Exploring Strategies for Guiding Symbolic Analysis with Machine Learning Prediction

Mingyue Yang, David Lie, Nicolas Papernot
Department of Computer Engineering
University of Toronto & Vector Institute
Toronto, Canada
Email: myshirley.yang@mail.utoronto.ca, {david.lie,nicolas.papernot}@utoronto.ca

***Abstract*—To improve the scalability of symbolic analysis tools, one observation is that analysis resources are wasted on analyzing unsatisfiable paths, which are not possible in reality. While existing works attempt to predict the satisfiability of a program path without spending resources to analyze it, the performance of these predictor models are far from perfect. In this work, we attempt to understand how model predictions, even if imperfect, can be most effectively used to reduce the time required to analyze satisfiable paths. This work studies the sometimes complex interactions between model performance, analysis domain properties such as the distribution of path analysis costs and distribution of satisfiable paths, the design of symbolic analysis tools being used, and the algorithm used to prioritize and select paths for analysis. Using a novel simulation methodology, we study this problem and find that a number of factors can have as large an effect on symbolic analysis performance as improved predictors. Finally, we conclude with a couple of observations about how to best integrate machine learning prediction into symbolic analysis.**

I. Introduction

Symbolic analysis is widely used in areas such as software verification, vulnerability finding and detecting privacy leakage [4], [11], [6], [20]. In the broadest sense, symbolic analysis works by executing a program (or some abstraction of it) with symbolic inputs (as opposed to concrete program inputs), and tracking the state of the program symbolically along the execution paths. A set of constraints can then be derived from the state and solved to generate concrete inputs that would execute the same path on the real program. However, real world applications may have a large number of paths, and thus require a significant amount of time and computing resources to perform symbolic analysis. Resources are wasted on collecting and solving constraints that cannot be satisfied by any concrete input. These paths are unsatisfiable and cannot be triggered in the real world. Ideally, one would avoid spending resources on unsatisfiable paths.

Unfortunately, there is no way to determine whether a path is satisfiable or not without collecting and solving its constraints. As a mitigation, a number of existing works use machine learning to predict which paths will be satisfiable [15], [10], [19], [16], [14]. However, these models either do not have high performance or are not generally applicable to all path constraints, which limit their effectiveness.

As it has proven difficult to improve the accuracy of predictions from these models, we take a different approach, which is to study the trade-offs of how machine learning models can be incorporated into a symbolic analysis tool across a range of model prediction performance. Specifically, we focus on factors that can influence how to integrate a predictor into symbolic analysis, such as the performance of the predictor, the design of the symbolic analysis tool, and the distribution of satisfiable versus unsatisfiable paths in analyzed applications. **Our research goal is to broadly answer how these factors can interact to improve path analysis throughput for a symbolic analysis tool**.

In this work, we evaluate strategies with two different domains and their corresponding tools: TIRO for Android applications [17] and Mythril for smart contracts [1]. We also develop a novel simulation tool that allows us to easily study the trade-offs between how predictions are applied to prioritize paths for analysis. In summary, our paper makes the following contributions:

1) We study and clarify the trade-offs and lessons on applying machine learning prediction for path analysis in symbolic analysis tools. We aim for results not specific to properties of the tools themselves or the application domains the tools are used to analyze, but the general situations they represent.
2) We design and use a novel symbolic execution simulation tool to quickly prototype and derive different symblic analysis strategies. We are not aware of any other work using simulation to study the performance of symbolic analysis tools.

II. Background

A. Background for Symbolic Analysis

Symbolic analysis explores a given program by assuming symbolic program inputs instead of using actual input values. It aims to determine what input is required to execute paths in the program. A ***path*** starts from an entry point of program execution, and ends at some point inside the program. We broadly consider symbolic analyses as having two iterative phases: ***path finding*** and ***path analysis***. During path finding, symbolic analysis tools discover paths in the program using various methods such as static analysis, concrete execution or abstract interpretation. During path analysis, the tool extracts

and solves the path constraint allowing it to determine an input, which if it exists, will cause the path to be executed. Optionally, between path finding and path analysis, tools can implement a ***path selection*** phase. This phase selects a subset of paths according to some criteria for analysis. Satisfiability prediction naturally fits into such path selection phases: for example, paths predicted as more likely to be satisfiable can be prioritized for path analysis.

B. Baseline Symbolic Analysis Tools

We assume the use of supervised machine learning to train a predictor for path satisfiability. We thus need a training set of labeled paths from a distribution that can be reasonably learned. As such, we assume that such a distribution would arise in domains where a large number of programs use a common set of APIs, and hence paths from such programs would perform similar operations before and after calling such APIs. As a result, in this work we use two such domains and their corresponding tools for study: TIRO, which is a tool designed to perform symbolic analysis on Android applications [17] and Mythril, which is designed for smart contracts [1]. Moreover, these tools differ in their design, which also allows us to explore how machine learning prediction can be incorporated into different symbolic analysis designs.

1) ***Path Finding and Path Analysis****:* TIRO and Mythril have vastly different path finding and analysis designs. TIRO is a *targeted* symbolic analysis tool, which means that it targets paths whose end points meet a certain criterion. It does this by indiscriminantly finding paths during its path analysis phase, but only selecting those that meet its targeting criteria during path selection (TIRO specifically targets locations performing de-obfuscation). As such, the paths analyzed by TIRO are ***independent***—while a path may have constraints that are a subconstraint of another path, TIRO does not explicitly track this and is not aware of such dependencies if they exist.

In contrast, Mythril is not targeted and tries to exhaustively explore every path in a program, making it more similar to other conventional symbolic execution tools like KLEE [4]. Thus, Mythril switches from path finding to path analysis whenever it hits a branch, so that it can check for the satisfiability of all successor paths of the branch, and solve for an input that can reach each successor path. Mythril then iteratively searches for all paths that include the satisfiable successor paths. This results in ***predecessor-successor relationships*** between paths, as each discovered path has constraints that are a subconstraint of a previously found path, and thus paths in Mythril are ***dependent*** on one another.

2) ***Path Analysis Costs****:* There are some differences and similiarities in path analysis time of TIRO and Mythril. To collect data, we run TIRO on 2,000 of the most popular Google Playstore apps downloaded in 2019, and discard apps with fewer than 100 paths and 1 hour of path analysis time. This results in 997 apps for TIRO. Also, we run Mythril on 3,845 smart contracts [21] with unique code hashes and names. Overall, we collect 4,243,189 paths for TIRO, and 78,977,920 paths for Mythril. As shown in Table I, unsatisfiable paths take shorter than average time to analyze for TIRO, while for Mythril, unsatisfiable path analysis takes longer than average.

