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

AI-Powered Hardware Verification: Your Non-Human Friend in Action

Sreedevi Sreekaladevi[4], Deepak Narayan Gadde[1], Aman Kumar[2],
Johannes Grinschgl[3], Linus Maurer[4], Georg Pelz[3]
[1]Infineon Technologies Dresden GmbH & Co. KG, Germany
[2]Infineon Technologies India Pvt. Ltd., India
[3]Infineon Technologies AG, Germany
[4]Universität der Bundeswehr Munchen, Germany

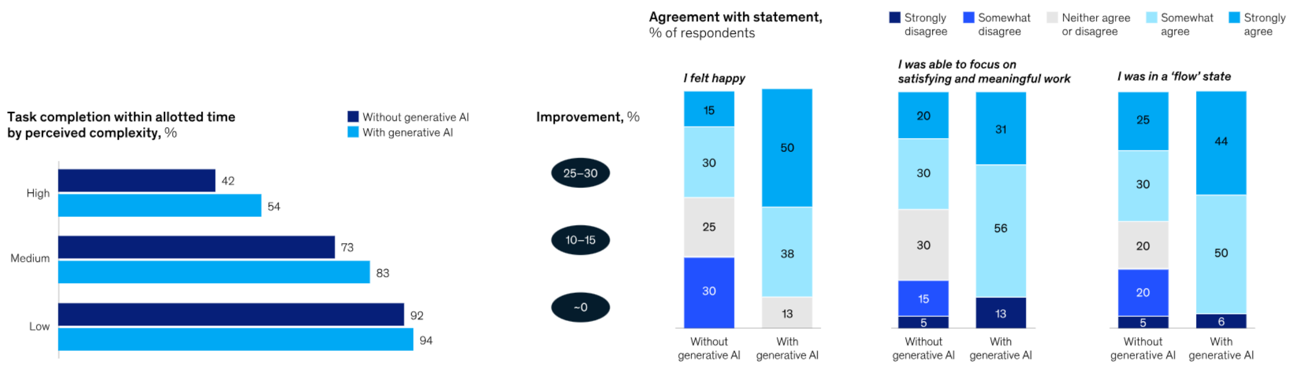
***Abstract*—Artificial Intelligence (AI) and Machine Learning (ML) have been increasingly adopted to enhance engineering productivity across various domains. In the context of hardware design and verification, where workflows are often complex, repetitive, and resource-intensive, Generative AI (GenAI) has emerged as a promising approach to support various engineering tasks. This paper investigates the use of GenAI in areas such as requirements engineering, verification setup, debugging, coverage closure, and related activities. Through prompt-driven and context-aware interactions, GenAI can help streamline the verification workflow, reduce manual effort, and allow engineers to focus on more complex and creative problem-solving. Preliminary evaluations demonstrate productivity improvements of up to 30% when GenAI is integrated into the verification process.**

***Keywords*—Verification, AI, ML, Large Language Models (LLMs), GenAI, Electronic Design Automation (EDA)**

I. Introduction

For several decades, engineers have sought to leverage AI to improve overall productivity. As early as 1989, A. Miller [1] published a paper exploring how AI could assist in Integrated Circuit (IC) and Printed Circuit Board (PCB) design and verification. Since then, AI has advanced significantly, with hundreds of research papers demonstrating its growing role in supporting engineers across a wide range of tasks [2] [3] [4].

Meanwhile engineers can achieve significant productivity gains with AI, in particular GenAI, which enables developers to complete tasks like process optimization, new code creation, code optimization, and documentation in a fraction of the usual time. A survey conducted by VALA [5] revealed that 52 % of their employees currently use AI tools, estimating an average efficiency improvement of 29 %. In Software test automation, developers showed the highest adoption and optimism, while testers expressed more skepticism. Despite uncertainties in estimating efficiency gains, there is a general agreement on the current benefits of AI tools and an expectation of future improvements, potentially reaching 40 % productivity gains.



(a) GenAI assistance in task completion **(b)** Developer experience with GenAI tools

Fig. 1. Accelerating developer productivity with GenAI [6]

Another study by McKinsey [6] shows that with proper upskilling and organizational support, GenAI can surpass previous advances in engineering productivity by combining innovative tools with refined processes. Their research identifies four key areas where GenAI significantly enhances developer productivity - *automating repetitive tasks, jump-starting new code creation, accelerating code updates, and supporting complex problem-solving*. As depicted in Fig. 1a, GenAI significantly increases task completion rates within the allotted time, particularly improvements are most notable for high-complexity tasks around 25 % to 30 %. Another observation from the survey is shown in Fig. 1b, which illustrates that the use of GenAI is associated with higher levels of happiness, focus on meaningful work, and experiencing a "flow" state among respondents. A greater proportion of users strongly agreed with positive statements when using GenAI compared to those without it. These findings suggest that GenAI may enhance the developers productivity and experience in various aspects.

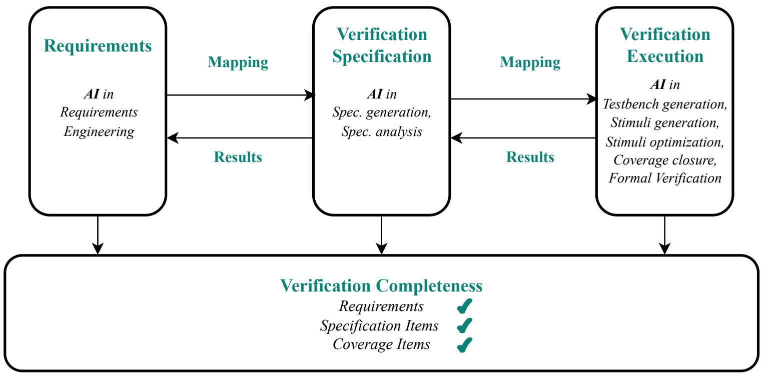

Requirements
AI in Requirements Engineering
Mapping
Results
Verification Specification
AI in Spec. generation, Spec. analysis
Mapping
Results
Verification Execution
AI in Testbench generation, Stimuli generation, Stimuli optimization, Coverage closure, Formal Verification
Verification Completeness
Requirements ✓
Specification Items ✓
Coverage Items ✓

Fig. 2. AI applications in requirements based verification flow

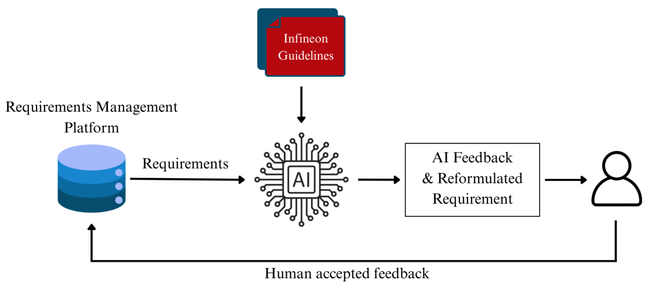
AI improves productivity in design verification across Requirements Engineering (RE) [7] [8] [9], Universal Verification Methodology (UVM) environment development [10], stimuli generation, regression optimization [11], debugging [12], coverage closure [13], and formal verification [14], [15]. Fig. 2 outlines a requirement-based verification flow: Requirement Engineering (capturing and refining requirements, largely text-centric); Verification Specification (requirement analysis, planning, and specification creation, also text-centric); Verification Execution (testbench development, test creation, formal property generation, and stimuli optimization to meet coverage goals); and a final Verification Completeness Check, where metrics are reviewed to confirm comprehensive coverage and that all requirements are addressed. These stages are time- and resource-intensive, but leveraging AI as a "non-human best friend" can reduce manual effort, streamline workflows, and boost productivity end-to-end.

