	0.863	0.928
joda	0.76	0.796	0.818	0.849	0.885	0.921
lafj	0.803	0.827	0.846	0.885	0.913	0.955
lang	0.64	0.692	0.709	0.74	0.762	0.808
msg	0.868	0.879	0.899	0.93	0.952	0.975
uaa	0.848	0.871	0.884	0.908	0.926	0.951
vrapt	0.852	0.868	0.89	0.906	0.922	0.949
wire	0.858	0.885	0.899	0.922	0.936	0.956

TABLE V
RANDOM SAMPLING PMT PERFORMANCE METRICS AT % OF TRAINING DATA THRESHOLDS. **BOLDED** VALUES EXCEED CROSS-PROJECT PMT ON FULL TEST SET, CROSS-SINGLE-VERSION PMT ON UNLABELED TEST SET. NO VALUES EXCEED THAT OF AL-PMT IN THE RESPECTIVE TEST SET

Project & test set	F1 score						ROC-AUC score					
	1%	*5%*	*10%*	*20%*	*30%*	*49%*	*1%*	*5%*	*10%*	*20%*	*30%*	*49%*
apns full	0.659	0.774	0.811	0.881	**0.888**	**0.92**	0.483	0.785	0.821	0.878	0.885	0.914
apns unlabeled	0.654	0.76	0.788	0.851	0.844	0.872	0.481	0.774	0.801	0.849	0.842	0.864
assj full	**0.937**	**0.961**	**0.965**	**0.968**	**0.972**	**0.976**	0.686	0.805	0.829	**0.855**	**0.878**	**0.899**
assj unlabeled	0.936	0.959	**0.961**	**0.962**	**0.965**	**0.965**	0.683	0.797	0.814	0.832	0.853	0.857
joda full	**0.897**	**0.909**	**0.917**	**0.925**	**0.929**	**0.939**	0.733	0.792	0.806	0.829	0.842	0.868
joda unlabeled	0.896	0.905	0.909	0.912	0.911	0.911	0.731	0.782	0.789	0.798	0.799	0.809
lafj full	**0.872**	**0.889**	**0.899**	**0.911**	**0.919**	**0.93**	0.794	0.813	0.833	0.865	0.874	**0.892**
lafj unlabeled	0.871	0.884	0.89	0.894	0.895	0.896	0.792	0.804	0.816	0.836	0.837	0.84
lang full	**0.917**	**0.925**	**0.93**	**0.935**	**0.939**	**0.946**	0.641	0.689	0.725	0.761	0.783	0.816
lang unlabeled	0.917	0.922	0.923	0.923	0.922	0.922	0.638	0.674	0.699	0.717	0.725	0.735
msg full	0.883	0.897	**0.913**	**0.915**	**0.923**	**0.932**	0.848	0.866	0.884	0.89	0.903	0.914
msg unlabeled	0.882	0.892	0.906	0.898	0.9	0.896	0.847	0.859	0.872	0.867	0.872	0.869
uaa full	0.757	0.807	0.833	0.855	0.865	0.881	0.774	0.82	0.845	0.866	0.876	0.891
uaa unlabeled	0.755	0.797	0.818	0.831	0.833	0.841	0.772	0.811	0.83	0.842	0.843	0.851
vrapt full	**0.92**	**0.934**	**0.939**	**0.946**	**0.95**	**0.96**	0.823	0.856	0.871	**0.888**	**0.902**	**0.922**
vrapt unlabeled	0.919	0.93	0.933	**0.935**	**0.934**	**0.939**	0.821	0.85	0.858	0.865	0.87	0.882
wire full	**0.88**	**0.926**	**0.935**	**0.942**	**0.946**	**0.951**	0.736	0.86	0.883	0.897	0.905	0.914
wire unlabeled	0.879	0.922	0.929	0.929	0.931	0.928	0.734	0.854	0.874	0.876	0.882	0.878

TABLE VI
SURPASS POINTS OF PMT'S PERFORMANCE

PMT project & scenario	AL-PMT surpasses at:			
	F1 score		ROC-AUC score	
	full	*unlabeled*	*full*	*unlabeled*
apns cross	9%	15%	36%	36%
apns single	36%	40%	-	-
assj cross	1%	1%	5%	11%
assj single	1%	1%	-	-
joda cross	1%	1%	48%	37%
joda single	8%	11%	-	-
lafj cross	1%	1%	15%	18%
lafj single	2%	3%	-	-
lang cross	1%	1%	-	-
lang single	-	38%	-	-
msg cross	1%	1%	18%	18%
msg single	-	28%	-	-
uaa cross	-	47%	-	-
uaa single	-	-	-	-
vrapt cross	1%	1%	8%	9%
vrapt single	1%	1%	-	-
wire cross	1%	1%	25%	17%
wire single	2%	2%	-	-

