2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 14-15, 2025

Accelerating Coverage Closure with Reinforcement Learning: A Case Study on FSM Verification

Tijana Misic, Verification Consulting doo, Belgrade, Serbia (tijana.misic.consulting@gmail.com)

***Abstract*—Verifying complex SoC hardware and software systems poses significant challenges, particularly when trying to balance fast verification cycles with growing design complexity. This paper explores how reinforcement learning (RL) can be applied to improve coverage closure time. We present an experiment involving the verification of a custom finite state machine (FSM), comparing results obtained through traditional constrained random verification with those enhanced by reinforcement learning. The findings demonstrate that integrating RL into the verification flow leads to a noticeable reduction in coverage closure time, showcasing the potential of machine learning to optimize and accelerate the verification process.**

Keywords— Constrained Random Verification; Reinforcement Learning; Coverage Closure; FSM

I. INTRODUCTION

The timeless fable of The Turtle and the Hare (Rabbit) has perhaps never been more relevant than it is today in the domain of hardware verification for complex systems. Verification engineers often find themselves balancing the urgency of tight project deadlines with the requirement of exhaustive coverage. The ideal approach would merge the Rabbit's velocity with the Turtle's methodical and analytical focus. But is such a synthesis truly feasible, or, as Aesop's tale suggests, fundamentally contradictory?

The current trends in verification go towards using Universal Verification Methodology (UVM) and constrained random verification. Testing is done randomly in many iterations covering all points which are requested to be covered by the verification plan but also revealing some defects we were not aware of and never targeted to test. Special attention should be paid to tracking configuration settings and what kind of firmware was run [1]. Random tests are running in many iterations as a part of long regressions which can significantly prolong feedback loops. The regression results are analyzed on the pass rate bases and coverage. Pass rate is impacted by the number of pass and failed tests. Failed tests can reveal defects in the design or some issues in the verification environment. Still, this approach takes a lot of time. Random tests can, in theory, achieve full coverage given infinite time. In practice, however, complex designs make this infeasible, as important edge cases may remain untested.

While there are many already in place or just potential applications of Machine Learning (ML) techniques in the Semiconductor industry, this paper is focused on optimizing the time and resources required to achieve coverage closure and functional correctness of the design. The aim is to provide practical improvement that directly addresses one of the critical bottlenecks in the development process. The paper describes a possible approach to enhance the time needed for verification coverage closure by applying machine learning algorithms. The goal is to reduce the time required for coverage closure by integrating reinforcement learning into the test selection process. Once an optimized set of tests capable of achieving full coverage is identified, it can be reused to significantly shorten regression runtime. While this approach is applicable to complex systems, a custom FSM was chosen as a clear and focused example to demonstrate the methodology.

The first part of this paper presents an overview of the challenges in coverage closure of complex systems by using constrained random verification. The second part of the paper demonstrates improvement of the coverage closure process by using reinforcement learning. The idea of using reinforcement learning is evaluated by presenting a verification example of coverage closure of custom FSM. Firstly, the coverage is closed by using constrained random tests in many iterations and the time needed for this is captured. Then, the same coverage closure is achieved by using the proposed approach of using a mixture of constrained random verification and

reinforcement learning approach. The last part is dedicated to the conclusion and proposals for future improvement of this methodology.

II. Related Work

Complex systems are playing an important role in the modern world and decreasing costs and time of verification become imperative in hardware verification. As modern System-on-Chip (SoC) designs continue to grow in complexity, the need for intelligent, adaptive test generation becomes more pressing. Verification remains one of the most resource-intensive phases in the design cycle, accounting for up to 70% of total development time [2], making optimization critical to reducing cost and accelerating time-to-market.

Current industry trends are focused on improving efficiency and significantly reducing the time to market of complex SoCs. One major area of opportunity lies in the time and computational resources required to achieve coverage closure, an area where machine learning can offer substantial benefits. Machine-learning based approaches (reinforcement learning in this case) can significantly improve functional coverage closure time by saving human and processor load. ML-based stimulus generation, for example, has been shown to dramatically improve the efficiency of defect discovery in both hardware and software domains [3].

