FuSeBMC AI

FuSeBMC AI: Acceleration of Hybrid Approach through Machine Learning

(Competition Contribution)

Kaled M. Alshmrany(✉)[1,2][0000−0002−5822−5435], Mohannad Aldughaim[2,3][0000−0002−5822−5435], Chenfeng Wei[2][0009−0008−0416−3006], Tom Sweet[4], Richard Allmendinger[2], and Lucas C. Cordeiro[2][0000−0002−6235−4272]

[1] Institute of Public Administration, Jeddah, Saudi Arabia
[2] University of Manchester, Manchester, UK
[3] King Saud University, Riyadh, Saudi Arabia
[4] SES Escrow, Handforth Cheshire, UK
shamranial@ipa.edu.sa

Abstract. We present *FuSeBMC*-AI, a test generation tool grounded in machine learning techniques. *FuSeBMC*-AI extracts various features from the program and employs support vector machine and neural network models to predict a hybrid approach's optimal configuration. *FuSeBMC*-AI utilizes Bounded Model Checking and Fuzzing as back-end verification engines. *FuSeBMC*-AI outperforms the default configuration of the underlying verification engine in certain cases while concurrently diminishing resource consumption.

1 Test-Generation Approach

The success of Machine Learning (ML) in automating diverse software engineering tasks is noteworthy, given the escalating complexity of modern software systems [1]. A hybrid approach of multiple techniques, including fuzzing, bounded model checking, and abstract interpretation, has proven effective in verifying software compliance with specified requirements [2]. However, challenges arise, particularly in software with intricate conditions or loops, where the primary obstacle lies in navigating the exponentially expanding program state space and managing resource consumption. Various efforts have been undertaken to enhance the hybrid approach, exemplified by initiatives such as *FuSeBMC* Interval Analysis [3] and Tracer [2]. *FuSeBMC* [4, 5] works as a test generator that synthesizes "smart seeds" with properties to enhance the efficiency of its hybrid fuzzer, achieving extensive coverage of programs. To address challenges related to program state explosion and resource usage, *FuSeBMC* provides the option of execution with diverse parameters (flags). Unfortunately, determining the optimal flags for a specific program requires expert knowledge, often leading to the execution of hybrid tools with default settings and subsequent compromises in performance. This paper presents the *FuSeBMC*-AI tool to predict the optimal configuration flags for a given program. Specifically, *FuSeBMC*-AI employs ML models, support vector machines (SVMs), and neural network (NN) models to predict optimal settings. These ML models undergo training to discern relevant features within the input C program. *FuSeBMC*-AI exhibits enhancements in some subcategories in Test-Comp 2024 [6], such as "ControlFlow", "Hardware", "Loops", and "Software Systems BusyBox Mem-

Safety”, if compared to the default configuration of *FuSeBMC*, achieving a 3% reduction in resource utilization as reported in Test-Comp 2024. [1]

2 Software Architecture

FuSeBMC-AI builds on top of *FuSeBMC*v4.2.1 [4,5]. The initial step involves analyzing the source code, extracting features that impact training and enhancing the capabilities of *FuSeBMC*-AI’s engines. Subsequently, these features are stored for future application in ML models. These models, in turn, forecast optimal scores for *FuSeBMC*-AI’s engines. After that, *FuSeBMC*-AI executes the target program using the recommended configuration. Fig.1 illustrates the *FuSeBMC*-AI framework.

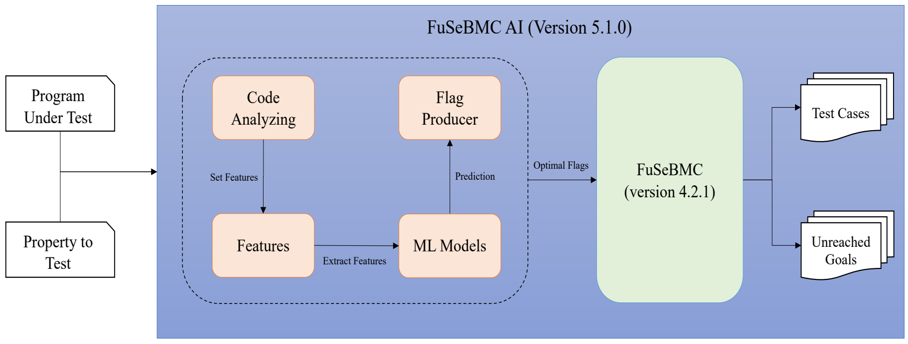

Fig. 1. The major components of the *FuSeBMC*-AI test generator and how they interact.