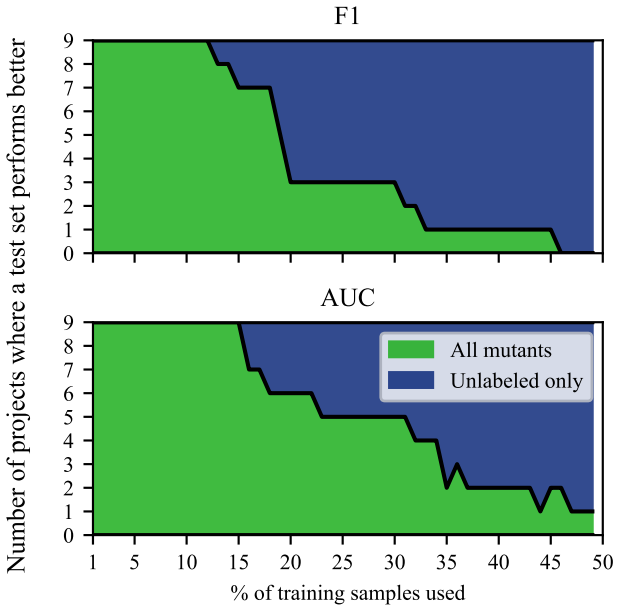

Fig. 7. Metric performance on different test sets

B. *RQ2: How does the choice of test set affect performance metrics?*

To support the arguments at the beginning of this section, and interpret the contents of Table VI, we consult Fig. 7. This shows which test set option gives higher performance metrics in base projects on how much training was done. This confirms our earlier suspicions that eliminating difficult points from the full test set by simulated test case execution sets up "unlabeled" for an advantage further into training. It also shows that F1 score is more influenced by this effect than ROC-AUC score; full set F1 falls behind at 20% of the training data labeled but ROC-AUC only does so at 35%.

We observe two things when comparing to the original PMT numbers. Firstly, PMT achieves higher performance when trained in the single-project, single-past-version scenario than in the cross-project one. 7 out of 9 of the cross-project metrics on base projects are immediately surpassed by both AL-PMT test set options. Secondly, the Github projects are in general much smaller than the base projects, and were only examined in the cross-project scenario.

We imagine AL-PMT being used in a "cold start" situation: a project where no previous mutation testing work was done, and where industry practitioners train it only on their own project, executing the test suite only on the mutants chosen by

AL-PMT. Once built this way it could even continue supplying predictions to future versions of the program. As such, the predictions would be used on the not-yet tested mutants. For **RQ2**, we believe that the most suitable comparison is therefore the cross-single-version experiment of PMT versus the "unlabeled" test set experiment of AL-PMT. For the Github projects we continue using "unlabeled" as the test set and compare with the available cross-project PMT.

C. *RQ3: How does the choice of initial data points affect model performance?*

We select three projects of different sizes from the 9 base projects: "apns" with 1105 mutants, "uaa" with 7618 mutants, and "wire" with 3047 mutants. We run the AL-PMT process on them 100 times, randomizing the initial 5 selected mutants each time.

Fig.8 shows the range and median of performance metrics achieved by the 100 runs. We see that the performance range narrows quickly, and that the effect of random initialisation is small beyond 5-10% of training data threshold. The slight exception to this is "apns", which is the project with the fewest mutants in the 9 base projects. However, when considering the absolute number of mutants labeled instead of percentages, the range of possible outcomes narrows down at around the same 200-400 mutants as for the other two projects here.

Fig.9 illustrates on how much agreement there is among the 100 runs on which mutants are to be labeled next. At each % of training data we count the number of mutants that have been selected for labeling by multiple (50, 80, 90, or all 100) differently initialised runs, divided by the total number of mutants in a run up to that point. For example, on 'apns' by 30% of the total maximum training data, 20% of the mutants up to that point - so, 6% of total max - have been chosen by all other runs as well, and around 50% of mutants up to that point have been chosen by at least 80 runs.

These figures indicate high agreement in which data points are considered informative by the query strategy in different situations, with clearer trends in larger projects. Different runs end up quickly sharing at least half of their selected data points right after initialisation. After this exploration of the dataset, agreement in all runs on some points starts between 8% and 20% of a project's mutants. The somewhat parallel lines of the 80, 90 and 100 thresholds, more pronounced on "uaa" and "wire", indicate increasing agreement on existing data points. All mutants that have been selected by 100 runs must necessarily have been selected by 90 previously - the lines trending upwards at the same rate indicate increased agreement on previous mutants, rather than newfound agreement on new mutants.