Electronic Design Automation (EDA) vendors provide advanced verification tools with a lot of useful features, and it is possible that machine learning techniques are employed in their implementation, but that is often not publicly detailed. One of the very useful features in coverage closure provided by different vendors is test ranking. There may be more capabilities available in commercial tools that are not widely documented or accessible. This limits the ability to compare and highlight portfolios of different EDA vendors and to precisely measure the contribution of the proposed solution compared to each one of them. However, to the best of my knowledge, EDA vendors currently do not offer machine learning–based efficiency improvement that is specifically targeting UVM constraint-random verification environments.

Traditional constrained random verification often requires running billions of simulation cycles to achieve sufficient coverage. In practice, coverage tends to accumulate unevenly: approximately 10% of total coverage is achieved by hitting the same bins, while the remaining 90% of bins are rarely exercised, potentially hiding critical bugs [4]. These insights highlight opportunities for improvement in both regression runtime and the efficiency of test selection.

Machine learning can address these challenges by training models—ranging from simple linear regressors to deep neural networks—to predict and prioritize stimuli with a higher probability of achieving meaningful coverage [5]. While the concept of using machine learning to accelerate coverage convergence is not entirely new, many existing methods are complex and tailored to simpler verification environments. Their implementation often involves significant changes to the testbench, including adding Python-based ML layers on top of established System Verilog UVM environments. Added complexity poses challenges to scalability and adoption.

In contrast, the approach proposed in this paper applies reinforcement learning in a lightweight and practical manner, during the post-processing stage of test generation and run. It requires minimal modifications to the testbench, making it easier to integrate into existing environments and more feasible for application to large-scale systems.

III. Baseline: Constrained Random Verification

Constrained Random Verification (CRV) generates randomized test cases while enforcing constraints to ensure the input stimuli meet certain design requirements. Despite its random nature, CRV still offers engineers the ability to bias or guide stimulus generation through constraints or manual overrides, allowing for targeted testing of specific corner cases. Automatically collected functional coverage provides us with a view of the final quality that verification accomplished and a clear overview of what is covered in overall verification. The coverage is a critical point when talking about measuring the effectiveness of the constrained random verification. It reveals untested parts of the design and lets verification and design engineers reconsider the quality of verification or/and quality of the design code. Each time when some hole in verification coverage is revealed, the time-consuming process is

triggered by verification engineers, but other teams are included as well (design engineers, architects, managers). Also, it is worth saying that this can be repeated in different phases of verification.

The first part of the experiment is done by using constraint random verification on a custom FSM for power control with 16 states. This FSM represents the operational behavior of the unit, transitioning between defined states in response to various input conditions or events. Each state corresponds to a specific operational mode or condition. The certain state is covered if FSM accessed that state during the time the test is run. The regression has as many tests as needed till the point when the coverage is collected, and all states of the FSM are covered. External factors such as reset signals, overtemperature events, overvoltage events, and access faults act as influencing stimuli that trigger state transitions or affect state behavior.

One random test with applied stimuli and selection of constraints is run in many iterations. In CRV flow, both stimuli and constraints are selected completely random. The selection of constraints is performed once per test case and may include application of one, multiple, or no constraints at all. It is driven by the value of a random selector variable, which can take any value within its defined range. The random selector is randomized by the verification tool, which internally chooses values without user intervention or bias. However, the stimuli themselves remain randomized and may differ between iterations, even when the same selector and, consequently, the same set of constraints are used. The number of random test iterations required to cover all verification cover points (FSM states) is captured.

The full coverage of FSM states is collected after 54 iterations when CRV verification flow is applied. The Figure 1. presents the iteration index together with the number of new states covered during that iteration. It is important to emphasize that many of these iterations had no contribution to the coverage, and they still spent some resources and consumed the time.