II. AI in Requirements Engineering

According to Generative Pretrained Transformer (GPT)-4o model, "Requirements Engineering is the structured process of identifying, documenting, analysing, validating, and managing stakeholder needs and expectations to ensure successful development of a system or product" [16]. Requirements Engineering is the first crucial step in any product development and lack of well-formulated requirements can drive up costs and time of development processes [17].

Using LLMs for various Natural Language Processing (NLP) tasks in RE [7] [8] [9] are studied extensively nowadays due to their ability to process, i.e., understand, analyse and generate, complex text, as seen above. We aim to harness this capability of LLMs and present our first attempt to create a GenAI-based requirements formulation tool that can support and the requirement authoring and review process within Infineon Technologies. Currently, both these tasks are done largely manually, starting with an engineer writing the requirements, followed by 3 rounds of reviews before they are accepted and recorded. It is anticipated that this tool can help speed up the process, thus contributing towards minimizing the time to market of a product.

While writing requirements, it is important to follow guidelines and several literature illustrating them are available in public domain [18] [19]. But the guidelines followed inside Infineon Technologies are proprietary and these guidelines can change across various domains. The focus of this work is to use an LLM to assess the quality of requirements against a custom set of domain specific guidelines and analyse if the feedback generated by the model would match the review by a human expert.


Infineon Guidelines
Requirements Management Platform
Requirements
AI
AI Feedback & Reformulated Requirement
Human accepted feedback

A well-formulated requirement should follow the below guidelines:

1. Provide requirements in complete sentences
2. Provide only one sentence per requirement
3. Use active voice
4. Put statements in a positive form
5. Avoid needless words
6. Keep related words together
7. Avoid ambiguity

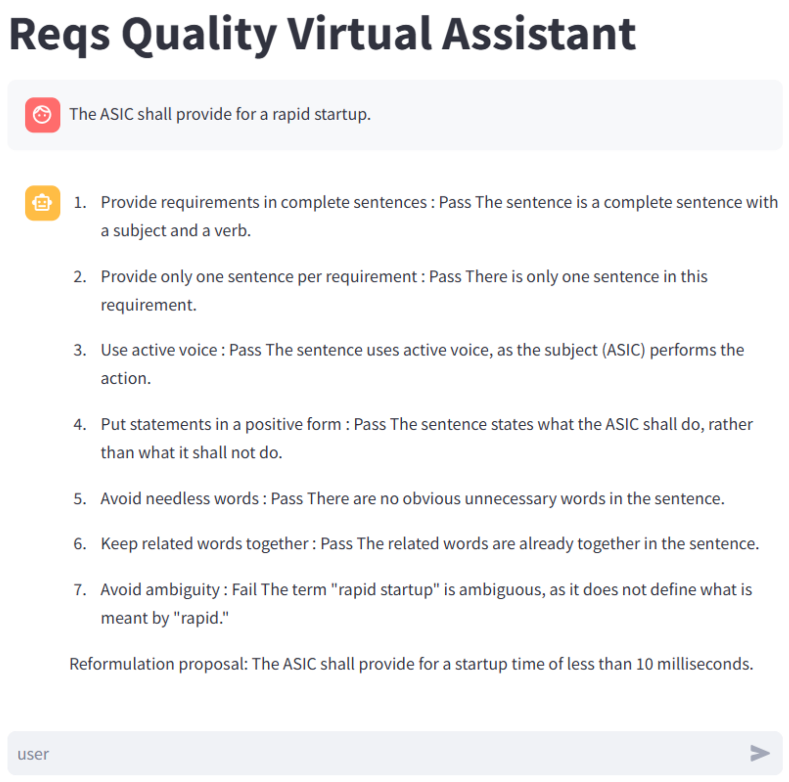
(a) Proposed workflow **(b)** Guidelines for checking the quality of requirements

Fig. 3. Reformulating requirements using GenAI

A. Methodology

The workflow of the proposed tool is given in Fig. 3a. Requirements, extracted from Infineon's requirements management platforms, are evaluated by an LLM against the provided quality guidelines. The LLM provides feedback and, where applicable, proposes revised formulations of the requirements. If the user accepts the suggested reformulations, they are then saved back to the management platform.

Due to the confidentiality nature of the data, both guidelines and product requirements, we have used only the LLMs which are hosted internally in Infineon's servers. Of the available models, at the time of development of this proof-of-concept, Llama 3.1 70B parameter model was best suited for this use case. The guidelines provided to the model were created by summarizing the different criteria observed for requirements' authoring within Infineon. The summary is also a subset of the rules given in [18] and INCOSE's guide to writing requirements [19].



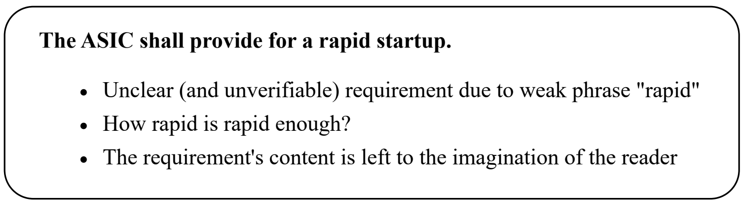
Reqs Quality Virtual Assistant

The ASIC shall provide for a rapid startup.

1. Provide requirements in complete sentences : Pass The sentence is a complete sentence with a subject and a verb.
2. Provide only one sentence per requirement : Pass There is only one sentence in this requirement.
3. Use active voice : Pass The sentence uses active voice, as the subject (ASIC) performs the action.
4. Put statements in a positive form : Pass The sentence states what the ASIC shall do, rather than what it shall not do.
5. Avoid needless words : Pass There are no obvious unnecessary words in the sentence.
6. Keep related words together : Pass The related words are already together in the sentence.
7. Avoid ambiguity : Fail The term "rapid startup" is ambiguous, as it does not define what is meant by "rapid."

Reformulation proposal: The ASIC shall provide for a startup time of less than 10 milliseconds.

Fig. 4. Feedback and reformulated requirement generated by Llama3.1 in tool interface



The ASIC shall provide for a rapid startup.

- Unclear (and unverifiable) requirement due to weak phrase "rapid"
- How rapid is rapid enough?
- The requirement's content is left to the imagination of the reader

Fig. 5. Feedback by human expert

B. Prompting

There are different ways to prompt an LLM, namely zero shot prompting, few-shot prompting [20] [21] and Chain of Thought (COT) [22] prompting. In this initial step, COT was not required, as the intended task does not involve not complex reasoning, but rather the model should use its knowledge of a natural language (English) to analyse the sentence structure and semantics.