D. *RQ4: How many labeled mutants are required for a good time/performance tradeoff?*

We look at all available projects and plot the difference between AL-PMT and cross-project PMT performance, under entropy-based uncertainty sampling. We use cross-project PMT values for the 9 base projects for this to keep it consistent

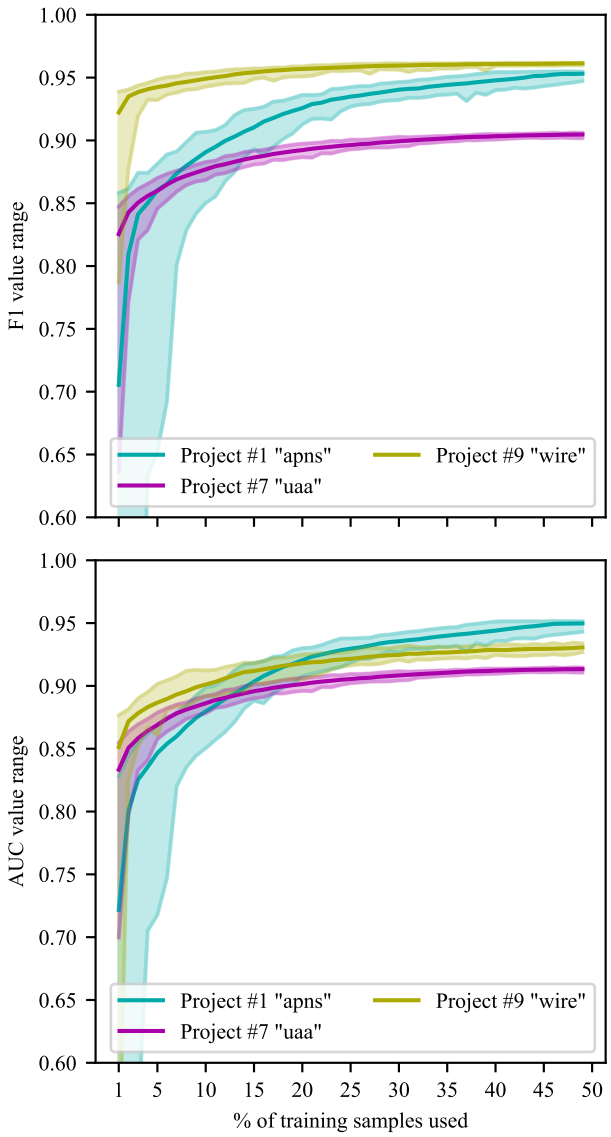

Fig. 8. 100-run performance range on apns, uaa and wire

with the Github project set. Some Github projects are very small (30-40 mutants) so to avoid further reducing the already tiny test set they provide in this experiment we use the full mutant set as test. We filter a small number of Github projects where the respective metric is 1.0 or 0.0 in at least 95% of measured points, due to prediction on these projects being too trivial. This removes 10 projects from the F1 graph and 14 from the AUC graph. We then show the full range of values over the learning curve in Fig. 10, including 5, 10, 20 and 50 percentile ranges. It shows that the median project quickly surpasses PMT's performance, and that almost 80% of the projects here achieve at least parity with PMT, and over 20% do noticeably better. There remain some outliers in the Github