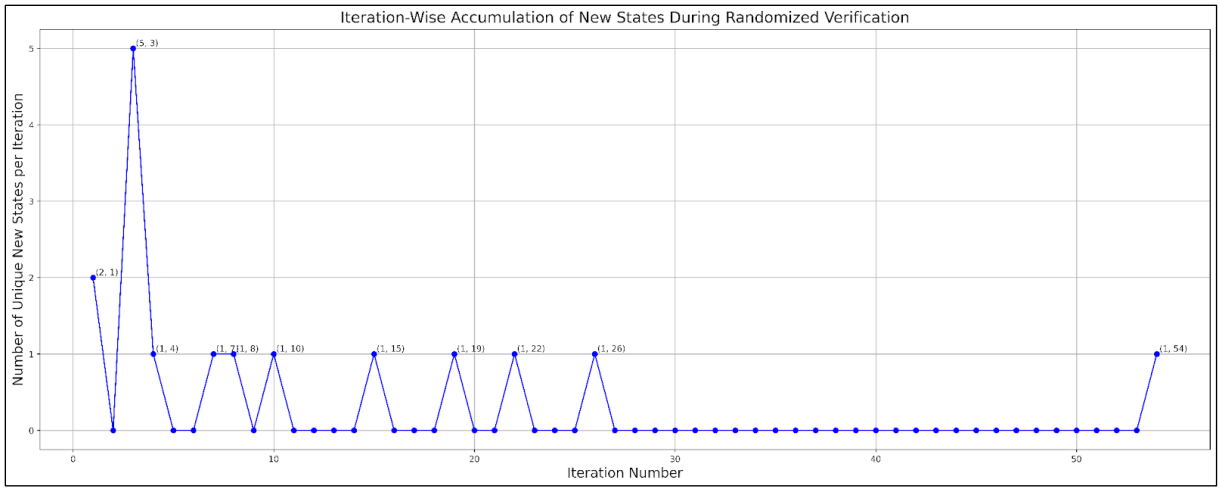

Figure 1. *Iteration-Wise Accumulation of New States in CRV flow*

The relationship between contributing and non-contributing tests in the sense of covering new states is outlined in Figure 2. The percentage of tests which were run in CRV regression without any new state coverage contribution is high and, in this regression, it takes 79.6%.

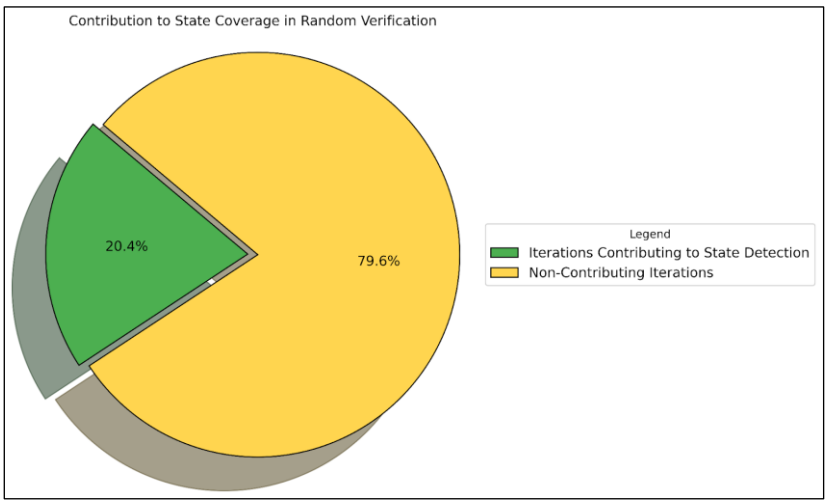

Figure 2. *State Coverage in CRV flow Based on New State Discovery*

This was an opportunity for us to analyze the correlation between the selector value and the number of tests in which a certain state is hit for that selector value. The heatmap is presented in Figure 3. From the heatmap, it is possible to identify which influencing factors most frequently affect specific FSM states. The conclusion is that some states may be highly sensitive to certain factors. For example, if the selector value selects the constraint that highly impacts overtemperature event, that selector value would highly correlate with the coverage of the state in which overtemperature event is handled. Conversely, states with minimal interaction with selector value may represent stable or less complex operational modes.