TABLE I: Path Processing Cost for TIRO and Mythril

Scenario	mean (ms)	std (ms)
TIRO Path Finding	33	514
TIRO Path Analysis	9,436	43,378
TIRO Unsatisfiable Path Analysis	1,447	21,160
Mythril Path Finding	6.4	462
Mythril Path Analysis	889	1658
Mythril Unsatisfiable Path Analysis	1656	3239

TABLE II: Distribution of Paths with Different Analysis Time

TIRO			Mythril		
Analysis Time	Num Paths		Analysis Time	Num Paths	
	Sat : Unsat	Total Paths		Sat : Unsat	Total Paths
<10ms	1.00	59,768 (1.41%)	<10ms	10.8	1,417,117 (31.5%)
10ms-100ms	0.0344	745,541 (17.6%)	10ms-100ms	28.7	717,611 (16.0%)
100ms-1s	0.0815	1,501,044 (35.4%)	100ms-1s	12.9	1,759,171 (39.1%)
1s-10s	1.35	1,209,168 (28.5%)	1s-10s	9.2	575,545 (12.8%)
10s-100s	18.6	658,899 (15.5%)	>10s	0.0499	27,542 (0.613%)
>100s	15.4	68,769 (1.62%)	N/A		

However, for both tools, path analysis, regardless of satisfiability, is generally several magnitudes more expensive than path finding. This means for both tools, it is potentially beneficial to avoid analyzing unsatisfiable paths, and use the saved time to find and analyze other paths that are more likely to be satisfiable.

3) ***Distribution of Satisfiable Paths****:* Because TIRO is targeted and Mythril is not, there are differences in the distribution of paths they find and analyze. Table II shows the distribution of paths, as well as the split between satisfiable and unsatisfiable paths, by analysis time (i.e. cost). We note that in general, the path analysis time is dominated by constraint analysis, whose running time grows with the number of constraints, and is roughly proportional to the length of a path. Because Mythril is not targeted, and it must find and analyze shorter predecessor paths before it can find and analyze the longer successor paths, Mythril's path distribution generally has more shorter paths than TIRO, which only targets paths that end in a point in the code that may be performing de-obfuscation. In addition, since Mythril only finds and explores paths that have a satisfiable predecessor, most of the short paths in Mythril tend to be satisfiable, with a sat:unsat ratio of 10.8 for paths that take less than 10ms to analyze, while most paths that run more than 10s are unsatisfiable. In contrast, TIRO has a more normal distribution of path lengths and analysis time, with the majority paths having analysis times between 100ms-10s. In addition, shorter running paths between 10ms-1s are mostly unsatisfiable, while paths that take more than 10s are mostly satisfiable.

III. Methodology

A. Symbolic Execution Simulator

To enable faster prototyping and controlled exploration of the design space, we build a symbolic execution simulator that estimates the time required to analyze a program. Because the running time of symbolic analysis tools is often dominated by the number of paths and the time to analyze the paths (i.e. extract and solve constraints), we expect that our simulator

can provide reliable predictions about the relative execution times when the order and prioritization of paths selected for analysis by our predictor changes. To build this simulator, we begin by running the underlying symbolic analysis tool without simulation to collect a corpus of paths found, along with the satisfiability and time to analyze each path.

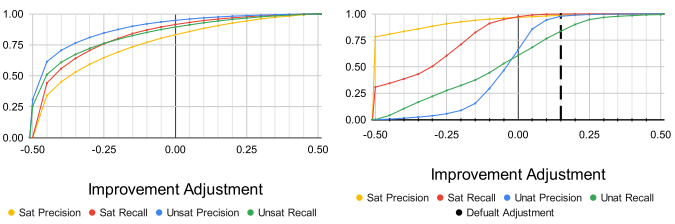
Our analysis is performed on a corpus of paths from programs in Section II-B2. For TIRO, we use 997 apps previously mentioned. For Mythril, we use 2,229 out of the 3,845 unique contracts due to time limitations [21]. We use an overall memory limit and timeout limit for the analysis, as well as a timeout for every path. For TIRO, we use 4 threads, 240GB of memory, a 5-minute timeout per path and a 24-hour overall timeout. For Mythril, we use 8GB of memory, a 10s timeout per path and a 2-hour overall timeout. Paths that exceed the memory limit are excluded. In TIRO, paths that exceed any timeout are excluded, while in Mythril, paths that exceed any timeout are marked as unsatisfiable. For paths whose satisfiability cannot be determined, we still mark them as satisfiable, because these paths and their successors are potentially worth exploring. Furthermore, we estimate costs such as path feature extraction, model prediction and time for saving/restoring analysis states by implementing these components and timing their execution. We acknowlege bias that may be introduced in these data collection methods.

Some assumptions from our simulation may not be satisfied by all symbolic analysis techniques. For example, we assume the analysis time for each path is fixed, and techniques such as caching constraints for reuse [18], [13] and incremental constraint solving [7] are out of our scope. Also, the default implementation of Mythril performs analysis at every instruction instead of every branch. However, we collect paths that end at branch and return instructions instead of at any executed instruction, as the constraints required to reach instructions in the same basic block usually have the same satisfiability: we currently do not consider exceptions within basic blocks.

B. Simulating Improved Prediction Accuracy

We anticipate that advances in representations for program analysis tasks may result in improved machine learning predictions in the future. Since we know the ground-truth satisfiability of each path, we can thus simulate and evaluate the performance of different symbolic analysis designs for increased and decreased model performance. However, models typically do not have uniform performance across all paths as some paths are easier to predict than others. For example, previous work has shown that machine learning predictors can have different accuracies across satisfiable and unsatisfiable paths, as well as across path analysis times [19].

We thus train baseline random forest models to predict path satisfiability, and then simulate a predictor of a different performance by applying an ***improvement adjustment*** to the model's output satisfiability likelihood: if a path is actually satisfiable, then we add the improvement adjustment, and otherwise we subtract the improvement adjustment if the path is unsatisfiable. We then clamp the resulting value to between



(a) TIRO (b) Mythril

Fig. 1: Adjusted Model Performance

0 and 1. Consequently, a positive improvement adjustment makes the model more accurate, while a negative improvement adjustment makes it less accurate. In the extreme case, an improvement adjustment greater than 0.5 results in the model always returning correct predictions, while an improvement adjustment less than -0.5 results in a model that always gives incorrect predictions.

C. Model Performance

As Figure 1 shows, increasing improvement adjustment increases the model performance for both datasets. For Mythril, as the model performance with 0 improvement adjustment is low, we use a default improvement adjustment of 0.15 to get decent model performance in Sections V-A and V-B.

IV. Design Space Exploration

A. Overview

A naïve approach for using a predictor is to use it to filter out and discard paths that are predicted as unsatisfiable. However, mispredictions from the predictor can cause symbolic execution to miss many satisfiable paths. For example, this can cause missing malicious behaviors for TIRO and undetected vulnerabilities for Mythril. Particularly, when there are dependencies among paths, missing a parent path would cause symbolic analysis to miss all child paths initiating from it, resulting in a large number of paths missed. For example, while our predictor on Mythril has 97.3% accuracy for satisfiable paths across applications, discarding all paths predicted as unsatisfiable will cause it to miss 57.7% of paths averaged over all applications. This is because wrongly discarding a path not only misses that path but also all the successor paths that depend on the discarded path.