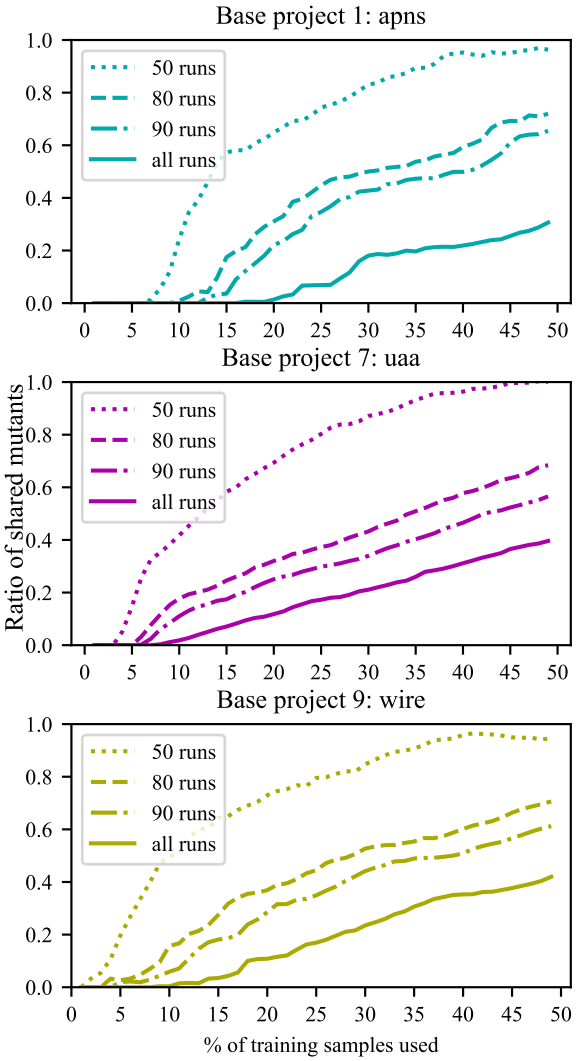

Fig. 9. Ratio of shared selected mutants in 100 runs

projects that the trivial filtering step does not catch, seen by the minimum lines on the percentile range graphs. Although AL-PMT is very data-efficient, some tiny projects can have enough mutants with all-different features that learning the feature-label link from only the project is impossible. Though, for projects sufficiently small for AL-PMT to fail on, the full mutant set could likely be ran in full anyway.

Developers looking to employ AL-PMT can consult the figure on at what point they can expect to stop training for a certain performance expectation. We supplement this information in Table VII. We take the total amount of improvement over the course of training a model (i.e. the difference between the maximum and the minimum score on the AL curve) and list how much of the potential performance improvement was achieved by a certain time, by quantity of projects it was done on. So, for example, AL-PMT reached 95% of its best performance potential on F1 score by 10% of the training set in 82% of cases.

To answer **RQ4** we claim that AL-PMT has high confidence in achieving PMT-par performance early in the training process when applied to an unknown project. Depending on the user's required accuracy we provide a lookup table of how long they should train for. In over 80% of projects, training on just one-tenth of the total available mutant set on the singular project puts AL-PMT at 98% of its performance cap. Said cap is also strongly expected to be close to or above to that of cross-project PMT, which was trained on the mutant set of all 9 large base projects, not just 1.

TABLE VII
AL-PMT PERFORMANCE EXPECTATIONS

Reached improvement	AL-PMT at X% training				
	1	*5*	*10*	*20*	*30*
98% of F1	64%	78%	81%	85%	88%
98% of AUC	64%	78%	80%	84%	88%
95% of F1	69%	78%	82%	86%	89%
95% of AUC	69%	78%	82%	86%	89%
90% of F1	72%	80%	84%	87%	90%
90% of AUC	72%	80%	84%	87%	90%
85% of F1	78%	82%	85%	89%	91%
85% of AUC	78%	82%	85%	88%	90%
80% of F1	78%	85%	88%	90%	92%
80% of AUC	78%	85%	87%	90%	91%
75% of F1	80%	85%	88%	91%	92%
75% of AUC	80%	85%	88%	91%	92%

V. DISCUSSION

A. Uncovered mutants

While PMT is a foundational work in this area, it is not without criticisms. In adopting a shared dataset and evaluation method or metrics, we were able to showcase the benefits and data efficiency of active learning in a very close comparison to PMT. This however means that observations on the overall validity of said metrics also hold for our research. In particular, it has been noted that the dataset includes a large number of unreached mutants which are not covered by any test [19], [20]. Excluding these cuts the overall size of the dataset significantly from 519 530 to 135 947. This filtering would reverse the previous class imbalance of the dataset, going from 20.22% killed and 79.78% survived mutants to 77.29% killed and 22.71% survived. In addition to the change in class-specific metrics, combined measures like F1 and ROC-AUC also suffer in this more difficult variant of the dataset, seeing as how being uncovered leading to a survived mutant is a trivial relationship to learn.

