

ChatTest: Coverage-Enhanced Testbench Generation for Agile Hardware Verification with LLMs

Gwok-Waa Wan^{1,2}, Shengchu Su¹, Jingyi Zhang¹, Sam Zaak Wong^{1,2},
Mengnv Xing², Lei Ji², Zhe Jiang^{*1,2}, Xi Wang^{*1,2}, Jun Yang^{1,2}

¹ National ASIC Center, School of Integrated Circuits, Southeast University, Nanjing 210096, China

² National Center of Technology Innovation for Electronic Design Automation, Nanjing 211800, China

Abstract—The growing complexity of modern hardware designs has rendered traditional functional verification increasingly time-consuming, with verification costs now dominating the design cycle. While large language models (LLMs) show promise in automating testbench generation, existing approaches struggle with real-world scalability, suffering from poor comprehension of long specifications and complex designs. To address these challenges, we propose ChatTest, a novel, end-to-end, multi-agent LLM framework for coverage-aware, agile hardware verification. Our key innovation lies in a function-mapped, divide-and-conquer architecture that integrates a Verification Description Language (VDL)—a structured, LLM-friendly DSL for precise specification encoding—with Constraint-Aware Segmental Adaptation (CASA) to enable coherent processing of long, heterogeneous design documents. By leveraging retrieval-augmented generation and supervised fine-tuning using multi-hierarchical specification-code alignment, ChatTest ensures accurate translation of functional points into targeted test stimuli. Furthermore, we introduce a coverage-driven feedback loop for automated test augmentation. Evaluated on a new benchmark of 20 complex RTL designs (up to 31K tokens of specification and 4K line-of-code), ChatTest achieves 1.46× higher toggle coverage and 2.28× higher line coverage than SOTA, with a 24.23% improvement in functional coverage, demonstrating its effectiveness in accelerating verification convergence.

Index Terms—TestBench Generation, LLM, Design Verification

I. INTRODUCTION

The rapid evolution of semiconductor technology, driven by the increasing demands of data-intensive applications such as machine learning, blockchain, and big data analytics, has led to a notable increase in the complexity of contemporary silicon designs [1]. Accompanied by continuous advancements in CMOS scaling and the rise of domain-specific architectures (DSAs) [2], the construction and verification of hardware prototypes have become exceedingly time-consuming and labor-intensive [3]–[5]. The traditional verification process now accounts for 50% to 70% of the overall design effort [6]–[8]. In response to these challenges, agile hardware design and verification techniques have become imperative.

Addressing this demand, the field has witnessed the emergence of innovative approaches aimed at automating and expediting the verification process. These include automatic stimulus generation, testbench template automation, and learning-based stimulus optimization. However, despite their potential,

these methods often require significant supervision and may fall short of ensuring comprehensive functional coverage [9].

The emergence and advancement of large language models (LLMs) [10] have presented a promising pathway to overcome the limitations mentioned above, paving the way for a new paradigm of agile hardware verification. Building upon their demonstrated success in software engineering [11], LLMs have spurred growing interest in LLM-Aided Design (LAD) [12]. LLMs have not only shown feasibility in tasks such as hardware code generation [13]–[16], RTL optimization [17]–[19], and error correction [20]–[22], but are also increasingly demonstrating potential in functional verification. Recent studies [23], including AutoBench [24], VerilogReader [25], CorrectBench [26], MEIC [21], and UVLLM [27], indicate potential for automated generation of test stimuli and testbench.

However, significant gaps remain in applying existing approaches to real-world hardware functional verification. First, they lack the capacity to reliably infer complex design intent and to reason over long, heterogeneous specifications [23]: when confronted with large, informally formatted documents, they exhibit misinterpretations, logical inconsistencies, and omission of functional features (low pass rates are illustrated in Figure 1). Moreover, this deficiency results in inadequate constraint reasoning, manifested in the inability of LLMs to extract and enforce timing, boundary, and protocol constraints, producing low coverage tests. Furthermore, these methods scale poorly to industrial-scale designs (evaluations typically target RTL under 1k lines and specifications of only hundreds to a few thousand tokens), and they do not establish a direct mapping between functional points and generated stimuli. This inefficiency is further compounded by the conventional practice of defining functional points from natural language design specifications, which often results in substantial gaps between initial functional descriptions and final stimulus generation, consequently prolonging verification convergence cycles.

Therefore, we introduce **ChatTest**, an advanced end-to-end, multi-agent-driven, automated test planning, coverage-aware generation platform using LLMs for agile hardware verification, specifically engineered to handle the complexities of real-world design challenges. Our main contributions are as follows:

- We analyze the challenges of LLM-assisted test generation and propose **ChatTest**, a multi-agent LLM system for function-mapped test generation, using a Divide-and-Conquer strategy to correlate functional features with test

*Corresponding author: Xi Wang. Email: xi.wang@seu.edu.cn. Zhe Jiang. Email: zhejiang.arch@gmail.com.

