Beyond Heuristics: AI/ML Driven Verification for Design Sign-off

Gulshan Kumar Sharma
Samsung(SSIR)
gks.92@samsung.com

Sougata Bhattacharjee
Samsung(SSIR)
sougata.b@samsung.com

Abhishek Raj
Samsung(SSIR)
r.abhishek@samsung.com

Avinash Bollu
Samsung(SSIR)
b1.avinash@samsung.com

Akshaya Kumar Jain
Samsung(SSIR)
akshaya.jain@samsung.com

***Abstract*- Functional verification consumes a major portion of semiconductor development effort, particularly at the IP/block level, where regression campaigns run thousands of tests daily. Traditional strategies such as brute-force execution or heuristic-based ranking and resequencing improve efficiency but remain static, manual, and limited in adaptability. These methods often waste computing resources, delay coverage closure, and postpone the discovery of corner-case bugs. In this paper, we present an Al/ML-driven regression optimization methodology that leverages historical regression data to predict test utility, dynamically prioritize execution, and continuously adapt to evolving verification goals. The flow integrates seamlessly into existing UVM-based regression infrastructure and creates a closed feedback loop that re-trains models with each iteration. Compared to heuristic-based flows, the proposed solution achieved 65-70% lesser regression time with respect to traditional flow and 25-30% faster regression turnaround with respect to ranking based approach. We achieved ~25% faster coverage closure and design sign-off in IP-level verification environments. These results demonstrate the practical value of applying Al/ML to verification, establishing a scalable foundation for IP, subsystem and SoC-level adoption.**

I. INTRODUCTION

Verification continues to dominate the overall effort in semiconductor design, often accounting for 60-70% of project cost and schedule. Even at the IP/block level, regression campaigns can involve thousands of tests and millions of simulation cycles, leading to significant challenges in compute efficiency, coverage closure, and bug discovery.
Three recurring challenges dominate:

A. Regression Efficiency: A significant fraction of regression cycles is wasted on redundant tests, slowing down verification closure.
B. Coverage Maximization: Achieving functional, code, and assertion coverage is slow, often requiring multiple iterations of manual gap analysis and test reruns.
C. Bug Hunting: High-value corner-case bugs are frequently detected late, driving costly debug cycles and increasing silicon risk.

Traditional regression optimization techniques, such as ranking and resequencing prioritize tests using heuristics, including failure recency, coverage contribution, and runtime. While effective in improving efficiency compared to brute-force execution, these methods are static and manual. They cannot adapt to daily changes in RTL design, evolving coverage goals, or shifts in regression content.

This limitation motivates the application of Al/ML-driven approaches to verification. Abundant regression data is already available in the form of logs, coverage metrics, bug history, and execution reports. By applying machine learning to this data, we can build predictive models that estimate the value of each test, enabling an adaptive and continuously improving regression methodology.

To validate the effectiveness of the proposed methodology, extensive experiments were conducted on industry-grade IP verification environments. The results showed a significant reduction in overall closure time, improved functional coverage convergence, and fewer redundant test iterations. Moreover, the AI/ML engine was able to highlight areas of interest that were previously overlooked, enabling earlier detection of corner-case bugs and reducing late-stage surprises.

Another key contribution of this work is its adaptive learning capability, which means the model keeps learning as it is used. As the design and verification environment changes—such as updates to the RTL, verification tools, or testbench—the system does not remain static. Instead, it continuously learns from each iteration and updates its AI/ML models accordingly. This adaptive learning approach allows the system to improve over time and remain effective without requiring repeated manual retraining. This adaptability ensures that the verification closure process remains relevant, effective, and forward-looking, even as project dynamics change.

This paper presents a practical implementation of an adaptive learning framework for verification. The framework employs supervised machine learning, where the model is trained using historical test data along with known outcomes (such as whether a test improved coverage or exposed a bug). By learning from these labelled examples, the model predicts the usefulness of each test—specifically, its likelihood of uncovering new coverage or detecting bugs—and prioritises high-value tests earlier in the regression process.

Unlike earlier approaches that rely on one-time training or vendor-specific automation, the proposed methodology introduces several key features. It supports continuous retraining after each regression, allowing the model to improve as more labelled data becomes available. The approach is independent of any commercial ML platform, ensuring portability and flexibility. It also incorporates a built-in cold-start strategy to handle new IPs with little or no historical data.

Finally, the framework enables seamless integration with standard UVM-based regression infrastructure, requiring minimal changes to existing verification workflows.

Table I compares the different verification approaches and the proposed AI/ML methodology based on implementation metrics.