Therefore, instead of discarding unsatisfiable paths, we ***prioritize*** the analysis of certain paths using output satisfiable likelihood from the predictor. This means no satisfiable path is missed. With the prioritization technique used, we aim to save time required to analyze some number of satisfiable paths.

B. Design Criteria

We define the following criteria that outlines the dimensions along which we explore the design space.

1) ***Types of Symbolic Analysis****:* As shown in Section II-B1, TIRO and Mythril are different types of symbolic analysis tools. TIRO targets specific paths that lead to given API calls, while Mythril explores all paths in a given program. These two tools have different path finding, path analysis and path

dependency relationships. We evaluate both TIRO and Mythril, for the two types of symbolic analysis designs they represent.

*2) **Constructing Path Set**:* The path selection phase should have access to a set of paths that the predictor assigns priorities to. We call this set the ***path set***. To have this set of paths to choose from, our simulation experiments do not analyze a path as soon as it is found, but insert found paths into a path set, and paths in the path set are ranked using outputs from the predictor. We only select the top ranking path from the path set for analysis.

We formally define a ***dependent path set*** as a path set where paths that are known to share constraints (i.e. constraint of one path is the subconstraint for another path in the set). In contrast, an ***independent path set*** is one where paths that are not known to share constraints. As discussed in Section II-B, TIRO generates an independent path set: because TIRO only finds paths that reach particular targets and no known dependency exists between found paths, it is not possible for TIRO to generate a dependent path set. On the other hand, while Mythril by default generates a dependent path set since every path found has a predecessor with its subconstraint, we can simulate a version of Mythril that produces an independent paths set. This is done by combining path finding with path selection: we implement a path finding algorithm that only explores the successors of a path when the path is selected from path set for analysis. Since paths selected are removed from the path set, this guarantees that no predecessor and successor can present in the path set at the same time, making the path set independent.

*3) **Path Set Size Limit**:* We can set a path set size limit for an independent path set with TIRO or a dependent path set with Mythril. With this limit, we only select paths for analysis when the path set size hits the limit. With a simple path selection strategy that selects the path predicted as most likely satisfiable, generally one may expect larger path set sizes to have better performance: larger path set size allows path selection to have more candidate paths to choose from. For more complex ranking objectives and path selection strategies, however, path set size can have subtle effects as discussed in Section V-A3.

In contrast, we cannot set a path set size limit for Mythril's independent path set. Recall that to enforce path independence, a path's successors cannot be added to the path set unless its predecessor is selected for analysis, thus removing it from the path set. This is in conflict with the need to find enough paths to reach the size limit before selecting a path for analysis. Thus, it is not possible to enforce a path set size limit for Mythril when it is configured to use an independent path set – the path set size limit can only work for Mythril on a dependent path set.

*4) **Path Pruning**:* We observe that path dependency can be used to infer satisfiability. If a path is satisfiable, then all its predecessor paths are all satisfiable, as otherwise no input can reach the satisfiable path. Similarly, if a path is unsatisfiable, none of its successor paths can be reached and these paths are thus all unsatisfiable. Therefore, we can omit the analysis for predecessors of satisfiable paths and successors of unsatisfiable paths, because their satisfiability is already determined. In our simulator, as soon as the analysis of a path is finished, based on its satisfiability, we can prune all predecessors/successors of the path from the path set, and the pruned paths will no longer be selected or analyzed.

Thus, path pruning can be applied on a ***dependent path set***. However, it cannot be applied on independent path sets, as there exists no dependency among paths in the set.

*5) **Ranking Objective**:* We propose the following ranking objectives for selecting paths from the path set. These different objectives can be used under different scenarios.

Sat Ranking: An obvious ranking objective is to prioritize paths with higher satisfiable likelihood, so paths that are more likely to be satisfiable are analyzed earlier.

Unsat Ranking: Path pruning, however, favours analyzing unsatisfiable paths earlier, as this can help prune unsatisfiable successor paths earlier. Path pruning also favours analyzing satisfiable paths later, so when longer satisfiable paths are analyzed, analysis cost for shorter satisfiable paths can be saved as they are pruned. Thus, with unsat ranking, we prioritize paths with a higher unsatisfiable likelihood: $1 - sat\_likelihood$.

*6) **Filtering Paths for Immediate Analysis**:* We can add an analysis filter before path selection, to allow immediate analysis of paths that meet certain requirements, before sending them to path selection. The analysis of these path have higher priority than all paths in the path set.

Immediate Timeout Filter: For cases when there are a large number of short-running satisfiable paths, as soon as a path is found, we can analyze it for a period of time. If the path analysis finishes within a timeout, there is no need to proceed to path selection as the path analysis is completed. If the analysis does not finish within the timeout, we save the analysis state for this path, and add this path to the path set for path selection. We name this timeout ***immediate timeout filter***. Later, when the path is selected for analysis, we restore its analysis state to analyze it again.

If a path does not finish analysis within the immediate timeout filter, its analysis time is also counted towards the total analysis time. Moreover, when using Mythril's independent path set, as soon as path analysis completes within the immediate timeout filter, we also inform path finding to explore successor paths.

Prediction Threshold Filter: We can also allow immediate analysis of a path, if it is highly likely to be satisfiable. We can set a value for the ***prediction threshold filter***, and analyze a path before path selection if its satisfiable likelihood is above the prediction threshold filter. Paths that are not analyzed immediately are sent to path selection for analysis later.

*7) **Model Performance**:* We study the effect of model performance on symbolic analysis by increasing or decreasing the improvement adjustment for model output, as described in Section III-B.

C. Metric

In general, symbolic analysis is limited by computing resources. As a result, users of symblic analysis generally implement a time limit on the analysis tool. We thus use the percentage of time saved at different points of the analysis as a metric to evaluate the effectiveness of different design points. Specifically, we use the amount of time required to analyze a certain % of satisfiable paths as compared to the run of the ***baseline*** implementation of the tool used to collect data for the simulator. By evaluating the time savings at different points, we can evaluate if a design confers more savings earlier or later in the analysis.

As the analysis of all paths in each application proceeds, we take a checkpoint as every 5% of the satisfiable paths in the application is analyzed until all paths have been analyzed. At each checkpoint, we measure the time savings compared to the baseline. We then average the time savings across all applications to compute an average saving at every checkpoint. This approach treats applications with many paths and few paths equally, so the result will not be dominated by applications that have a large number of paths.

V. Evaluation

In this section, we evaluate the performance of different path selection strategies. We explore options in our design space as described in Section IV. As mentioned in Section III-C, we use a ***default*** model improvement adjustment of 0.15 in Sections V-A and V-B. We do not adjust the accuracy of TIRO's baseline model.

