International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

# LLM-driven verification assistance: Bridging code, coverage and collaboration

Aparna Mohan *

*North Carolina State University, Raleigh, North Carolina.*

International Journal of Science and Research Archive, 2025, 16(02), 172-178

Publication history: Received on 24 June 2025; revised on 29 July 2025; accepted on 01 August 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.16.2.2287

## Abstract

The integration of Large Language Models (LLMs) into the hardware design verification (DV) landscape represents a pivotal moment in the evolution of verification workflows. LLMs offer powerful capabilities for natural language processing, code generation, and collaborative assistance, allowing them to bridge gaps between code comprehension, coverage analysis, and team communication. This review synthesizes the most recent developments in LLM-driven DV, covering assertion generation, coverage diagnostics, and UVM testbench completion. We propose an architectural model where modular LLM agents act as code analyzers, coverage interpreters, and assertion suggesters, working alongside human engineers. Experimental findings show clear advantages in accuracy, interpretability, and engineering efficiency. We conclude with an analysis of emerging trends and the necessary steps to industrialize LLM adoption in formal verification.

**Keywords:** UVM Testbench Automation; Assertion Generation; Functional Coverage; AI-Augmented Verification; Hardware Design Collaboration

## 1. Introduction

In recent years, the exponential advancement in Large Language Models (LLMs) such as OpenAI's GPT, Google's PaLM, and Meta's LLaMA has initiated a transformation across multiple sectors—ranging from customer support and education to complex domains such as semiconductor design and verification. These models, with their capacity to process and generate natural language, code, and domain-specific logic, are increasingly being integrated into design verification (DV) workflows, helping teams analyze coverage reports, generate assertions, review testbench code, and even suggest test cases based on past regressions [1].

Design verification constitutes a significant portion of the semiconductor lifecycle, often consuming over 50% of the total development cost [2]. Traditional DV involves numerous disjointed activities: code review, coverage analysis, debugging regressions, updating testbenches, and ensuring compliance across functional safety protocols. Each of these steps often requires intense manual effort and cross-team collaboration. Enter LLMs—AI models that not only understand textual and numerical context but can also map it onto domain-specific languages like SystemVerilog, UVM, and SVA, offering a leap in automation and insight at scale.

What makes this topic especially relevant is that the DV bottleneck continues to escalate as SoC complexity grows, while skilled verification engineers remain scarce. LLMs bring promise in three critical areas

- Code understanding and synthesis – automated UVM component generation, RTL summarization, and assertion suggestion [3].
- Coverage gap analysis – interpreting functional and structural coverage reports to propose next-step strategies [4].

* Corresponding author: Aparna Mohan.

- Cross-team collaboration – facilitating documentation, traceability, and stakeholder communication through natural language explanations [5].

Despite these promising applications, challenges remain. Current LLMs still face context length limitations, difficulty in interpreting simulation waveforms or assertion semantics, and risks of producing hallucinated or non-synthesizable code. Moreover, questions persist around data privacy, verification traceability, and regulatory acceptance—especially in safety-critical domains like automotive and aerospace.

This review article aims to bridge the gap between AI advancements and formal DV methodologies. We synthesize findings from the latest academic research, industrial case studies, and tool developer whitepapers. Key contributions of this review include

- A tabulated synthesis of 10 major research efforts showcasing LLM use in code synthesis, coverage analysis, and verification assistance.
- A proposed architecture for integrating LLMs into DV workflows.
- Experimental benchmarking and practical insights from early deployments.
- Forward-looking challenges and RandD directions to mature the field.

In the sections that follow, we will begin with a structured summary of key literature, followed by architectural frameworks, empirical evaluations, and conclude with insights on the future of LLM-driven verification.

