HWFixBench: Benchmarking Tools for Hardware Understanding and Fault Repair

Weimin Fu Kansas State University Manhattan, Kansas, USA weiminf@ksu.edu

Yier Jin University of Science and Technology of China Hefei, Anhui, China jinyier@ustc.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated remarkable potential in automating coding tasks, making their application in Electronic Design Automation (EDA) an increasingly popular research direction. However, existing benchmarks for evaluating LLMs' hardware-related capabilities are overly simplistic and fail to capture the complexity of real-world hardware projects. To address this gap, we constructed HWFixBench, a comprehensive benchmark derived from 500 pull requests and 1,481 bug fixes across 12 widely-used open-source hardware projects. HWFixBench reflects the challenges and complexity of real-world hardware tasks, providing a rigorous testbed for assessing whether LLMs can truly automate hardware development. We evaluate general-purpose LLMs, hardware-specialized LLMs, and prompt-based methods on HWFixBench, offering insights into their performance and limitations. To facilitate further research, we open-source the dataset and provide full hardware simulation support, enabling robust crosscategory comparisons of solutions.

CCS Concepts

 \bullet Hardware \rightarrow Software tools for EDA; \bullet Computing methodologies \rightarrow Language resources.

Keywords

Large Language Model, Hardware Benchmarking, Semi-Synthetic Dataset, Electronic Design Automation, Metric

ACM Reference Format:

Weimin Fu, Shijie Li, Yier Jin, and Xiaolong Guo. 2025. HWFixBench: Benchmarking Tools for Hardware Understanding and Fault Repair. In *Great Lakes Symposium on VLSI 2025 (GLSVLSI '25), June 30-July 2, 2025, New Orleans, LA, USA*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3716368. 3735171

\odot

This work is licensed under a Creative Commons Attribution 4.0 International License. GLSVLSI '25, New Orleans, LA, USA © 2025 Copyright held by the owner/author(s). ACM ISBN 979-8-4007-1496-2/2025/06 https://doi.org/10.1145/3716368.3735171 Shijie Li University of Science and Technology of China Hefei, Anhui, China shijie_li@mail.ustc.edu.cn

> Xiaolong Guo Kansas State University Manhattan, Kansas, USA guoxiaolong@ksu.edu



Figure 1: Three Key Tasks in HWFixBench and Their Corresponding Examples in Real-World Version Control

1 Introduction

Large Language Models (LLMs) have been rapidly integrated into products, offering question-answering [30] and code assistance [43] services. While popular in the software domain, increasing efforts [20, 27, 33] have been made to explore and refine LLM in the hardware domain. Hardware-related benchmarks designed to evaluate LLM have also emerged. However, existing benchmarks [1, 16, 18, 22] have reached a state of saturation in assessing model performance and fail to reflect real-world hardware tasks. These benchmarks do not effectively delineate the performance boundaries of state-of-the-art (SOTA) LLMs.

In the software domain, the SWE-Bench [15] constructed from real-world projects has become the de facto standard for evaluating software engineering performance. Similarly, there is a pressing need to establish a similar benchmark in the hardware domain. Given the rapid advancements in LLMs, existing evaluation benchmarks quickly become obsolete, failing to capture these models' evolving complexity and capabilities. This underscores the urgent need for a forward-looking benchmark to track meaningful progress over the forthcoming years. As a dataset-centric study, HWFixBench sets rigorous and realistic baselines by leveraging real-world hardware repair tasks supported by meticulous data selection.

HWFixBench is derived from high-quality open-source hardware projects. Specifically, we include projects with over 50 citations in academic literature or those that have served as evaluation targets

Weimin Fu, Shijie Li, Yier Jin, and Xiaolong Guo

in EDA security competitions, as such projects have rigorous validation. Additionally, we leverage version control data to trace the discovery and resolution of hardware bugs, forming the foundation of HWFixBench. For usability, we modify dependencies in the original designs to ensure they can be simulated as standalone components. The Figure 1 illustrates the three tasks in HWFixBench and their real-world counterparts in version control systems.

- (1) **Task 1**: Bug Detection: The LLM identifies bugs directly from the hardware design, analogous to issues reported by users or test engineers during the hardware design's deployment.
- (2) Task 2: Bug Fixing: Given a reported issue and the corresponding hardware design, the LLM generates a fix, mirroring the process where a developer submits a pull request to address a specific issue.
- (3) Task 3: Patch Generation from Design: The LLM generates patches directly from the hardware design without an explicit issue report. While such an approach is uncommon in conventional development workflows, it represents a prevalent application of LLMs in hardware design automation.

To summarize, our key contributions are as follows:

- High-Quality Dataset Construction: We developed a high-quality dataset based on open-source hardware projects with outstanding reputations. Our preprocessing techniques removed design dependencies and enhanced issue descriptions to ensure stability, mitigating the impact of user expertise variability on issue quality.
- Comprehensive Model Evaluation: We systematically evaluated various models, including proprietary models, open-source models with general coding capabilities, hardware domain expert fine-tuned models, and prompt-based approaches. Performance was assessed across all three tasks. For fine-tuned models, we measured their performance gains over baseline models. For prompt-based methods, we compared the performance improvement achieved using task-specific prompt formats versus standard evaluation settings within the same model.
- Open-Source Benchmark and Testing Framework: We release our benchmark to enable researchers testing their fine-tuned or other accessible models. We also provide a testbench, allowing for more rigorous evaluation endpoints beyond the current version.

2 Background

2.1 Version Control

Version control documents a project's entire lifecycle, from the initial code creation to deployment, including feature development, **bug fixes**, and refactoring. These records capture the codebase's evolution and provide contextual information, such as developer discussions, design decisions, and performance optimizations. With the hardware open-source communities' rise, version control has become indispensable in development [10]. Notable open-source hardware groups utilize Git [32] to track modifications throughout the stages of the hardware development, including RTL design, simulation, verification, and synthesis.

In this work, version control serves as a raw information source.

- Issue Tracking: Documents design bugs.
- Patch: The patches to correct errors.

2.2 LLM Backend: Use LLM as a judge

Human-processed datasets have become prohibitively expensive with the increasing sizes. Leading AI companies have begun employing PhDs in computer science, mathematics, and other subjects under the title PhD Data Partner to annotate data. However, the limited availability of PhD-level experts constrains the speed, and the high associated costs have sparked interest in fully automated annotation. LLM Judges have been demonstrated as alternatives [5, 21, 48]. LLMs are considered well-suited for evaluating code semantic correctness. CodeJudge [31], [40], and [31] have explored the feasibility of using LLMs for code evaluation and have found that they can deliver stable performance. This evaluation cannot replace strict functional testing but is the best assessment for current LLM applications in the hardware domain. Currently, LLMs struggle to generate hardware that passes functional verification. However, compared to starting from scratch, having an initial output with flaws is often considered a better start [36].

The assessment of abilities is often conducted in a closed-book manner [21], so HWFixBench exclusively provides the test to LLMs.