Also, our metrics take checkpoints at every 5% of satisfiable paths analyzed. However, our final checkpoint is taken not when 100% of satisfiable paths are analyzed, but when the satisfiability of all paths are determined, including all satisfiable and unsatisfiable paths in the analyzed application.

Our later evaluation sections answer the following research questions:

- **RQ1**: For each symbolic analysis tool, what are the best ranking objectives and path sets for using a predictor?
- **RQ2**: How can analysis filters be leveraged to obtain more time savings?
- **RQ3**: How does performance of symbolic analysis change with the performance of the predictor?

A. RQ1: Path Set Evaluation

In this section, we simulate our two symbolic analysis tools with different types of path sets and different path set size limits. We evaluate the basic sat ranking and unsat ranking objectives. The option to use path pruning for dependent path set is also examined.

For these cases, we sweep the path set size limit to see the effect on time savings to analyze the same percentage of satisfiable paths. From this, we can limit later experiments to a single or a small range of path set sizes. In this section, the ***baseline*** run for time saving measurements is the original symbolic analysis without any path set.

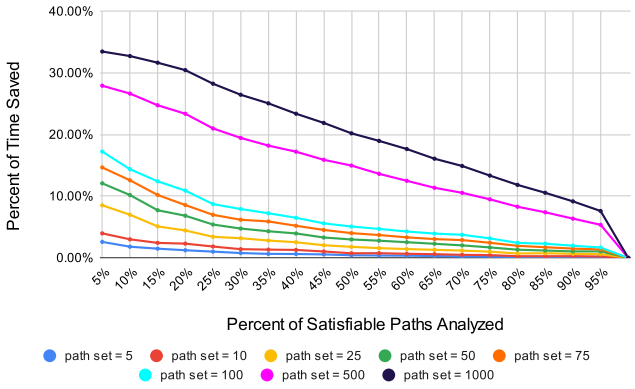


Fig. 2: Time Savings for TIRO's Independent Path Set, Sat Ranking (Baseline: Vanilla)

1) ***TIRO + Independent Path Set****:* We evaluate the time savings for TIRO's independent path set with different path set size limits. Sat ranking is used with this path set, as it prioritizes satisfiable paths. As mentioned, our baseline run is from the original symbolic analysis tool without any path set ranking. We denote this baseline as ***Vanilla*** in our figures.

Figure 2 shows that a larger path set size increases the time savings at all analysis checkpoints. This is because a larger path set allows the symbolic analysis to have more candidate paths to select from and thus prioritizes paths that are more likely to be satisfiable.

2) ***Mythril + Independent Path Set****:* We evaluate Mythril's independent path set with sat ranking. We do not apply path pruning as it is not necessary for independent path sets (discussed in Section IV-B4). Consequently, we use the sat ranking objective and omit unsat ranking, which is beneficial only with path pruning. Also, as shown in Section IV-B2, there is no option to adjust path set size limit for Mythril's independent path set, because path finding is impacted by path selection. Therefore, there is no parameter tuning required when using Mythril's independent path set on its own.

The time savings for Mythril's independent path set are shown as the no-filter option in Figure 4. Mythril's independent path set has a slow down at the earlier stage of the symbolic analysis, but achieves slight time savings after the 50% checkpoint. At the 95% analysis checkpoint, it can save about 3.4% of time.

The reason is that in Mythril, paths visited at early stage of the analysis are shallow and have small analysis cost. The extra cost for finding deeper paths and maintaining path set thus becomes large compared to saved time for path analysis. For example, at the 5% checkpoint, maintaining a path set takes up 5.12% of time, while this overhead is 0 for the baseline. Also, as the path set keeps a number of candidate paths, the time required to find these candidate paths also adds up. Thus, although Mythril's independent path set omits analysis of about 1/3 encountered unsatisfiable paths at the 5% checkpoint, it still takes longer. However, this effect inverts later on in the analysis since the paths analyzed later tend to be longer and have higher analysis cost. At this stage, path

analysis cost overwhelms cost for path finding and maintaining path set. This results in a speedup for Mythril's independent path set at later stage of the analysis.

This means Mythril's independent path set on its own saves time when larger analysis time is used, and in particular, when the user is willing to spend enough computing resources to analyze at least 50% of satisfiable paths. It only benefits users who are willing to spend the time to do analyses close to completion for each application instead of only a shallow analysis. Note Mythril's independent path set does not provide an overall time saving, as it reorders the paths for analysis rather than omitting to analyze any of them.

3) ***Mythril + Dependent Path Set****:* For Mythril's dependent path set, we study the effect of sat/unsat ranking objectives and path pruning.

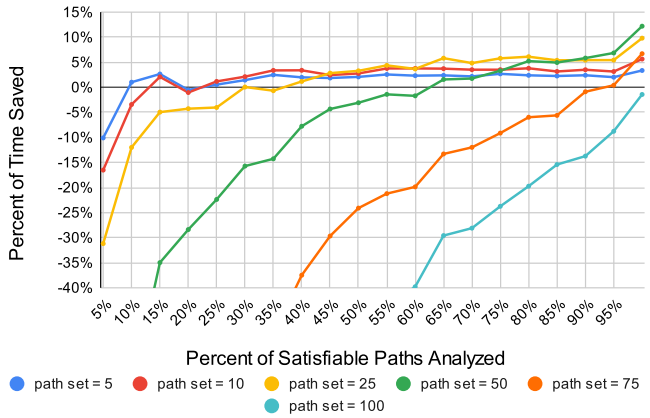
Effect of Path Pruning: Path pruning plays a crucial role in Mythril's dependent path set, as it avoids analysis cost of satisfiable predecessors and unsatisfiable sucessors. In our experiments, the analysis cost becomes several times greater without path pruning. This means path pruning should be applied along with dependent path set to save time.

Sat Vs. Unsat Ranking + Path Pruning: The option to use sat/unsat ranking objective is evaluated with Mythril's dependent path set. Path pruning is turned on as it can provide savings. We compare the sat ranking and unsat ranking objectives in Figure 3. For ease of demonstration, we limit the minimum time savings in Figure 3 to -40% (actually a slowdown), and clip negative values below this number.