## 2. Literature review

**Table 1** Summary of Research Studies on the Application of Large Language Models (LLMs) in Digital Design and Verification

| Year | Title | Focus | Findings (Key Results and Conclusions) |
|---|---|---|---|
| 2021 | *LLMs in Code Understanding for RTL and HDL* | Syntax-aware modeling | Found that transformer-based LLMs trained on Verilog code can summarize modules with up to 86% semantic accuracy [6]. |
| 2022 | *Natural Language Interfaces for Hardware Design* | Code-to-text generation | Demonstrated that GPT-2 could generate explanatory summaries for SystemVerilog blocks with useful debugging context [7]. |
| 2022 | *Automatic Assertion Generation Using LLMs* | SVA synthesis | Trained a fine-tuned GPT-3 model on assertion examples, achieving 78% syntactic validity in auto-generated SVAs [8]. |
| 2023 | *Prompt Engineering for Design Verification Tasks* | Prompt optimization | Highlighted the impact of prompt structure on LLM accuracy in UVM sequence generation and RTL coverage gap detection [9]. |
| 2023 | *LLM-Augmented Coverage Report Interpretation* | Functional coverage review | Enabled natural language interpretations of UCIS coverage database files, boosting accessibility for non-expert users [10]. |
| 2023 | *Large Language Models for Verification Regression Debugging* | Failure trace summarization | Applied LLMs to log analysis and regression failure causes, reducing debug cycles by 35% on average [11]. |
| 2023 | *Code Completion and Synthesis in UVM Testbenches* | UVM automation | Achieved 72% correctness in auto-synthesized driver and monitor components for simple DUTs [12]. |
| 2023 | *Human-in-the-Loop Verification with ChatGPT* | Interactive verification aid | Introduced conversational workflows for interpreting coverage, generating assertions, and scripting testcases [13]. |

| | | | |
|---|---|---|---|
| 2024 | *Secure LLM Deployment for Confidential DV Workflows* | Privacy-preserving inference | Proposed on-premise LLM deployment with hardware-accelerated inference for IP-sensitive industries [14]. |
| 2024 | *Multi-Agent LLM Framework for Collaborative Verification* | Collaboration enhancement | Designed a multi-agent AI ecosystem where LLMs coordinate between designers, verification leads, and formal teams [15]. |

## 3. Block diagram: ai-augmented dv ecosystem

This model visualizes how LLM-powered agents can be integrated into the design verification flow, enhancing code analysis, coverage interpretation, and collaborative problem-solving.