Although few-shot prompting can outperform zero-shot in some applications [23], two factors disfavor labeled examples here: (i) there are no universally accepted "perfect" requirements correctness is inherently subjective; and (ii) each Infineon domain augments the guidelines in Fig. 3b with domain-specific rules, limiting cross-domain transferability and scalability. Consequently, a well-engineered zero-shot approach was adopted. To ensure consistent evaluation and appropriate user control, the guidelines are fixed in the system prompt to define task context, while the user prompt contains only the requirement under review. This prevents user-side modification of the guidelines.

C. Results and Evaluation

The LLM evaluates each requirement against the guideline's criteria; if any criterion is unmet, it reformulates the requirement so all conditions are satisfied. Over 1500 Infineon product requirements across multiple domains were reviewed. Responses were consistently aligned (Fig. 4). Prompts were optimized via a human-in-the-loop process, and the model's comments and reformulations were reviewed by RE experts. Expert feedback was largely positive, noting improved understanding of requirement structure. One team reported that AI comments matched expert reviews for over 50 % of tested requirements, estimating 36 % human effort savings and about 45 % reduction in overall cycle time. Two limitations remain: inconsistent detection of sentence voice and occasional misinterpretation of abbreviations without context. These can be mitigated through further prompt tuning or a feedback loop in which a second LLM reviews outputs and its comments are fed back to the first—planned as future work.

III. AI in Development of Verification Environments

AI can be pivotal in generating basic infrastructure for the verification environment, such as several components of the UVM. LLMs are efficient at generating templates to serve as baseline and boilerplate code to facilitate the quick start of verification activities. Several AI-based productivity improvement tools such as GitHub Copilot [24], DVT AI Assistant [25] and others have proved to increase productivity up to 20 % in the design verification domain.

IV. AI in Regression Optimization

In pre-silicon hardware verification, regression testing serves as a critical phase where a large suite of tests is run repeatedly to ensure design stability across code changes. However, these regressions are often computationally intensive, time-consuming, and costly—especially for complex System-on-Chips (SoCs) and Intellectual Property (IP) blocks with expansive design and verification spaces. Traditional regression strategies rely on static test selection or heuristic-based prioritization, which often fails to capture evolving design nuances and changing coverage landscapes.

GenAI presents a transformative approach to optimizing regression strategies by learning patterns from historical test execution data and dynamically generating or prioritizing test subsets that maximize verification efficiency. Unlike conventional AI methods that focus purely on classification or prediction, GenAI can synthesize new test stimuli, adapt regression sequences, and even propose architectural-level regression plans tailored to current design states.

A. Key Contributions of GenAI in Regression Optimization

- **Coverage-Aware Test Selection:** Leveraging generative models trained on prior coverage reports, assertion failures, and waveform data, GenAI can generate prioritized test lists that are more likely to uncover corner-case bugs or maximize functional coverage within limited time windows.
- **Test Stimuli Generation:** Using models such as generative transformers or diffusion models, GenAI can generate syntactically correct and semantically meaningful input sequences for constrained-random environments (e.g., UVM testbenches), targeting unexplored states or transitions in the Design Under Verification (DUV).
- **Adaptive Regression Scheduling:** GenAI systems can incorporate feedback from real-time regression runs—such as failure logs, simulation times, and resource usage—to iteratively refine the schedule for test execution, balancing bug-hunting efficacy with compute cost.
- **Failure-Aware Clustering and Grouping:** By embedding test outcomes and waveform signatures into latent spaces, GenAI models can cluster similar failing tests, helping verification engineers to isolate root causes faster and reduce debug cycles.

B. Architectural Integration

A typical GenAI-enabled regression flow includes the following components:

1) **Data Aggregator:** Collects regression metadata, test outcomes, coverage deltas, and assertion hits.
2) **Representation Model:** Encodes the test characteristics, simulation logs, and coverage vectors into machine-interpretable formats.
3) **Generative Planner:** A transformer-based or reinforcement learning-driven engine that proposes an optimized regression plan.
4) **Execution Monitor:** Feeds back live execution metrics to fine-tune future generations.

V. AI in Debugging Support

The growing complexity of SoC architectures has posed significant challenges to the effectiveness of traditional manual verification and debugging techniques. Industry reports indicate that verification processes account for nearly 70 % of the overall development time [26], highlighting their substantial impact on project timelines. To address this critical bottleneck, EDA vendors and researchers have increasingly adopted GenAI as a transformative solution within verification workflows. This shift marks a notable transition toward intelligent and autonomous systems. Unlike conventional verification tools, GenAI-based approaches exhibit advanced proficiency in understanding design specifications articulated in natural language, analyzing complex Hardware Description Language (HDL) code, and automatically deriving formal properties such as assertions and invariants [27]. By accelerating root-cause analysis and enabling automated code corrections, these AI-driven solutions significantly reduce the manual effort required for debugging, thereby fostering more efficient, reliable, and scalable verification frameworks tailored to the demands of modern semiconductor design.

A. Commercial AI Debugging Tools

*1) **Cadence Verisium AI-Driven Verification Platform**:* Cadence has been at the forefront of AI-driven verification innovation with its Verisium platform, which is built on the Joint Enterprise Data and AI (JedAI) framework [28]. This architecture facilitates the aggregation of data from diverse verification sources such as waveforms, coverage reports, source code, and log files into a unified repository as shown in Fig. 6. The centralized data infrastructure plays a critical role, as AI models rely on large volumes of high-quality data to develop precise predictive capabilities for identifying complex hardware bugs across various design domains.

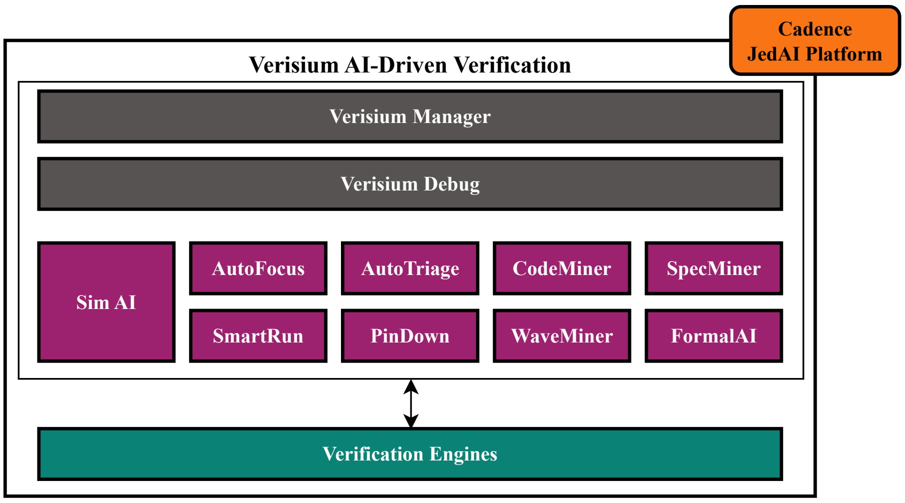


Fig. 6. Verisium AI-Driven Verification Platform [28]