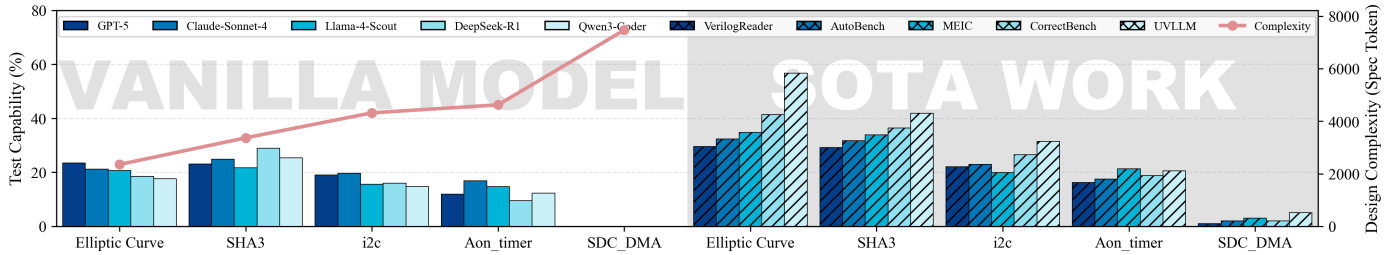


Fig. 1: Test Generation Ability Test on Complex Designs for SOTA LLM and Methods.

generation. Additionally, we enhance sub-task generation by selectively applying retrieval augmented generation (RAG) and supervised fine-tuning (SFT) with multi-hierarchical specification-code alignment.

- We propose Verification Description Language (VDL), a new LLM-friendly Domain Specific Language (DSL) tailored for fine-grained hardware verification specifications, eliminating ambiguities in test planning, constraint analysis, functional feature design, and test case generation.
- Building on VDL, we propose Constraint-Aware Segmental Adaptation (CASA), a mechanism that pre-processes long, mixed-format design specifications by segmenting them into coherent semantic chunks, which improves the ability of models to process complex designs.
- We propose a coverage-aware test augmentation to achieve automatic optimizations towards quality enhancement.
- Considering that the benchmarks used in the previous works were simple designs, on this basis, we have designed a new benchmark that includes 20 complex RTL designs, with specifications up to 31592 tokens and 4473 lines of code. The effectiveness of our method has been validated through extensive experiments and analysis with state-of-the-art (SOTA) LLM and methods.

The remaining sections are organized as follows: Section 2 presents related work and an analysis of current LLM capabilities; Section 3 introduces the design of ChatTest; Section 4 showcases the evaluation; and Section 5 concludes the paper.

II. BACKGROUND

A. LLM-Aided Design Verification

Recent advancements in LLMs have showcased significant capabilities in natural language processing (NLP) [28]. These models utilize contextual information [29] effectively, offering the potential to generate intricate hardware designs from high-level natural language inputs [30]. Several studies have explored how LLMs can be applied in hardware verification. The LLM4DV [31] was subsequently proposed to automate the test stimulus generation using LLMs. However, the LLM4DV framework requires the coverage plan input from developers rather than full-stack automation. Similarly, AssertLLM [32] focuses exclusively on assertion generation, limiting its application scope. Meanwhile, VerilogReader [25] utilizes GPT-4 for testing but is evaluated on relatively simple cases. AutoBench [24] is an LLM-based testbench generation framework

that focuses on the syntactic correctness and functional accuracy of the generated testbench but overlooks coverage requirements. In comparison to [24], CorrectBench [26] improves generation success rates through functional self-verification and iterative self-correction; however, its evaluation is confined to designs from the HDLBits benchmark [33], which typically comprises only tens of lines of code, limiting its generalizability to more complex systems. MEIC [21] and UVLLM [27] adopt the UVM-based paradigm and emphasize test case augmentation and refinement by leveraging simulation logs from iterative executions, but with limited support for effectively processing and reasoning over long-content specifications.

B. LLM-Aided Complex Test Generation Analysis

We observed that previous studies have insufficiently validated their methodologies on complexity designs. These designs typically feature long specifications, structured modular hierarchies, sophisticated interface logic, and a high density of signals at their boundaries. To confirm the limitations of current methods, we selected five designs, as shown in Figure 1, for a preliminary evaluation. We utilized the arithmetic mean of line, toggle [34], and functional coverage [35] metrics to calculate the overall performance. The overall test capability (TC) of each LLM is quantified using the arithmetic mean of the these coverages percentages achieved across all designs as follows:

$$TC = \frac{1}{n} \sum_{i=1}^n C_i \quad (1)$$

where C_i represents the coverage achieved on the i -th design and n is the total number of evaluated designs. If an LLM failed to produce a viable testbench within these iterations, its capability for that particular design was recorded as zero. Figure 1 presents the results. We observed a consistent decline in verification capability across all LLMs and previous works as design complexity increased. This reflects that existing methods are incapable of handling complex designs, lack the ability to extract key constraints, and thus fail to meet the requirements of testbenches targeting coverage. These findings underscore the challenges LLMs face in handling complex design verification.

