SvedbergOpen
DISSEMINATION OF KNOWLEDGE
International Journal of Artificial Intelligence and Machine Learning
Publisher's Home Page: https://www.svedbergopen.com/
Research Paper Open Access

Scalable UVM Frameworks Augmented With ML For Multi-Protocol Storage Soc Verification

Amit kumar

Independent Researcher, USA

Abstract
In the modern semiconductor industry, storage system-on-chip designs must support multiple high-speed protocols, including PCIe, NVMe, SATA, and NAND flash interfaces with increasingly complex design topologies. Customary Universal Verification Methodology (UVM) frameworks continue to face important limitations, as they do not consider the exponential growth of design state spaces, protocol interplays, and system-level integration scenarios in modern storage controller designs. Other suggested extensions to customary verification methodologies include smart test generation from a reinforcement learning agent, automated coverage optimization and clustering from machine learning techniques and predictive analytics, and discovery of cross-protocol interactions through graph neural networks and sequence-to-sequence modeled topologies. Coupled with existing UVM base classes, a machine learning inference engine (or engines) can be selectively instantiated, preserving the modularity and reusability intrinsic to the UVM methodology while introducing artificial intelligence to the production verification flow. While practical benefits will be realized beyond pilot programs by addressing compute overhead management, training data collection and curation, interpretability and machine learning integration within legacy electronic design automation (EDA) tool frameworks, machine learning methods mapped to Universal Verification Methodology (UVM) frameworks represent state-of-the-art incremental steps in the progression of semiconductor verification in improved coverage ratio, bug discovery efficiency and resource efficiency backward compatible to legacy flows to help verification teams combat the verification complexity crisis of next-generation storage system-on-chip design.

Keywords: Machine Learning-Augmented Verification, Universal Verification Methodology, Multi-Protocol Storage Controllers, Reinforcement Learning Test Generation, Cross-Layer Protocol Verification

1. Introduction

As the number of high-speed communication and control protocols integrated into a storage system on a chip (SoC) has increased, contemporary storage controllers integrate multiple protocols such as PCIe, NVMe, SATA, and NAND flash into a single semiconductor device, creating the most complex verification environments ever, posing new verification challenges. There are statistically important industry studies that disclose that the functional verification phase has emerged as the major bottleneck in the semiconductor development cycle, and this is substantiated by studies that show that functional verification activities consume a majority of the design effort, posing a major challenge for organizations that need to meet tight time-to-market deadlines. The increasing design complexity has resulted in systems whose verification environments need to not just address protocol-level correctness but also system-level interactions, power, and performance aspects across various conditions.

Machine learning (ML) technologies applied to hardware verification are considered a new approach to the customary verification scalability problem that has limited conventional techniques. Machine learning techniques can be applied to many aspects of the hardware verification process and can be used to automate tasks customarily performed by skilled human verification engineers. Early research on Bayesian networks and ML for verification demonstrated that statistical learning can be used to guide test generation in unexplored areas of the design space, which has since inspired further machine learning research. It has been shown that statistical models can be used to model complex relationships between test stimuli and coverage results and provide a mathematical framework for clever test generation that goes beyond using only random or human-guided test stimuli.

The storage controller domain is particularly amenable to ML-based augmentation due to the deterministic nature of protocol specifications and the large number of states and protocol interactions, with many modern storage SoCs containing advanced features such as hardware cryptographic engines, end-to-end data protection, QoS arbitration logic, and complex power management state machines. The verification of any hardware system to ensure it operates correctly has extended past functional correctness and now also includes such characteristics as performance in varying workloads, power consumption in varied operating modes, and security features for cryptographic and data path integrity algorithms. By blending established UVM frameworks with AI, practical solutions to verification problems increasingly intractable with conventional methods are beginning to emerge, providing semiconductor verification with the means to continually scale along with the ever-increasing complexity of designs being realized in the market.