TABLE I. Comparative Analysis between Verification Approaches

Aspect	Traditional	Ranking	AI/ML (Proposed)
Regression Efficiency	Complete regression running	Relies on manual test selection, also removes redundant runs	ML model prioritize high value tests, reducing redundant runs
Coverage Closure	Takes long time	Faster but manual tuning	Faster & Automated
Stressed Testing	Manual	Manual	Automated
Resource Utilization	Poor	Moderate	Optimum
Adaptability	Low	Moderate	High, Self-learning

II. RELATED WORK

Regression prioritization and optimization have been widely studied in both software and hardware verification. Early work by Rothermel et al. [5] formalized regression test selection, laying the foundation for ranking-based strategies. In the hardware domain, Yim et al. [6] and Jang et al. [7] demonstrated process automation and data-driven verification planning, while Automated Seed Selection [8] and Novelty-Driven Verification [9] used ML to identify high-value tests and novel coverage bins.

More recent research [7][8][9][10] introduced supervised and reinforcement learning for regression throughput optimization, reporting 40–70% savings in simulation effort. However, most prior methods trained models offline, lacked self-adaptive feedback, or were closely tied to specific commercial ecosystems.

Our approach builds upon these foundations but adds a self-learning regression optimizer that operates in a continuous feedback loop, retrains incrementally, and maintains tool independence. It further demonstrates industrial-scale deployment at IP level and extrapolates scalability to subsystem and SoC verification.

III. PROPOSED METHODOLOGY

Figure 1 illustrates the proposed AI/ML assisted workflow for accelerating verification closure in UVM based environments. The proposed methodology leverages machine learning to analyze the coverage metrics data, create intelligent test sequences, and iteratively improve the verification regressions effectiveness.

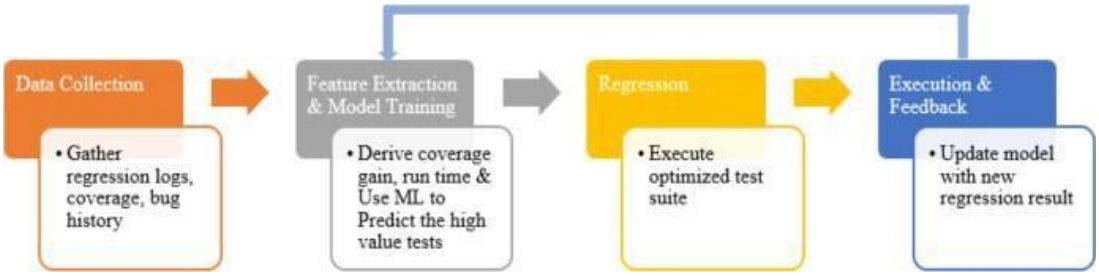

Figure 1. Proposed Methodology Flow

A. Methodology

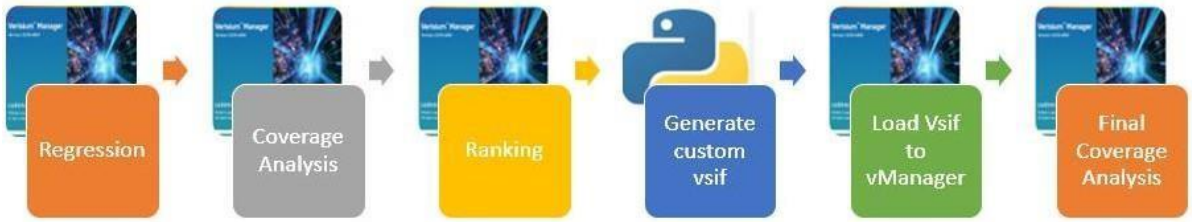

Figure1.1 Ranking & Resequencing Flow

Before introducing the AI/ML approach, a ranking-based regression flow Figure 1.1 was implemented as a baseline for comparison. In this heuristic method, each test is assigned a static priority score based on factors such as past coverage gain, failure recency, and runtime. Tests with higher scores are executed first to accelerate coverage closure. While this method improves efficiency compared to brute-force execution, it relies on manually tuned weights and static thresholds. As design functionality and coverage goals evolve, these rankings quickly lose relevance, requiring periodic human recalibration. This limitation motivated the transition toward a data-driven, self-adapting AI/ML framework.

The implementation flow is built around 5 key stages:

1. Data Collection

   Each regression generates metadata including:

   Test and seed identifiers, Simulation outcome (pass/fail), Runtime and resource metric, Coverage delta (per test and per seed)

   These logs are collected automatically via regression scripts. The data is cleaned to remove corrupted entries, and coverage deltas are normalised relative to the total number of functional bins.