C. Problem Formulation

We identify and define three primary challenges ($C1-C3$):

- **C1. Degraded Performance in Complex Scenarios:** The accuracy of existing methods drops significantly when applied to complex designs. This limitation stems primarily

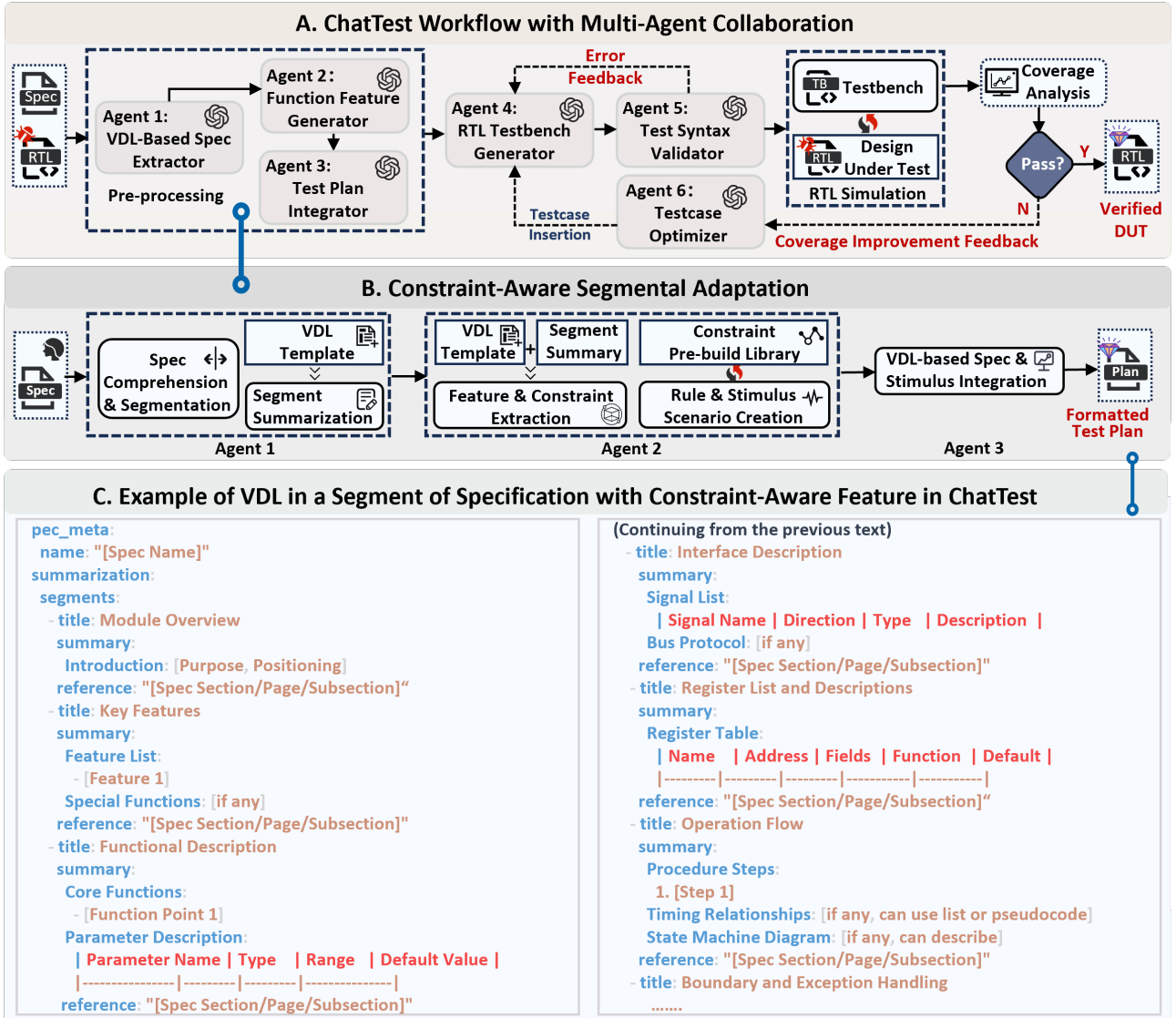


Fig. 2: ChatTest Workflow

from the inability to effectively process long and heterogeneous specifications or code, which undermines reliability and scalability in practical verification workflows.

- **C2. Inadequate Constraint Reasoning:** Verifying complex designs requires models to infer timing, boundary, and protocol constraints from the specification to produce matching stimuli. Current methods lack such optimization, yielding low-quality or incorrect tests.
- **C3. Limited Test Coverage:** Effective test generation involves a series of complex and implicit tasks: functional feature extraction from specifications, test planning, mapping these functional features to stimuli, and optimizing coverage rates, which are missing from existing methods.

III. DESIGN & PHILOSOPHY

A. Solutions & Strategies

In response to the test generation challenges (C1–C3) of LLMs identified in Section II-C, we propose the ChatTest

framework, incorporating the following key solutions (S1–S3):

- **S1. Verification Description Language (VDL) for Constraint-Aware Segmental Adaptation (CASA):** We introduce VDL, a new DSL that structures verification-relevant information for LLMs and provides a unified, model-friendly schema for robust test generation and validation. The CASA procedure segments source specifications, summarizes each segment, integrates constraint-aware features, and applies multi-round reasoning to produce structured VDL outputs that address C1 and C2.
- **S2. Divide-and-Conquer with Multi-Agent Collaboration Featuring Hierarchical Spec–Code Alignment:** We propose a function-mapped test-generation methodology that employs multi-agent decomposition to transform complex design requirements into smaller, well-scoped test cases to mitigate C1. Knowledge enhancement is achieved by constructing a fine-grained spec–code aligned dataset.