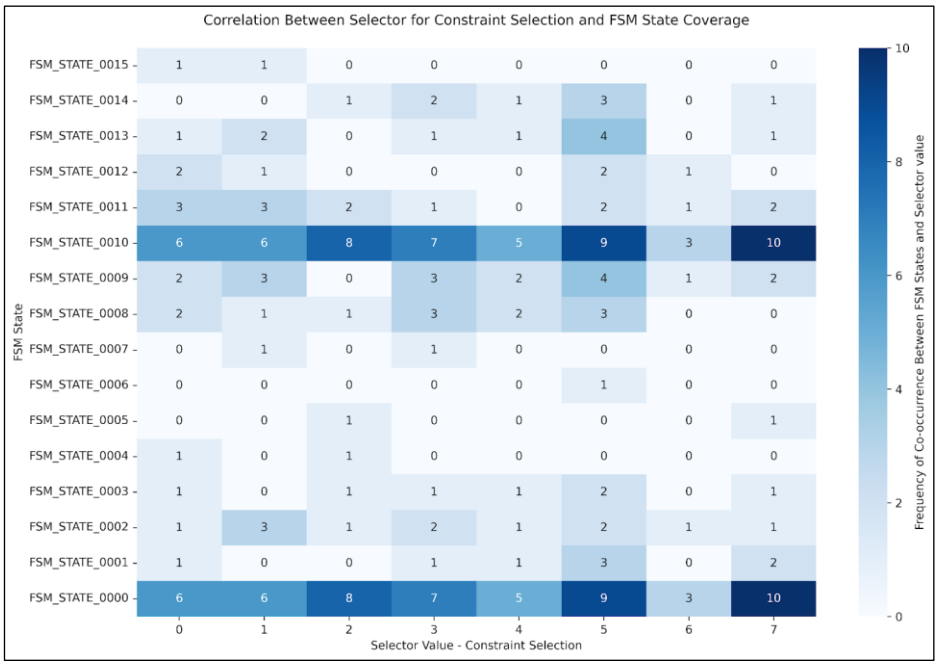

Figure 3. Correlation Between Selector for Constraint Selection and FSM state coverage

The idea of this paper is to take the knowledge we have from the CRV process and use that to improve the efficiency of regression by keeping the quality measured by coverage at the same level. We learned from analyzing CRV results that many iterations (in this case, 79.6% of overall number) introduce overhead without delivering any additional value and that applying some constraints can bring us faster to the wanted result. These are the inputs that can be supplied to Reinforcement Learning flow to make decisions on a subset of tests that will bring the full coverage in an efficient manner.

IV. Reinforcement Learning-Augmented Flow

In today's fast-paced development environment, accelerating the coverage closure process is crucial, as traditional methods often take longer than the market is willing to wait. AI-driven techniques offer a solution by prioritizing test cases based on their likelihood of detecting defects, thus optimizing the allocation of testing resources. Integrating these advanced technologies into Constrained Random Verification (CRV) enables new testing strategies that dynamically adjust the stimuli set, allowing the same results to be achieved in a shorter time frame. Reinforcement Learning (RL) is a type of machine learning where an agent learns how to behave in an environment, by performing actions and receiving feedback in the form of rewards or penalties. The goal of reinforcement learning is for the agent to learn a strategy, called a policy, that maximizes the cumulative reward over time [6]. The part of the logic responsible for decision-making regarding penalization in reinforcement learning is referred to as the Agent.

Reinforcement learning is selected as ML technique of choice because it is closely aligned with verification itself: generate and run tests, collect coverage and analyze, adjust test stimuli and/or environment and repeat the process. It matches the RL loop: State-Action-Reward-New State. CRV already explores a broad input space and that is a valuable feature we want to keep. RL can prioritize test sequences that are more likely to close coverage holes, while still exploring new areas. For Supervised learning, it would be required to have labeled input-output pairs which is not immediately clear how this approach could be implemented in verification. Unsupervised learning techniques have their potential in analyzing and organizing coverage data which would be outside the scope of this paper. However, Supervised and Unsupervised learning are not designed to generate stimuli or drive decision-making in the verification process.

