AFCML: Accelerating the Functional Coverage through Machine Learning within a UVM Framework

Syed Jawad Shah, Majeed Ahmed, Muhammad Imran, Haroon Waris, Nasir Mohyuddin, Mahboob Ur Rehman

{sshah.msee22seecs, mahmed.msee18seecs muhammad.imran1}@seecs.edu.pk,
{haroonwaris, nasir.mohyuddin, chudrymahboob}@gmail.com

National University of Science & Technology (NUST), Pakistan

ABSTRACT- As system complexity continues to increase, there is a growing need for more effective verification methods. This work proposes a novel solution to tackle this issue by leveraging machine learning techniques, including K-means++ clustering, to pinpoint the most relevant stimuli from a broad set for a particular design within the UVM environment. By examining the design's features and the effects of various stimuli, the machine learning system will strategically choose and generate the most impactful stimuli, thereby enhancing the efficiency of the verification process. With the deployment of the machine learning model, improvements in transaction efficiency by 49 percent and a notable decrease in the time needed to reach complete functional coverage were observed.

I. INTRODUCTION

The rapid advancement in semiconductor technology has significantly escalated the complexity of Integrated Circuit (IC) designs. Functional verification, which ensures that the design under test (DUT) behaves as expected under all possible circumstances, accounts for nearly 70% of the overall design effort [1]. Even with improvements in verification techniques, the quest for full functional coverage closure is still an ongoing struggle especially where designs are becoming more complex. The Universal Verification Methodology (UVM) has become a standard framework to create scalable, modular, and reusable, verification setups. Yet even with its organized approach, UVM-based setups still depend a lot on constrained random stimulus creation. This approach may result in repetitious or useless stimuli that extend simulation times longer than required [2].

The main challenge is to complete verification, making sure all possible design specifications have been verified without making and running redundant test cases that waste valuable simulation resources. As industry struggles with the demand for high performance and power-efficient chips, smart solutions have become imperative to improve the efficiency of the verification process. One such solution involves adding machine learning (ML) techniques to verification setups. The ability of ML to analyze large data sets, recognize patterns and make data driven decisions brings a tremendous opportunity to enhance generation of effective stimulus and reduce the number of simulation cycles, required to achieve full functional coverage.

Machine learning methods are widely used to improve the verification process, as also reported by existing studies. In [3] researchers implemented classification in a homeomorphism algorithm to suppress repeating stimuli, with which covering cases are detected to shrink dynamic stimulus generation on Godson-2 processor workload (62.5% usages). Similarly, [4] used an unsupervised support vector analysis model to achieve a 65.3 % test count reduction on ALU but with regards to complex designs further validation was recommended against.

An artificial neural network (ANN) to partition and prioritize the assertions-based stimuli for execution, achieving a 24.5x simulation speedup on CPU tests while requiring only 60% of the stimuli is presented in [5]. Furthermore, [6] showed that word-level assertions result in more comprehensible and efficient code coverage; offering the same 90% input coverage while writing half as much new lines, with less bugs founded on a commercial SMP (Symmetric multi-processing) processor. Finally, K-means clustering was used to segment the signals making it much easier for existing bug detection methods to identify design errors earlier in time by exploiting non-frequent triggered signals that were clustered out (21%−40% shorter bug-detection times across multiple DUTs) with broader applicability into different companies [7].

The proposed study builds on existing work by introducing a new verification framework. It combines machine learning methods, like K-means++ clustering and decision trees, with the UVM environment. Choosing the K-means++ for its speed, simplicity, and improved initialization over other clustering algorithms, along with its scalability compared to hierarchical and density-based methods, making it ideal for clustering large, high-dimensional datasets typically encountered in IP verification environments. The framework aims to handle the creation and running of stimuli. It groups them based on how they affect coverage and predicts which ones will lead to full coverage closure. Unlike previous methods that often saw stimuli as separate units, the clustering approach used here allows the framework to group similar stimuli. This enables more targeted testing of areas that haven't been verified yet. The study then uses decision trees, which are easy to understand and explain, to predict if a particular stimulus will contribute to coverage or not. It gives priority to those that will and less importance to those that won't.