- **S3. Coverage-Aware Feedback Reinforcement:** To address **C3**, we design an iterative, coverage-aware test-augmentation paradigm that adjusts generated stimulus based on coverage metrics to reinforce coverage closure.

B. Verification Description Language

We designed a novel domain-specific language (DSL) [36] named Verification Description Language to encapsulate comprehensive design information and verification strategies for each DUT. Figure 2.C illustrates the template of VDL. VDL includes key details such as I/O pins, timing information, functionality, interfaces, functional coverage targets, stimulus sequences, checkpoints, and boundary conditions in YAML [37] format. By serving as a structured intermediate representation, VDL bridges the gap between high-level natural language specifications and machine-friendly verification descriptions.

C. Constraint-Aware Segmental Adaptation

Constraint-Aware Segmental Adaptation (CASA) is a pre-processing and synthesis pipeline that centers constraints to improve LLM handling of long, heterogeneous specifications, as shown in Figure 2.B. CASA enforces structured, constraint-centric segmentation and summarization so timing, boundary, and protocol requirements are explicitly captured and preserved for downstream test generation.

Specifically, CASA segments a specification (spec) along semantic axes: feature descriptions, functional units, I/O mappings, register definitions, procedures, temporal/boundary conditions, and exceptions. Then CASA summarizes the spec: for each stimulus scenario, CASA (1) classifies it as typical, boundary, or exceptional, (2) derives value ranges, rules, and temporal constraints (e.g., setup/hold, handshake, timing), and (3) aggregates these into a validated VDL fragment cross-checked against the spec. To ensure correctness and accelerate stimulus generation, CASA is backed by a prebuilt retrieval library of 15,000 VDL templates covering common primitives and protocols. During generation, the LLM could retrieve and instantiate a template, invoke a protocol-specific incentive generator from the retrieval module, or extract rule predicates and synthesize a stimulus generator that enforces DUT input constraints for custom modules. This hybrid strategy, which combines extracted constraints with a standard DSL template, produces constraint-aware, temporally consistent VDL outputs.

In ChatTest, CASA is implemented by three agents: Agent 1 segments and summarizes; Agent 2 instantiates per-scenario constraints; Agent 3 integrates and validates consolidated VDL fragments. The pipeline ensures timing and protocol constraints are explicit, validated inputs to subsequent function-mapped test generation, as shown below.

D. Function-Mapped Test Generation

Based on CASA, we design a function-mapped test generation methodology with a divide-and-conquer [38] strategy to decompose complex DUTs and streamline the generation process using multiple LLM agents [39]. The ability to break down the spec into fine-grained test points ensures that the generated stimulus are closely aligned with the desired functionality,

leading to enhanced functional coverage while minimizing the overhead associated with unnecessary stimulus simulations.

In the test planning phase, Agent 2 leverages the VDL retrieved from Agent 1 to generate function points of the DUT. Each one represents a specific function coverage to be covered during the testing and simulation processes. Subsequently, Agent 3 processes the function points to produce concise and precise test cases, defining patterns and value ranges of stimulus, and incorporating both directed and random stimulus.

In the testbench generation stage, Agent 4 generates the actual testbench based on the received test points and randomization constraints. By processing fine-grained and well-defined test points, Agent 4 simplifies the logical reasoning required for testbench generation, minimizing ambiguity and hallucination issues commonly associated with LLMs. Finally, the generated test is subjected to a sanity check by Agent 5 to ensure syntactical and functional correctness.

Notably, as the generated test stimuli are derived step by step from the function metrics, this workflow naturally establishes the one-to-one mapping between a generated test and a specific function metric. Through this mapping, we can simply associate the generated tests with the functional coverage status of the DUT to expedite verification convergence.

In addition, the RAG employed in the function-mapped test generation distinguishes between different functional blocks within a module, employing guidance to analyze control logic, state machines, and computational logic. Therefore, LLMs can improve the accuracy and coverage of the generated tests by harnessing the detailed hints from this paradigm.

E. Coverage-Aware Test Augmentation

To further enhance the efficiency of convergence, the ChatTest workflow incorporates a coverage-aware test augmentation. Upon each iteration of the DUT simulation, an automated script extracts the verification logs and reports to analyze the coverage metrics. By adding Agent 6, the ChatTest framework becomes capable of capturing and analyzing the coverage report generated by the RTL simulation and cross-validation with the golden reference model. If the coverage rate meets the user-defined threshold, the verification process is then completed. Otherwise, Agent 6 will analyze the coverage status and append additional test cases to the test plan. New stimulus will be generated by Agent 4 and go through the entire iteration.