The Verisium platform comprises a suite of specialized applications tailored to streamline different aspects of the verification process:

- **Verisium AutoTriage** leverages ML models to automate regression failure triage by predicting and categorizing test failures based on common root causes, thereby significantly reducing manual intervention.
- **Verisium SemanticDiff** employs algorithmic methods to compare multiple source code revisions, classify them, and rank updates based on their potential impact on the overall system.
- **Verisium WaveMiner** utilizes AI-driven engines to analyze waveforms from multiple simulations, pinpointing specific signals at particular time intervals that are most likely linked to root causes of failures.
- **Verisium PinDown** integrates with version control systems to train ML models on data from source code changes, test reports, and log files, enabling the prediction of specific code modifications that may have introduced defects.
- **Verisium Debug** provides an end-to-end debugging solution, spanning individual IP blocks to full SoC designs. It includes advanced features such as waveform visualization, schematic tracing, and SmartLog to enhance both interactive and post-process debugging workflows [29].

Industry deployments of the Verisium platform have demonstrated significant productivity gains. For instance, Renesas reported up to a sixfold improvement in debugging efficiency and a twofold increase in simulation probing performance, underscoring the platform's potential to revolutionize verification processes [29].

*2) **Synopsys.ai Copilot and Verdi Platform**:* Synopsys has introduced Synopsys.ai Copilot [30], a comprehensive GenAI solution for chip design that integrates collaborative, generative, and autonomous capabilities. The platform harnesses GenAI to provide advanced tool guidance, optimize EDA workflows, and accelerate the development of key verification components, including Register Transfer Level (RTL) code, formal verification assertions, and UVM testbenches. By enhancing these early-stage processes, Synopsys.ai Copilot aims to transform the chip design pipeline with more intelligent and efficient workflows.

One of the platform's key innovations is its focus on automating upstream verification tasks, signalling a strategic shift from reactive debugging to proactive bug prevention. By improving the quality and efficiency of initial design and verification intellectual property, the platform reduces the downstream burden of addressing complex bugs later in the design cycle [31].

Additionally, Synopsys has extended its AI-driven debug capabilities through the next-generation Verdi platform [32] as illustrated in Fig. 7. This platform offers an Intefrated Development Environment (IDE) and sophisticated verification management features designed to streamline debugging and verification processes. Enhancements include reduced debug turnaround times through real-time development checks and AI-powered failure analyses, improved regression

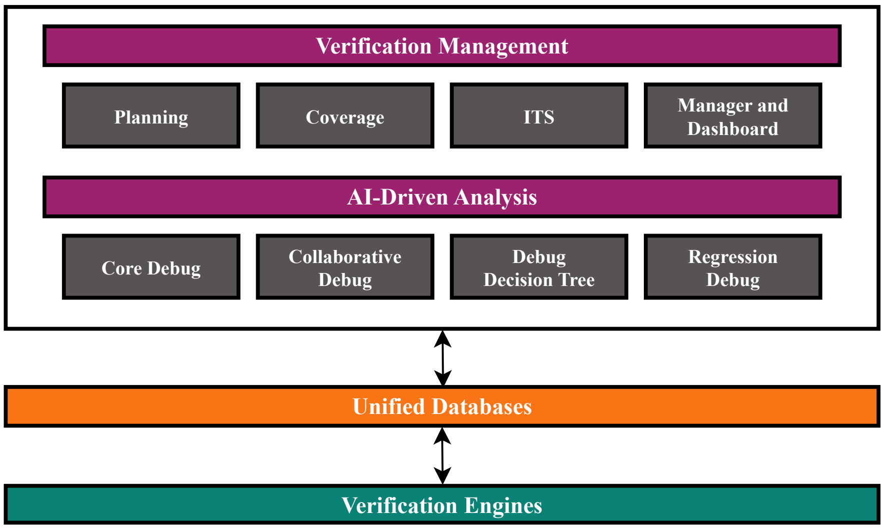


Fig. 7. Verdi Automated Debug System [32]

throughput enabled by a comprehensive verification management system, and cross-domain debugging capabilities. These advanced features enable the platform to analyze design behavior across various abstraction levels, facilitating seamless debugging across diverse domains. Together, these advancements underscore Synopsys's commitment to revolutionizing chip design and verification through intelligent, AI-driven technologies.

*3) **Siemens EDA Questa One Platform**:* Siemens EDA has strategically integrated classical ML, GenAI, and emerging agentic AI technologies to tackle domain-specific challenges across a wide array of design tools and workflows [33]. This approach signifies a deliberate progression of AI applications beyond basic chatbot functionalities and into more sophisticated roles such as flow orchestration, root-cause analysis, and autonomous interactions between tools. By leveraging these advanced AI methodologies, Siemens aims to optimize design and verification processes, pushing the boundaries of efficiency and innovation in EDA.

Questa One combines static and formal verification engines with predictive and GenAI to enhance the verification process [34]. Questa One SFV (Stimulus-Free Verification) integrates linting, auto-correction, and GenAI to automate the creation and verification of SystemVerilog Assertion (SVA), seamlessly executing a sequence of linting, auto-fixing, and equivalency checking. Questa Verification IQ goes a step further by incorporating predictive and GenAI to provide actionable insights and improve verification efficiency. Notably, its GenAI engine can translate natural language descriptions into corresponding SVAs, simplifying the creation of verification properties.

The Siemens EDA platform demonstrates significant advancements in productivity and effectiveness through its "smart" capabilities. These include smart analysis powered by unsupervised ML to detect coverage gaps, smart regression that uses AI for intelligent test selection and prioritization, and smart debug features that automate failure detection and root-cause analysis based on historical data. Collectively, these innovations underscore Siemens' emphasis on leveraging cutting-edge AI technologies to enhance design and verification workflows, driving substantial gains in productivity, accuracy, and scalability.

*4) **Industry Impact and Paradigm Shift**:* Commercial platforms signal a shift in EDA: from reactive, manual bug hunts to proactive, predictive, automated root-cause analysis. The traditional, labor-intensive "art" of hardware debugging dependent on human expertise and trial-and-error is giving way to AI-driven methods that predict and classify failures, cutting manual effort and accelerating debug. Agentic AI advances autonomy in EDA workflows. Beyond earlier AI, these systems navigate complex toolchains, diagnose intricate issues, and sometimes resolve them without human input. This elevates automation, streamlines design, boosts productivity, and points to more intelligent, scalable, efficient debugging and verification in semiconductor design.

B. *Emerging Commercial Solutions*

In addition to established EDA vendors, a number of innovative companies are developing specialized GenAI debugging tools to address targeted challenges in hardware design verification. These next-generation tools leverage advanced AI methodologies to streamline debugging, enhance productivity, and improve the quality of chip design processes.