The main contributions of this work is development of a three-stage machine learning pipeline that improves the UVM-based verification process as follows:

1. Test Classification: Classify tests based on the variables that are being randomized, this allows us to group similar types of tests and analyze them more effectively.
2. Unsupervised Clustering: The stimuli are clustered automatically in K-means++ clustering according to their functional coverage attributes and should enable the verification environment to concentrate on areas of DUT that remain unverified.
3. Prediction of Functional Coverage: Using decision trees to predict when stimuli in a cluster will hit uncovered coverage points can make the process deterministic and reduce waste generation from generating redundant motivations.

We have evaluated the proposed framework through a set of experiments on AHB-Lite (Advanced High-performance Bus) DUT and improved both verification efficiency, as well as coverage closure time. While running the machine-learning optimized UVM test bench, it eliminated most of the redundant stimuli and used scarce simulation resources for high-priority access sequences leading to a maximum 49.7% reduction in verification time.

The remainder of this paper is structured as follows: Section II outlines the contributions made by this research work, Section III presents the results, highlighting the improvements achieved through the proposed approach. Finally, Section IV concludes the paper and suggests areas for future research.

II. CONTRIBUTION

For efficient verification methodology, the primary challenge lies in identifying an intelligent method to apply stimuli that maximizes functional coverage while minimizing the number of transactions and simulation time. To address this very challenge, this research demonstrates the use of machine learning (ML) techniques to address the following key questions:

i. Can machine learning techniques effectively generate and prioritize stimuli to achieve higher functional coverage with fewer transactions?
ii. Is it possible to apply stimuli in a specific order, based on their effectiveness, to maximize functional coverage closure in a shorter time?
iii. How can we integrate machine learning into the verification process to apply effective stimuli in a specific order during run-time?
iv. What types of feedback mechanisms or metrics can be used to guide and adjust stimuli application, dynamically during verification?

Before diving into the novel integrated machine learning model, we'll briefly cover the DUT and the UVM environment to provide context. This includes a quick overview of the design's key functions and the structure of the verification environment, which will help illustrate how the new model improves the verification process.

A. Design Under Verification (DUV)

The Advanced High-performance Bus (AHB), which we generated from [8] is specifically designed for high-performance systems, supporting multiple bus masters and high-bandwidth operations. A typical AMBA system setup includes an AHB master, AHB slave, AHB arbiter, and AHB decoder. In an AHB system, masters are responsible for initiating transactions, while slaves respond based on the incoming requests. Only one master can access the bus at a time, so each must request access (grant) and wait its turn in line with the arbitration process before it can initiate a transaction. But the scope of this research effort is not limited to the AHB, but can be implemented on any Intellectual Property (IP).

B. UVM Verification Environment

The pillars of verification are foundational to ensure that a design's intent, initially captured at the Register Transfer Level (RTL), is accurately realized. This process involves applying a carefully selected set of stimuli to target specific areas of the design and then verifying if it behaves as expected. Simulation continues until all critical and significant scenarios in the design space are verified for correct functionality. Figure 1 illustrates these pillars of verification.