The improvement in verification efficiency achieved by the RL-augmented flow is demonstrated through direct comparison of its regression results with the results gained by standard CRV flow. RL-augmented flow uses Reinforcement Learning Agent that rewards the test each time a new cover point is hit, or a defect is detected. If the stimuli do not result in new coverage or bug detection, the test running those stimuli is penalized. The Agent is implemented in Python and operates in the post-processing phase, analyzing coverage reports. The environment is set as presented in Figure 1. In this setup, the Agent interacts with the environment by taking actions and receiving feedback in the form of rewards or penalties.

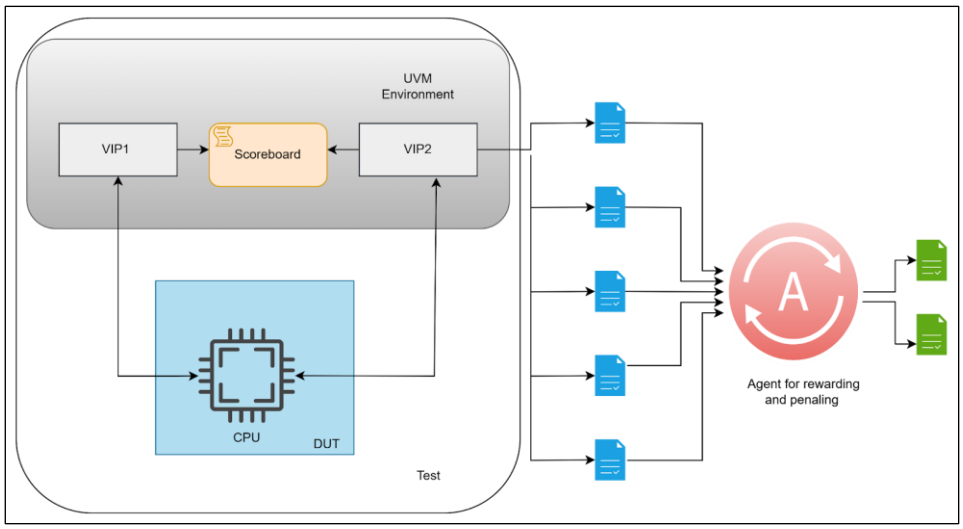

Figure 4. Applying Reinforcement Learning to UVM Verification via Post-Processing

The constrained random verification and RL-augmented flow are two approaches being compared in this paper. In the first flow, all stimuli are completely random, and we calculate the number of iterations which are needed to cover all verification cover points. In the second flow, reinforcement learning is applied to initial random stimuli. After the CRV regression is run, reports for each test are analyzed by RL Agent and rewarding and

penalizing per test is produced as presented in Table I. In the report, we have knowledge about constraints that are applied in the certain test by capturing selector value from that run.

Table I. Result of RL Agent processing — The Part of Test Ranking Report

```
[NEW]         [TEST_1] State: 0000,1010                Constraints: [0, 0, 0, 0] => Reward: 1.0
[NO_NEW_COV] [TEST_2] State: /                         Constraints: [0, 1, 1, 1] => Penalty: -0.5
[NEW]         [TEST_3] State: 0100,1000,1011,1100,1111 Constraints: [0, 0, 0, 0] => Reward: 1.0
[NEW]         [TEST_4] State: 1001                     Constraints: [0, 1, 1, 1] => Reward: 1.0
[NO_NEW_COV] [TEST_5] State: /                         Constraints: [0, 1, 1, 1] => Penalty: -0.5
[NO_NEW_COV] [TEST_6] State: /                         Constraints: [0, 0, 1, 0] => Penalty: -0.5
[NEW]         [TEST_7] State: 0001                     Constraints: [0, 1, 0, 1] => Reward: 1.0
[NEW]         [TEST_8] State: 1101                     Constraints: [0, 1, 0, 1] => Reward: 1.0
[NO_NEW_COV] [TEST_9] State: /                         Constraints: [0, 1, 0, 1] => Penalty: -0.5
```

As a result of this process, there is a list of selector values which has the highest probability that the new state will be covered. RL Agent identified a subset of selector values (0,7,5,3,1) that maximize the probability of exercising previously uncovered states. An RL-guided regression was then executed using this subset. There were 2 RL-Improved regressions ran to be compared with random regression. RL-Improved regression with 3 iterations was run in the way that each selector value was applied to three consecutive random tests before advancing to the next value in the subset. RL-Improved regression with 5 iterations determines that each selector value was applied to five consecutive random tests before changing to the next value.