This collaborative approach with the multi-agent LLM system allows for continuous improvement through coverage-aware feedback, facilitating the achievement of desired verification goals. The resulting logs and data are used to update the knowledge base, further enhancing ChatTest capabilities through RAG iteration. This iterative process progressively improves the capability of LLMs to design effective test plans with minimal human intervention.

F. RAG & SFT

The ChatTest workflow incorporates Retrieval-Augmented Generation (RAG) to enhance the test generation capability of LLMs. RAG enables LLMs to dynamically utilize external

knowledge from comprehensive verification databases, including design specifications, verification methodology guidelines, and previously validated verification cases. This approach addresses the challenges of few-shot learning due to the scarcity of pre-training datasets, ensuring that the generated verification content aligns closely with practical verification requirements. Our knowledge database includes verified open-source and proprietary RTL designs, corresponding specifications, testbenches, educational materials, IEEE technical standards, and detailed examples from our VDL, alongside chain-of-thought (COT) guidance. We employ the GraphRAG [40] to structure and correlate this knowledge, enabling precise and contextually relevant prompts for LLMs within the ChatTest.

To support end-to-end alignment with our workflow, we constructed a multi-hierarchical, spec-code aligned dataset in which each entry comprises a verified RTL design together with its associated specification, functional points, test points, test cases, and testbench. We began by crawling public repositories containing RTL sources; approximately 10% of the repositories we collected included executable testbenches, and we selected these as the initial seed corpus. Because the original repositories rarely provided explicit mappings between specifications, functional points, and test cases, we used GPT-5 to perform spec-code alignment by supplementing it with metadata. For entries that lacked human-written specifications, we performed reverse-engineering from the RTL to infer likely spec fragments and functional annotations using GPT-5. Finally, we assembled a dataset of 6,350 end-to-end aligned verification examples suitable for model fine-tuning. We used this corpus to fine-tune Llama4-8B with LoRA, which significantly improved the model’s ability to internalize the ChatTest workflow and to generate high-quality testbenches.

G. Put Them All Together: ChatTest Flow

The complete workflow of ChatTest is illustrated in Figure 2. Initially, the user submits the design specification alongside the corresponding RTL code. ChatTest extracts and aligns these specifications with the RTL, subsequently transforming them into test requirements using the VDL framework. Subsequently, ChatTest generates function features based on these VDL-defined test requirements. These functional features serve as the basis for detailed test cases, which together form a comprehensive test plan. This test plan is then input into Agent 4, which crafts tailored testbench code from various templates. Agent 5 preliminarily verifies the generated testbench, checking its format and syntax before RTL simulation. If errors are detected, feedback mechanisms prompt correction or regeneration by Agent 4. Upon successful verification, simulations generate coverage reports, with modules processing and analyzing data to ensure compliance with coverage criteria.

Subsequently, a coverage analysis evaluates the data in depth to ascertain whether the current coverage meets the specified criteria. If the test fails, Agent 6 analyzes the results and provides feedback. This feedback prompts the insertion of additional test cases and refines the test plan to improve coverage. Through this iterative process, ChatTest ensures comprehensive and automated test verification under various conditions.

TABLE I: Experimental Environment Specifications

Server	Configurations
CPU	Intel(R) Xeon(R) Platinum 8358 CPU @2.60GHz 128 Core
GPU	8xA100(80GB,PCIe4.0)
Memory	1024GB
CUDA	12.0
Disk	7.0TB
OS	Ubuntu 22.04

IV. EVALUATION

We fine-tuned Llama4 as Agent 2 and Agent 4 on an NVIDIA server detailed in Table I. Additionally, we employed *gpt-5-2025-08-07* as the remaining agents. Based on their tasks, we assigned different roles and standardized prompts to these models with a connection to the corresponding RAG database.

A. Experimental Setup

1) *Benchmark*: To evaluate our proposed approach, we have developed a complex benchmark suite comprising open-source design modules from OpenCores [41] that are silicon-proven. These designs encompass a variety of functionalities, and the details are presented in Table II. Additionally, we also conducted evaluations on the benchmarks used in previous work to enhance the completeness of the results.

2) *Metrics*: Each DUT underwent a maximum of ten iterations for evaluation. We then compared the generated testbenches in below metrics: **pass rate**, **functional coverage**, **line coverage**, and **toggle coverage**. The pass rate refers to the proportion of generated testbenches that execute without syntax error; functional coverage indicates whether the generated stimulus hits the functional points/features for testing; line/toggle coverage is directly assessed using Synopsys VCS.

B. Result

1) *Coverage Analysis*: Table III shows that ChatTest achieves an average functional coverage of **87.75%** across all tests, surpassing the SOTA model (GPT-5) and prior work (UVLLM) by **50.09%** and **21.50%**, respectively, thereby demonstrating both its usability and capability. On our complex design benchmark, this advantage becomes even more pronounced: in terms of line coverage, ChatTest outperforms GPT-5 by **4.55×** and UVLLM by **1.46×**. For toggle coverage, the improvements are **6.36×** and **2.28×**, respectively, underscoring the robustness of ChatTest in handling complex designs.