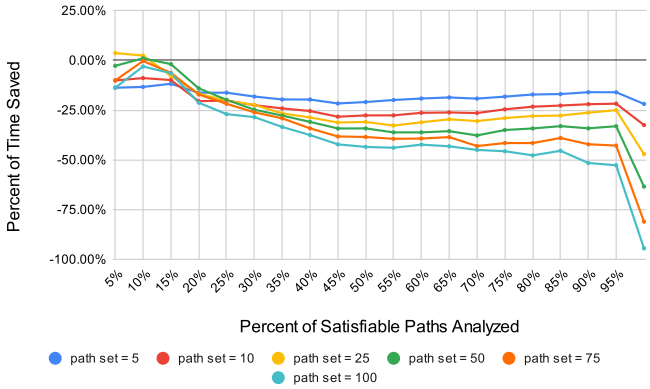
With path pruning, the unsat ranking performs much better than sat ranking. The unsat ranking option in Figure 3a can achieve time savings at several checkpoints and overall, while sat ranking in Figure 3b only adds more analysis time overhead compared to the baseline for almost all checkpoints. While this sounds counterintuitive, this is because path pruning saves analysis time for satisfiable predecessors and unsatisfiable successors. As discussed in Section IV-B4 and Section IV-B5, delaying the analysis of satisfiable path is desirable as it provides more possibilities for saving, in case their satisfiable successor is analyzed. Also, analyzing unsatisfiable paths earlier can provide savings by omitting the finding/analysis of their unsatisfiable successors in future for Figure 3a. This also explains why sat ranking should not be used: it causes unsatisfiable paths and their successors to stay in the path set for longer, and savings from satisfiable paths to be eliminated as they are picked early, resulting in an overall slow down.

In addition, both ranking objectives have startup overhead at early analysis stage. One explanation is the same reason for slow down in the early stages of Mythril + independent path set: saved path analysis cost is not as significant compared to added path finding cost and strategy overhead.

Also, Figure 3a shows when the path set size limit increases from 5 to 50 for unsat ranking, startup overhead and time savings overall both increase. This is because when the path set becomes large, more satisfiable paths are left in the path set not analyzed. At the early analysis stage, as unsatisfiable paths



(a) Unsat Ranking + Path Pruning



(b) Sat Ranking + Path Pruning

Fig. 3: Time Savings for Mythril's Dependent Path Set + Path Pruning (Improve Adj = 0.15, Baseline: Vanilla)

are prioritized for analysis, it requires more time to analyze the same number of satisfiable paths, resulting in a slow down. In contrast, as analysis proceeds, these satisfiable paths left in the path set are pruned once their satisfiable successors are analyzed, causing time savings overall and in the later stage of the analysis. However, as the path set limit further increases to 75 and 100, more unsatisfiable paths are added to the path set. As there are more candidate unsatisfiable paths in the path set, it is less likely for individual unsatisfiable paths to be selected. As these unsatisfiable paths keep staying in the path set without being selected, their unsatisfiable successors are visited/analyzed without being pruned. This thus causes a slowdown for almost all analysis checkpoints.

Therefore, Mythril's dependent path set benefits from path pruning and unsat ranking. With this combination, it is desirable to keep the path set size limit below 50. Within this range, a larger path set size is desirable to analyze more paths, while a smaller path set size is better when fewer paths are to be analyzed. While the improvement from Mythril's dependent path set on its own is not much, it can be combined with analysis filters to achieve better savings as shown later.

B. RQ2: Immediate Analysis Filter

We study the effect of immediate analysis filters and prediction threshold filter on different combination of symbolic analysis tools and path sets. To measure time savings, we use the vanilla ***baseline*** run, which is the original analysis tool without any path set ranking, same as in Section V-A.

1) ***TIRO + Independent Path Set****:* Our experiment shows the timeout filter is overall not helpful to TIRO's independent path set. As shown in Section II-B3, for TIRO, there are not a big proportion of the satisfiable paths under 1s: indeed, paths between 10ms-1s are mostly unsatisfiable. Thus, using a small timeout filter below 1s prioritizes unsatisfiable paths, and causes significant slowdown. A larger timeout filter is also not as helpful: because most paths regardless of satisfiability can finish within the time filter, the ability for the path set to prioritize analysis of satisfiable paths is eliminated.

Also, we find the prediction threshold filter does not provide extra time savings for TIRO's independent path set. This is because this filter allows immediate analysis of all paths above the prediction threshold, where as without the filter, only the path with highest predicted satisfiable likelihood is selected for analysis. Therefore, compared to the no-filter option, the prediction threshold filter causes TIRO to select more unsatisfiable paths for immediate analysis, and thus does not save time.

2) ***Mythril + Independent Path Set****:* We now analyze the effect of analysis filters on Mythril's independent path set. We find the immediate timeout filter to be more effective than the prediction threshold filter for Mythril.

Immediate Timeout Filter: For Mythril's independent path set, a well-tuned timeout filter value can bring in time savings. As shown in Figure 4, a small timeout filter between 50ms to 1s provides good time savings. This immediate timeout filter is effective because for Mythril, paths that finish within 1s are mostly satisfiable. Because the timeout filter allows short-running satisfiable paths to be analyzed first before long-running satisfiable paths, it requires less time to analyze the same number of satisfiable paths. As the time filter increases from 10ms, more short running satisfiable paths can pass the time filter to be analyzed immediately and save time. However, for paths that do not finish within the timeout, the time filter thresholds spent to analyze these paths are also counted towards the required analysis cost.

There is thus a tradeoff between having a large and small timeout filter. For Mythril, the optimal timeout filter is at 250ms, where it has 11% of time savings at the 5% checkpoint, and around 5% of time savings before the 95% checkpoint.

Prediction Threshold Filter: We find time savings from even a well-tuned prediction threshold filter to be minor. A prediction threshold value of 0.5-0.6 can reduce the startup overhead at early analysis stage, as described in Section V-A2, where path analysis cost is small for shallow paths. This is because the satisfiability predictor is blind to path analysis cost. The most likely satisfiable path picked from the path set is not always the cheapest. Without the prediction filter, when a

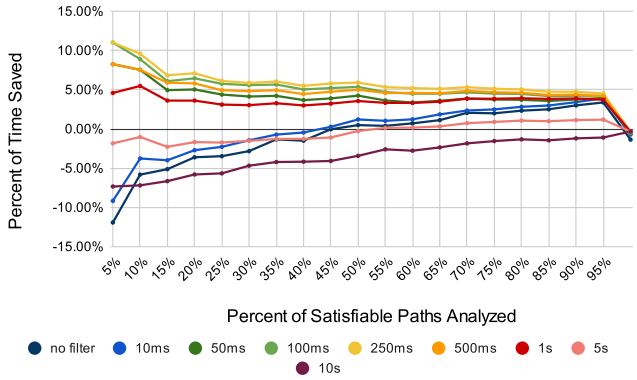


Fig. 4: Effect of Immediate Timeout Filter on Mythril's Independent Path Set, Sat Ranking (Improve Adj= 0.15, Baseline: Vanilla)

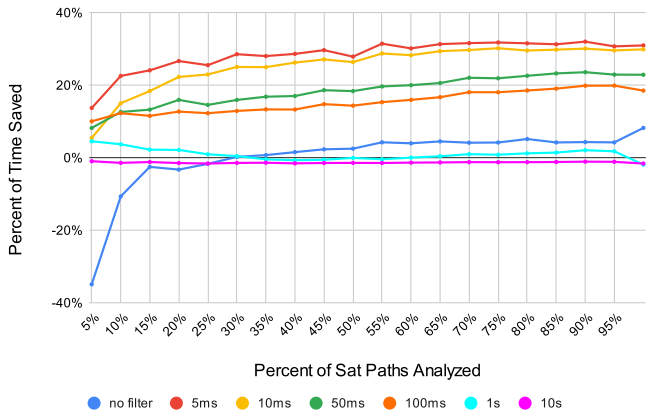
path and its deeper satisfiable successors are predicted as more likely satisfiable than other shallow satisfiable paths in the path set, symbolic analysis keeps exploring and analyzing these deep paths and their successors. As deeper paths generally take longer to analyze, this results in more computation overhead at early analysis stage. In contrast, the prediction filter saves time by allowing immediate analysis of all paths above the prediction threshold, instead of focusing analysis on a number of deep satisfiable paths that require more analysis time.