ChipAgents has introduced an agentic AI environment specifically designed to optimize the debugging phase, which typically accounts for $40\,\%$ of the overall verification cycle [35]. Its platform utilizes LLM-powered agents to perform a "chain of reasoning complemented by experiments," an advanced methodology where simulations with potential code modifications are executed to test hypotheses and iteratively isolate root causes of design issues. This dynamic, experimental approach advances beyond traditional static analysis by closely mimicking the iterative reasoning processes of human debugging experts. Engineers interact with the platform via natural language queries, enabling seamless analysis of waveform files and simulation logs. The system identifies first-level candidate behaviors in waveforms and either allows engineers to refine the results or autonomously continues the localization process. Recognizing the limitations of AI models in this specialized domain, ChipAgents has invested heavily in training its system with both synthetic and human-annotated data specific to chip design. Additionally, the company has developed specialized tools for parsing large simulation dump and log files, which are critical for its agents' analysis and traceback capabilities. By addressing these challenges, ChipAgents aims to deliver a 10x productivity improvement in RTL design and verification.

PrimisAI, on the other hand, has developed RapidGPT, which features PAI Agents designed to assist with a range of hardware design tasks, including RTL coding, debugging, and verification [36]. Its AIVRIL framework has demonstrated an $88.46\,\%$ success rate in achieving verification objectives. RapidGPT incorporates a natural language interface that guides hardware engineers through the entire design process, ensuring accessibility and ease of use. For debugging and quality assurance, RapidGPT offers two key features: AutoReview, an AI-driven HDL auditing tool, and AutoComment, which generates intelligent comments for HDL files. These tools embody a "shift-left" philosophy, integrating quality assurance and bug identification earlier in the design cycle during the initial stages of code creation and documentation. By proactively addressing potential issues at this early stage, the platform minimizes the risk of bugs propagating downstream, where they would be more complex and costly to resolve.

Collectively, these specialized platforms illustrate a growing trend within the industry to deploy GenAI technologies that extend beyond generic AI functionality. By focusing on domain-specific challenges, these tools are reshaping the debugging landscape, enabling higher efficiency, accuracy, and scalability in hardware design verification.

C. Research Advancements in GenAI-Based Debugging for Hardware Design

Academic research is rapidly advancing the application of GenAI in hardware design verification, with a strong focus on developing novel methodologies for bug identification, automated patching, and the creation of intelligent RTL design assistants. These efforts aim to integrate the capabilities of LLMs with traditional EDA workflows to address existing challenges and push the boundaries of verification and debugging processes. By exploring the underlying mechanisms of LLMs and tailoring them to the specific needs of hardware design, researchers are setting the stage for the next generation of AI-driven verification tools.

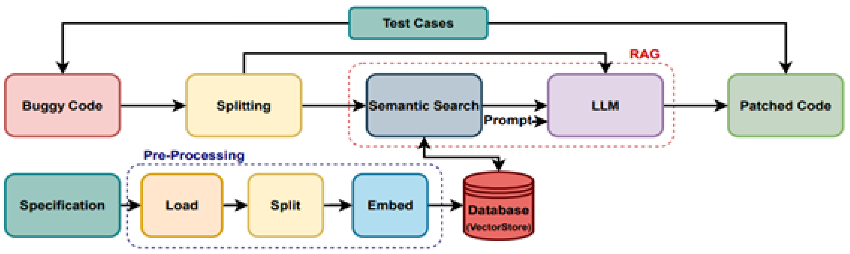


Fig. 8. Bug identification and patching with LLMs and Retrieval Augmented Generation (RAG) [27]

[27] introduces a three-stage methodology for detecting and patching functional bugs in HDL code, validated through experiments on OpenTitan IP blocks. The framework begins with pre-processing to create vector stores from specification files, followed by semantic search for line-by-line HDL code analysis, and culminates in iterative bug identification using adaptive context adjustments for failed initial attempts as illustrated in Fig. 8. Complementing this, the HDLxGraph framework integrates Graph RAG with LLMs, leveraging Abstract Syntax Trees and Data Flow Graphs for HDL-specific graph representations. This approach yields a $12.22\,\%$ improvement in debugging efficiency compared to standard similarity-based RAG by providing richer domain-specific context tailored to the structural and behavioral intricacies of HDL.

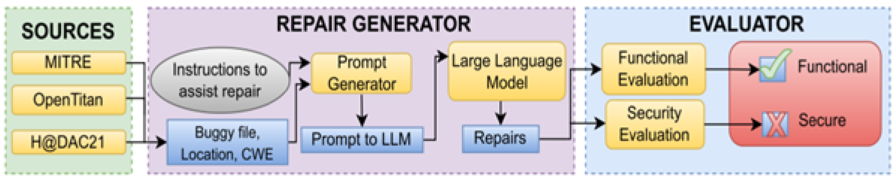


Fig. 9. Components of the LLM-based repair evaluation framework [37]

In [37] researchers examine the ability of LLMs to repair security-critical bugs in Verilog hardware designs through automated replacement code generation as depicted in Fig. 9. While the results demonstrated successful bug fixes across all fifteen benchmarks, approximately $60\,\%$ of the AI-generated designs exhibited vulnerabilities linked to Common Weakness Enumerations (CWEs). These findings underscore the necessity of robust verification and debugging mechanisms, particularly in security-sensitive applications, and highlight the growing importance of formal verification to ensure functional correctness and guarantee safety.

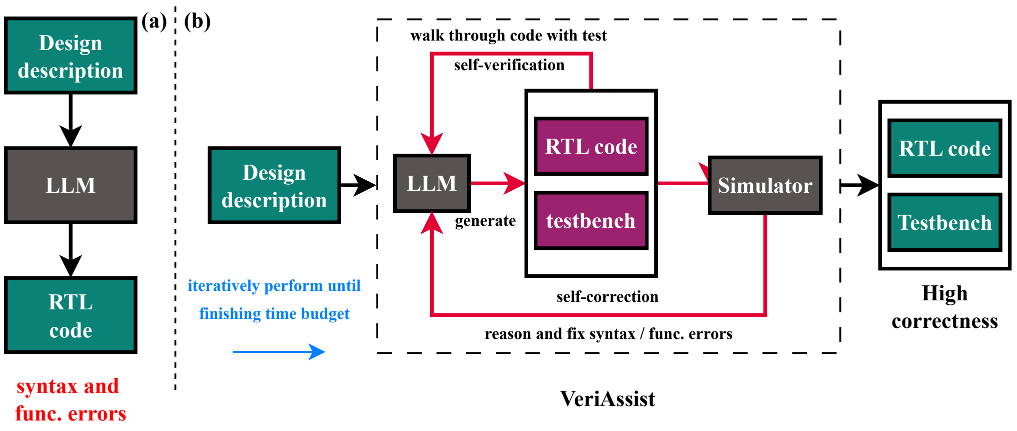


Fig. 10. Comparison of workflows between traditional methods and VeriAssist [38]

VeriAssist [38], an LLM-powered Verilog RTL assistant integrates self-verification and self-correction with Verilog simulators. It employs iterative generation cycles through a “prompt-generate-feedback-revise” loop, mimicking human debugging workflows as shown in Fig. 10. This iterative process has led to a 10.4 % improvement in functionality pass rates, with the system capable of generating and refining test benches for improved debugging efficiency. VeriAssist demonstrates that the quality of verification environments directly impacts the effectiveness of AI-driven debugging, forming a positive feedback loop to enhance both debugging and verification outcomes.