2) *Ablation Analysis*: We evaluated the individual contributions of ChatTest. Figure 3 illustrates that the coverage-aware feedback reinforcement within ChatTest significantly enhanced test coverage. Specifically, line coverage increased by an average of **13.72%**, with a maximum improvement of **29.77%**. Additionally, the combination of VDL and CASA demonstrated a notable impact on improving functional convergence across all tested designs. The contribution of this approach ranged from 8% to 31%, with an average improvement of **17.65%**.

V. CONCLUSION

We present ChatTest, an end-to-end test planning and generation platform utilizing LLMs. By adopting a Divide-and-Conquer strategy and leveraging the multi-agent framework

TABLE II: Components of Test Designs

ID	Design Name	Description	Line of Code	Spec Token	Feature Number
01	Adapter	Interface between 2 subsystems at different clock domains	110	785	9
02	TBU	A state-driven data processing unit	208	1024	8
03	FPU_mul	Multiplication on 2 64-bit IEEE 754 floating-point numbers	228	1243	10
04	FPU_div	Division on 2 64-bit IEEE 754 floating-point numbers	356	1287	9
05	Cam_slave	Interface with a camera sensor and an AHB	406	1300	12
06	Streamdh264ct8	Decode VLC for H.264/MPEG-4 AVC standards	432	1542	8
07	Streamdmpg2cbp	Decode coded block pattern	433	1756	8
08	A7WrapSM	A state machine for managing address transactions	475	1345	13
09	Streamdh264ct4	Decode VLC data specific to an H.264 video compression stream	507	1453	8
10	Timer	A multi-functional timer unit	574	1985	8
11	Uart_apb_reg	Interface between the UART and the APB	587	2102	9
12	SHA3	A bridge between the AXI and the AHB	652	3369	14
13	Ct_had_ctrl	Control and manage various debug and trace functionalities	813	2358	12
14	AXI_interconnect	An AXI interconnect for routing AXI transactions	853	3180	17
15	AXI2ahb	A bridge between the AXI and the AHB	1030	2312	19
16	Elliptic Curve	A core for computing the addition of two elements in the elliptic curve group	1554	2355	26
17	I2C	A two-wire, bi-directional serial bus protocol	1087	4320	43
18	Aon_timer	Wakeup and watchdog timers running on a low-power, always-on clock	1981	4624	48
19	SDC_DMA	A direct memory access implementation for sd card controller	3576	7471	56
20	Spi_host	Serial peripheral interface for host mode	4473	31592	62

TABLE III: Result of All Tested Models and Works on Different Benchmarks

		VerilogEval (156 Tasks)				RTLML (30 Tasks)				ChatTest-Bench (20 Tasks)			
		Pass Rate	Line Cov.	Toggle Cov.	Func. Cov.	Pass Rate	Line Cov.	Toggle Cov.	Func. Cov.	Pass Rate	Line Cov.	Toggle Cov.	Func. Cov.
LLM*	GPT-5	47.64%	84.62%	56.67%	44.63%	40.00%	58.91%	34.72%	36.75%	30.00%	17.63%	11.56%	30.91%
	Claude-4	45.73%	86.09%	58.06%	46.17%	36.67%	55.10%	42.33%	32.64%	35.00%	16.33%	9.13%	32.44%
	Llama-4	43.20%	82.90%	63.33%	40.06%	33.33%	37.89%	29.08%	32.59%	25.00%	16.67%	7.07%	24.72%
	DeepSeek	46.85%	85.43%	43.25%	42.90%	33.33%	32.34%	18.45%	28.70%	20.00%	21.72%	4.35%	20.93%
	Qwen3	51.64%	78.54%	35.83%	39.99%	26.66%	45.67%	24.58%	28.75%	20.00%	9.45%	4.58%	22.17%
Work†	VerilogReader	55.53%	90.39%	60.61%	49.00%	40.00%	69.83%	36.72%	36.44%	30.00%	27.63%	19.24%	36.64%
	AutoBench	59.16%	93.70%	65.49%	52.34%	43.33%	76.32%	36.15%	45.93%	30.00%	25.91%	17.32%	35.60%
	MEIC	84.52%	96.15%	75.83%	75.62%	76.67%	87.70%	47.68%	61.53%	45.00%	49.16%	23.79%	45.48%
	CorrectBench	83.87%	95.42%	69.47%	57.31%	66.67%	86.39%	45.29%	59.06%	40.00%	27.98%	25.67%	41.40%
	UVLLM	98.53%	100.00%	83.55%	79.31%	83.33%	90.00%	72.16%	67.32%	50.00%	54.85%	32.26%	52.10%
	ChatTest	96.76%	100.00%	95.92%	95.24%	93.33%	98.48%	93.19%	91.67%	90.00%	80.15%	73.49%	76.33%

*: The detailed version of LLM: GPT-5-2025-08-07, Claude4-Sonnet, Llama4-Scout, DeepSeek-R1 and Qwen-3-Coder.

†: For all works, if there is no fine-tuned model or dedicated API, the latest GPT-5-2025-08-07 API consistent with ChatTest is used to ensure fairness. The introduction of these baseline works can be found in Section II-A. VerilogEval or RTLML is used by these works.