2. Feature Engineering

   From the raw data, a compact yet informative feature set is derived:

   I. Coverage novelty factor: Proportion of new bins hit by the test.
   II. Failure likelihood: Historical probability of test failure.
   III. Runtime efficiency: Coverage-per-second ratio.
   IV. Execution recency: Time since the last run.
   V. Seed diversity index: Entropy-based measure of seed outcomes for the test.

   Each test record becomes a vector of numerical and categorical features representing its historical verification value.

3. Model Families and Training Workflow

   To ensure portability, training is implemented using open-source ML frameworks (e.g., scikit-learn). Three model families were evaluated:

   I. Logistic Regression – Baseline linear model for interpretability.
   II. Random Forest (RF) – Non-Linear ensemble offering robustness)
   III. Gradient Boosting (GBM) – High-accuracy model for complex feature interactions.

Models were trained on 20,000–25,000 regression samples per IP using stratified 5-fold cross-validation. Random Forest (ML technique that makes decisions by combining the results of many simple models instead of relying on a single complex model) achieved the best trade-off between interpretability and performance, with precision and recall exceeding 0.9 for high-value test prediction.
Retraining occurs automatically after every 2–3 regression iterations, using the most recent logs. Model metrics (accuracy, F1 score) are tracked via the same regression dashboard to ensure transparency and prevent overfitting

4. Test Utility Prediction and Prioritization
   For each upcoming regression, the trained model predicts a utility score (U) representing expected coverage or bug yield per test:

$$U_i = f(C_{novelty}, F_{prob}, R_{eff}, S_{entropy}, T_{recency})$$

   Thresholds are dynamically adjusted to maintain a minimum coverage confidence level of 95%.

   Low-utility tests are deprioritized but not deleted; they remain candidates for occasional exploratory runs or when coverage stagnation is detected.

5. Regression Execution and Feedback
   The prioritized test list is fed into the existing regression scheduler (unchanged infrastructure). After each run:
   I. Results are appended to the regression database
   II. Coverage and failure updates are extracted
   III. The model restrains incrementally using the new data

This creates a self-correcting feedback loop, where the AI continuously refines its understanding of what constitutes a high-value test.
While Cadence SimAI was used to automate log extraction, all ML processing and regression planning are performed via a custom pipeline. This ensures portability: any verification team can reproduce the flow using their own simulator or coverage tools. The architecture requires only standard coverage reports and test result logs as inputs.

B. Application of Proposed methodology

The proposed AI/ML-driven regression methodology can be applied across multiple levels of verification:

A. Coverage Maximization
   Targeting functional and code coverage closure earlier by learning which tests contribute most effectively to specific holes.
B. Bug Hunting
   Prioritizing tests with high failure probability increases chances of catching corner-case issues earlier.
C. Continuous Verification
   The adaptive feedback loop makes it especially suitable for nightly regressions, where learning from previous runs can guide the next day's execution plan.

II. RESULTS

The proposed methodology is implemented leveraging cadence SimAI tool.

A. Test count optimization:

The proposed framework is implemented on 2 Multimedia IP's. The test count reduction after implementation is shown in Figure 2 and Figure 3. IP1 have 814 tests (including seeds) and IP2 has 24000 test. It clearly shows that the test count reduced around ~65% for both IP1 and IP2, which is a significant number of reduction in tests to be run. Overall regression efficiency and resource utilization will improve because of this optimization.

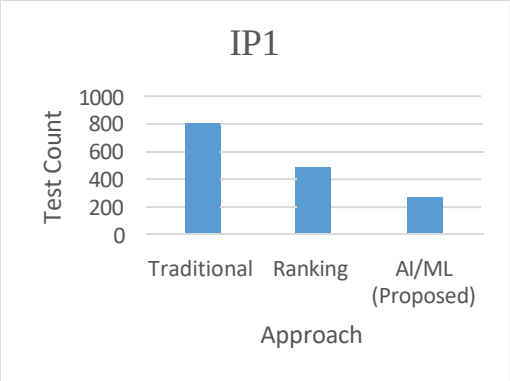

Figure 2. Test count reduction IP1

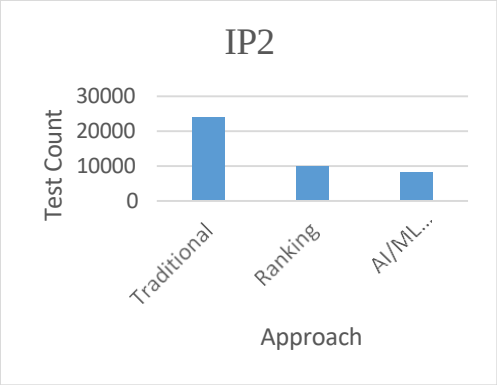

Figure 3. Test count reduction IP2

B. Regression Efficiency:

Figure 4 and Figure 5 shows the efficiency improvement in terms of reduction in regression time.