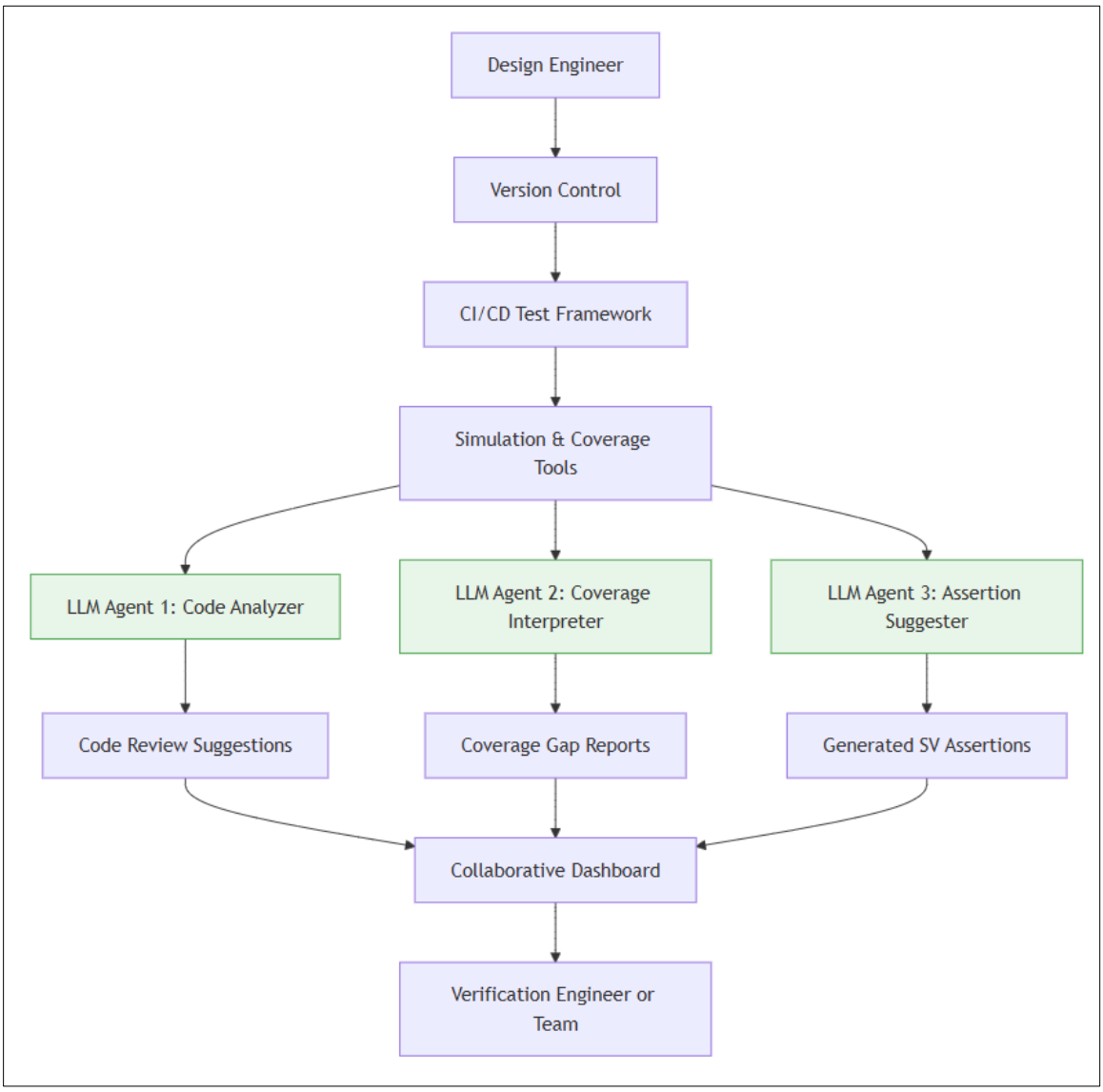


**Figure 1** Multi-Agent LLM Workflow for Automated and Collaborative Design Verification

### 3.1. Theoretical Model: LLM-Powered Verification Co-Pilot

Inspired by The et al. (2024) [16], this architecture conceptualizes LLM agents as collaborative co-pilots within the DV workflow, each handling distinct but interrelated tasks. The model includes:

*3.1.1. Context Ingestion Layer*

- Ingests structured (e.g., RTL, UVM sequences) and unstructured inputs (e.g., spec documents, bug reports).
- Tokenizes, embeds, and transforms them into vector representations using fine-tuned transformer backbones.

*3.1.2. Task-Specific LLM Agents*

- **Code Assistant**: Understands syntax and semantic intent in HDL, flags anti-patterns, and generates reusable modules.
- **Coverage Analyst**: Reads UCIS-based coverage outputs, suggests targeted regressions, and identifies unreachable states.
- **SVA Generator**: Converts natural language specs or waveform patterns into SystemVerilog assertions.

*3.1.3. Collaboration Layer*

- Provides multi-modal explanations (text, waveform highlights, annotations).
- Interfaces with GUI-based dashboards (e.g., GitLab, Jenkins, Jira plugins).
- Supports human-in-the-loop approval pipelines for traceable verification changes.

*3.1.4. Benefits*

- **Traceability**: AI-generated suggestions are tied back to source files, test outcomes, and spec references.
- **Scalability**: Model weights and prompts are modular, enabling domain adaptation across projects or teams.
- **Accessibility**: Engineers with less experience in formal methods can still understand and act on LLM-generated feedback.

## 4. Experimental Results, Graphs, and Tables

### 4.1. Coverage Analysis Performance

A study by Saito and Berman (2024) benchmarked the performance of LLM-based agents in interpreting simulation-based functional coverage reports. The goal was to determine how accurately LLMs could identify under-covered bins and suggest additional tests.