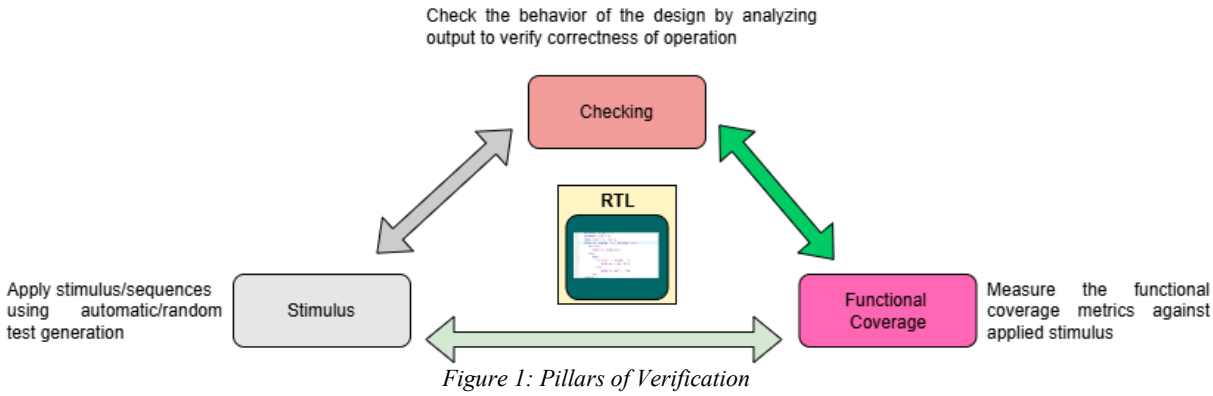

Figure 1: Pillars of Verification

Our verification environment, developed in Mentor Questa Sim Version 10.6, is structured into three key parts to ensure effective testing and validation of the design. First, we use an agent to simulate behavior of a master, to initiate transactions and produce control signals. Next comes slave agent which replicates that slave behavior, it will respond

to requests based on the data received. In between these two categories of agents are the specialized components that facilitate critical functions like sampling and transaction driving, assertion checking to ensure that expected behavior is exhibited, and coverage data is being collected.

Figure 2 illustrates how a typical UVM Verification Environment works. The master agent generates signals, which are sent to the DUT through a dedicated interface. Similarly, slave agent receives its input from an interface connected to the DUT. During testing, all signals exchanged between the agents and the DUT are carefully monitored. These signals are sent to a scoreboard that checks their validity, ensuring the design behaves as expected. Additionally, functional coverage data is collected throughout the process to confirm that all important scenarios have been tested. This structured approach not only improves the reliability of the design but also helps catch potential issues early on.