Table 1: Verification Methodology Adoption and Challenges [1,2]

Aspect	Traditional UVM Characteristics	ML-Enhanced Capabilities
Industry Adoption	Widespread usage across semiconductor companies for complex ASIC and SoC projects	Incremental integration preserving established infrastructure
Verification Effort	Consumes the substantial majority of the total design development cycle	Reduces cycle time through intelligent automation
Coverage Direction	Manual and constrained-random approaches	Statistical learning methods directed toward uncovered spaces
Test Generation	Engineer-intensive development requiring extensive experience	Automated learning from coverage feedback
Scalability	Limited by exponential state space growth	Addresses complexity through adaptive algorithms

2. Architectural Framework for ML-Enhanced UVM Verification Environments

The challenge of integrating ML capabilities into UVM frameworks is to do so while maintaining the modularity and reusability that have helped UVM become the de facto industry-standard verification methodology. Therefore, the architecture needs to bbe builton existing UVM component hierarchies while maintaining compatibility with years of production experience built around well-known verification flows. Surveys of industry methodologies indicate that UVM has reached industry-standard levels of usage. UVM-based methodologies are already the most common ASIC and SoC verification methodologies. Therefore, utilizing machine learning in conjunction with the industry's existing UVM methodology has become a best practice. The verification industry has made important investments in UVM-based testbench architectures, libraries of

reusable verification components, and UVM education. Thus, backward compatibility and incremental adoption are key requirements for any machine learning enhancement of UVM-based verification [1].

The proposed machine learning architecture adds inference engines to the existing UVM architectural elements used in verification. This new architecture allows verification teams to plug machine learning into their existing UVM environment. The inference engines communicate with the legacy UVM environment through APIs. As a result, the verification team was able to use machine learning, which offers maximum benefit, and continue to use customary methods elsewhere, thanks to the layered architecture. Statistical learning has been applied to coverage-directed test generation. These studies suggest that test generation can be guided by learned models that describe what parts of the design under verification are covered and what parts are not, demonstrating the technical feasibility of using learning algorithms in verification feedback loops with little added complexity to the verification architecture [2]. Production verification processes are increasingly adopting learning-based methods. This work has validated the concept of augmenting rather than replacing customary verification processes, and it is clearer that constrained-random generation can be more powerful when combined with a smart learning-based direction than either method used alone.

These protocol-specific verification agents (such as PCIe or NAND flash) require domain-specific machine learning augmentation strategies to cater to different protocol characteristics and unique verification challenges. For PCIe, for example, it is essential to consider link training sequences, power management transitions, error detection and recovery mechanisms, and transaction-level ordering requirements across a wide range of operational scenarios. PCIe protocol specification focuses on complex state machines that must be verified during link initialization, power state transitions, and underlying system architecture when they need to conform to protocol semantics. NAND flash verification focuses on wear leveling algorithms that must distribute write cycles evenly across a large number of physical flash blocks to improve device lifetime, ECC logic adapted to the increasing number of errors in data as flash cells age, and FTL logic that creates the logical abstraction of a block addressable storage device while managing flash erase before write restrictions and garbage collection logic. On the architecture level, protocol-specific machine-learning models are deployed autonomously, while a common knowledge repository is used to acquire and share cross-protocol dependencies and system-level verification cases leveraging protocol interactions, which are required to ensure the correctness of each protocol and the interactions of protocols in the complete storage controller implementation.

Table 2: Reinforcement Learning for Test Generation [3,4]

3. Machine Learning Techniques for Intelligent Test Generation and Coverage Optimization

Using machine learning toolkits for test generation is an evolution of constrained-random and directed testing methodologies that have existed in hardware verification for decades. Reinforcement learning algorithms provide a well-defined framework for formalizing test generation as a sequential decision-making task in which a software agent learns to identify efficient policies for generating tests by interacting with a verification environment and receiving rewards for covering more of the design under test and identifying bugs. Further work on the application of deep reinforcement learning to hardware verification has shown that using neural networks as agents can allow test generation policies to learn to direct test generation to regions of the design state space that are not being well tested, rather than randomly sampling the state space [3]. The reinforcement learning framework also matches the iterative nature of hardware verification well since the agent can treat the test results as feedback information to guide further test generation.

Test case generation in reinforcement learning is typically driven by multi-objective reward functions that take into account multiple verification objectives, including functional coverage reachability, heuristic or historical likelihood of a bug discovery, or computational resource efficiency. More advanced reinforcement learning architectures support the trade-off between exploring new or under-explored design states vs. Furthermore, when learning test generation strategies that often achieve high coverage or bug-finding rates in prior verification case studies, more complicated forms of reward shaping, which provide intermediate rewards for achieving incremental progress towards verification goals, allow for faster learning and for reinforcement learning agents to discover more effective test generation strategies to support verification than would otherwise be possible with the naively structured rewards that are often too sparse for reinforcement learning to yield solutions in reasonable time frames [4]. Successful reward functions for hardware verification tasks

are difficult to design because they require an understanding of machine learning and hardware verification, as a poorly constructed reward function may, in some cases. After all, agents exploit strategies that maximize the reward function but do not result in an improvement in another useful metric.