However, a prediction threshold smaller than 0.3 causes overall slowdown, as only very few satisfiable paths with satisfiable likelihood smaller than this value can be prioritized by the path set. A prediction threshold greater than 0.7 causes the performance of the filter to approach the no-filter option: as the small number of paths with predicted satisfiable likelihood greater than the threshold are very similar to the ones prioritized by the path set using satisfiable likelihood.

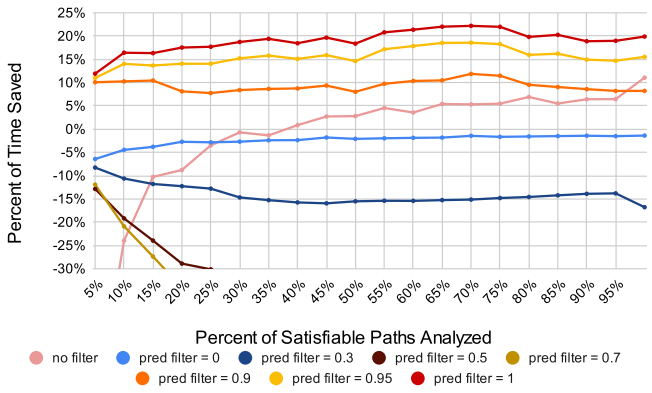
On Mythril's independent path set + prediction threshold filter, the optimal saving at the prediction threshold of 0.6 is minor compared to the immediate timeout filter. However, its decreased startup overhead demonstrates possible savings from taking path analysis time into account, which is a direction for future work.

3) ***Mythril + Dependent Path Set****:* For Mythril's dependent path set, both the immediate timeout filter and the prediction threshold filter have opportunities to provide a good amount of time savings.

Immediate Timeout Filter: For the immediate timeout filter, our experiments use a path set size of 20 for Mythril's dependent path set, as it has a good performance. In Figure 5a, small timeout filter values from 5ms-100ms provide time savings for all analysis checkpoints. The optimal 5ms timeout filter improves the overall time saving from around 8% to over 31%. This is because, as mentioned in Section V-B2, short-running paths in Mythril are likely to be satisfiable. With path pruning, the timeout filter allows opportunities to finish analysis of a short-running satisfiable path and prune all its satisfiable parent paths in the path set that can take more analysis time. Also, prioritizing short-running satisfiable paths



(a) Immediate Timeout Filter, Path Set Size =20



(b) Prediction Threshold Filter, Path Set Size = 30

Fig. 5: Effect of Analysis Filters on Mythril's Dependent Path Set, Unsat Ranking, Path Pruning (Improve Adj = 0.15, Baseline: Vanilla)

itself is favorable, as it requires less time to analyze the same number of satisfiable paths.

Thus, for Mythril's dependent path set, an immediate timeout filter with small timeout can achieve good time savings.

Prediction Threshold Filter: As shown in Figure 5b, a large prediction threshold above 0.9 brings significantly more time savings for Mythril's dependent path set. This is because a high prediction threshold filter prioritizes analysis of a small number of satisfiable paths, with the rest of satisfiable paths inserted into the path set. With path pruning, this omits a large number of satisfiable predecessor paths in the path set with immediate analysis of a small number of satisfiable paths.

However, a small prediction threshold filter of 0 does not provide much savings because all paths are immediately analyzed without being inserted into the path set. As the prediction threshold increases from 0 to 0.7, more unsatisfiable paths are discarded by the filter and inserted into the path set. As a result, an unsatisfiable path is less likely to be selected as it has more candidates to compete against. In turn, more satisfiable successors are visited and analyzed. This causes the curve for time saving performance to become worse.

Overall, it is preferable to use a high prediction threshold value for the prediction threshold filter. While the savings from the prediction threshold filter is not as significant as the timeout filter, it is still able to achieve good time savings.

C. RQ3: Model Performance

We study how the performance of symbolic analysis is affected upon change in model performance. We try different values of the improvement adjustment and evaluate their impact. Note for model performance, the baseline that time savings we measure against is the scenario with ***an improvement adjustment of 0***, and other parameters unchanged. This is different from the baseline of our previous evaluation sections, which is the original analysis without any path set.

*1) **Basic Path Sets**:* For simplicity, we first study how model performance affects path sets without any analysis filter. Sat/unsat ranking objective and path set size limits are chosen using evaluation from Section V-A.

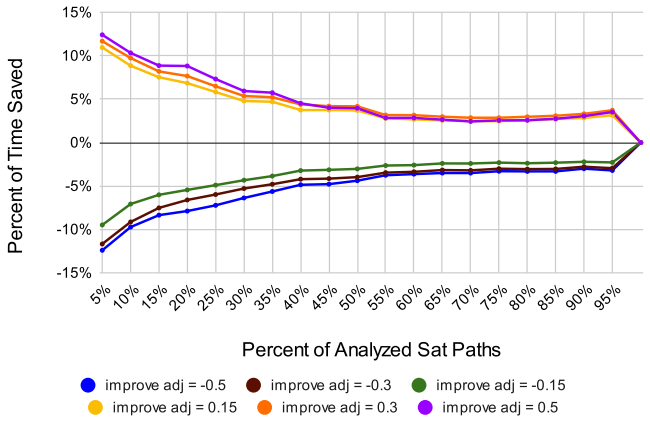
TIRO + Independent Path Set: Our experiments find the time savings generally increases, when a bad predictor improves (as improvement adjustment increases from -0.5 to 0). However, as the improvement adjustment further increases beyond 0, the time savings decrease. This is because in TIRO, a large number of satisfiable paths are long-running, taking more than 10s (Section II-B3). When the model performance is already high, further increasing the improvement adjustment results in a number of long-running satisfiable paths being analyzed. As these satisfiable paths take more time to run, it requires more time to analyze the same number of satisfiable paths.

Therefore, for TIRO's independent path set, it is preferable to have a satisfiable predictor with reasonable performance, but does not prioritize too many long-running path.