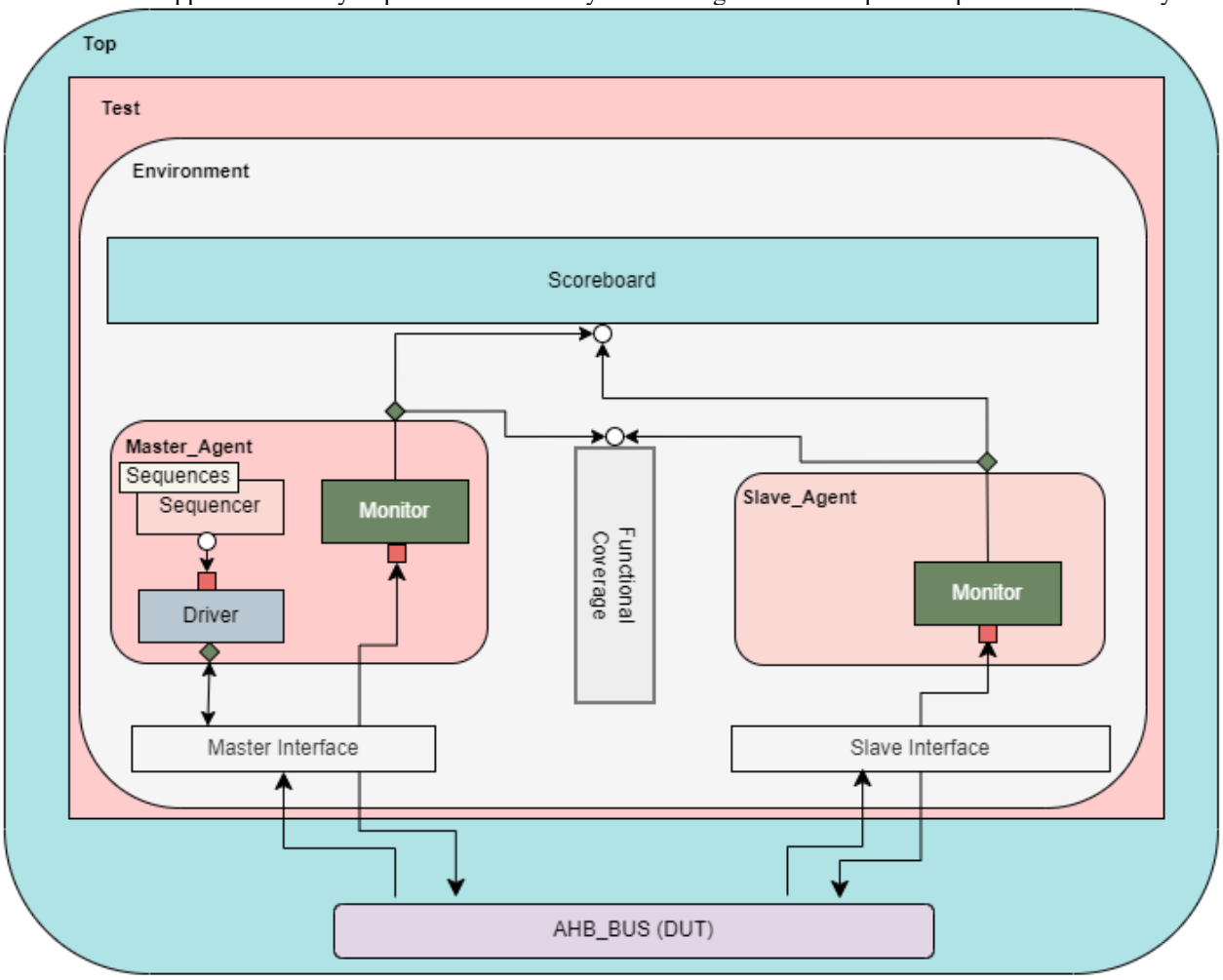

Figure 2: UVM Verification Environment

A. Test Plan

Our tests are grouped into two main categories: normal constrained random tests and directed tests. We're focused on testing specific AHB functionalities to gather timing results rather than verifying all features. Once we have these results, we plan to enhance them using machine learning techniques. Only the normal tests will be optimized with machine learning, as the directed tests are straightforward and do not require further enhancement. In random test, we will be focused upon bus request test, which is simpler one and the purpose of this test to check for masters acquiring their grants for certain times for different stimulus.

B. Machine Learning Model

Our novel ML model consists of three stages, each utilizing different machine learning techniques. First, we categorize tests into groups based on the main variables being randomized. This groups similar tests together, as tests focused on "read" operations, for instance, will likely have closer coverage metrics than those testing protocol implementation. Next, we cluster stimuli into several groups based on coverage-related variables. Finally, after training our model on tests within each class, we predict whether each stimulus will achieve coverage within its respective cluster. Figure 3 shows our verification environment with the integrated ML model, where stimuli flow to the model, which then decides if they should be executed.