2.3 Simulator Backend

The gold standard in the hardware domain remains simulation. To pave the way for fully automated in the future, HWFixBench provides a simulator backend. We expanded the macros, implemented minimal external functions, and developed simulation testbenches.

Our evaluation follows a *zero prior knowledge testing* approach, meaning that no additional inputs are provided aside from the design (and issue descriptions in Task 2). This test scenario aligns with the current LLM automation trend of turning labor into software.

As discussed in Subsection 2.2, imposing strict functional simulation requirements would result in all LLMs yielding **zero** performance. This paper does not report detailed metrics for the simulator backend. However, we have open-sourced the benchmark, allowing approaches with non-LLM applications and future powerful LLMs.

2.4 Evaluation Methods

We use the pass rate as the performance metric, which measures the success rate of a test given a single attempt. This is a stringent criterion that is not inherently LLM-friendly. LLMs are regarded as experts that produce judgments and conclusions by reading and comparing information in the hardware domain [10].

3 Methodology

3.1 Benchmark Construction

3.1.1 Preprocessing Pipeline. The Figure 2 black lines represent the data selection process, illustrating the steps to filter data from version control systems for inclusion in HWFixBench. Our benchmark collects all pull requests (PRs) from 12 open-source hardware projects. For each PR, we identify and retrieve the corresponding issue it resolves, selecting only those related to bugs. PRs containing file modifications and aligned patches are filtered, providing the raw dataset. For Bug Detection, the issue reports extracted from version control serve as ground truth. For Bug Fixing, the relationship between issues and PRs provides paired data for testing LLMs in generating fixes given an issue and its associated design. For HWFixBench: Benchmarking Tools for Hardware Understanding and Fault Repair



Figure 2: Relationship Between Version Control Data and Specific Tasks

Patch Generation from Design, PR modifications (patches) serve as reference solutions. Table 1 presents a step-by-step statistical breakdown, detailing the data refinement process—from the total number of collected PRs to the final set of bug-related, issue-resolved file modifications in HWFixBench (excluding projects that were not selected). After filtering, HWFixBench includes nine projects, 500 pull requests, and 1,481 bug-fix-related file modifications.

Table 1: Statistical Breakdown of Version Control Data from Selected Open-Source Projects: From Total Pull Requests to Bug-Fix-Related File Modifications

Project	Total Pull Request	Issue Solved PR	Selected PR	Selected File Change
OpenTitan	19297	1513	300	556
Ibex	1403	583	99	271
CVA6	1641	607	39	211
CVW	1009	339	25	288
CV32E40X	774	240	17	73
CV32E40P	532	172	11	31
CV32E40S	438	90	6	31
CV-HPDCache	42	17	2	7
Core-V MCU	188	41	1	7
Total	25471	3640	500	1481

3.1.2 *Post-Processing.* We observed that community issues are often vague and do not explicitly describe the bugs, making them unreliable as ground truth. To address this, we manually curated and enhanced the issue descriptions to ensure they clearly articulate the specific bugs being addressed. Additionally, we performed data sanitization and preprocessing on the design files, including de-identifying sensitive information, expanding macros, and minimizing external dependencies referenced within the files. We also give testbenches to facilitate future simulation-based comparisons.

3.1.3 Alpaca Style Dataset Construction. HWFixBench defines a well-defined Alpaca-style instruction-input-output format to ensure interpretability and systematic evaluation of LLM performance.

Task 1: The LLM to identify the corresponding bug report associated with a given hardware design file. The input comprises the full hardware design; the expected output is the relevant issue describing the bug. This task simulates how test engineers or users detect and report hardware issues.

Task 2: The LLM generates a corrective patch based on a reported issue and the corresponding hardware design. The input includes both the bug report and the affected design, while the output is a patch that addresses the issue. This process aligns with real-world debugging workflows, where developers submit fixes via pull requests in version control systems.

Task 3: Unlike the previous task, this task requires the LLM to identify and fix errors autonomously without an explicit bug report. The input consists solely of the hardware design file, and the output is a minimal corrective patch if an issue is detected; otherwise, the

model should return "No modification needed." This task evaluates the model's ability to perform self-directed debugging and repair, mimicking an automated hardware verification and correction system.

3.2 Benchmark Statistics

Table 2 provides two complementary statistical perspectives to comprehensively analyze HWFixBench. Line counts reflect traditional EDA metrics, offering insights into the structural complexity of hardware designs. Token counts, computed using the LLaMA-3 tokenizer, align with the requirements of LLM research by capturing the semantic granularity of the code. Additionally, we present statistical results for the comparative benchmarks discussed in Section 5, enabling a thorough evaluation of HWFixBench in relation to existing benchmarks. The high variability in line/token counts and high skewness underscores the inherent complexity of real-world hardware design changes. HWFixBench contains a large amount, minimal edits, and outlier cases where designs or patches exhibit immense scopes. HWFixBench is a stress hardware domain test for LLM across issue detection, patch generation, and implicit bug resolution.

 Table 2: Statistical Comparison of HWFixBench and Common Hardware LLM Benchmarks

Matric	Count	Line				Token				
Metric	count	mean	max	0.25	skew	mean	max	0.25	skew	
VerilogEval	156	19.15	76	11.00	1.77	123.31	571	53.50	1.88	
FVEval	192	421.95	2139	103.00	2.14	2816.23	14587	651.00	2.14	
RTLLM	50	54.64	195	24.75	1.74	415.20	1607	166.25	1.86	
CreativeEval	119	21.03	108	10.00	2.55	150.16	1019	66.50	2.84	
CorrectBench [28]	156	18.60	77	11.00	1.85	119.16	571	49.75	1.93	
HWFixBench	1232	487.65	3488	12.00	2.28	4564.29	203	1266.50	3.08	

3.3 Metric Evaluation

We designed the prompt based on the assumption in Sub-Section 2.4. The prompt that instructs the evaluator to compare the predicted output with the ground truth. The prompt asks ChatGPT40 whether both results convey the same solution and solve the same issue. If they align, the evaluator returns "MATCHED"; otherwise, "NOT _ MATCHED", accompanied by a short explanation. This structured evaluation ensures consistency in assessing model performance across different tasks.

4 Experiment

4.1 LLMs Under Test

HWFixBench was evaluated across four categories of LLMs: generalpurpose open-source models, proprietary models, fine-tuned models and prompt methods.