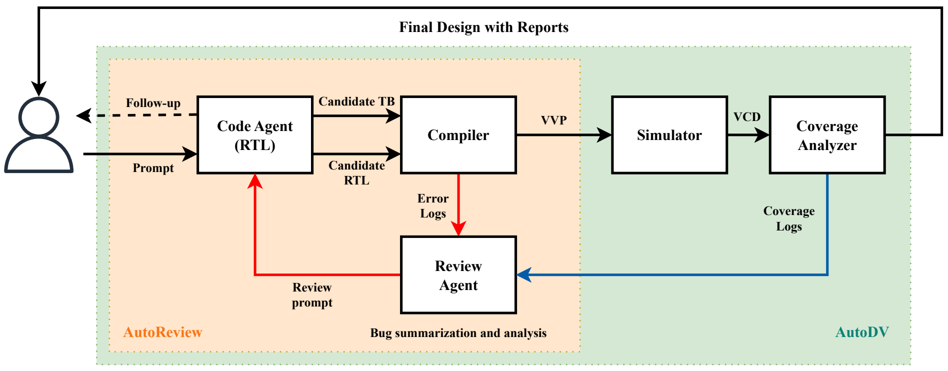


Fig. 11. Overall architecture of the AIVRIL framework [39]

[39] introduces RTLSquad, a multi-agent framework designed to streamline hardware design and debugging. Specialized agents collaborate across exploration, implementation, and verification stages as depicted in Fig. 11, mirroring human team workflows to decompose complex tasks into manageable sub-processes. This framework achieves an 88.46 % success rate in meeting verification objectives and demonstrates nearly a 2x improvement in code quality through iterative verification-fix loops that incorporate agent feedback directly into the design generation process.

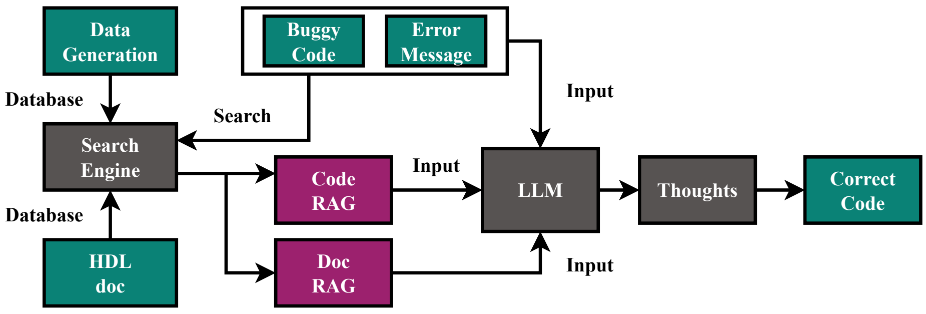


Fig. 12. Framework overview of HDLdebugger [40]

[40] address the significant challenge of limited high-quality HDL training data through reverse-engineering techniques that systematically introduce realistic errors into correct HDL code as seen in Fig. 12. The framework consists of three core components: synthetic data generation, optimized RAG for retrieving relevant debugging information, and fine-tuning LLMs on retrieval-augmented data. HDLDebugger achieves pass rates of up to $81.93\,\%$ and demonstrates that supervised fine-tuning is more impactful than RAG alone, although hybrid approaches combining both strategies deliver optimal results.

These academic efforts collectively highlight the transformative potential of GenAI in hardware design verification. By addressing challenges such as bug identification, error resolution, and iterative design improvement, these methodologies pave the way for more efficient, reliable, and intelligent verification frameworks tailored to the complexities of modern semiconductor design.

VI. AI in Coverage Closure

Coverage closure is essential in verification to ensure all functional aspects of a design are thoroughly tested, minimizing undetected flaws. Traditional methods depend on resource-intensive regression tests, but advancements in AI and ML have introduced more efficient alternatives. ML models prioritize uncovered areas, neural network-based test selectors optimize test generation, and RTL-integrated verification dynamically adjusts test sequences to accelerate coverage closure. LLMs further enhance this process by identifying coverage gaps and refining test scenarios [41]. These innovations streamline verification by reducing redundancy, improving resource efficiency, and shortening cycles. Overall, AI promises to further optimise coverage-driven workflows and strengthen design verification.

A. *Enhancing Coverage Metric Analysis*

One of the primary challenges in coverage closure involves analyzing coverage metrics to identify gaps and determine whether all functionality of the DUV has been adequately tested. GenAI models, trained on large datasets of verification scenarios, coverage reports, and coverage closure workflows, can automatically parse coverage reports and identify areas of incomplete testing. These models can intelligently classify uncovered areas into categories such as missing functional scenarios, insufficient stimulus coverage, or unverified corner cases. By doing so, GenAI accelerates the process of pinpointing deficiencies and aids engineers in prioritizing test development to address the most critical gaps.

Additionally, GenAI enables the correlation of coverage metrics with design specifications and test plans. Given a design specification in natural language or formal documentation, a GenAI model can cross-reference uncovered coverage points to specific requirements, ensuring that all functional and performance goals are met. This capability reduces the likelihood of specification oversights and improves traceability throughout the verification process.

B. *Automated Stimulus Generation*

Generating high-quality stimulus to target uncovered scenarios is a key task in achieving coverage closure. Manual stimulus generation, while effective for simple scenarios, struggles with complex and rare corner cases. GenAI-powered tools can dynamically generate stimulus sequences tailored to maximize coverage, combining symbolic reasoning with learned patterns from historical verification data. These tools use reinforcement learning to iteratively refine stimulus generation strategies based on real-time feedback from coverage metrics, ensuring efficient convergence towards closure.

Moreover, GenAI can augment Constrained Random Verification (CRV) by identifying gaps in constraints or bias in the randomization process that might leave certain regions of the state space unexplored. By analyzing DUV design characteristics and the simulation environment, GenAI suggests constraint modifications or generates targeted inputs to exercise hard-to-reach states. This proactive approach enhances the quality of random stimulus generation and reduces the number of simulation iterations required to close coverage gaps.

C. *Adaptive Learning and Continuous Improvement*

One of the most promising aspects of using GenAI in coverage closure is its ability to learn and improve over time. By aggregating data from verification cycles, GenAI builds knowledge of design patterns, coverage gaps, and effective test strategies, enabling adaptation to future designs and environments and improving automation. It can detect recurring gaps across projects and propose reusable tests or methodology refinements particularly valuable for SoC programs with shared IP blocks or standardized protocols, where insights transfer across designs.