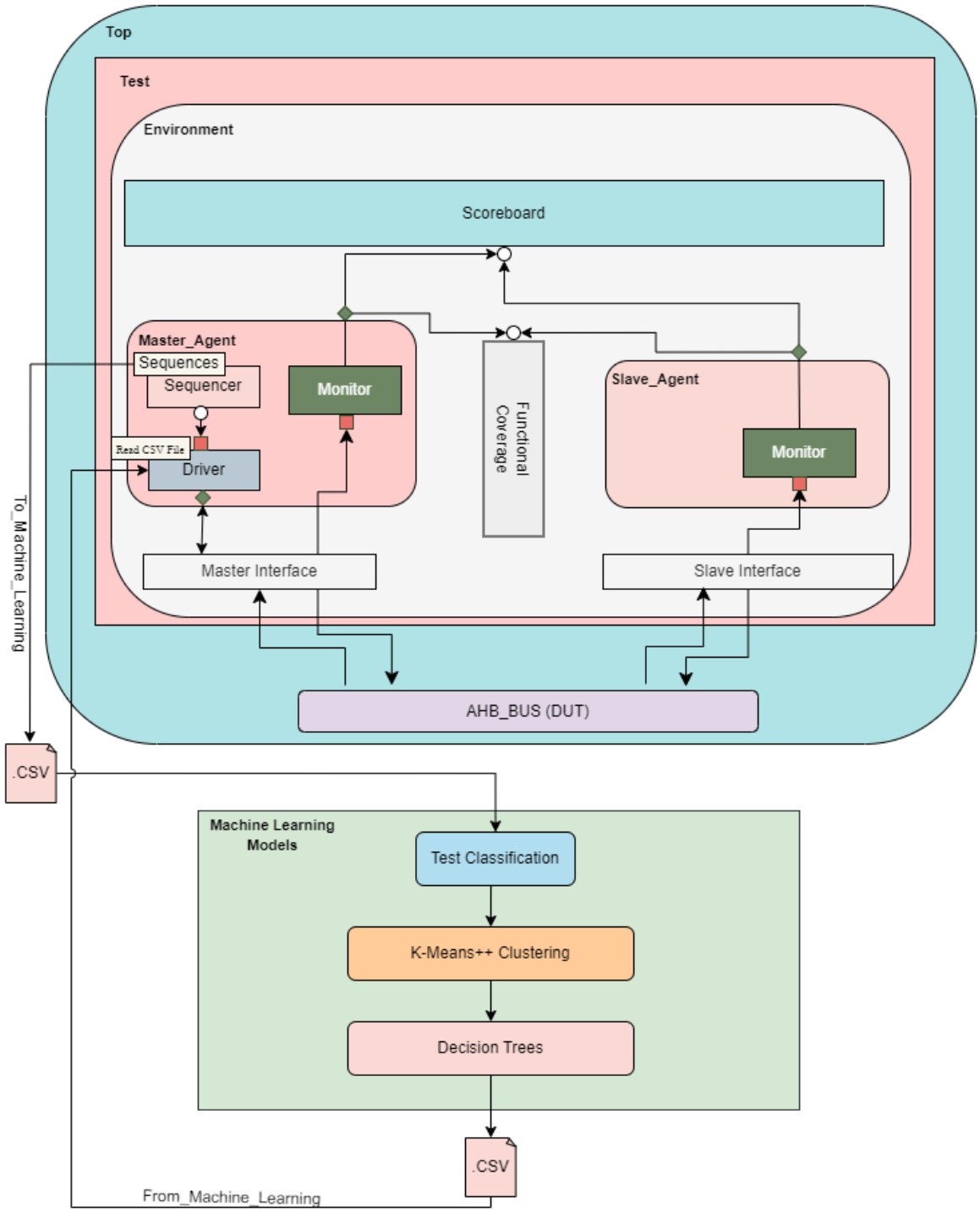

Figure 3: Verification Environment after ML Enhancement

1). Stage 1: Classifying Tests

In the first stage of our model, incoming stimuli are sorted into specific categories. Predicting success across all stimuli for various tests is challenging, as each test has a unique purpose. Classifying tests, however, is straightforward and doesn't require ML; a verification engineer can easily handle this. By using simple coding, we determine which tests focus on specific inputs and instruct the system to create test classes accordingly.

2). Stage 2: Unsupervised Machine Learning

In the next stage, we independently cluster stimuli into 'k' clusters for each test class. Here, stimuli are clustered based on data relevant to the functional coverage process, grouping similar items to improve the accuracy and efficiency of coverage predictions. Figure 4 presents coverage against random stimuli, while Figure 5 illustrates the creation of different clusters using K-Means clustering.

M_HBUSREQ	M_HADDR	M_HTRANS	M_HSIZE	M_HBURST	M_HPROT	M_HLOCK	M_HWRITE	M_HWDATA	Coverage
0	0	0	0	0	0	0	0	0	0
0	89	0	0	0	0	0	0	2439	0
0	187	0	0	0	0	0	0	1619	0
0	275	0	0	0	0	0	0	3203	0
0	137	0	0	0	0	0	0	4390	0
1	269	0	0	0	0	0	1	2295	0
1	392	0	0	0	0	0	0	3537	0
1	361	0	0	0	0	0	0	1926	0
1	20	0	0	0	0	0	0	4801	0
1	235	0	0	0	0	0	1	2916	1
1	103	0	0	0	0	0	0	3060	0
1	90	0	0	0	0	0	0	3946	0
0	378	0	0	0	0	0	1	3407	0
0	224	0	0	0	0	0	1	4599	0
1	57	0	0	0	0	0	1	2857	0
1	173	0	0	0	0	0	1	3211	1