Although this is vital information with regards to the predictability of mutant classes we argue that the impact of it is far less severe in the case of AL-PMT. As shown in

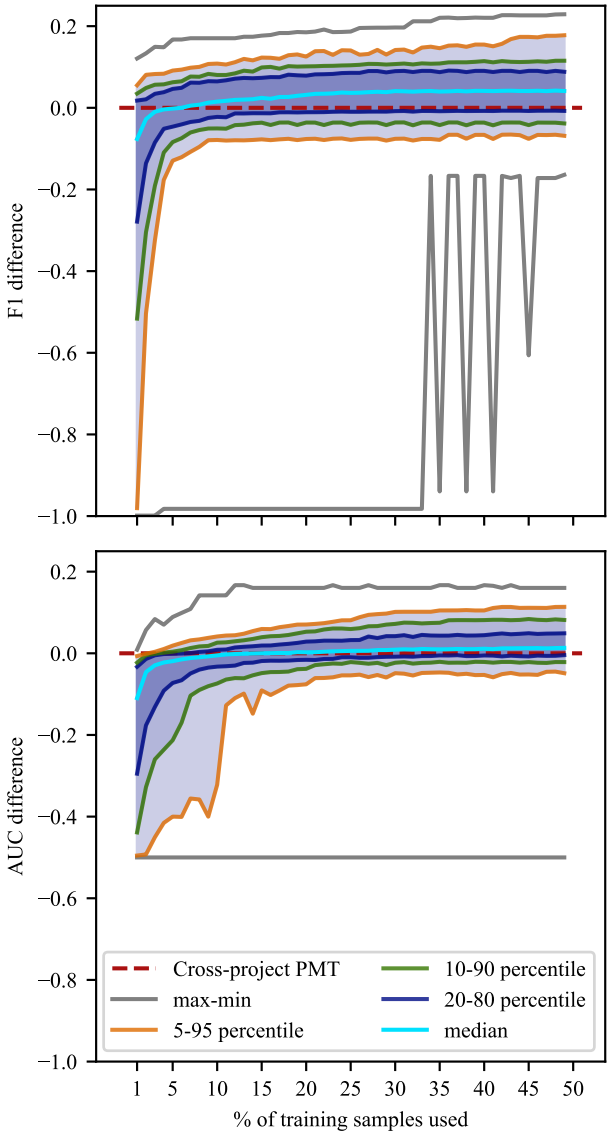

Fig. 10. All projects performance comparison with percentile bounds

the difference between the choice of test set in Figs. 3-6 and the rate of reaching performance milestones in Table VII, AL-PMT reaches close to the comparable cap of the model's performance quickly. The rate is usually faster, 1/20th to 1/5th of the dataset, than the previous filtering to uncovered mutants of approximately 1/4th would suggest. In addition, we see a frequent continuing upward trend with less of a clear performance cap when using only the unlabeled remaining set, as opposed to the full mutant set. The combination of these two factors suggest that AL-PMT does learn the trivial relationship quickly, then continues to query the more difficult samples in the dataset. The involvement of a query strategy can be viewed as a sampling and filtering method, which are often suggested in dealing with the imbalanced dataset having a share of easy points. While this means that the F1 and ROC-AUC scores are not directly translatable to a more challenging dataset, it does not invalidate the AL-PMT application scenario. Rather, it shows that selecting which mutants to test based on a combination of a predictive model and a query strategy can very quickly separate the easy and difficult mutants and suggest the most relevant ones for actual test execution instead of prediction, or show them to developers for further examination.

B. Common practical obstacles of active learning

Here we list applicable common issues that active learning faces in practical implementation as identified by [22]. As our goal is easy practical applicability, these common issues represent threats to validity. We discuss how they might apply to mutation testing and AL-PMT specifically.

Querying in batches raises the point that there can be a notable delay between sending out a query and receiving a label, often the case with human labelers. As processing a data point can strongly influence the informativeness of the remaining points, a simple greedy selection of the top N can be highly inefficient. Batched active learning may situationally require query strategies specifically designed for it [28]–[30]. We ran all AL-PMT experiments with a batch size of 1 and did not check how batching would impact performance. We argue that in the basic case without integrated developer feedback this is a fair assumption; the computer can swap between executing a test suite to training the model on the result quickly, and experiences no significant downtime in between. Any extension of AL-PMT involving developer input should keep batching in mind.

Noisy oracles asks what if the information received from the oracles can be inaccurate [31]–[33]. Requesting multiple labels or even querying a set of different oracles can severely complicate matters. The use case of mutation testing is thankfully deterministic and answered by a computer. There may be cases where a test's outcome is non-deterministic, e.g. race conditions or device-dependent errors. We did not consider cases in which the oracle-supplied label could be incorrect. In these cases the strength of predictive mutation testing is dependent on whether the underlying test suite is capable of catching them. Projects where such non-deterministic errors are particularly common should employ noisy oracle-tolerant query strategies.