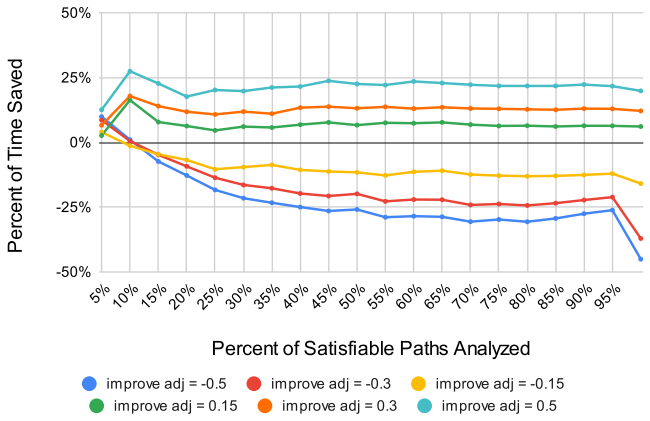
Mythril + Independent / Dependent Path Set: Next, change in model performance is evaluated on Mythril's independent path set with sat ranking, as well as Mythril's dependent path set with path pruning and unsat ranking. In both scenarios, the performance of symbolic analysis generally increases when the improvement adjustment increases, as shown in Figure 6.

In Figure 6a, for Mythril's independent path set, when the model performance is very poor or very good, the time saving is insensitive to changes in the improvement adjustment. In contrast, from Figure 6b, for Mythril's dependent path set, changing performance of a very poor or close-to-perfect model has much larger effect on time savings. This is because Mythril's dependent path set stops exploring unsatisfiable successors when an unsatisfiable path is first hit. Even when a very poor or very good model is used, it does not make much difference as the unsatisfiable paths Mythril visits or omits are bounded. However, with path pruning and dependent path set, it can potentially provide either a large time savings by omitting satisfiable predecessors or significant performance degradation by keeping unsatisfiable successors in the path set. The change in model performance can thus still affect time savings from symbolic analysis even when the model is close to perfect or very poor.

Thus, for Mythril's independent path set, improving mediocre models brings in much more time savings than



(a) Independent Path Set



(b) Dependent Path Set, Path Set Size = 20

Fig. 6: Effect of Model Performance on Mythril (Baseline: Improve Adj = 0)

improving models that already perform well. For Mythril's dependent path set, however, improving model performance generally results in a small increase in time savings.

*2) **Path Set + Analysis Filter**:* We further examine the effect of model performance on path sets combined with immediate analysis filters. We pick Mythril's independent path set with timeout filter, and Mythril's dependent path set with timeout and prediction threshold filter. These combinations and their parameters are selected as they bring in general performance improvement from Section V-B. Studies on combinations that do not improve symbolic analysis by much are omitted, as these combinations are not worth applying.

Mythril's Independent Path Set + Timeout Filter: We evaluate the effect of model performance change on Mythril's independent path set, with an optimal immediate timeout filter value of 250ms. We find the time saving is the best at the ***default*** improvement adjustment of 0.15 in Section V-B2. Although we find better model performance decreases time saving, it is mostly because the better model prioritizes long-running satisfiable paths as the independent path set explores deeper, more time-consuming paths. In fact, the percent of time spent on analyzing satisfiable paths increases with increase in improvement adjustment.

Although analyzing deeper paths is not always undesirable, when the goal is to minimize analysis time, an unfortunate side effect is that the model may correctly pick these long-running paths and end up hurting analysis throughput. Ultimately, effective models need to take analysis time prediction into account as well, which could be interesting future work.

Mythril's Dependent Path Set: We evaluate the effect of model performance on Mythril's dependent path set with immediate timeout filter and prediction threshold filter respectively. The path set size, timeout filter and predictin threshold filter values are parameters with best performance selected from Section V-B3.

- ***Mythril's Dependent Path Set + Timeout Filter***

With the selected timeout filter, time savings generally increase as the improvement adjustment increases. However, when the improvement adjustment increases beyond 0.1, the increase in time savings becomes insignificant. With the time filter and a relatively good model, Mythril's dependent path set can already omit a good number of satisfiable paths and avoid analyzing more unsatisfiable paths. As part of the time saving comes from a well-tuned time filter, keeping increasing model performance is not as helpful.

Thus, for Mythril's dependent path set with timeout filter, it comes with good time savings to improve models that do not perform very well, but continuing to improve models that already have good performance brings in little benefit.

- ***Mythril's Dependent Path Set + Prediction Filter***

With the selected prediction filter, the relationship between time savings and model performance is unstable. Time savings decrease as the improvement adjustment increases from -0.5 to -0.3: more unsatisfiable paths are discarded by the prediction filter and inserted into the path set. Since the path set contains more unsatisfiable candidate paths to select from, unsatisfiable paths stay in the path set for longer and thus more unsatisfiable paths are visited/analyzed. In addition, the time savings decrease when the improvement adjustment increases from 0.15 to 0.5: as the improvement adjustment increases, more satisfiable paths have their satisfiable likelihood above the prediction threshold. These satisfiable paths are analyzed immediately, and their analysis time cannot be saved since they are not inserted into the path set and cannot be pruned.

Therefore, due to complex effect with the prediction threshold filter, improving the model performance is not necessarily a good option.

D. Cross-Validation

To check how results generalize across different subsets of apps, we randomly split apps from both TIRO and Mythril into 5 folds. We measure time savings for each fold of apps upon changing parameters. Despite variability of path analysis time across folds, trends and optimal parameters in Sections V-A, V-B, and V-C generally hold within the same folds. However, in some folds, a perfect model for both Mythril's dependent

and independent path sets without any filter does not result in the best time savings before the 20% checkpoint. This is possibly due to good models picking long running paths, whose analysis time is significant at early stage of analysis.

VI. Related Work

To save time for symbolic execution, recent works build machine learning models for path satisfiability prediction. ICON [15] and DeepSolver [14] use DNNs to predict constraint satisfiability for symbolic execution. Their models only allow constraints with fixed sizes. They query the constraint solver only when a path is predicted as unsatisfiable or when the constraint size does not satisfy their models' requirements. Paths that are predicted satisfiable are skipped without analysis and the satisfiability of these paths are left unknown. This is different from our analysis which determines the satisfiability of all paths found. We believe it brings valuable information to certainly know whether a path is satisfiable, especially for targeted symbolic analysis tools like TIRO.

Learch [8] and Homi [5] learn models and probabilities to prioritize program state exploration. They improve code coverage of the symbolic execution tool KLEE [4] with predecessor-successor relationships during path finding. But they do not do pruning using dependencies between program states. This is similar to our Mythril's independent path set scenario. They also directly learn path selection strategies instead of using a path satisfiability predictor like in our strategies.