Figure 4: Tracking Coverage Responses to Random Stimuli

M_HBUSREQ	M_HADDR	M_HTRANS	M_HSIZE	M_HBURST	M_HPROT	M_HLOCK	M_HWRITE	M_HWDATA	Cluster
1	161	0	0	0	0	0	1	4961	0
0	134	0	0	0	0	0	0	80	0
0	221	0	0	0	0	0	0	2770	2
0	28	0	0	0	0	0	0	399	3
1	129	0	0	0	0	0	0	2334	0
1	175	0	0	0	0	0	0	3631	0
1	39	0	0	0	0	0	0	1831	3
1	327	0	0	0	0	0	0	1133	1
1	211	0	0	0	0	0	1	4967	2
0	316	0	0	0	0	0	1	1029	1
0	166	0	0	0	0	0	0	890	0
1	233	0	0	0	0	0	1	3685	2
0	115	0	0	0	0	0	0	3242	0
1	99	0	0	0	0	0	1	3124	0
1	100	0	0	0	0	0	1	4695	0
1	192	0	0	0	0	0	1	1177	0

Figure 5: Clustering Similar Stimuli for Pattern Recognition

3). Stage 3: Supervised Machine Learning

The third and final stage of our model focuses on predicting coverage for each cluster of stimuli. Predicting coverage on a cluster level is beneficial, as similar stimuli generally target similar coverage areas. Stimuli predicted to meet coverage requirements are run first; if these don't achieve 100% coverage, the remaining stimuli are executed. This way, if our predictions are less accurate for a particular test, all stimuli still get run just in a different order. Redundant stimuli are also identified and queued last, ensuring efficiency. Figure 4 illustrates this third stage. For this stage, we'll use decision trees, as they have a fast-learning rate. While more complex models like Deep Neural Networks (DNNs) could capture intricate relationships or uncover hidden features, and Recurrent Neural Networks (RNNs) could handle sequence dependencies, a decision tree is sufficient for our current tests. Our main goal is to prototype this approach and assess its effectiveness, allowing room for more advanced models in future refinements.

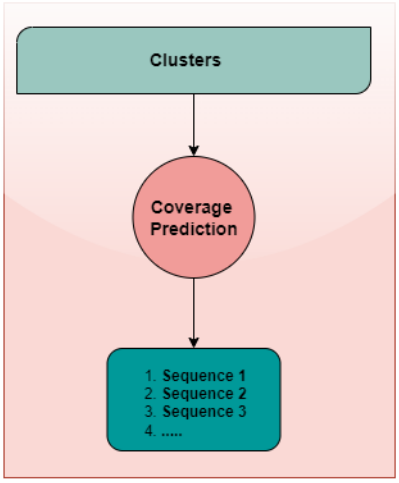

Figure 6: Coverage Prediction

A. Running Mechanism in the UVM Environment After ML Enhancement

Our verification environment is inspired by the principles of Coarse Grain Multi-threading, applying similar logic to manage cluster-based testing. In Coarse Grain Multi-threading, a controller issues new instructions from a single thread each clock cycle until it encounters an instruction with a long wait time, prompting it to switch to a new thread.

In our verification setup, we typically have four coverage-predicted clusters. We run stimuli from a file until we encounter two consecutive stimuli that don't change coverage, at which point we switch to the next file. This approach keeps testing efficient, reducing time spent on stimuli less likely to reach coverage and maintaining a streamlined testing flow.

III. RESULTS

The experimental results of this research demonstrate the significant advantages of integrating machine learning techniques into the UVM framework. This section analyzes performance improvements in terms of functional coverage closure, reduction in simulation cycles, and time savings achieved through the machine learning driven verification environment. Figure 5 illustrates the comparison between coverage and the number of transactions. In the conventional UVM approach (without ML), 100% functional coverage is achieved after 1997 transactions. However, with the introduction of the ML model, the same functional coverage is obtained in just 1003 transactions, reflecting a substantial reduction in the number of stimuli required.

Similarly, Figure 6 highlights the comparison between coverage and simulation time. By integrating ML, the total simulation time is reduced to 10,016 ns, representing an approximate improvement of 49.7% compared to the non-ML approach.

Figure 7 presents a detailed summary of these results, showcasing how the machine learning model significantly enhances overall verification efficiency by reducing both the transaction count and simulation time while maintaining full coverage. These results emphasize the effectiveness of ML in streamlining the functional verification process, providing a more efficient solution for complex IC design validation.