For general-purpose open-source LLMs, we selected a diverse set of models spanning multiple organizations and parameter scales:

- DeepSeek Coder [13] (6.7B, 7B, 33B), R1 Distill Models [7] (Qwen 1.5B, 7B, 14B, 32B, LLaMA 8B, 70B)
- Google Gemma 7B [24], Gemma2 27B [29]
- Meta AI LLaMA (LLaMA 2 70B [37], LLaMA 3 8B, 70B, LLaMA 3.1 8B, 70B, LLaMA 3.2 11B, LLaMA 3.3 70B [9])
- Microsoft models (Phi-3.5 MoE [3], Phi-4 [2])
- Qwen models [42] (Qwen2.5 14B, 32B, QwQ 32B)

For proprietary models, we included:

• OpenAI models (GPT-40, GPT-40 mini, O1, O1mini, O3mini)

For fine-tuned(FT) models, we applied a selection process to ensure compatibility with modern LLM pipelines. Specific models were excluded due to obsolescence (MEIC [41], VeriGen [34], AutoV-Coder [12]) or lack of availability (CraftRTL [19], LLM4SecHW [11], [26]). The final selection included:

• CodeV [46], OriGen [6], RTLCoder [20], and VeriSeek [39]

For prompt-based methods, we categorized them into multi-agent systems and normal prompt:

- Multi-Agent Systems: SPEC in [28] were removed. For VerilogCoder [14], we provided additional simulation.
- Normal Prompt: We adapted the general prompt format to align with methodologies: PromptV [25], RTLRewriter [44], VerilogReader [23], [4], RTLFixer [38], and [35].

4.2 Overall Performance across models

4.2.1 Task 1. The best-performing model was OpenAI's proprietary GPT-40, achieving 11.44%. The top-performing open-source model, Microsoft's Phi-4, reached 10.06%, while most other open LLMs achieved between 3% and 8%, a performance level that renders them unusable for real-world applications. Among the finetuned models, OriGen demonstrated a substantial 61.02% relative improvement; however, its absolute performance remained low at 7.71%. This raises questions about whether the observed improvement stems from the weak baseline performance of the underlying DeepSeek Coder series or if fine-tuning genuinely offers meaningful enhancements when applied to powerful models.

4.2.2 Task 2. The strongest performer was o1mini, achieving an impressive accuracy of 83.44%. Among open-source models, DeepSeek R1 Distill Llama 80B obtained the highest score at 43.83%. Notably, models designed for advanced reasoning, often considered superior in general-purpose tasks, did not demonstrate a clear advantage in this project. Model performance positively correlated with scale within the same generation and institution. Only OriGen demonstrated a performance gain among fine-tuned models, while the remaining three models experienced degradation.

This result aligns with the expectation that LLMs can assist in code repair; however, it also highlights a significant gap between proprietary and open-source models. The disparity suggests that OpenAI's closed-source models may have been explicitly trained with hardware-related data, whereas most open-source models lack sufficient exposure to this domain.

4.2.3 Task 3. Despite being the most challenging task, the models' performances exceeded results from Task 1. QwQ 32B Preview achieved the highest score among all models with 29.95%. Only VeriSeek among the fine-tuned models showed a performance gain. However, it remains unclear whether the relative improvement is genuinely due to the fine-tuning process or reflects the weak baseline performance of the DeepSeek Coder 7B model (12.74% vs. 15.50% from the 6.7B model). The fine-tuned model's score of 13.23% also lags behind the highest-performing fine-tuned model, RTLCoder (14.20%).

4.3 Knowledge Cutoff Analysis

A common question in evaluating LLMs is whether their performance reflects genuine understanding or mere memorization of internet-crawled data. Since time progresses linearly and unidirectionally, bug fixes made public after an LLM's knowledge cutoff date could not have been present in its training data. Using each model's reported knowledge cutoff date as a reference, we partition the benchmark into pre-cutoff and post-cutoff subsets. Then, we analyze the performance gap between them to assess whether an LLM truly understands the task as shown in Table 4.

While this approach appears ideal theoretically, hardware data inside LLMs is often actively removed. A notable example is removing hardware code in StarCoder [17] between iterations. This omission creates a fundamental limitation: only a post-cutoff performance drop can reliably indicate that the model lacks proper understanding of hardware tasks. Conversely, an unexpected performance increase cannot be considered evidence of experience, as the model's training data may have contained little hardware knowledge, rendering the experiment's assumptions invalid.

4.3.1 Task1. Only the fine-tuned OriGen model exhibited performance degradation after the knowledge cutoff. Given the overall low baseline for this task, we can infer that LLMs generally perform poorly in directly identifying bugs in hardware designs. Still, their performance does not strongly depend on whether such data was present in the training set.

4.3.2 Task 2. DeepSeek's Coder series maintained stable performance, while the Distill series degraded significantly. Since the latter models did not undergo post-training and only relied on supervised fine-tuning, this suggests a stronger dependence on the training dataset. Similarly, Google's Gemma 2, most Meta AI models, and Microsoft's models exhibited similar post-cutoff performance drops. However, Qwen models improved across iterations—while the Qwen 2.5 series still suffered post-cutoff degradation, the nextgeneration models reversed this trend. For OpenAI models, GPT-40 and GPT-40 mini did not perform poorly, while the "thinking" models exhibited a decline.

Among FT models, those designed for hardware code generation (CodeV and RTLCoder) did not show a drop, whereas debug-specific models suffered a significant decrease. This indicates that the latter depends on explicitly presenting test cases in the training data.

4.3.3 Task 3. The results were similar to Task 2, except that "thinking" models did not exhibit negative optimization this time. However, fine-tuned models experienced an overall performance decline, suggesting that models trained explicitly for hardware debugging struggle to generalize beyond their pre-existing knowledge.

4.4 Design Length Evaluation

The ability to extrapolate LLM performance from small-scale to large-scale designs has long been an open question, and this challenge is particularly pronounced for LLMs if they were trained on short designs. Based on two typical hardware LLM datasets[20, 45], we identified a typical maximum token length of 2, 567 tokens. To simulate real-world scenarios where additional context is appended, we extended this length by a factor of $1.5\times$, partitioning the benchmark into a short subset (\leq 3, 852 tokens) and a long subset (> 3, 852