Despite its potential, applying GenAI to coverage closure faces challenges. Training requires large, curated historical verification datasets that are often scarce or fragmented. Additionally, limited interpretability of AI-generated stimuli and coverage gap analyses necessitates engineer validation. Addressing these issues calls for robust data curation, hybrid AI-human workflows, and advances in explainable AI. Future directions include multimodal models that unify textual specifications, simulation traces, and waveform data within a single framework, and domain-specific LLMs tailored to verification tasks to improve precision and applicability.

VII. AI in Formal Verification

Formal verification techniques, such as model checking and theorem proving, are widely used to validate RTL designs in safety-critical applications. However, the effectiveness of these methods heavily relies on human-written properties and deep domain expertise. As the complexity of SoCs increases, the cost of writing formal properties and debugging Counter Examples (CEXs) grows significantly. GenAI, especially models like ChatGPT, Codex, and other LLMs, has shown promise in understanding, generating, and reasoning about code. When trained on HDL and verification methodologies, these models can assist in formal verification workflows. This paper investigates the intersection of GenAI and formal verification, proposing methodologies, presenting case studies, and discussing current limitations and future directions.

In formal verification, text-to-code capabilities of GenAI models can be leveraged to [14]:

- Generate SVAs
- Suggest formal properties
- Summarize CEXs
- Propose constraints or abstraction hints

A. *Automatic Property Generation*

Writing SVAs is a major bottleneck in formal verification. GenAI can generate candidate properties based on:

- Design specification documents
- RTL code pattern recognition
- Past assertion datasets

Example:

Listing 1: An example property generated by GPT-4 with a default temperature setting of 0.7

```
assert property (@(posedge clk) !empty |-> ##1 valid);
```

B. *Specification Mining and Natural Language Understanding*

GenAI can translate English descriptions into formal specifications.

Input: “Data should not be written when reset is active.”

Output (SVA):

Listing 2: SVA property asserting that write is inactive when reset is active

```
assert property (@(posedge clk) reset |-> !write);
```

C. *Debugging counterexamples and Explanation*

GenAI can:

- Parse waveform dumps (e.g., VCD, FSDB)
- Summarize failing scenarios in plain English
- Suggest potential fixes or abstractions

D. *Formal Abstraction and Constraint Suggestions*

Formal verification tools often struggle with state space explosion, particularly in large-scale or highly concurrent systems. To manage this complexity, designers must supply abstraction models and environmental constraints that guide the verification engine toward tractable and meaningful analysis. However, developing effective abstractions and constraints typically requires expert insight and iterative refinement. GenAI presents a promising avenue to assist in this process by analyzing the design intent and suggesting useful abstractions or constraints [15]. These may include:

- **Input Constraints**: Restricting formal inputs to realistic operational ranges, such as limiting burst sizes in AXI transactions or constraining request frequencies in FIFOs. GenAI can infer such constraints from protocol documentation or prior simulations.
- **State Abstraction**: Identifying irrelevant internal state variables or redundant logic that can be abstracted away to reduce model complexity. For example, GenAI can flag debug registers or status counters that do not influence the safety properties under verification.
- **Modular Decomposition**: Proposing boundary interfaces or assume-guarantee pairs to decompose a complex verification task into smaller, composable modules. GenAI can analyze module dependencies and suggest candidate cut points.
- **Property Partitioning and Scoping**: GenAI can assist in scoping assertions to appropriate clock domains, reset conditions, or subcomponents, thus preventing vacuous proofs and improving convergence.
- **Overconstraint Detection**: By comparing AI-generated assumptions with actual RTL behavior, the tool can warn against overly strict constraints that may suppress legitimate CEXs.

E. Integration with Verification Tools

For GenAI to be effectively adopted in real-world verification environments, seamless integration with existing EDA tools is essential. Several approaches can be used to embed GenAI capabilities into formal verification workflows:

- **IDE Plugins:** GenAI can be integrated as a plugin within RTL development and formal verification IDEs (e.g., Cadence JasperGold, Synopsys VC Formal). This allows for features such as in-line assertion generation, natural language specification parsing, and guided debug within the environment developers already use.
- **Natural Language to Assertion Pipelines:** Documentation written in natural language (e.g., from Confluence, markdown specs, or PDF) can be converted into SVA significantly reduces the manual effort in bridging the gap between specification and implementation.
- **Chatbot-based Debugging Assistants:** LLM-powered assistants can be used to interpret CEXs, answer verification-related queries, and propose strategies for convergence. These assistants can be trained on waveform dumps, formal logs, and past debug patterns to enhance relevance.
- **Integration with Version Control and CI/CD Pipelines:** GenAI can analyze commit diffs, generate verification intents, and recommend assertions or constraints for newly added RTL modules. In regression environments, GenAI can also monitor property convergence trends and suggest fixes based on failure logs.
  item **Context-aware Code Completion:** Advanced GenAI models can offer formal-aware auto-completion for assertion templates, property refinements, and temporal logic patterns, accelerating assertion development for verification engineers.
- **Feedback Loop with Formal Engines:** Integration with formal solvers allows GenAI to learn from engine feedback (e.g., vacuity warnings, CEX traces) and iteratively refine assertions or constraints in real time.

Overall, tight coupling between GenAI systems and formal verification tools opens the door to highly interactive and intelligent verification workflows, reducing turnaround time and democratizing formal methods for less experienced engineers.

VIII. Conclusion

This paper has highlighted the transformative potential of AI, particularly GenAI, in hardware design verification. It examined how AI can enhance productivity across the verification lifecycle, including requirements engineering, test-bench development, regression optimization, debugging, coverage closure, and formal verification. Practical examples demonstrated that AI tools are already reducing manual effort, improving efficiency, and delivering measurable benefits in time savings and overall quality.

Notably, requirements engineering and debugging emerged as areas where AI assists engineers by automating repetitive tasks, identifying gaps, and accelerating workflows. Likewise, tools that optimize regression and support formal verification enable faster and more accurate testing, helping to close coverage gaps and minimize undetected bugs. Preliminary evaluations indicate that AI-driven solutions can improve productivity by up to 30 %, underscoring the growing importance of these technologies in engineering workflows.

Realizing these benefits requires careful integration, including upskilling teams, adapting workflows, and establishing trust in AI-generated solutions. Continued attention to the accuracy and interpretability of AI outputs remains essential, as addressing these challenges is critical for sustained adoption.

In conclusion, AI is no longer merely a buzzword in hardware verification; it is becoming an indispensable tool. By reducing time-to-market and enhancing quality, AI enables teams to concentrate on innovation while offloading routine tasks. As the technology evolves, its role in shaping the future of verification will continue to expand, making it a valuable partner in modern engineering.

References

[1] A. Miller, *From expert assistant to design verification: applications of AI to VLSI design*, Apr. 1989. DOI: 10.1109/SECON.1989.132410

[2] D. Yu et al., "A survey of machine learning applications in functional verification," *DVCon US*, 2023.