ML coverage optimization includes efficient test generation that optimizes code coverage as well as post-execution analysis of test suites to identify redundancies, predict saturation points, and dynamically allocate computational resources between multiple verification goals competing for limited resources. Clustering and dimensionality reduction can group large test suites into classes of tests that exercise similar design functionality, enabling clever test suite reduction that eliminates redundant tests while maintaining quality test coverage of the design under verification. Machine learning techniques applied to coverage analysis enable the prediction of upcoming progress rates by leveraging the current coverage state and verification rate using predictive statistical models built from existing coverage data. This leads to providing verification teams with useful quantitative information for planning their verification. These predictive models for coverage information allow a team to know how to allocate its verification resources optimally, and this is a meaningful advance over customary coverage analysis techniques that provide just summary statistics about current coverage but supply no information on how fast future coverage may grow and which verification activities will best improve coverage toward completion.

4. Multi-Protocol Interaction Verification and Cross-Layer Dependency Analysis

NLP techniques have been explored for the automatic verification tasks, such as to extract verification information and generate verification artifacts (including testbenches) from specifications, design documents, and issue reports. The transformer architecture for language models has had a transformational impact on NLP across a wide variety of tasks and can be used to analyze the documents (e.g., hardware specifications and design documents) written in a domain-specific language. NLP applications for specifications include automatic extraction of verification requirements, detection of ambiguous or contradictory statements within specification documents, specification error detection, and specification disambiguation. Another important application is building an initial verification plan based on a natural language specification, which summarizes the major verification scenarios described in the specification. Other papers have studied other NLP tasks to bridge between human-readable specs and machine-usable verification code, showing that language models can learn some hardware design concepts and can generate verification artifacts that are meaningfully aligned with the specs when trained on technical domain-specific data. Human revision and editing are often still necessary to create programmatically correct and complete verification materials.

The verification of storage SoC designs is complicated due to cross-layer protocol interference and the need to validate system-level properties that cannot be checked by merely verifying individual protocol layers. For example, a storage controller with a PCIe host interface and a NAND flash memory interface must tightly coordinate these two domains' transaction flows. It must also ensure that PCIe transaction layer packets will always trigger the appropriate NAND operations and will generate PCIe completion responses in a manner that satisfies protocol ordering semantics, error recovery, and performance specifications. Therefore, it is not sufficient to rely solely on customary protocol verification methods, but it is necessary to address cross-protocol dependencies and high-level integration scenarios that would not be adequately exercised by testing single protocol interfaces in isolation. Industry surveys of functional verification practices have identified system-level verification and multi-domain integration validation as being two of the hardest problems in SoC verification, with integration bugs discovered late in the verification flow or during post-silicon validation often being due to insufficient verification of cross-protocol interactions in pre-silicon verification stages [1].

Machine learning techniques are well-suited for discovering and exercising cross-protocol interaction scenarios by automatically training models to learn relationships between events from different protocol domains. Sequence-to-sequence learning architectures achieved state-of-the-art results in several tasks that require accurately modeling the complex temporal relationships between the input and output sequences. They provide a natural fit for the task of learning mappings between protocol events in multiple communication interfaces. Active research into applying ML techniques to cross-layer protocol verification has concentrated on graph representations of protocol-layer architectures, where nodes represent functional units that implement protocol functionality and edges represent communicating and dependent protocols [5]. These protocol-layer architecture graphs can be directly used as input to graph neural networks. The model could learn to verify integration properties of the protocol architecture, such as deadlock in request-response protocols due to circular dependencies, flow control bottlenecks due to buffer capacity, or credit management

mismatches between protocol boundaries, as well as error propagation through the protocol architecture that conforms to the error handling semantics of the protocol layers in question.

Testing of QoS functions and bandwidth arbitration policies across multiple communication protocols requires knowledge of the performance characteristics of the protocols and the generation of test workloads to expose fairness violations, starvation, and priority inversion problems. Storage controllers use various complex arbitration policies based on many factors, including competing demands of host-initiated PCIe interface reads and writes, and background and internal controller-initiated processes such as GCE and wear. Access to one or more NAND flash components and controller-level error scrubbing and refresh cycles are leveled. Sensors and neural network (NN) architectures can find useful patterns in time-series data for abnormal detection in temporal data streams. Specifically, recurrent neural networks (RNNs) and transformer architectures can successfully identify small anomalies in the expected behavior of systems and processes with complex temporal dynamics [6]. It can be usedto analyzee time-series data analysis techniques to analyze irregularities in latencies, throughput, and QoS failures, helping to detect subtle bugs in the arbitration or resource management algorithms.