Variable labeling costs questions the assumption that minimizing the number of requested labels minimizes actual labeling cost. A difficult-to-label example might provide more information but could also lead to more time spent producing said label. For software engineering, this could be the difference between a test failing quickly or by timeout. In this paper we minimized only the number of requested labels. There is evidence that relying on only the number of mutants during random sampling can distort estimates of the true runtime of mutation testing [34]. If some subset of tests in a suite is particularly slow, or the model is intended to predict particular

cells in a kill matrix rather than the kill status of a single mutant, then one should consider choosing some type of cost-aware query strategy [35], [36].

In summary, the basic AL-PMT scenario avoids most of the problems associated with successfully implementing active learning in practice. However, these common obstacles are to be kept in mind for codebases with some uncommon properties or when taking developer feedback on mutant usefulness into account.

VI. Related work

Code coverage often has low correlation with test suite effectiveness at detecting real faults [37]. Mutation testing has considerably better performance in this respect, and the gap between mutation score and coverage can also be useful information [38]. Mutants have been shown to be a good substitute for real faults [39], [40]. Mutation testing is seeing increased usage in industry practice [16], including safety-critical applications [14], and leading tech giants like Google [13], [15] and Meta [12].

Mutation testing has a broad and active academic landscape [41], [42]. Beyond just evaluating test suite effectiveness, mutation testing sees wider usage as a tool for automated program repair [43], [44], test oracle generation [45], [46] or fault localization [47], [48]. Reducing the computational cost of mutation testing is a frequent pursuit [3]. Researchers have proposed classification models to predict the outcomes of mutations rather than execute the test. We closely base our work on advancements in this field. [6], [7] propose feature-based models and [8] shows that the findings are transferable from Java to C. [19], [20] analyze the PMT dataset to find a strong class imbalance and a trivially predictable relation, and suggest ways of correcting for this. [10] builds a language-based approach to PMT, [9] does so by fine-tuning CodeBERT and [11] creates contrastive text pairs to encode mutations.

Industry studies of mutation testing [12]–[16] emphasize developer training and workflow changes for the successful integration of mutation testing. A low overhead and only showing useful mutants to developers are also strong requirements. This aligns with the measure of Test Completeness Advancement Probability advocated by [18]. The design of the AL-PMT scenario was strongly influenced by these industry observatios and priorities.

VII. Conclusion and future work

In this paper we have shown the efficacy of active learning on the predictive mutation testing problem. To our knowledge we are the first to apply active learning to predictive mutation testing.

The **main takeaways** are:

- AL-PMT provides high confidence in good performance despite using only a fraction of the training data of the original PMT model it is based on.
- AL-PMT solves a more challenging, online, single-project version of the predictive mutation testing setting. This setting is closer to practical industry application.
- AL-PMT is a good showcase of active learning in practice, as it avoids the common pitfalls of applied AL.

This reduction in data requirements, and the ability to observe and prioritize mutants through a custom query strategy, enable a much smoother integration into a practical software development workflow. Rather than having a monolithic pre-trained model with high training data requirements, or relying only on heuristics like the best-performing mutation operators, AL-PMT progressively updates and finds the most relevant mutants for a given project. Combined with some source of insight into which mutants developers find useful, this moves the PMT use case from providing a coverage metric quickly with some uncertainty, to actually guiding development and improvements in the test suite efficiently in both computational and developer time.

The addition of a query strategy to PMT offers a way to involve developer expertise in determining which mutants are most useful to execute the test suite on, and log to developers as a task item. While a user study was outside the scope of this paper, we do want to give direction to how AL-PMT could be implemented in a future study or an industry application. Implementing a user feedback parameter into AL-PMT is trivial, one just needs some measure, e.g. test completeness advancement probability [18] or a custom mutant-usefulness function [13], in a weighted sum with the chosen query strategy optimizer (e.g. entropy). Even simpler, the system could also treat "rated useful/not useful by a developer" as another hidden label to predict and therefore learn it from developer feedback or mutation tool usage over the course of the testing process.

The concepts behind active learning are not strictly restricted in applying to the models showcased in the original PMT paper. PMT's offline competitors may also offer insights when adapted into the online scenario and augmented with active learning. In the future we plan to investigate how much efficient prioritization of actually executed mutants can improve training times of PMT's other offline competitors. We also see potential in predicting not only the mutant kill status but the specifics of which test actually does so. With active learning's viability soundly proven, we also plan on performing an in-depth examination similar to [7]. There is a large range of other potential query strategies, feature engineering methods and additional data sets to test. We believe that AL-PMT has the potential to grow from a powerful data-efficient PMT alternative to state-of-the art in the field.

VIII. Acknowledgements

The author would like to thank Dr. Peter Reimann for his help during and after the rebuttal phase. The author would also like to thank Prof. Klemens Böhm and Prof. Ina Schaefer for their support and discussions in the early stages of this paper.