Open Source Model					Proprietary Model								
Affiliation	Name	Task1	Task2	Task3	Affiliation	Name		Thinking	Task1	Task2	Task3		
	Coder 6.7b base	4.79%	17.13%	15.50%		g	gpt 4o		11.44%	83.04%	21.27%		
	Coder 7b Instruct v1.5	6.82%	22.00%	12.74%		gpt 40 mini		INO	7.22%	75.41%	21.92%		
	Coder 33b Instruct	6.90%	16.48%	13.88%	OpenAI		01		5.76%	78.57%	24.59%		
	R1 Distill Llama 8B	7.06%	38.07%	23.38%		01	o1 mini		5.52%	83.44%	22.73%		
DeepSeek	R1 Distill Llama 70B	6.66%	43.83%	24.43%		03	mini		6.98%	75.65%	26.22%		
	R1 Distill Qwen 1.5B	3.25%	23.86%	17.21%			Supervise	d Finetuning I	Method				
	R1 Distill Qwen 7B	5.84%	29.87%	24.11%	Task	Affiliation	Model	Pass Rate	Base Model	Pass Rate	Benefit		
	R1 Distill Qwen 14B	5.44%	38.72%	25.41%		CAS	CodeV DS [46]	7.22%	DeepSeek		50.85%		
	R1 Distill Qwen 32B	7.14%	42.13%	27.27%		PKU	OriGen [6]	7.71%		4.79%	61.02%		
	Gemma 7B	2.35%	23.30%	9.98%	Task1	HKUST	RTLCoder [20]	6.33%	Couer 0.70 base		32.20%		
Google	Gemma2 27B	3.73%	10.23%	9.58%		CUHK VeriSeek [39]	VeriSeek [39]	4.30%	DeepSeek Coder	6.82%	-36.90%		
									7b Instruct v1.5				
	Llama 2 70b hf	5.76%	12.66%	10.23%		CAS	CodeV DS [46]	12.42%	DeepSeek Coder 6.7b base	17.13%	-27.49%		
	Llama 3 8B Instruct	7.39%	28.73%	17.37%		PKU	OriGen [6]	21.59%			26.07%		
	Llama 3 70B Instruct	5.68%	31.01%	15.99%	Task2	HKUST	RTLCoder [20]	9.58%			-44.08%		
	Llama 3.1 8B Instruct	7.14%	31.25%	21.43%		CUHK	VeriSeek [39]	12.74%	DeepSeek Coder	22.00%	-42.07%		
Meta						0.10	e luperd	10.0/	/b instruct v1.5		0.05		
	Llama 3.1 70B Instruct	7.06%	35.31%	20.13%		CAS	CodeV DS [46]	13.96%	DeepSeek Coder		-9.95%		
	Llama 3.2 11B Vison	7.31%	31.33%	22.32%		PKU	OriGen [6]	13.31%	6.7b base	15.50%	-14.14%		
	Llama 3.3 70B Instruct	7.22%	39.53%	18.02%	Task3	HKUST	RTLCoder [20]	14.20%			-8.38%		
	Qwen2.5 14B	6.41%	32.31%	19.81%		CUHK	VeriSeek [39]	13.23%	DeepSeek Coder 7b Instruct v1.5	12.74%	3.82%		
Qwen	Qwen2.5 32B	7.14%	34.90%	18.10%									
	QwQ 32B Preview	8.93%	40.02%	29.95%									
NC 0	Phi3.5 MoE Instruct	5.60%	11.77%	9.82%									
Microsoft	Phi-4	10.06%	36.61%	27.92%									

Table 3: Overall Performance on HWFixBench across LLMs under Test

Table 4: Knowledge Cutoff Analysis across All LLMs

A (C1: ++: +	Madal	Knowledge		Task1			Task2			Task3		
Annation	Model	Cutoff Time	Pre Perf	Post Perf	Perf Diff	Pre Perf	Post Perf	Perf Diff	Pre Perf	Post Perf	Perf Diff	
		Ge	neral Purpos	e LLM or Cod	e LLM (not h	ardware don	iain expert)					
	Coder 6.7b base	Mar-23	2.38%	5.69%	3.31%	10.71%	19.53%	8.82%	9.23%	17.86%	8.63%	
	Coder 7b Instruct v1.5	Mar-23	5.06%	7.48%	2.42%	17.26%	23.77%	6.51%	8.93%	14.17%	5.25%	
	Coder 33b Instruct	Mar-23	5.06%	7.59%	2.53%	12.50%	17.97%	5.47%	14.29%	13.73%	-0.56%	
	R1 Distill Qwen 1.5B	Jul-24	2.19%	7.96%	5.78%	24.75%	19.91%	-4.84%	19.28%	7.96%	-11.32%	
DeepSeek	R1 Distill Qwen 7B	Jul-24	4.57%	11.50%	6.93%	30.42%	27.43%	-2.98%	26.14%	15.04%	-11.10%	
	R1 Distill Llama 8B	Jul-24	4.97%	16.37%	11.40%	38.47%	36.28%	-2.19%	25.15%	15.49%	-9.66%	
	R1 Distill Qwen 14B	Jul-24	4.47%	9.73%	5.26%	38.57%	39.38%	0.81%	27.14%	17.70%	-9.44%	
	R1 Distill Qwen 32B	Jul-24	5.77%	13.27%	7.51%	41.95%	42.92%	0.97%	29.22%	18.58%	-10.64%	
	R1 Distill Llama 70B	Jul-24	5.57%	11.50%	5.94%	43.84%	43.81%	-0.03%	27.24%	11.95%	-15.29%	
Google	Gemma2 27B	Jun-24	2.21%	7.90%	5.69%	11.30%	7.29%	-4.00%	11.41%	4.56%	-6.85%	
	Llama 2 70b hf	Jul-23	2.83%	7.12%	4.29%	8.74%	14.47%	5.73%	6.94%	11.74%	4.80%	
	Llama 3 8B Instruct	Mar-23	3.57%	8.82%	5.25%	25.60%	29.91%	4.32%	15.48%	18.08%	2.60%	
	Llama 3 70B Instruct	Dec-23	3.00%	7.97%	4.97%	34.22%	28.27%	-5.94%	19.93%	12.63%	-7.30%	
Meta	Llama 3.1 8B Instruct	Dec-23	4.06%	9.77%	5.72%	33.33%	29.47%	-3.86%	31.22%	13.08%	-18.13%	
	Llama 3.1 70B Instruct	Dec-23	3.53%	10.08%	6.55%	35.45%	35.19%	-0.26%	25.04%	15.94%	-9.10%	
	Llama 3.2 11B Vison	Dec-23	4.41%	9.77%	5.37%	34.92%	28.27%	-6.65%	29.45%	16.24%	-13.21%	
	Llama 3.3 70B Instruct	Dec-23	3.70%	10.23%	6.52%	42.68%	36.84%	-5.84%	26.10%	11.13%	-14.97%	
Microsoft	Phi3.5 MoE Instruct	Oct-23	2.11%	7.45%	5.35%	7.49%	14.04%	6.54%	10.30%	9.57%	-0.74%	
MICIOSOIT	Phi-4	Jun-24	6.87%	18.84%	11.98%	37.65%	33.74%	-3.91%	24.69%	15.64%	-9.05%	
	Qwen2.5 14B	Dec-23	3.70%	8.72%	5.02%	34.57%	30.38%	-4.19%	23.81%	13.23%	-10.58%	
Qwen	Qwen2.5 32B	Dec-23	4.06%	9.77%	5.72%	36.33%	33.68%	-2.65%	32.80%	27.52%	-5.29%	
	QwQ 32B Preview	Dec-23	4.59%	12.63%	8.05%	38.80%	41.05%	2.25%	37.74%	43.16%	5.42%	
	gpt 4o	Oct-23	7.49%	13.54%	6.05%	82.67%	83.23%	0.56%	18.50%	22.73%	4.23%	
	gpt 40 mini	Oct-23	7.03%	7.33%	0.30%	74.00%	76.15%	2.14%	17.33%	24.35%	7.02%	
OpenAI	01	Oct-23	3.75%	6.83%	3.09%	81.26%	77.14%	-4.12%	21.78%	26.09%	4.31%	
	o1 mini	Oct-23	4.22%	6.21%	2.00%	83.61%	83.35%	-0.25%	18.03%	25.22%	7.18%	
	o3 mini	Oct-23	3.98%	8.57%	4.59%	77.05%	74.91%	-2.14%	23.42%	27.70%	4.28%	
				Supervised	Finetuning N	1ethod						
CAS	CodeV DS [46]	7/17/2024	6.14%	13.23%	7.09%	12.37%	12.70%	0.33%	14.96%	8.47%	-6.49%	
PKU	OriGen [6]	10/15/2024	7.92%	5.26%	-2.65%	21.99%	16.84%	-5.15%	13.63%	9.47%	-4.16%	
HKUST	RTLCoder [20]	1/21/2024	3.90%	8.97%	5.07%	7.64%	11.68%	4.03%	14.82%	13.54%	-1.28%	
CUHK	VeriSeek [39]	7/19/2024	3.07%	11.17%	8.11%	13.12%	10.64%	-2.48%	14.66%	5.32%	-9.34%	

tokens). This evaluation assessed whether LLMs could generalize beyond their maximum training lengths while maintaining robust performance on extended hardware design inputs.