This controlled sequencing allows the impact of each high-priority selector value to be evaluated under both shorter and longer reuse windows. The results showed significant improvement of the regression efficiency keeping up the same quality observed by coverage results. Regression with 3 iterations resulted with a full coverage after 18 iterations. Regression with 5 iterations resulted with the same result after only 15 iterations. The comparison of all 3 regressions in the iteration-wise accumulation of the new states is presented in Figure 5.

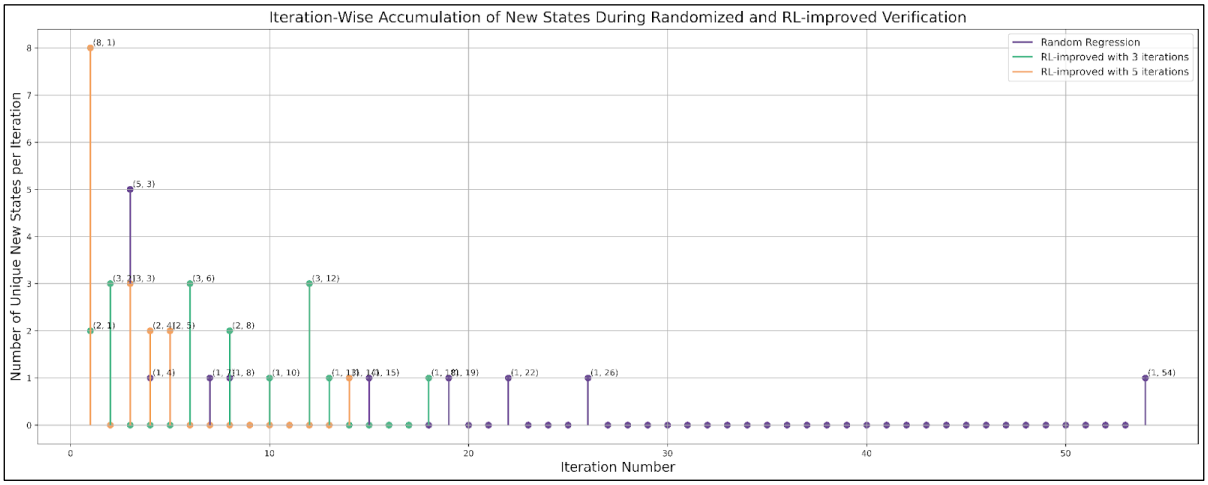

Figure 5. Iteration-Wise Accumulation of New States During Randomized and RL-Improved Verification

The results are analyzed in the sense of coverage contribution iterations and that is presented in Figure 6. There is visible improvement in the regression efficiency when we use the report generated by RL-Agent about the minimal test selection that can be run to have full coverage.