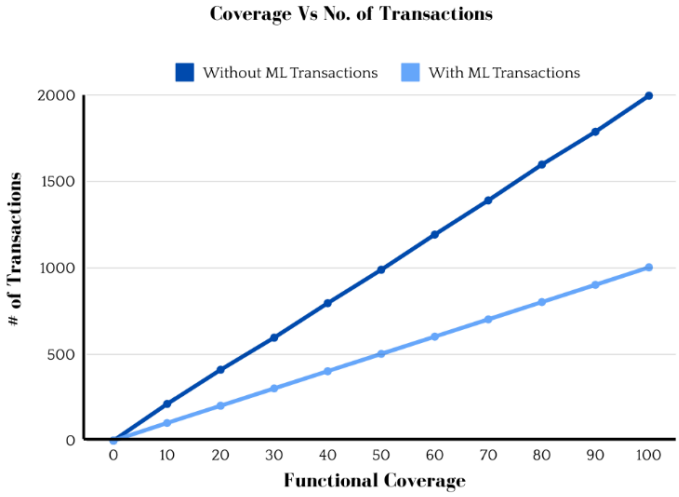

Figure 7: Comparison b/w Coverage and Transactions

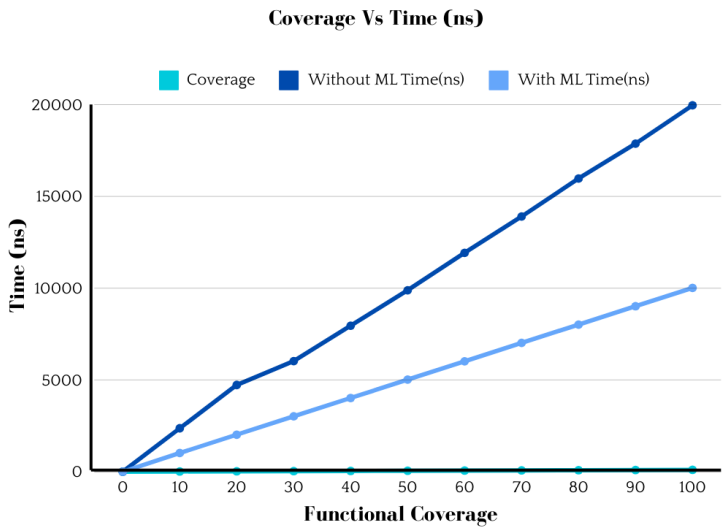

Figure 8: Comparison b/w Coverage and Simulation Time

Coverage	Without ML Time (ns)	Without ML Transactions	With ML Time (ns)	With ML Transactions
0	0	0	0	0
10	2366	212	1016	102
20	4726	410	2016	202
30	6026	596	3016	302
40	7956	796	4016	402
50	9886	989	5016	502
60	11926	1193	6016	602
70	13906	1391	7016	702
80	15976	1598	8016	802
90	17876	1788	9016	902
100	**19966**	**1997**	**10016**	**1003**

Figure 9: Coverage Closure with ML and without ML

The experiments primarily focused on verifying the improvements introduced by the three-stage machine learning pipeline test classification, K-means++ clustering, and decision tree-based functional coverage prediction. The results are compared against a baseline UVM environment that relied on constrained random testing (CRT) without machine learning assistance.

A major improvement is noted in the time it took to achieve coverage closure. In conventional verification settings that utilize constrained random stimulus generation, reaching complete coverage often demands many simulation cycles. Nevertheless, the ML-augmented framework minimized the number of cycles needed by strategically focusing on stimuli that is more probable to aid in coverage.

On average, the UVM environment driven by machine learning decreased the total time for coverage closure by 10% to 49.7%, varying with the DUT's complexity. For simpler AHB-Lite configurations that had fewer master-slave interactions, coverage is achieved in 30% fewer cycles. In more complex cases, including locked transactions or error injections, the reduction in coverage closure time is even more significant, achieving up to 49.7%.