Graph-based representations of systems have been combined with machine-learning-based algorithms to allow reasoning about system properties that would customarily be impossible or expensive to realize via simulation. The hierarchical structure of functional blocks, the communication between components, and the dependencies between the modes of operation of a multi-protocol system-on-chip can be formalized as graph structures and analyzed using machine learning algorithms, such as graph neural networks. The study of machine learning techniques for manufacturing systems and their applicability to hardware verification problems, which share similar characteristics including high-dimensional state spaces and complex dependencies between components, has shown that graph-based models and machine learning techniques can discover optimization opportunities and failure conditions via analysis that is not achievable using other analysis techniques [7]. For hardware verification problems, similar graph-based Machine learning techniques can be applied to automatically discover potential deadlock conditions by learning from circular dependency graphs, predicting performance bottlenecks through the analysis of bandwidth capacities and traffic patterns on interconnect edges, and verifying the completeness of error handling schemes by automatically tracing paths of error propagation through the system to determine whether error handling is defined for all possible error types along those paths in each architectural element.

5. Implementation Challenges and Practical Deployment Considerations

The computational overhead of machine learning inference is arguably the most important practical issue facing the adoption of machine learning-augmented verification flows for production semiconductor design and testing, and could restrict such flows to academic research prototypes and experimental pilot programs. Since verification flows typically run millions of simulation cycles to meet coverage and bug-hunting goals, even small amounts of additional time per transaction may be impractical. Some work has been done to identify architectures of machine learning components in verification flows that ensure the simulation throughput remains on pace for verification [8]. Another challenge associated with machine learning components in verification flows is balancing the improvements to verification from employing machine learning techniques against the overhead associated with performing inference from these techniques. Different machine learning models also make trade-offs in terms of their complexity, inference latency, and how well they improve verification performance.

A major problem in the practical application of machine learning is the amount and quality of data that is needed for a machine learning model to be trained to a level of effectiveness. This problem is particularly pronounced for projects that do not have a rich amount of historical verification data or past projects that are similar enough to provide data to learn from, such as novel design categories. Machine learning, particularly deep learning, relies on large neural networks with millions of tunable parameters that require enormous datasets. The industry is aware of the time and cost associated with collecting, curating, and labeling data for chip design settings. They state that obtaining suitable training datasets is among the most high-cost aspects of machine learning initiatives [9]. The verification space poses a somewhat different problem, as it needs to capture corner cases and bug manifestations that happen infrequently with random simulations but on which the verification effort is focused. This careful curation ensures that the training data is representative of all verification-relevant cases and accurately reflects the problems encountered in actual verification, avoiding bias toward the most common, unrepresentative cases.

Interpretability and debuggability of verification decisions made by machine learning agents is a topic where the interests of machine learning and customary verification engineers collide, because customary verification relies on the traceability and auditability of verification activity. When a machine learning agent generates a sequence of tests that exposes a bug or achieves coverage of a previously unreachable design state, verification engineers would like to understand the reasons behind the agent's decisions to maximize learning and to verify that the agent does not achieve success by inappropriate means. Interpretability has been identified as a critical success factor in using machine learning in hardware design and verification. Black-box machine learning systems that provide recommendations or decisions without explanations are unlikely to be widely adopted by engineering teams who would expect the rationale for automated decisions that considerably impact project outcomes to be understandable. Explainable artificial intelligence techniques, including attention visualizations that highlight input features that most influenced the decision, feature importance that indicates how much each input factor contributes to the output of a model, and counterfactuals that highlight the smallest change to an input that would result in a different decision, have been successfully used to provide interpretability and transparency in verification systems for machine learning.

The practical issue of integration is more complex: the tool has to fit into the existing organization's engineering process and practices to support legacy verification practices. Production verification systems may include commercial electronic design automation tools, proprietary in-house verification platforms, regression management systems that may start thousands of simulation jobs, coverage analysis and reporting tools, and bug tracking systems. These components may be integrated into a multi-team, multi-step verification workflow. Studies of using machine learning in verification have found that successful use cases involve integration with existing tools and workflows, not a replacement of existing tools [9]. This suggests that standard interfaces and open-source infrastructure for integrating machine learning with verification could help accelerate the uptake of these tools in industry by lowering the cost for individual companies to make use of them and allowing verification teams to build upon community implementations instead of maintaining proprietary ones. These efforts are complicated by having to coordinate with multiple competitors and the fact that competitors' incentives to work together decrease unless there are strong incentives towards standardization.