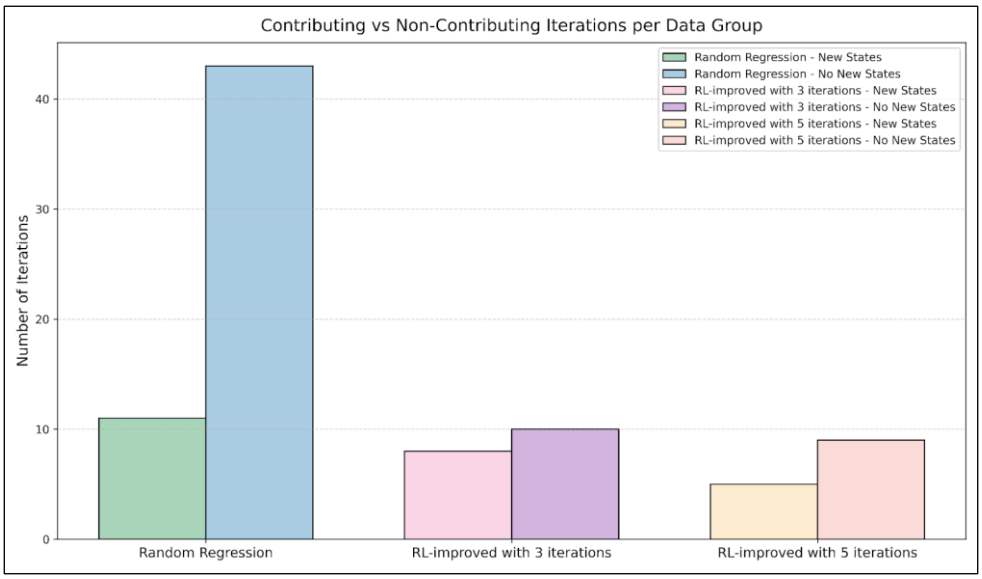

Figure 6. Coverage Contributing vs. Non-Contributing Iterations During Randomized and RL-Improved Verification

The proposed methodology has been demonstrated on a complex FSM, which allowed us to validate the core principles of the proposal. This setup offers clarity and reproducibility; however, it is acknowledged that further work is needed to evaluate the simplicity and scalability of the approach on more complex designs.

The reward model can be extended to include more inputs as part of decisioning criteria like:

- Test runtime - to prefer faster tests
- Critical features - to prioritize tests which are covering critical features
- Diversity- to encourage test variety

The result is particularly important for sanity regressions, which are run before any changes are integrated into the common development branch. Given the value of processor resources and time, we cannot afford to run long regressions multiple times per day. The flow would be to run full random regression and select best tests to have them in regression. Each time the commit is ready to be submitted, the set of selected tests is run to confirm the sanity of the design and verification environment. How many commits to the development branch do we have per a day on large projects? It can be measured in tens or hundreds of commits and considering that each commit will be preceded by sanity regression tells how much resources can be saved by implementing RL constrained random verification methodology on complex projects.

V. CONCLUSION

Just like the Rabbit and Turtle race, where the quick but overconfident rabbit misses key opportunities while the steady turtle maintains consistent progress, Constrained Random Verification and Machine Learning each bring their own strengths to the table. Constrained Random Verification is wide-reaching, much like the rabbit's sprint at the start of the race. However, it can miss the hard-to-reach corner cases.

On the other hand, Machine Learning acts like the turtle — steady, adaptive, and focused on ensuring thorough exploration of every condition, no matter how elusive. By combining both approaches, we merge the breadth of constrained random verification with the intelligence and precision of Machine Learning, ensuring a comprehensive and efficient verification process. This hybrid approach not only accelerates coverage closure time but also ensures that complex systems are rigorously tested, providing a more reliable, safe, and predictable outcome. In essence, this hybrid approach blends the rabbit's speed with the turtle's persistence, delivering a smarter, faster, and more robust verification process keeping it easily applicable and scalable.

REFERENCES

[1] A. Meyer and H. Foster, "Metrics in SoC Verification: Not just for coverage anymore," in DVCon, San Jose, CA, 2013.

[2] A. Grosse, D. Patel, and R. White, “Standing up to the semiconductor verification challenge,” McKinsey & Company, Sep. 2014.

[3] W. Hughes, S. Srinivasan, R. Suvarna, and M. Kulkarni, "Optimizing design verification using machine learning: Doing better than random," in DVCon Europe, 2021.

[4] S. Sokorac, "Optimizing random test constraints using machine learning algorithms," in DVon Europe, 2021.

[5] S. M. Ambalakkat and E. Nelson, "Simulation runtime optimization of constrained random verification using machine learning algorithms," in DVCon U.S. Proceeding, 2018.

[6] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA: MIT Press, 2018.