Regression runtime reduced by ~71% vs random execution, ~31% vs ranking/resequencing for IP2 and ~66% vs random execution for IP1.

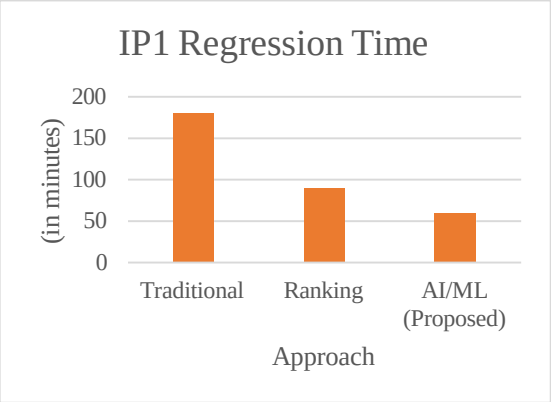

Figure 4. Regression time efficiency IP1

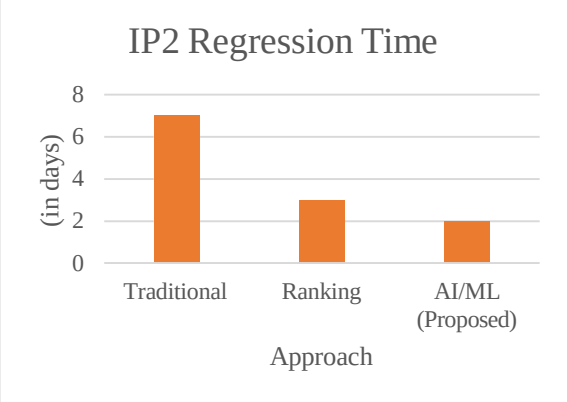

Figure 5. Regression time efficiency IP2

C. Coverage Closure:


```
{
    "Original dataset stats": {
        "Number of bins": 448509,
        "Number of runs": 814,
        "# bins not covered": 139510,
        "# bins covered": 308999,
        "# bins always hit": 44093,
        "# rare bins (<0.05)": 53742,
        "Total CPU time [hrs]": 52.944
    },
    "Result dataset stats": {
        "Number of bins": 448509,
        "Number of runs": 244,
        "# bins not covered": 160894,
        "# bins covered": 287615,
        "# bins always hit": 186416,
        "# rare bins (<0.05)": 33709,
        "Total CPU time [hrs]": 11.323
    },
    "Comparison": {
        "Target coverage": 308999,
        "New coverage": 287615,
        "# new hits": 1261,
        "# regained bins": 286354,
        "# NOT regained bins": 22645,
        "Num rare bins in original dataset": 53742,
        "Num regained rare bins": 33240
    }
}
```


Figure 6. Regain result analytics IP1

After running the generated optimized vsif, we achieved 92.7% coverage for IP1, by regaining 286,354 out of 308,999 target bins. This reflects strong coverage closure progress for the design.

Achieved similar coverage with ~67% fewer seeds, 71% less time vs Random execution and 18.5 % fewer seeds and 33% less time vs Ranking methodology results for IP2.

D. Hard to cover bins:

It learns and creates vsif to target uncovered bins. Regain or covered bins increases with each iteration. This iterative process leverages coverage feedback to generate targeted stimulus, improving hit efficiency. As illustrated in figure 7, the number of newly hit original holes grows progressively.


```
{
    "num_runs_init_iter": 814,
    "num_covered_bins_init_iter": 1348672,
    "newly_hit_original_holes_count": 1416,
    "newly_hit_original_holes": [
        "Instances/top/dut/GIC400_WRAPPER/GIC400/u_gic_distributor/g_spis[2]/u_spi/42/1/4",
        "Instances/top/dut/GIC400_WRAPPER/GIC400/u_gic_distributor/g_spis[2]/u_spi/43/1/8",
```


Figure 7 Learning based approach for covering hard to hit bins

E. Resource Utilization:

As the redundant tests are removed from the generated optimized test suite, the compute resource requirement also comes down.

F. Analysis efficiency improvement with Analytics:

The interactive plot shown in Figure 8 provides a 2D view. By selecting a specific point in the plot, the corresponding hit ratio along with the test and its seed can be visualized, enabling a detailed analysis of coverage distribution

G. Cold-start Behaviour:

For new IPs with minimal history, the system begins with heuristic ranking for 3–5 regressions. Once ~1,000 regression samples accumulate, the ML model is initialized. Using feature normalization and prior models from similar designs, coverage and runtime efficiency stabilize after the first week, indicating strong transfer learning capability.