References

[1] T. T. Chekam, M. Papadakis, Y. Le Traon, and M. Harman, "An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption," in *2017*

IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 597–608.

[2] M. Ojdanic, A. Khanfir, A. Garg, R. Degiovanni, M. Papadakis, and Y. Le Traon, "On comparing mutation testing tools through learning-based mutant selection," in *2023 IEEE/ACM International Conference on Automation of Software Test (AST)*, 2023, pp. 35–46.

[3] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, and M. Ribeiro, "A systematic literature review of techniques and metrics to reduce the cost of mutation testing," *Journal of Systems and Software*, vol. 157, p. 110388, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301554

[4] L. Chen and L. Zhang, "Speeding up mutation testing via regression test selection: An extensive study," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 58–69.

[5] C. Brito, V. H. S. Durelli, R. S. Durelli, S. R. S. d. Souza, A. M. R. Vincenzi, and M. E. Delamaro, "A preliminary investigation into using machine learning algorithms to identify minimal and equivalent mutants," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 304–313.

[6] J. Zhang, Z. Wang, L. Zhang, D. Hao, L. Zang, S. Cheng, and L. Zhang, "Predictive mutation testing," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 342–353. [Online]. Available: https://doi.org/10.1145/2931037.2931038

[7] D. Mao, L. Chen, and L. Zhang, "An extensive study on cross-project predictive mutation testing," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, 2019, pp. 160–171.

[8] A. Duque-Torres, N. Doliashvili, D. Pfahl, and R. Ramler, "Predicting survived and killed mutants," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020, pp. 274–283.

[9] K. Jain, U. Alon, A. Groce, and C. Le Goues, "Contextual predictive mutation testing," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 250–261. [Online]. Available: https://doi.org/10.1145/3611643.3616289

[10] J. Kim, J. Jeon, S. Hong, and S. Yoo, "Predictive mutation analysis via the natural language channel in source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 4, Jul. 2022. [Online]. Available: https://doi.org/10.1145/3510417

[11] Y. Zhao, Y. Chen, Z. Sun, Q. Liang, G. Wang, and D. Hao, "Spotting code mutation for predictive mutation testing," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1133–1145. [Online]. Available: https://doi.org/10.1145/3691620.3695491

[12] M. Beller, C.-P. Wong, J. Bader, A. Scott, M. Machalica, S. Chandra, and E. Meijer, "What it would take to use mutation testing in industry—a study at facebook," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 268–277.

[13] G. Petrovic and M. Ivankovic, "State of mutation testing at google," in *Proceedings of the 40th International Conference on Software Engineering 2017 (SEIP)*, 2018.

[14] S. Vercacmmen, M. Borg, and S. Demeyer, "Validation of mutation testing in the safety critical industry through a pilot study," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2023, pp. 334–343.

[15] G. Petrovic, M. Ivankovic, B. Kurtz, P. Ammann, and R. Just, "An industrial application of mutation testing: Lessons, challenges, and research directions," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 47–53.

[16] J. Örgård, G. Gay, F. G. de Oliveira Neto, and K. Viggedal, "Mutation testing in continuous integration: An exploratory industrial case study," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2023, pp. 324–333.

[17] S. A. van Heijningen, T. Wiik, F. G. d. O. Neto, G. Gay, K. Viggedal, and D. Friberg, "Integrating mutation testing into developer workflow: An industrial case study," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2110–2120. [Online]. Available: https://doi.org/10.1145/3691620.3695273

[18] S. J. Kaufman, R. Featherman, J. Alvin, B. Kurtz, P. Ammann, and R. Just, "Prioritizing mutants to guide mutation testing," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1743–1754.

[19] S. Balderas-Díaz, G. Guerrero-Contreras, P. Delgado-Pérez, and I. Medina-Bulo, "An analysis of class imbalance challenges in predictive mutation testing," in *Intelligent Information and Database Systems*, N. T. Nguyen, T. Matsuo, F. L. Gaol, Y. Manolopoulos, H. Fujita, T.-P. Hong, and K. Wojtkiewicz, Eds. Singapore: Springer Nature Singapore, 2025, pp. 319–331.

[20] A. Aghamohammadi and S.-H. Mirian-Hosseinabadi, "An ensemble-based predictive mutation testing approach that considers impact of unreached mutants," *Software Testing, Verification and Reliability*, vol. 31, no. 7, p. e1784, 2021.

[21] B. Settles, "Active learning literature survey," 2009.