Most models exhibited performance degradation on all tasks when presented with sequences exceeding the estimated training threshold, highlighting a general limitation in handling long hardware tasks. The degradation was particularly pronounced in opensource and fine-tuned models. Only OpenAI's proprietary models demonstrated no degradation on Task 2, suggesting superior long-sequence processing capabilities or a more practical approach in hardware scenarios.

4.5 Prompt methods Analysis

We conducted prompt method experiments on open-source models in the 27B-32B range, FT LLMs, and base models. The results,

A (C1) - 41 - 11	M. 1.1		Task1			Task2		Task3			
Amilation	Short Perf Long Perf Diff			Short Perf	Long Perf	Perf Diff	Short Perf	Long Perf	Perf Diff		
General Purpose LLM or Code LL						are domain ex	pert)		_		
	Coder 33b Instruct	7.92%	5.40%	-2.52%	24.59%	4.60%	-19.99%	17.62%	8.40%	-9.22%	
	Coder 6.7b base	5.74%	3.40%	-2.34%	25.68%	4.60%	-21.08%	21.58%	6.60%	-14.98%	
	Coder 7b Instruct v1.5	6.97%	6.60%	-0.37%	31.69%	7.80%	-23.89%	16.39%	7.40%	-8.99%	
	R1 Distill Llama 70B	7.24%	5.80%	-1.44%	58.33%	22.60%	-35.73%	32.51%	12.60%	-19.91%	
DeepSeek	R1 Distill Llama 8B	8.61%	4.80%	-3.81%	52.46%	17.00%	-35.46%	31.42%	11.60%	-19.82%	
	R1 Distill Qwen 1.5B	3.55%	2.80%	-0.75%	34.02%	9.00%	-25.02%	22.68%	9.20%	-13.48%	
	R1 Distill Qwen 14B	6.28%	4.20%	-2.08%	51.50%	20.00%	-31.50%	35.25%	11.00%	-24.25%	
	R1 Distill Qwen 32B	8.74%	4.80%	-3.94%	57.65%	19.40%	-38.25%	36.20%	14.20%	-22.00%	
	R1 Distill Qwen 7B	7.51%	3.40%	-4.11%	43.17%	10.40%	-32.77%	33.61%	10.20%	-23.41%	
Google	Gemma2 27B	4.51%	2.60%	-1.91%	12.70%	6.60%	-6.10%	12.98%	4.60%	-8.38%	
Google	Gemma 7B	2.60%	2.00%	-0.60%	27.73%	16.80%	-10.93%	12.98%	5.60%	-7.38%	
	Llama 2 70b hf	6.69%	4.40%	-2.29%	18.03%	4.80%	-13.23%	13.93%	4.80%	-9.13%	
	Llama 3.1 70B Instruct	7.10%	7.00%	-0.10%	46.99%	18.20%	-28.79%	24.73%	13.40%	-11.33%	
	Llama 3.1 8B Instruct	7.51%	6.60%	-0.91%	42.35%	15.00%	-27.35%	23.91%	17.80%	-6.11%	
Meta	Llama 3.2 11B Vison	8.06%	6.20%	-1.86%	43.31%	13.80%	-29.51%	28.28%	13.60%	-14.68%	
	Llama 3.3 70B Instruct	6.56%	8.20%	1.64%	50.00%	24.20%	-25.80%	20.22%	14.80%	-5.42%	
	Llama 3 70B Instruct	5.19%	6.40%	1.21%	41.53%	15.60%	-25.93%	17.62%	13.60%	-4.02%	
	Llama 3 8B Instruct	8.47%	5.80%	-2.67%	36.89%	16.80%	-20.09%	18.85%	15.20%	-3.65%	
Migrosoft	Phi3.5 MoE Instruct	6.42%	4.40%	-2.02%	17.08%	4.00%	-13.08%	12.02%	6.60%	-5.42%	
MICIOSOIT	Phi-4	11.07%	8.60%	-2.47%	51.23%	15.20%	-36.03%	35.52%	16.80%	-18.72%	
	Qwen2.5 14B	6.69%	6.00%	-0.69%	45.36%	13.20%	-32.16%	23.09%	15.00%	-8.09%	
Qwen	Qwen2.5 32B	6.56%	8.00%	1.44%	47.68%	16.20%	-31.48%	20.36%	14.80%	-5.56%	
	QwQ 32B Preview	9.70%	7.80%	-1.90%	57.10%	15.00%	-42.10%	40.16%	15.00%	-25.16%	
	gpt-4o	12.43%	10.00%	-2.43%	83.20%	82.80%	-0.40%	24.45%	16.60%	-7.85%	
	gpt-4o-mini	7.65%	6.60%	-1.05%	74.45%	76.80%	2.35%	24.59%	18.00%	-6.59%	
OpenAI	01	6.83%	4.20%	-2.63%	76.09%	82.20%	6.11%	26.64%	21.60%	-5.04%	
	o1-mini	5.87%	5.00%	-0.87%	83.20%	83.80%	0.60%	24.59%	20.00%	-4.59%	
	o3-mini	8.20%	5.20%	-3.00%	72.95%	79.60%	6.65%	27.87%	23.80%	-4.07%	
S				upervised Fin	netuning Method						
CAS	CodeV DS [46]	7.65%	6.60%	-1.05%	18.17%	4.00%	-14.17%	20.08%	5.00%	-15.08%	
PKU	OriGen [6]	9.56%	5.00%	-4.56%	31.42%	7.20%	-24.22%	17.62%	7.00%	-10.62%	
HKUST	RTLCoder [20]	7.10%	5.20%	-1.90%	14.48%	2.40%	-12.08%	20.63%	4.80%	-15.83%	
CUHK	VeriSeek [39]	5.19%	3.00%	-2.19%	19.40%	3.00%	-16.40%	18.99%	4.80%	-14.19%	

Table 5: Design Length Analysis across All LLMs

presented in Table 6, should be interpreted with caution, as this experiment is not directly comparable to previous evaluations.