Conclusion

The incorporation of machine learning techniques into UVM provides a practical approach to solving the scalability problem. The storage system-on-chip (SoC) is becoming increasingly complex due to the integration of multiple communication protocols and advanced control units, which is causing verification productivity to face scalability issues related to the time required and capacity. The architectural approaches and machine learning techniques described in this article show how artificial intelligence augmentation can improve various aspects of verification, such as smart test generation by reinforcement learning agents exploring design state spaces, coverage optimization by predictive analytic approaches and redundancy elimination techniques, cross-protocol interactions discovery via sequence-to-sequence models and graph neural networks, and performance verification through anomaly detection on time series data while maintaining the existing Universal Verification Methodology construct that represents a meaningful investment by most organizations. However, our experience in practicing machine learning-augmented verification has highlighted that realizing machine learning's full potential involves addressing its computational overhead (e.g. through the calculated use of inference operations), data quality and quantity (e.g. by collecting and curating training datasets that include not only frequent but also important corner cases), interpretability (e.g. using explainable artificial intelligence techniques to help verification engineers to understand and trust machine learning decisions), and integration (e.g. through interfaces that support adopting machine learning without disrupting the established verification process and tool ecosystem). The combination of formal verification guarantees and machine learning scalability is promising for future hybrid approaches, which can exploit their respective strengths, and the use of open-source tools and benchmarks can help innovation and fair comparison. In particular, as SoC architectures involve more and more integration density, compatibility between protocols, and functional complexity, machine learning-enabled UVM frameworks are key to enabling the IC industry to develop storage systems capable of meeting increasing performance, reliability, and feature requirements and reducing time to market in response to global competition and roadmap schedules.

References

[1] Harry Foster, "2020 Wilson Research Group Functional Verification Study," Siemens EDA, 2020. [Online]. Available: https://uobdv.github.io/Design-Verification/WilsonResearchGroupFunctionalVerificationStudy/2020-WRGFV-Study/2020-WrG-FV-Study-Webinar-Oct13.pdf
[2] S. Fine; A. Ziv, "Coverage directed test generation for functional verification using Bayesian networks," IEEE, 2003. [Online]. Available: https://ieeexplore.ieee.org/document/1219010
[3] Jingyi Chen, et al., "Deep Reinforcement Learning-Based Automatic Test Case Generation for Hardware Verification," ResearchGate, 2024. [Online]. Available: https://www.researchgate.net/publication/386230850_Deep_Reinforcement_Learning-Based_Automatic_Test_Case_Generation_for_Hardware_Verification
[4] Mihai-Corneliu Cristescu, "Machine Learning Techniques for Improving the Performance Metrics of Functional Verification," ResearchGate, 2021. [Online]. Available: https://www.researchgate.net/publication/350552019_Machine_Learning_Techniques_for_Improving_the_Performance_Metrics_of_Functional_Verification
[5] Suruchi Kumari et al., "Optimizing Coverage-Driven Verification Using Machine Learning and PyUVM: A Novel Approach," arXiv, 2025. [Online]. Available: https://arxiv.org/pdf/2503.11666
[6] Rashid Mustafa, et al., "Cross-Layer Analysis of Machine Learning Models for Secure and Energy-Efficient IoT Networks," Sensors, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/12/3720
[7] Thomas Schlegl, et al., "Scalable anomaly detection in manufacturing systems using an interpretable deep learning approach," ScienceDirect, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827121011598
[8] Nícolas Pfeifer, et al., "Coverage directed test generation for functional verification using Bayesian networks," IEEE, 2003. [Online]. Available: https://ieeexplore.ieee.org/document/9116198
[9] Brucek Khailany et al., "Accelerating Chip Design With Machine Learning," ResearchGate, 2020. [Online]. Available: https://www.researchgate.net/publication/346885327_Accelerating_Chip_Design_With_Machine_Learning
[10] Christopher Bennett, Kerstin Eder, "Review of Machine Learning for Micro-Electronic Design Verification," arXiv, 2025. [Online]. Available: https://arxiv.org/html/2503.11687v1