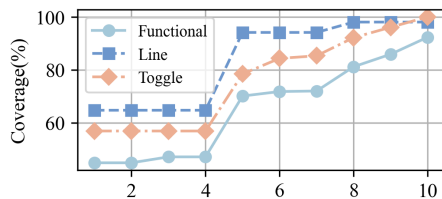


Fig. 3: Coverage Opt

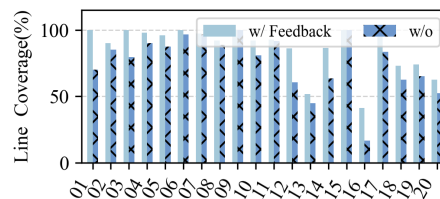


Fig. 4: Feedback Analysis

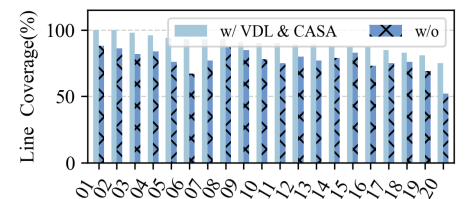


Fig. 5: VDL & CASA Ablation

with RAG and SFT, ChatTest automates testbench generation and enhances functional coverage. Besides, the introduction of the VDL and CASA refines AI-friendly design verification for complex designs. Through a systematic evaluation, we demonstrate that ChatTest surpasses SOTA across several key metrics. ChatTest achieves a $2.28\times$ increase in toggle coverage and a $1.46\times$ improvement in line coverage compared to UVLLM on the complex design scenario. Furthermore, ChatTest achieves at least a 24.23% improvement in functional

coverage over UVLLM. These findings underscore the potential in accelerating the real-world hardware functional verification.

VI. ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program (Grant No.2024YFB4405600), the National Natural Science Foundation of China under NSFC (Grant No. 92464301), and the Key Research and Development Program of Jiangsu Province (Grant No.BG2024010).