Setting Features. We focus on discerning the features whose values could impact the efficacy and limitations of the engine’s performance. This emphasis arose from recognizing that certain programs need specific values for effective handling, particularly those involving arrays and loops. We analyze the Program Under Test (PUT) and extract the features that *FuSeBMC*-AI prioritized, which are based on determining the optimal flags and values that could be supplied to the engines of *FuSeBMC*-AI (Tab. 1).

Dataset. The SV-Comp benchmarks were selected as the dataset for our training and testing phases for ML models. Our emphasis was on diversity, considering various scenarios, and minimizing repetition to enhance the precision of our approach. We addressed multiple categories: “no-overflow”, “termination”, “unreach-call”, and “valid-memsafety”[2]. However, for the Test-Comp 2024, our focus narrowed to “coverage-error-call” and “coverage-branches” encompassing a total of 3352 benchmarks. In detail, the training set contains 4% (111 benchmarks) of the coverage-branches benchmarks and 11% (67 benchmarks) of the coverage-error benchmarks in Test-Comp 2024.

Training and Testing models. We focused on four models: Decision Tree Classification (DTC) [7], Support Vector Classification (SVC) [8], Neural Network Regression

[1] https://test-comp.sosy-lab.org/2024/results/results-verified/

[2] https://doi.org/10.5281/zenodo.10458701

Program Features	**Sub Features**
For Loops	For count, For max depth, For depth avg
While Loop	While count, While infinite count While max depth, While depth avg, While infinite with NonDetCall count
Do Loop	Do Count, Do max depth, Do depth avg, Do infinite count
If – Else condition	If count, If max depth, If depth avg, nested If count, Else count, Else depth avg
NonDetCall	Non DetCall count, Non DetCall depth avg, has Non DetCall in loop

Flags	**Values**
Strategy	incr, kinducti
Solver	boolector, z3
Encoding	floatbv, fixedbv
KStep	[1,2,3]
ContextBound	[2,4]
Unwind	[10, -1] #-1 default
Fuzz1Enabled	[0,1]
Fuzz1Time	[25,83,188] for 250 seconds, (300 - 50) 75% ,33.3% ,10%
Total run	2*2*2*3*2*2*4 = 384 (for each program)

Table 1. presents the features that *FuSeBMC*-AI prioritized, along with illustrative examples of flags that could be supplied to the engines of *FuSeBMC*-AI.

(NNR) [9], and a multi-model (DTC then SVC then NNR). The training phase was executed, followed by using the aforementioned benchmarks. The four models underwent supervised and guided training, ensuring a balanced approach to mitigate repetition during the training phase. The training process involved teaching the models to predict optimal flags for *FuSeBMC*-AI's engines, thereby assisting these engines in determining the most suitable flag values for each category of programs. The classification of outputs was dedicated to facilitating model training (Tab. 2). The classification process involved categorizing "Cover-Error" and "Cover-Branches". This categorization was based on the extent of coverage or error detection and the corresponding time duration. Comprehensive testing with 384 different combinations of flags (for each program) was conducted. Consider the cover branches as an illustrative example to provide a more comprehensive understanding of the scale of the conducted experiments. With 111 benchmarks within the Cover-Branches category, *FuSeBMC* is executed approximately $42,624$ times (111 multiplied by 384). Subsequently, we compile a summary encompassing the verification time and verdict for each of the $55,488$ training samples, categorized into "Cover Error" ($12,864$ instances) and "Cover-Branches" ($42,624$ instances). These samples are assigned ordinal labels ranging from 0 to 5, as per the classification outlined in Tab 2. Lower values within the output class are considered more favorable, indicating swift and accurate verdicts.

Testing Result (Cover-Error)	**Coverage Result (Cover-Branches)**	**Class**
detect bug & IF restTimeRatio >= 0.8	score coverage >= 0.85	0
detect bug & ELSE IF restTimeRatio >= 0.6	score coverage >= 0.68	1
detect bug & ELSE IF restTimeRatio >= 0.4	score coverage >= 0.51	2
detect bug & ELSE IF restTimeRatio >= 0.2	score coverage >= 0.34	3
detect bug & ELSE IF restTimeRatio >= 0.0	score coverage >= 0.17	4
Unknown	score coverage >= 0.0	5

Table 2. The classification process for "Cover-Error" and "Cover-Branches."

Machine Learning Models. DTC, SVC, and NNR models undergo supervised training using the Scikit-learn library [10]. Each sample is weighted based on class frequency to address class imbalances within the training set. These ML models are trained to predict the output class from 0 to 5, considering the features of the Program Under Test (PUT) and a specific set of flags. The trained ML models are then employed to predict the optimal set of flags for *FuSeBMC*-AI. Specifically, all 384 possible flag combinations are tested, and the one resulting in the lowest output class is selected. Due to the computational efficiency of these models, this process is executed rapidly, typically concluding within a few seconds.