A key finding is that **the assumption made by prompt method studies, that their effectiveness is independent of the underlying LLM, does not hold**. Instead, each prompt method must be strictly tailored to specific LLMs, making them techniques for optimizing particular models rather than universal solutions. While prompting can enhance LLMs that are otherwise poorly suited for hardware tasks by reformatting input information, it cannot substitute for model advancements. For example, Gemma2 27B achieved a twofold performance increase but remained the weakest model. In contrast, QwQ 32B and DeepSeek R1 Distill Qwen 32B maintained relatively strong performance despite not being optimized for these prompt methods, demonstrating excellent model stability. Qwen2.5 32B showed a notable positive gain, making it a promising base model for prompt-based techniques. However, prompt methods alone cannot bridge model size and fundamental performance gaps.

Combining fine-tuned models with prompt methods did not lead to cumulative performance gains. Notably, OriGen, the bestperforming fine-tuned model, suffered a severe performance drop when used in conjunction with prompt methods. VeriSeek exhibited similar degradation. Meanwhile, RTLCoder and CodeV, two hardware design-specific models, demonstrated strong compatibility with prompt-based approaches.

5 Related Work

Unlike SWEBench[15], which focuses primarily on Python code, our work extends coverage to SystemVerilog implementations of typical chip functional modules. In practical hardware development, testing and debugging are typically handled by two separate roles: test engineers and develop engineers. Our benchmark reflects this division by structuring tasks into two categories: Task 1 for testing and Task 2 for repair. While mainstream LLM-based approaches often attempt to consolidate these roles or directly generate solutions in a single step, we evaluate this strategy through Task 3, testing the LLM's ability to handle both scenarios simultaneously. We show the SOTA performance of existing benchmarks

VerilogEval [18] is a pedagogical tool for digital logic design courses rather than an accurate measure of hardware design capability. Its most complex benchmarks involve simple finite-state machine applications, while its most straightforward cases cover empty designs or constant-zero outputs. Similarly, CreativeEval[8] and CorrectBench [28] are built on the same HDLBits foundation, focusing on basic Verilog-level exercises for educational purposes. This fundamentally differs from HWFixBench, which covers realworld chip functional module designs with practical applications and significantly higher complexity.

While RTLLM [22] shares similarities with our work by including simple module designs, HWFixBench is unique in drawing modules directly from real-world hardware repair workflows, making its tasks more representative of practical engineering challenges. FVEval [16], on the other hand, focuses on specific edge cases in FIFO and Pipeline scenarios, assessing LLM-generated SystemVerilog assertions using JasperGold. While these tasks are not inherently challenging, their low tolerance for error makes them unsuitable as general LLM benchmarks.

Model	Base Performance	Task	Affiliation	Prompt Style	Pass Rate	Benefit	Model	Pass Rate	Task	Affiliation	Prompt Style	Pass Rate	Benefit
	4.79%	Task1	Uni Siegen	CorrectBench [28]	4.40%	-8.10%		7.22%	Task1	Uni Siegen	CorrectBench [28]	7.58%	4.92%
DeepSeek			Uconn	AdvancedLLM [35]	19.52%	13.96%				Uconn	AdvancedLLM [35]	13.40%	7.94%
			Uni Siegen	CorrectBench [28]	18.26%	6.60%				Uni Siegen	CorrectBench [28]	14.03%	12.96%
	17 1207	Tooka	UNSW	[4]	17.92%	4.61%		12.420	Took 2	UNSW	[4]	13.79%	11.07%
Coder 6.7b	17.15%	1ask2	Stevens	PromptV [25]	18.16%	6.03%	CodeV DS [46]	12.42%	Task2	Stevens	PromptV [25]	14.38%	15.79%
base			Nvidia	RTLFixer [38]	18.18%	6.16%				Nvidia	RTLFixer [38]	15.72%	26.62%
			PKU	VerilogReader [23]	18.21%	6.34%				PKU	VerilogReader [23]	13.40%	7.94%
	15 50%	Tack3	Nvidia	RTLRewriter [44]	14.67%	-5.40%		13.06%	Tack3	Nvidia	RTLRewriter [44]	18.21%	30.45%
	15.50%	Tasks	CUHK	VerilogCoder [14]	18.85%	21.57%		15.70%	Tasks	CUHK	VerilogCoder [14]	18.04%	29.22%
	6.82%	Task1	Uni Siegen	CorrectBench [28]	5.72%	-16.05%		3.73%	Task1	Uni Siegen	CorrectBench [28]	5.69%	52.42%
			Uconn	AdvancedLLM [35]	21.06%	-4.25%				Uconn	AdvancedLLM [35]	29.65%	189.89%
DeenSeek			Uni Siegen	CorrectBench [28]	19.08%	-13.28%				Uni Siegen	CorrectBench [28]	28.12%	174.93%
Coder 7h	22.00%	Tack2	UNSW	[4]	20.15%	-8.40%		10.23%	Tack2	UNSW	[4]	25.08%	145.24%
Instruct	22.00%	1 ask2	Stevens	PromptV [25]	19.64%	-10.73%	Gemma2 27B	10.23%	Task2	Stevens	PromptV [25]	26.89%	162.97%
v1 5			Nvidia	RTLFixer [38]	20.15%	-8.40%				Nvidia	RTLFixer [38]	27.18%	165.80%
			PKU	VerilogReader [23]	17.00%	-22.72%				PKU	VerilogReader [23]	23.33%	128.09%
	12 74%	Task3	Nvidia	RTLRewriter [44]	13.81%	8.40%		9.58%	Task3	Nvidia	RTLRewriter [44]	19.38%	102.33%
	10.7 170	Tusks	CUHK	VerilogCoder [14]	19.93%	56.42%			Tusks	CUHK	VerilogCoder [14]	24.20%	152.64%
	7.71%	Task1	Uni Siegen	CorrectBench [28]	0.00%	-100.00%		8.93%	Task1	Uni Siegen	CorrectBench [28]	7.88%	-11.70%
	21.59%		Uconn	AdvancedLLM [35]	0.08%	-99.61%				Uconn	AdvancedLLM [35]	29.83%	-25.45%
			Uni Siegen	CorrectBench [28]	0.08%	-99.61%				Uni Siegen	CorrectBench [28]	30.80%	-23.04%
		Task2	UNSW	[4]	0.42%	-98.04%	OwO 32B	40.02%	Task2	UNSW	[4]	29.59%	-26.05%
OriGen [6]		1 HUSINE	Stevens	PromptV [25]	0.42%	-98.04%	Preview	10.0270	Tuste	Stevens	PromptV [25]	30.37%	-24.10%
			Nvidia	RTLFixer [38]	0.17%	-99.22%				Nvidia	RTLFixer [38]	31.08%	-22.34%
			PKU	VerilogReader [23]	0.42%	-98.05%				PKU	VerilogReader [23]	29.10%	-27.28%
	13 31%	Task3	Nvidia	RTLRewriter [44]	0.17%	-98.71%		20.05%	Task3	Nvidia	RTLRewriter [44]	18.50%	-38.24%
	15.51%	Links	CUHK	VerilogCoder [14]	0.34%	-97.46%		27.7570	Tusks	CUHK	VerilogCoder [14]	29.96%	0.04%
	6.33%	Task1	Uni Siegen	CorrectBench [28]	6.19%	-2.25%		7.14%	Task1	Uni Siegen	CorrectBench [28]	7.71%	7.95%
			Uconn	AdvancedLLM [35]	16.18%	68.96%				Uconn	AdvancedLLM [35]	40.83%	8.78%
			Uni Siegen	CorrectBench [28]	15.82%	65.22%				Uni Siegen	CorrectBench [28]	40.50%	8.24%
	9.58%	Task2	UNSW	[4]	13.58%	41.74%		34.90%	Task2	UNSW	[4]	40.42%	7.53%
RTLCoder [20]			Stevens	PromptV [25]	14.25%	48.78%	Qwen2.5 32B			Stevens	PromptV [25]	42.86%	14.52%
			Nvidia	RTLFixer [38]	15.15%	58.17%				Nvidia	RTLFixer [38]	40.50%	7.71%
			PKU	VerilogReader [23]	14.20%	48.29%				PKU	VerilogReader [23]	38.72%	3.94%
	14.20%	Task3	Nvidia	RTLRewriter [44]	19.30%	35.90%		18.10%	Task3	Nvidia	VerilogCoder [14]	39.45%	117.94%
			CUHK	VerilogCoder [14]	16.86%	18.68%				CUHK	RTLRewriter [44]	31.90%	76.23%
	4.30%	Task1	Uni Siegen	CorrectBench [28]	4.24%	-1.41%		7.14%	Task1	Uni Siegen	CorrectBench [28]	6.48%	-9.23%
			Uconn	AdvancedLLM [35]	9.37%	-26.47%				Uconn	AdvancedLLM [35]	33.15%	-21.32%
			Uni Siegen	CorrectBench [28]	11.54%	-9.48%				Uni Siegen	CorrectBench [28]	32.37%	-23.16%
	12.74%	Task2	UNSW	[4]	9.88%	-22.44%	DeepSeek	42.13%	Task2	UNSW	[4]	33.12%	-21.39%
VeriSeek [39]			Stevens	PromptV [25]	10.86%	-14.76%	R1 Distill			Stevens	PromptV [25]	32.83%	-22.06%
			Nvidia	RILFixer [38]	9.98%	-21.72%	Qwen 32B			Nvidia	RTLFixer [38]	32.28%	-23.38%
			PKU	VerilogReader [23]	9.58%	-24.86%				PKU	VerilogReader [23]	30.99%	-26.43%
	13.23%	3% Task3	Nvidia	KILKewriter [44]	11.84%	-10.49%	-	27.27%	Task3	Nvidia	KILKewriter [44]	20.76%	-23.89%
1	1	1	I CUHK	verligs.oder [14]	1 13.02%	-1.58%			I CUHK	verligs.oder [14]	1 30.57%	12.08%	