REFERENCES

- [1] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Masterrtl: A pre-synthesis ppa estimation framework for any rtl design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.
- [2] Irv Englander and Wilson Wong. *The architecture of computer hardware, systems software, and networking: An information technology approach*. John Wiley & Sons, 2021.
- [3] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, et al. Towards Developing High Performance RISC-V Processors Using Agile Methodology. In *MICRO*, 2022.
- [4] Philippe Coussy and Adam Morawiec. *High-level synthesis*, volume 1. Springer, 2010.
- [5] Anmol Mathur and Venkat Krishnaswamy. Design for verification in system-level models and rtl. In *Proceedings of the 44th annual Design Automation Conference*, pages 193–198, 2007.
- [6] Gwok-Waa Wan, SamZaak Wong, Shengchu Su, Chenxu Niu, Ning Wang, Xinlai Wan, Qixiang Chen, Mengnv Xing, Jingyi Zhang, Jianmin Ye, Yubo Wang, Rongchang Song, Tao Ni, Qiang Xu, Nan Guan, Zhe Jiang, Xi Wang, Yong Chen, and Jun Yang. Towards end-to-end benchmarking of llm-aided design verification. In *Proceedings of the 40th AAAI Conference*, pages 1–9, Singapore, Jan 2026.
- [7] Prakash Rashinkar, Peter Paterson, and Leena Singh. *System-on-a-chip Verification: Methodology and Techniques*. Springer Science & Business Media, 2001.
- [8] Carl Pixley, Aruna Chittor, Fred Meyer, Steve McMaster, and Dan Benua. Functional verification 2003: technology, tools and methodology. In *ASIC, 2003. Proceedings. 5th International Conference on*, volume 1, pages 1–5. IEEE, 2003.
- [9] Aruna Jayasena and Prabhat Mishra. Directed test generation for hardware validation: A survey. *ACM Computing Surveys*, 56(5):1–36, 2024.
- [10] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.
- [11] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572, 2023.
- [12] Yingbing Huang, Lily Jiaxin Wan, Hanchen Ye, Manvi Jha, Jinghua Wang, Yuhong Li, Xiaofan Zhang, and Deming Chen. New solutions on llm acceleration, optimization, and application. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–4, 2024.
- [13] Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: controlled verilog generation with discriminative guidance. In *Proceedings of the 41st International Conference on Machine Learning*, pages 40145–40153, 2024.
- [14] Sam-Zaak Wong, Gwok-Waa Wan, Dongping Liu, and Xi Wang. Vgv: Verilog generation using visual capabilities of multi-modal large language models. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–5.
- [15] Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. Chatcpu: An agile cpu design and verification platform with llm. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.
- [16] Tianyang Liu, Qi Tian, Jianmin Ye, LikTung Fu, Shengchu Su, Junyan Li, Gwok-Waa Wan, Layton Zhang, Sam-Zaak Wong, Xi Wang, and Jun Yang. Chatchisel: Enabling agile hardware design with large language models. In *Proceedings of the ACM Conference on Integrated System Electronic Design Automation*, 2024.
- [17] Xufeng Yao, Yiwen Wang, Xing Li, Yingzhao Lian, Ran Chen, Mingxuan Yuan, Hong Xu, and Bei Yu. Rtlrewriter: Methodologies for large models aided rtl code optimization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–7, 2024.
- [18] Gwok-Waa Wan, Yubo Wang, Sam-Zaak Wong, Jia Xiong, Qixiang Chen, Jingyi Zhang, Mingchi Zhang, Tao Ni, Mengnv Xing, Yusheng Hua, et al. Genben: A generative benchmark for llm-aided design. In *Arxiv*, 2025.
- [19] Jason Blocklove, Shailja Thakur, Benjamin Tan, Hammond Pearce, Siddharth Garg, and Ramesh Karri. Automatically improving llm-based verilog generation using eda tool feedback. *ACM Transactions on Design Automation of Electronic Systems*, 2025.
- [20] YunDa Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.
- [21] Ke Xu, Jialin Sun, Yuchen Hu, Xinwei Fang, Weiwei Shan, Xi Wang, and Zhe Jiang. Meic: Re-thinking rtl debug automation using llms. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.
- [22] Junyan Li, Sam-Zaak Wong, Gwok-Waa Wan, Xi Wang, and Jun Yang. Eda-debugger: An llm-based framework for automated eda runtime issue resolution. In *2025 26th International Symposium on Quality Electronic Design (ISQED)*, pages 1–7. IEEE, 2025.
- [23] Nan Wu, Yingjie Li, Hang Yang, Steve Dai, Cong Hao, Cunxi Yu, and Yuan Xie. Survey of machine learning for software-assisted hardware design verification: Past, present, and prospect. *ACM Transactions on Design Automation of Electronic Systems*, 29(4):1–42, 2024.
- [24] Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. Autobench: Automatic testbench generation and evaluation using llms for hdl design. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, pages 1–10, 2024.
- [25] Ruiyang Ma, Yuxin Yang, Ziqian Liu, Jiayi Zhang, Min Li, Junhua Huang, and Guojie Luo. Verilogreader: Llm-aided hardware test generation. *arXiv preprint arXiv:2406.04373*, 2024.
- [26] Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. Correctbench: Automatic testbench generation with functional self-correction using llms for hdl design. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2025.
- [27] Yuchen Hu, Junhao Ye, Ke Xu, Jialin Sun, Shiyue Zhang, Xinyao Jiao, Dingrong Pan, Jie Zhou, Ning Wang, Weiwei Shan, et al. Uvllm: An automated universal rtl verification framework using llms. *arXiv preprint arXiv:2411.16238*, 2024.
- [28] Bonan Min, Hayley Ross, Elinor Sulem, Amir Poursan Ben Veysseh, Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.
- [29] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4425–4445, 2022.
- [30] Jan Kocóń, Igor Cichecki, Oliwier Kaszyca, Mateusz Kochanek, Dominika Szydio, Joanna Baran, Julita Bielaniec, Marcin Gruza, Arkadiusz Janz, Kamil Kanclerz, et al. Chatpzt: Jack of all trades, master of none. *Information Fusion*, 99:101861, 2023.
- [31] Zixi Zhang, Greg Chadwick, Hugo McNally, Yiren Zhao, and Robert Mullins. LLM4DV: Using large language models for hardware test stimuli generation. In *Machine Learning for Systems 2023*, 2023.
- [32] Wenji Fang, Mengming Li, Min Li, Zhiyuan Yan, Shang Liu, Hongce Zhang, and Zhiyao Xie. AssertLLM: Generating hardware verification assertions from design specifications via multi-llms. In *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024.
- [33] Mingjie Liu, Nathaniel Pinckney, Bruce Khailany, and Haoxing Ren. Verilogval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2023.
- [34] Khader Abdel-Hafez, Michael Dsouza, Likith Kumar Manchukonda, Elddie Tsai, Karthikeyan Natarajan, Ting-Pu Tai, Wenhao Hsueh, and Smith Lai. Comprehensive power-aware atpg methodology for complex low-power designs. In *2022 IEEE International Test Conference (ITC)*, pages 334–339. IEEE, 2022.
- [35] Azade Nazi, Qijing Huang, Hamid Shojaei, H Asghari Esfeden, Azalia Mirhosseini, and Richard Ho. Adaptive test generation for fast functional coverage closure. *DVCON USA*, 2022.
- [36] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
- [37] Farima Farmahinfarahani, Petr Babkin, Salwa Alamir, and Xiaomo Liu. From zero to sixty at the speed of rag: Improving yaml recipe generation via retrieval. In *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, pages 49–56. IEEE, 2025.
- [38] Bingxuan Li, Yiwei Wang, Tao Meng, Kai-Wei Chang, and Nanyun Peng. Control large language models via divide and conquer. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15240–15256, 2024.
- [39] Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025.
- [40] William Jones Beckhauser and Renato Fileto. Echorag: a framework for enhancing language models with graph-rag and in-context learning. *Machine Learning*, 114(10):215, 2025.
- [41] OpenCores. <https://opencores.org/>. [Online; accessed 21-05-2024], 2024.