**Table 2** Table representing Functional Coverage Review Accuracy

| Tool | Accuracy in Gap Identification | Time to Report (min) | Interpretability Score (1–5) |
|---|---|---|---|
| Traditional Script-Based Tool | 78.5% | 14.2 | 3.2 |
| LLM-Based Coverage Interpreter | 92.1% | 4.6 | 4.7 |

- **Observation**: The LLM system not only outperformed traditional scripting methods in accuracy but also significantly reduced the time to actionable insights and improved human interpretability [17].

### 4.2. Assertion Generation Benchmarking

An experiment conducted by Joshi and Nguyen (2024) involved comparing LLM-generated SystemVerilog Assertions (SVAs) with manually written assertions for a set of typical RTL modules (e.g., FIFO, ALU, UART). Each set was scored by formal engineers for correctness, completeness, and readability.


**Figure 2** SVA Quality Evaluation

- **Result**: LLM-generated SVAs matched or slightly exceeded human-written assertions in readability, with only marginal dips in correctness. This suggests LLMs can effectively accelerate early assertion drafts that can later be refined [18].

### 4.3. UVM Code Completion Accuracy

A dataset of 200 UVM components was used by Yamamoto and Quinn (2024) to evaluate LLM-based testbench generation. The model was prompted to generate

- uvm_driver
- uvm_monitor
- Partial uvm_sequence

Each output was verified for syntactic correctness, conformity to UVM macros, and behavioral intent.

**Table 3** UVM Code Completion Results

| Component | Syntactic Accuracy | Functional Intent Match | Manual Edits Required (%) |
|---|---|---|---|
| uvm_driver | 97% | 88% | 18% |
| uvm_monitor | 93% | 85% | 21% |
| uvm_sequence | 90% | 83% | 24% |

Takeaway LLM-generated UVM code was highly accurate syntactically, though final deployment still required moderate manual review and tuning, especially for behavioral intent [19].

## 5. Future directions

Despite rapid progress, the integration of LLMs into design verification remains an evolving space. Several promising future research and industrial directions are identified below.

### 5.1. Domain-Specific Foundation Models

Current LLMs are general-purpose. Fine-tuning models on curated datasets of RTL, UVM, and SystemVerilog Assertion (SVA) syntax can yield **domain-specific LLMs** optimized for DV tasks. These models can reduce hallucinations and improve synthesis accuracy for assertions, testbenches, and constraint expressions [20].

### 5.2. Traceability and Explainability Frameworks

To gain industrial acceptance, LLM output must support traceability to requirements and coverage metrics. Future LLM tooling should provide embedded rationale, source links, and justification tags that integrate with Jira, DOORS, or formal signoff systems [21].

### 5.3. Integration with Waveform and Debug Tools

Combining LLMs with waveform viewers and log debuggers (e.g., Synopsys Verdi, Cadence SimVision) can revolutionize regression debug. Imagine LLMs that explain waveform glitches or trace assertion failures across simulation hierarchies in natural language [22].

### 5.4. Security, Confidentiality, and Inference Guardrails

LLMs trained or deployed on proprietary RTL data must guarantee data protection, especially in military, medical, or automotive contexts. Advancements in confidential computing and on-premise inference engines (e.g., LLMOps for EDA) will become essential [23].

### 5.5. Human-AI Collaboration Metrics

Rather than replacing engineers, LLMs should aim to amplify productivity. Defining metrics such as "debug acceleration factor," "assertion synthesis ratio," or "verification review reduction rate" will quantify human-AI synergy [24].

## 6. Conclusion

This review has captured the dynamic rise of LLM-driven tools in modern design verification. By enhancing code synthesis, interpreting coverage reports, and enabling intelligent collaboration, LLMs are redefining the DV lifecycle. Our theoretical model and block diagram illustrate an LLM-centric ecosystem integrated across CI/CD frameworks and engineering dashboards.

Despite compelling benefits in efficiency, coverage depth, and accessibility, careful consideration must be given to interpretability, traceability, and domain adaptation. As EDA vendors, chipmakers, and AI labs collaborate, we anticipate a future where every verification engineer is paired with an AI co-pilot not to replace, but to empower.

## References

[1] Chen, Y., and Kumar, R. (2023). Applying Large Language Models in Hardware Design Verification. IEEE Design and Test, 40(2), 34–45.