Table 6: Prompt methods Analysis based on various LLMs

Table 7: State-of-the-Art (SOTA) Performance Comparison on Benchmarks

					Н	WFixBench		
	Benchmark	VerilogEval	FVEval	RTLLM	LLM backend		Simulator	
					Open	Proprietary	backend	
	SOTA Perf	94.8\$	35%	99.99%	26.30%	39%	0	
	SOTA Method	[47]	FVEval	[25]	QwQ32 Preview	GPT-40	-	

In contrast, HWFixBench targets more challenging hardware designs but prioritizes providing a better starting point for users rather than enforcing strict simulation pass criteria. With 500 repair instances and more than 1,000 modifications, our benchmark substantially exceeds the combined scale of existing evaluation metrics. Moreover, the designs included in our dataset are of significantly higher practical value, with more intricate testing and repair tasks, making HWFixBench a suitable metric for assessing LLM capabilities in EDA tasks.

6 Conclusion

HWFixBench is a benchmark designed to evaluate the emerging trend of LLM-driven EDA automation. This work introduces a highvalue benchmark to assess existing and future LLM applications in hardware design and verification. Through comprehensive evaluations of general-purpose LLMs, fine-tuned models, and promptbased methods, we provide a clear performance landscape and identify challenges in adapting LLMs to hardware-related tasks.

Our findings highlight the significant gap between proprietary and open-source models, the task-dependent effectiveness of finetuning, and the model-specific nature of prompt methods. They emphasize that LLM-based automation in EDA requires tailored approaches rather than one-size-fits-all solutions. The benchmark is a foundation for future research, guiding the development of more capable LLMs for hardware applications and providing a standardized evaluation framework for measuring progress in this rapidly evolving domain.

Acknowledgments

Portions of this work were supported by the National Science Foundation (2340949 and 2419880).

References

- Manar Abdelatty, Jingxiao Ma, et al. 2024. MetRex: A Benchmark for Verilog Code Metric Reasoning Using LLMs. *CoRR* abs/2411.03471 (2024), 7. https: //doi.org/10.48550/ARXIV.2411.03471 arXiv:2411.03471
- [2] Marah I Abdin, Jyoti Aneja, et al. 2024. Phi-4 Technical Report. CoRR abs/2412.08905 (2024), 36. https://doi.org/10.48550/ARXIV.2412.08905 arXiv:2412.08905
- [3] Marah I Abdin, Sam Ade Jacobs, et al. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *CoRR* abs/2404.14219 (2024), 24. https://doi.org/10.48550/ARXIV.2404.14219 arXiv:2404.14219
- [4] Jason Blocklove, Siddharth Garg, et al. 2024. Evaluating LLMs for Hardware Design and Test. CoRR abs/2405.02326 (2024), 6. https://doi.org/10.48550/ARXIV. 2405.02326 arXiv:2405.02326
- [5] Chi-Min Chan, Weize Chen, et al. 2024. ChatEval: Towards Better LLM-based Evaluators through Multi-Agent Debate. In *The Twelfth International Conference* on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, Vienna, Austria, 15. https://openreview.net/forum?id=FQepisCUWu
- [6] Fan Cui, Chenyang Yin, et al. 2024. OriGen:Enhancing RTL Code Generation with Code-to-Code Augmentation and Self-Reflection. CoRR abs/2407.16237 (2024), 9. https://doi.org/10.48550/arXiv.2407.16237
- [7] DeepSeek-AI, Daya Guo, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *CoRR* abs/2501.12948 (2025), 22. https://doi.org/10.48550/ARXIV.2501.12948 arXiv:2501.12948