The decision tree models, which are trained using initial coverage data, can accurately predict coverage-hitting stimuli in over 85% of cases. This prediction capability significantly reduced the generation of redundant stimuli and ensured that the verification environment focused on areas that had not yet been fully explored.

From the experimental results, using machine learning in UVM saves a huge amount of time during verification to avoid generating unnecessary stimuli and gives an easier way for faster coverage closure. The use of the decision tree model for classifying and providing predictions on the covering hitting stimuli is made possible through the K-means++ algorithm, which offers an all-embracing but concentrated verification method. However, some random tradeoffs must be considered. One of the key disadvantages is that the first training phase in implementing the decision tree model is somewhat more time-consuming to simulate in the beginning stage for larger and more complex designs for the model.

IV. CONCLUSION

In this paper, we proposed a machine learning-enhanced UVM environment applied to the AHB-lite framework. Our machine learning approach combines K-means++ clustering and decision tree-based method to select and execute effective stimuli in a specific order. The effectiveness of a stimulus is determined by its contribution to achieving higher functional coverage. By implementing this machine learning strategy, we successfully reduced functional coverage closure time by 49.7%. Further enhancements could be achieved by incorporating more advanced machine learning techniques, such as deep neural networks (DNNs) or reinforcement learning, to address more complex verification scenarios. Additionally, establishing real-time feedback between the machine learning models and the UVM environment could significantly improve the dynamism of stimulus generation, allowing for more adaptive and efficient testing processes. This integration would enable the system to learn from ongoing test results and adjust stimulus strategies, accordingly, leading to even better functional coverage and performance outcomes. This significant improvement demonstrates the potential of machine learning in the design verification domain and offers encouragement to future researchers in this field.

ACKNOWLEDGEMENT

The authors express their sincere gratitude to **Mr. Mahmoud Tamer (**The German University in Cairo**)** for his instrumental support in broadening the scope of this work. The research exhibition is a testament to his dedicated efforts in developing the research community. Additionally, we would also like to acknowledge the valuable contributions of our friends, whose thoughtful reviews have enriched this paper.

REFERENCES

[1] R. Klein and T. Piekarz, "Accelerating functional simulation for processor based designs," *Fifth International Workshop on System-on-Chip for Real-Time Applications (IWSOC'05)*, Banff, AB, Canada, 2005, pp. 323-328, doi: 10.1109/IWSOC.2005.34.

[2] N. Kitchen and A. Kuehlmann, "Stimulus generation for constrained random simulation," *2007 IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, USA, 2007, pp. 258-265, doi: 10.1109/ICCAD.2007.4397275.

[3] Q. Guo, T. Chen, H. Shen, Y. Chen and W. Hu, "On-the-Fly Reduction of Stimuli for Functional Verification," *2010 19th IEEE Asian Test Symposium*, Shanghai, China, 2010, pp. 448-454, doi: 10.1109/ATS.2010.82.

[4] O. Guzey, L. -C. Wang, J. Levitt and H. Foster, "Functional test selection based on unsupervised support vector analysis," *2008 45th ACM/IEEE Design Automation Conference*, Anaheim, CA, USA, 2008, pp. 262-267, doi: 10.1145/1391469.1391536.

[5] Fanchao Wang, Hanbin Zhu, Pranjay Popli, Yao Xiao, Paul Bodgan, and Shahin Nazarian. 2018. Accelerating Coverage Directed Test Generation for Functional Verification: A Neural Network-based Framework. *In Proceedings of the 2018 Great Lakes Symposium on VLSI (GLSVLSI '18).* Association for Computing Machinery, New York, NY, USA, 207–212. https://doi.org/10.1145/3194554.3194561

[6] L. Liu, C. -H. Lin and S. Vasudevan, "Word level feature discovery to enhance quality of assertion mining," *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2012, pp. 210-217.

[7] Eman Mandouh and Amr G. Wassal. 2018. Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization. *J. Electron. Test. 34, 2* (April 2018), 163–181. https://doi.org/10.1007/s10836-018-5716-y

[8] *Adki/gen_amba_2021: Amba bus generator including Axi4, Axi3, AHB, and APB*, *GitHub*. Available at: https://github.com/adki/gen_amba_2021.