[22] ——, "From theories to queries: Active learning in practice," in *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, ser. Proceedings of Machine Learning Research, I. Guyon, G. Cawley, G. Dror, V. Lemaire, and A. Statnikov, Eds., vol. 16. Sardinia, Italy: PMLR, 16 May 2011, pp. 1–18. [Online]. Available: https://proceedings.mlr.press/v16/settles11a.html

[23] B. Settles, M. Craven, and L. Friedland, "Active learning with real annotation costs," in *Proceedings of the NIPS workshop on cost-sensitive learning*, vol. 1. Vancouver, CA:, 2008.

[24] R. King, K. Whelan, F. Jones, P. Reiser, C. Bryant, S. Muggleton, D. Kell, and S. Oliver, "Functional genomic hypothesis generation and experimentation by a robot scientist," *Nature*, vol. 427, pp. 247–252, 01 2004.

[25] P. Zhao and S. C. Hoi, "Cost-sensitive online active learning with application to malicious url detection," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 919–927. [Online]. Available: https://doi.org/10.1145/2487575.2487647

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] D. Kottke, M. Herde, T. P. Minh, A. Benz, P. Mergard, A. Roghman, C. Sandrock, and B. Sick, "scikit-activeml: A Library and Toolbox for Active Learning Algorithms," *Preprints*, 2021. [Online]. Available: https://github.com/scikit-activeml/scikit-activeml

[28] S. C. Hoi, R. Jin, J. Zhu, and M. R. Lyu, "Batch mode active learning and its application to medical image classification," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 417–424.

[29] B. Demir, C. Persello, and L. Bruzzone, "Batch-mode active-learning methods for the interactive classification of remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 3, pp. 1014–1031, 2010.

[30] Z. Wang and J. Ye, "Querying discriminative and representative samples for batch mode active learning," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 9, no. 3, pp. 1–23, 2015.

[31] W. Wu, Y. Liu, M. Guo, C. Wang, and X. Liu, "A probabilistic model of active learning with multiple noisy oracles," *Neurocomputing*, vol. 118, pp. 253–262, 2013.

[32] P. Donmez and J. G. Carbonell, "Proactive learning: cost-sensitive active learning with multiple imperfect oracles," in *Proceedings of the 17th ACM conference on Information and knowledge management*, 2008, pp. 619–628.

[33] C. Zhang and K. Chaudhuri, "Active learning from weak and strong labelers," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[34] G. Guizzo, F. Sarro, and M. Harman, "Cost measures matter for mutation testing study validity," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1127–1139. [Online]. Available: https://doi.org/10.1145/3368089.3409742

[35] K. Tomanek and U. Hahn, "A comparison of models for cost-sensitive active learning," in *Coling 2010: Posters*, 2010, pp. 1247–1255.

[36] A. Krishnamurthy, A. Agarwal, T.-K. Huang, H. D. III, and J. Langford, "Active learning for cost-sensitive classification," *Journal of Machine Learning Research*, vol. 20, no. 65, pp. 1–50, 2019. [Online]. Available: http://jmlr.org/papers/v20/17-681.html

[37] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 435–445.

[38] K. Jain, G. T. Kalburgi, C. Le Goues, and A. Groce, "Mind the gap: The difference between coverage and mutation score can guide testing efforts," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 102–113.

[39] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[40] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 654–665. [Online]. Available: https://doi.org/10.1145/2635868.2635929

[41] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Chapter six - mutation testing advances: An analysis and survey," ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2019, vol. 112, pp. 275–378. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245818300305

[42] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[43] A. Ghanbari, S. Benton, and L. Zhang, "Practical program repair via bytecode mutation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–30. [Online]. Available: https://doi.org/10.1145/3293882.3330559

[44] V. Debroy and W. E. Wong, "Using mutation to automatically suggest fixes for faulty programs," in *2010 Third International Conference on Software Testing, Verification and Validation*, 2010, pp. 65–74.

[45] G. Fraser and A. Zeller, "Mutation-driven generation of unit tests and oracles," in *Proceedings of the 19th international symposium on Software testing and analysis*, 2010, pp. 147–158.

[46] M. Staats, G. Gay, and M. P. E. Heimdahl, "Automated oracle creation support, or: How i learned to stop worrying about fault propagation and love mutation testing," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 870–880.

[47] L. Zhang, L. Zhang, and S. Khurshid, "Injecting mechanical faults to localize developer faults for evolving software," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 765–784. [Online]. Available: https://doi.org/10.1145/2509136.2509551

[48] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 169–180. [Online]. Available: https://doi.org/10.1145/3293882.3330574