H. Generalization Across Architectures

The same framework was applied (without retraining code changes) to a protocol IP and an interconnect IP. Even with different coverage models and runtimes, the regression reduction trend (60–70%) held, validating feature-domain independence.

Preliminary experiments on subsystem-level regressions show 2–3× faster throughput due to overlapping test coverage among integrated IPs.

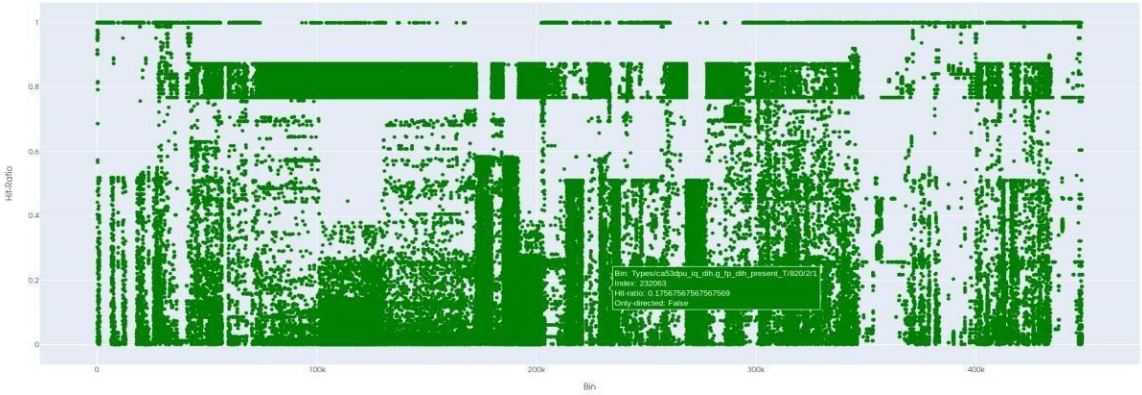

Figure 8. Bins Hit Ratio Analytics

These results demonstrate that AI/ML-driven flow provides tangible improvements in verification efficiency and effectiveness.

III. CONCLUSION

AI/ML-driven verification represents a significant advancement beyond static test strategies. While ranking and resequencing improve upon brute-force regression, they remain rigid and manual. The proposed approach delivers measurable improvements in regression efficiency, coverage closure, and bug hunting. Beyond immediate benefits in simulation cost and time, the framework establishes a scalable foundation for applying AI/ML across subsystem and SoC-level verification. This work demonstrates that integrating machine learning into verification flows is not just an optimization, but a necessity for future complex designs.

REFERENCE

[1] Seonghee Yim, Hanna Jang, Sunchang Choi and Seonil Brian Choi, "Accelerating SOC Verification using Process Automation and Integration" Design Verification Conference (DVCon), 2020

[2] Hanna Jang, Seonghee Yim, Sunchang Choi, Seonil Brian Choi, "Machine Learning Based Verification Planning Methodology Using Design and Verification Data" Design Verification Conference (DVCon),2022

[3] David C, Brian C, Jim S, and Brandon S, “Automated Seed Selection Algorithm for an Arbitrary Test Suite”, in DVCON US 2018

[4] Tim B, Rhys H, Sebastian S, R. Nicole, “Novelty-Driven Verification: Using Machine Learning to Identify Novel Stimuli and Close Coverage”, in DVCON US 2021

[5] G. Rothermel and M. J. Harrold, “A Safe, Efficient Regression Test Selection Technique,” ACM Trans. Softw. Eng. Methodol., vol. 6, no. 2, pp. 173–210, 1997

[6] S. Yim, A. Sinha, and J. Lee, “Accelerating SoC Verification Using Process Automation and Integration,” DVCon U.S. Proc., 2018.

[7] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, “Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration,” Proc. IEEE/ACM Int. Conf. Softw. Eng. (ICSE), 2018.

[8] A. Sharif, D. Marijan, and M. Liaaen, “DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing,” IEEE Trans. Softw. Eng., 2021.

[9] J. Zhang, Y. Liu, M. Gligoric, and A. Shi, “Comparing and Combining Analysis-Based and Learning-Based Regression Test Selection,” Proc. IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE), 2019.

[10] S. Arun, P. Lakshmikanthan, A. Aggarwal, and S. Ananthakrishnan, “Methodology for Verification Regression Throughput Optimization Using Machine Learning,” DVCon India Proc., 2021.