Yang et al. [19] uses path-level code features instead of constraint features to predict path satisfiability for the Android app analysis tool TIRO. PCC [16] extracts features from path constraints and predicts the fastest solver. Luo et al. [10] adaptively updates its model to gain knowledge specific to analyzed programs, and predicts whether a constraint can be solved under timeout. Unlike these existing works that try to improve model performance for savings, we explore ways to use model prediction for better symbolic analysis, even when the model is not super accurate.

Instead of predicting constraint satisfiability, some existing works approximate parts of the program when constraints are too complex to solve [9], [3], [12], [2]. These works attempt to solve for inputs when constraint solving is difficult, rather than to prioritize path analysis.

VII. Limitations and Threats to Validity

We estimate strategy overheads such as model prediction, pushing elements to list, feature extraction and restoring analysis states. The two tools we work on also have their path analysis time generally several magnitudes more expensive than our estimated costs. Thus, some performance improvements in our work may not exist for scenarios that have large strategy overheads compared to path analysis time.

We only use random forest models with path features collected during symbolic analysis on our two tools. The performance of our evaluated predictors may not be representative of all satisfiability predictors. Also, we adjust model performance by adding an improvement adjustment to model output satisfiability likelihood. This may not precisely represent the distribution of model prediction upon change in model performance in real world.

Our analysis is limited to two symbolic analysis tools for Android apps and smart contracts. The smart contracts we analyze are also relatively small programs. These programs may not be fully representative of all programs in general.

As mentioned in Section III-A, there may be bias in our data collection, and assumptions from our simulation may not hold for all symbolic analysis tools.

VIII. Conclusion

In conclusion, our study has demonstrated several valuable lessons for applying machine learning prediction to symbolic analysis. First, while existing works focus on improving path satisfiability prediction, our experiments show that as most likely satisfiable paths are not necessarily the cheapest to analyze, prioritizing such paths can lead to a drop in analysis throughput at early stages in a symbolic analysis run. This suggests that when the analysis resources per application is limited, it would be best to not only predict satisfiability, but extend the model to predict analysis cost. Second, leveraging the distribution of path costs and satisfiability for a particular problem domain can be as effective as improving performance of the predictor. For example, adding a well-tuned timeout filter, Mythril's dependent path set can achieve good savings with a reasonably good model. Further improving the model performance in this setting brings little extra time saving.

Finally, we summarize a few key rules of thumb for incorporating model prediction into symbolic analysis. First, for TIRO, a moderate to large path set size is desirable, while for Mythril, extremely large path set sizes result in too many unsatisfiable paths being added to the path set, ultimately reducing analysis throughput. Second, while path pruning leverages path dependencies for time savings, counter-intuitively, prioritizing unsatisfiable paths instead of satisfiable paths works well with path pruning on Mythril's dependent dataset. This combination achieves an overall time saving for analyzing all paths. Lastly, for Mythril, in which most short-running paths are satisfiable, adding an immediate timeout filter with a small timeout value significantly improves time savings for most analysis checkpoints. Similarly, the prediction threshold filter also confers results, when it is able to prioritize these short paths.

IX. Ackowledgements and Data and Code Availability

This research was supported by a Tier 1 Canada Research Chair in Secure and Reliable Systems, NSERC Alliance Grant ALLRP-586310-23, and NSERC Discovery Grant RGPIN-2018-05931. We also thank CIFAR for a Canada CIFAR AI Chair. Resources used in this research were provided in part, by the Province of Ontario, the Government of Canada, and companies sponsoring the Vector Institute. Our data and simulation code is available at https://github.com/dlgroupuoft/simulate-predicted-symbex.

REFERENCES

[1] "Mythril," https://github.com/ConsenSys/mythril, 2023.

[2] B. K. Aichernig, R. Bloem, M. Ebrahimi, M. Tappler, and J. Winter, "Automata learning for symbolic execution," in *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2018, pp. 1–9.

[3] L. Bu, Y. Liang, Z. Xie, H. Qian, Y.-Q. Hu, Y. Yu, X. Chen, and X. Li, "Machine learning steered symbolic execution framework for complex software code," *Formal Aspects of Computing*, vol. 33, no. 3, pp. 301–323, 2021.

[4] C. Cadar, D. Dunbar, D. R. Engler *et al.*, "KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs." in *OSDI*, vol. 8, 2008, pp. 209–224.

[5] S. Cha and H. Oh, "Making symbolic execution promising by learning aggressive state-pruning strategy," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 147–158.

[6] D. Davidson, B. Moench, T. Ristenpart, and S. Jha, "FIE on firmware: Finding vulnerabilities in embedded systems using symbolic execution," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 463–478.

[7] B. N. Freeman-Benson, J. Maloney, and A. Borning, "An incremental constraint solver," *Communications of the ACM*, vol. 33, no. 1, pp. 54–63, 1990.

[8] J. He, G. Sivanrupan, P. Tsankov, and M. Vechev, "Learning to explore paths for symbolic execution," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2526–2540.

[9] X. Li, Y. Liang, H. Qian, Y.-Q. Hu, L. Bu, Y. Yu, X. Chen, and X. Li, "Symbolic execution of complex program driven by machine learning based constraint solving," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2016, pp. 554–559.

[10] S. Luo, H. Xu, Y. Bi, X. Wang, and Y. Zhou, "Boosting symbolic execution via constraint solving time prediction (experience paper)," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 336–347.

[11] C. S. Păsăreanu and N. Rungta, "Symbolic pathfinder: symbolic execution of java bytecode," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 179–180.

[12] S. Shen, S. Shinde, S. Ramesh, A. Roychoudhury, and P. Saxena, "Neuro-symbolic execution: Augmenting symbolic execution with neural constraints." in *NDSS*, 2019.

[13] W. Visser, J. Geldenhuys, and M. B. Dwyer, "Green: reducing, reusing and recycling constraints in program analysis," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[14] J. Wen, M. Khan, M. Che, Y. Yan, and G. Yang, "Constraint solving with deep learning for symbolic execution," *arXiv preprint arXiv:2003.08350*, 2020.

[15] J. Wen, T. Mahmud, M. Che, Y. Yan, and G. Yang, "Intelligent constraint classification for symbolic execution," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 144–154.

[16] S.-H. Wen, W.-L. Mow, W.-N. Chen, C.-Y. Wang, and H.-C. Hsiao, "Enhancing symbolic execution by machine learning based solver selection," in *Proceedings of the NDSS Workshop on Binary Analysis Research*, 2019.

[17] M. Y. Wong and D. Lie, "Tackling runtime-based obfuscation in android with TIRO," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1247–1262.

[18] G. Yang, C. S. Păsăreanu, and S. Khurshid, "Memoized symbolic execution," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, 2012, pp. 144–154.

[19] M. Yang, D. Lie, and N. Papernot, "Accelerating symbolic analysis for android apps," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, pp. 47–52.

[20] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "Appintent: Analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1043–1054.

[21] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in *IJCAI*, 2020, pp. 3283–3290.