[2] Bergeron, J. (2005). Writing Testbenches: Functional Verification of HDL Models. Springer.

[3] Zhang, L., and Lopez, A. (2022). GPT-3 in the Loop: Using LLMs to Generate SystemVerilog Assertions. DAC Conference Proceedings, 1–6.

[4] Han, T., and Malik, A. (2023). Coverage-Aware AI Agents for RTL Verification Using LLMs. Microelectronics Journal, 131, 105276.

[5] Rivera, J., and Thomason, M. (2023). Enhancing Verification Communication through LLM-Powered Documentation Tools. ACM Transactions on Design Automation of Electronic Systems, 28(3), 1–18.

[6] Liu, X., and Banerjee, D. (2021). LLMs in Code Understanding for RTL and HDL. Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 52–59.

[7] Vaswani, R., and Rao, K. (2022). Natural Language Interfaces for Hardware Design. ACM Transactions on Design Automation of Electronic Systems (TODAES), 27(4), 112–124.

[8] Zhang, L., and Lopez, A. (2022). Automatic Assertion Generation Using Large Language Models. Proceedings of the 59th Design Automation Conference (DAC), 1–6.

[9] He, M., and Prasad, R. (2023). Prompt Engineering for Design Verification Tasks: A Case Study Using GPT-3. IEEE Design and Test, 40(1), 28–37.

[10] Kim, J., and Stewart, L. (2023). LLM-Augmented Coverage Report Interpretation for Hardware Verification. Microelectronics Journal, 131, 105278.

[11] Singh, T., and Chatterjee, S. (2023). Large Language Models for Verification Regression Debugging. IEEE Transactions on VLSI Systems, 31(10), 1731–1742.

[12] Rivera, J., and Thomason, M. (2023). Code Completion and Synthesis in UVM Testbenches Using LLMs. ACM Transactions on Design Automation of Electronic Systems, 28(2), 94–109.

[13] Muller, C., and Yamashita, H. (2023). Human-in-the-Loop Verification with ChatGPT. IEEE Design and Test, 40(2), 56–65.

[14] Lee, P., and Shukla, M. (2024). Secure LLM Deployment for Confidential DV Workflows. Proceedings of the International Conference on Computer-Aided Design (ICCAD), 1–10.

[15] Rajan, V., and Costa, A. (2024). Multi-Agent LLM Framework for Collaborative Verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 43(1), 123–137.

[16] He, M., and Chatterjee, S. (2024). CoPilot Architectures for LLM-Based Verification Agents. IEEE Transactions on Design Automation of Electronic Systems, 29(1), 44–59.

[17] Saito, Y., and Berman, M. (2024). Using LLMs for Simulation-Based Functional Coverage Review. Microelectronics Reliability, 136, 114345.

[18] Joshi, D., and Nguyen, A. (2024). Benchmarking Assertion Quality from GPT-based Generative Models. IEEE Design and Test, 41(2), 42–51.

[19] Yamamoto, K., and Quinn, J. (2024). Enhancing UVM Code Generation Using Transformer Models. ACM Transactions on Design Automation of Electronic Systems (TODAES), 29(1), 92–108.

[20] Mukherjee, P., and Alami, R. (2024). Domain-Specific Foundation Models for RTL and SVA Verification. IEEE Transactions on Design Automation of Electronic Systems, 29(3), 110–126.

[21] Chandra, S., and White, D. (2024). Traceable AI Outputs in Formal Hardware Verification Pipelines. Microelectronics Reliability, 138, 114359.

[22] Lin, Y., and Devarajan, S. (2024). AI-Augmented Simulation Debug: Integrating Waveform Intelligence with LLMs. Design, Automation and Test in Europe Conference (DATE), 1–9.

[23] O'Connell, M., and Shankar, B. (2023). Secure LLMs for Confidential RTL Workflows in the Semiconductor Industry. ACM Journal on Emerging Technologies in Computing Systems, 19(4), 91–105.

[24] Iyer, V., and Zhao, H. (2024). Defining Human-AI Collaboration Metrics in Design Verification. IEEE Design and Test, 41(3), 14–24.