3 Strengths and Weaknesses

Our proposed hybrid approach demonstrates efficacy in identifying vulnerabilities and attains optimization of computational resources through the AI-based optimisation of fuzzer invocation. An AI model is trained on the classified database, generating optimal flags for executing *FuSeBMC*-AI. For instance, the state system generated during the BMC process can be minimized by correctly configuring the unwinding times guided by the training models. This strategic adjustment results in a reduction of traversal spaces and smart seeds generation time. The impact of these enhancements is discernible in the benchmark sets of “ControlFlow,” “Hardware,” “Loops,” “Software Systems BusyBox MemSafety,” and “Termination-Main ControlFlow” when compared with the default *FuSeBMC*. In detail, *FuSeBMC*-AI successfully preserved the test case quality in “ControlFlow” and “XCSP” within the Cover-Error category, achieving a reduction in resource consumption of approximately 86% in “ControlFlow” and 41% in “XCSP”. In the Cover-Branches category, *FuSeBMC*-AI demonstrated increased coverage by 4% in “Software Systems BusyBox MemSafety” and 1.2% in “Hardware”. Furthermore, it achieved an 84% reduction in resource consumption in “Termination-Main ControlFlow” while maintaining the same coverage as the default *FuSeBMC*. However, our approach still exhibits limitations. Due to the training process primarily relying on the code sourced from SV-Comp 2023 and Test-Comp 2023, there is an insufficiency in the number and program structures of the training samples. Our ongoing efforts are directed towards addressing this limitation by enriching the training dataset and extending the application of our methodology to open-source software projects.

4 Tool Setup and Configuration

FuSeBMC-AI can be used via the python wrapper `fusebmc.py` to simplify its usage for the competition. Please refer to its help message (`-h`) for usage instructions. This wrapper runs the *FuSeBMC*-AI executable with command line options specific to each supported property. Also, *FuSeBMC*-AI offers a graphical user interface (GUI) for enhanced usability [1].

5 Software Project

FuSeBMC-AI is publicly available under the terms of the MIT License at GitHub.[2] *FuSeBMC*-AI (version 5.1.0) dependencies and instructions for building from source code are all listed in the `README.md` file.

[1] https://doi.org/10.5281/zenodo.10458701

[2] https://github.com/kaled-alshmrany/FuSeBMC/tree/FuSeBMC-AI

6 Data-Availability Statement

All files necessary to run the tool are available on Zenodo [11].

References

1. Rossi, B. & Pitner, T. Towards a Definition of Complex Software System. *Position Papers Of The 18thConference On Computer Science And Intelligence Systems*. pp. 119 (2023)
2. Alshmrany, Kaled M., Mohannad Aldughaim, Ahmed Bhayat, Fedor Shmarov, Fatimah Al-jaafari, and Lucas C. Cordeiro. “FuSeBMC v4: Improving code coverage with smart seeds via fuzzing and static analysis.” arXiv preprint arXiv:2206.14068 (2022).
3. Aldughaim, M., Alshmrany, K., Gadelha, M., Freitas, R. & Cordeiro, L. FuSeBMC v.5: Interval Analysis and Methods for Test Case Generation. DOI:https://doi.org/10.5281/zenodo.7473124 (Zenodo,2022,12)
4. Alshmrany, K., Aldughaim, M., Bhayat, A. & Cordeiro, L. FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. *International Conference On Tests And Proofs*. pp. 85-105 (2021)
5. Alshmrany, K., Aldughaim, M., Bhayat, A. & Cordeiro, L. FuSeBMC v4: Smart Seed Generation for Hybrid Fuzzing. *International Conference On Fundamental Approaches To Software Engineering*. pp. 336-340 (2022)
6. Beyer, Dirk. “Software testing: 5th comparative evaluation: Test-Comp 2023.” Fundamental Approaches to Software Engineering LNCS 13991 (2023): 309.
7. Quinlan, J. Induction of decision trees. *Machine Learning*. 1 pp. 81-106 (1986)
8. Cortes, C. & Vapnik, V. Support-vector networks. *Machine Learning*. 20 pp. 273-297 (1995)
9. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.
10. Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. “Scikit-learn: Machine learning in Python.” the Journal of machine Learning research 12 (2011): 2825-2830.
11. Alshmrany, K., Aldughaim, M., Wei, C., Allmendinger, R., & Cordeiro, L. FuSeBMC AI: Acceleration of Hybrid Approach through Machine Learning. DOI:https://doi.org/10.5281/zenodo.10199336 (Zenodo,2023,11)