[3] M. Abdollahi et al., "Hardware design and verification with large language models: A scoping review, challenges, and open issues," *Electronics*, vol. 14, no. 1, 2025, ISSN: 2079-9292. DOI: 10.3390/electronics14010120 [Online]. Available: https://www.mdpi.com/2079-9292/14/1/120

[4] K. A. Ismail et al., "Survey on machine learning algorithms enhancing the functional verification process," *Electronics*, vol. 10, no. 21, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10212688 [Online]. Available: %7Bhttps://www.mdpi.com/2079-9292/10/21/2688%7D

[5] VALA, *Efficiency gains of AI assisted testing in 2024 and near future*, May 2024. [Online]. Available: https://www.valagroup.com/blog/efficiency-gains-of-ai-assisted-testing-in-2024-and-near-future/

[6] B. K. Deniz et al., *Unleashing developer productivity with Generative AI*, Jun. 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/unleashing-developer-productivity-with-generative-ai#/

[7] K. Ronanki et al., *Requirements Engineering using Generative AI: Prompts and Prompting Patterns*, 2023. arXiv: 2311.03832 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2311.03832

[8] K. Ronanki et al., *Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes*, 2023. arXiv: 2307.07381 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2307.07381

[9] H. Cheng et al., *Generative AI for Requirements Engineering: A Systematic Literature Review*, 2025. arXiv: 2409.06741 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2409.06741

[10] D. N. Gadde et al., "Efficient Stimuli Generation using Reinforcement Learning in Design Verification," in *2024 20th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2024, pp. 1–4. DOI: 10.1109/SMACD61181.2024.10745410

[11] D. N. Gadde et al., *Improving Simulation Regression Efficiency using a Machine Learning-based Method in Design Verification*, 2024. arXiv: 2405.17481 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2405.17481

[12] Texas A&M University, "Accelerating Functional Verification with Machine Learning," DVCon U.S., 2025.

[13] S. Kumari et al., *Optimizing Coverage-Driven Verification Using Machine Learning and PyUVM: A Novel Approach*, 2025. arXiv: 2503.11666 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2503.11666

[14] A. Kumar et al., *Saarthi: The First AI Formal Verification Engineer*, 2025. arXiv: 2502.16662 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2502.16662

[15] A. Kumar et al., "Generative AI Augmented Induction-based Formal Verification," in *2024 IEEE 37th International System-on-Chip Conference (SOCC)*, 2024, pp. 1–2. DOI: 10.1109/SOCC62300.2024.10737803

[16] OpenAI, *GPT-4o model*, Accessed: 2025-06-23, 2024. [Online]. Available: https://openai.com/index/chatgpt/

[17] I. J. Sebastiany et al., "Requirements Engineering in the New Product Development Process: Bibliometric and Systemic Analysis," in *Transdisciplinary Engineering: A Paradigm Shift*. IOS Press, 2017. DOI: 10.3233/978-1-61499-779-5-214 [Online]. Available: http://dx.doi.org/10.3233/978-1-61499-779-5-214

[18] C. Rupp, *Requirements-Engineering und Management*. Munich: Carl Hanser Verlag, 2017.

[19] L. Wheatcraft et al., *Guide to Writing Requirements* (INCOSE), English. INCOSE, 2022.

[20] T. B. Brown et al., "Language Models are Few-Shot Learners," *CoRR*, vol. abs/2005.14165, 2020. arXiv: 2005.14165. [Online]. Available: https://arxiv.org/abs/2005.14165

[21] A. Chowdhery et al., *PaLM: Scaling Language Modeling with Pathways*, 2022. arXiv: 2204.02311 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2204.02311

[22] J. Wei et al., "Chain of Thought Prompting Elicits Reasoning in Large Language Models," *CoRR*, vol. abs/2201.11903, 2022. arXiv: 2201.11903. [Online]. Available: https://arxiv.org/abs/2201.11903

[23] X. V. Lin et al., "Few-shot Learning with Multilingual Language Models," *CoRR*, vol. abs/2112.10668, 2021. arXiv: 2112.10668. [Online]. Available: https://arxiv.org/abs/2112.10668

[24] *GitHub Copilot · Your AI pair programmer — github.com*, https://github.com/features/copilot, [Accessed 15-09-2025].

[25] *AI Assistant | DVT SystemVerilog IDE for VS Code User Guide — eda.amiq.com*, https://eda.amiq.com/documentation/vscode/sv/toc/ai-assistant/index.html, [Accessed 15-09-2025].

[26] H. Foster, *Wilson Research Group IC/ASIC functional verification trend report*, Siemens Blog, 2024.

[27] K. Qayyum et al., "From Bugs to Fixes: HDL Bug Identification and Patching using LLMs and RAG," in *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024, pp. 1–5. DOI: 10.1109/LAD62341.2024.10691874

[28] Cadence Design Systems, *Verisium: AI-Driven Verification*, https://www.cadence.com/en_US/home/tools/system-design-and-verification/ai-driven-verification.html.

[29] Cadence Design Systems, *Verisium Debug: AI-Driven Verification*, https://www.cadence.com/en_US/home/tools/system-design-and-verification/ai-driven-verification/verisium-debug.html.

[30] Synopsys, *Copilot: Generative AI for Chip Design*, https://www.synopsys.com/blogs/chip-design/copilot-generative-ai-chip-design.html.

[31] Synopsys, *Debug Solutions for Verification*, Synopsys Official Website. [Online]. Available: https://www.synopsys.com/verification/debug.html

[32] Synopsys, *Verdi Automated Debug System*, https://www.synopsys.com/verification/debug/verdi.html.

[33] Siemens Digital Industries Software, *EDA and AI: Transforming Electronic Design Automation with Artificial Intelligence*, https://eda.sw.siemens.com/en-US/trending-technologies/eda-ai-page.

[34] Siemens Digital Industries Software, *Questa One: Unified Verification for IC Design*, https://eda.sw.siemens.com/en-US/ic/questa-one/.

[35] ChipAgents AI, *ChipAgents: The Agentic AI Chip Design Environment — chipagents.ai*, https://chipagents.ai/.

[36] Primis AI, *Welcome to the Future of Hardware Design | PrimisAI — primis.ai*, https://primis.ai/.

[37] B. Ahmad et al., "On Hardware Security Bug Code Fixes by Prompting Large Language Models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4043–4057, 2024. DOI: 10.1109/TIFS.2024.3374558

[38] H. Huang et al., *Towards LLM-Powered Verilog RTL Assistant: Self-Verification and Self-Correction*, 2024. arXiv: 2406.00115 [cs.PL]. [Online]. Available: https://arxiv.org/abs/2406.00115

[39] M. ul Islam et al., *AIvril: AI-Driven RTL Generation With Verification In-The-Loop*, 2024. arXiv: 2409.11411 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2409.11411

[40] X. Yao et al., *HDLdebugger: Streamlining HDL debugging with Large Language Models*, 2024. arXiv: 2403.11671 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2403.11671

[41] C. Xiao et al., "LLM-based Processor Verification: A Case Study for Neuromorphic Processor," in *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2024, pp. 1–6. DOI: 10.23919/DATE58400.2024.10546707