- [8] Matthew DeLorenzo, Vasudev Gohil, et al. 2024. CreativEval: Evaluating Creativity of LLM-Based Hardware Code Generation. CoRR abs/2404.08806 (2024), 5. https://doi.org/10.48550/ARXIV.2404.08806 arXiv:2404.08806
- [9] Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. CoRR abs/2407.21783 (2024), 92. https://doi.org/10.48550/ARXIV.2407.21783 arXiv:2407.21783
- [10] Weimin Fu, Shijie Li, et al. 2025. A Generalize Hardware Debugging Approach for Large Language Models Semi-Synthetic, Datasets. *IEEE Transactions on Circuits* and Systems I: Regular Papers 72, 2 (2025), 623–636. https://doi.org/10.1109/TCSI. 2024.3487486
- [11] Weimin Fu, Kaichen Yang, et al. 2023. LLM4SecHW: Leveraging Domain-Specific Large Language Model for Hardware Debugging. In Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2023, Tianjin, China, December 13-15, 2023. IEEE, Tianjin, China, 1–6. https://doi.org/10.1109/ASIANHOST59942.2023. 10409307
- [12] Mingzhe Gao, Jieru Zhao, et al. 2024. AutoVCoder: A Systematic Framework for Automated Verilog Code Generation using LLMs. In 42nd IEEE International Conference on Computer Design, ICCD 2024, Milan, Italy, November 18-20, 2024. IEEE, Milan, Italy, 162–169. https://doi.org/10.1109/ICCD63220.2024.00033
- [13] Daya Guo, Qihao Zhu, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence. *CoRR* abs/2401.14196 (2024), 23. https://doi.org/10.48550/ARXIV.2401.14196 arXiv:2401.14196
- [14] Chia-Tung Ho, Haoxing Ren, et al. 2024. VerilogCoder: Autonomous Verilog Coding Agents with Graph-based Planning and Abstract Syntax Tree (AST)based Waveform Tracing Tool. CoRR abs/2408.08927 (2024), 30. https://doi.org/ 10.48550/ARXIV.2408.08927 arXiv:2408.08927
- [15] Carlos E. Jimenez, John Yang, et al. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, Vienna, Austria, 51. https://openreview.net/forum?id=VTF8yNQM66
- [16] Minwoo Kang, Mingjie Liu, et al. 2024. FVEval: Understanding Language Model Capabilities in Formal Verification of Digital Hardware. CoRR abs/2410.23299 (2024), 42. https://doi.org/10.48550/ARXIV.2410.23299 arXiv:2410.23299
- [17] Raymond Li, Loubna Ben Allal, et al. 2023. StarCoder: may the source be with you! Trans. Mach. Learn. Res. 2023 (2023), 43. https://openreview.net/forum?id= KoFOg41haE
- [18] Mingjie Liu, Nathaniel Ross Pinckney, et al. 2023. Invited Paper: VerilogEval: Evaluating Large Language Models for Verilog Code Generation. In IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023. IEEE, San Francisco, USA, 1–8. https://doi. org/10.1109/ICCAD57390.2023.10323812
- [19] Mingjie Liu, Yun-Da Tsai, et al. 2024. CraftRTL: High-quality Synthetic Data Generation for Verilog Code Models with Correct-by-Construction Non-Textual Representations and Targeted Code Repair. CoRR abs/2409.12993 (2024), 46. https://doi.org/10.48550/ARXIV.2409.12993 arXiv:2409.12993
- [20] Shang Liu, Wenji Fang, et al. 2024. RTLCoder: Fully Open-Source and Efficient LLM-Assisted RTL Code Generation Technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* Early Access (2024), 1–1. https: //doi.org/10.1109/TCAD.2024.3483089
- [21] Yang Liu, Dan Iter, et al. 2023. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 2511–2522. https://doi.org/10.18653/V1/2023.EMNLP-MAIN.153
- [22] Yao Lu, Shang Liu, et al. 2024. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. In Proceedings of the 29th Asia and South Pacific Design Automation Conference, ASPDAC 2024, Incheon, Korea, January 22-25, 2024. IEEE, Incheon, Korea, 722–727. https://doi.org/10.1109/ASP-DAC58780.2024.10473904
- [23] Ruiyang Ma, Yuxin Yang, et al. 2024. VerilogReader: LLM-Aided Hardware Test Generation. CoRR abs/2406.04373 (2024), 5. https://doi.org/10.48550/ARXIV.2406. 04373 arXiv:2406.04373
- [24] Thomas Mesnard, Cassidy Hardin, et al. 2024. Gemma: Open Models Based on Gemini Research and Technology. CoRR abs/2403.08295 (2024), 17. https: //doi.org/10.48550/ARXIV.2403.08295 arXiv:2403.08295
- [25] Zhendong Mi, Renming Zheng, et al. 2024. PromptV: Leveraging LLM-powered Multi-Agent Prompting for High-quality Verilog Generation. *CoRR* abs/2412.11014 (2024), 5. https://doi.org/10.48550/ARXIV.2412.11014 arXiv:2412.11014
- [26] Bardia Nadimi and Hao Zheng. 2024. A Multi-Expert Large Language Model Architecture for Verilog Code Generation. *CoRR* abs/2404.08029 (2024), 5. https: //doi.org/10.48550/ARXIV.2404.08029 arXiv:2404.08029
- [27] Zehua Pei, Hui-Ling Zhen, et al. 2024. BetterV: Controlled Verilog Generation with Discriminative Guidance. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net, Vienna, Austria, 9. https://openreview.net/forum?id=iKnW7r7de1
- Austria, 9. https://openreview.net/forum?id=jKnW7r7de1
 [28] Ruidi Qiu, Grace Li Zhang, et al. 2024. CorrectBench: Automatic Testbench Generation with Functional Self-Correction using LLMs for HDL Design.

CoRR abs/2411.08510 (2024), 7. https://doi.org/10.48550/ARXIV.2411.08510 arXiv:2411.08510

- [29] Morgane Rivière, Shreya Pathak, et al. 2024. Gemma 2: Improving Open Language Models at a Practical Size. CoRR abs/2408.00118 (2024), 21. https://doi.org/10. 48550/ARXIV.2408.00118 arXiv:2408.00118
- [30] Pat Rondon, Renyao Wei, et al. 2025. Evaluating Agent-based Program Repair at Google. arXiv:cs.SE/2501.07531 https://arxiv.org/abs/2501.07531
- [31] Minju Seo, Jinheon Baek, et al. 2024. Rethinking Code Refinement: Learning to Judge Code Efficiency. In Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 11045–11056. https://aclanthology.org/2024.findings-emnlp. 645
- [32] Diomidis Spinellis. 2012. Git. IEEE Softw. 29, 3 (2012), 100–101. https://doi.org/ 10.1109/MS.2012.61
- [33] Shailja Thakur, Baleegh Ahmad, et al. 2023. Benchmarking Large Language Models for Automated Verilog RTL Code Generation. In Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023. IEEE, Antwerp, Belgium, 1–6. https://doi.org/10.23919/DATE56975.2023. 10137086
- [34] Shailja Thakur, Baleegh Ahmad, et al. 2024. VeriGen: A Large Language Model for Verilog Code Generation. ACM Trans. Design Autom. Electr. Syst. 29, 3 (2024), 46:1–46:31. https://doi.org/10.1145/3643681
- [35] Kiran Thorat, Jiahui Zhao, et al. 2023. Advanced Large Language Model (LLM)-Driven Verilog Development: Enhancing Power, Performance, and Area Optimization in Code Synthesis. CoRR abs/2312.01022 (2023), 8. https://doi.org/10. 48550/ARXIV.2312.01022 arXiv:2312.01022
- [36] Weixi Tong and Tianyi Zhang. 2024. CodeJudge: Evaluating Code Generation with Large Language Models. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, FL, USA, 20032–20051. https://aclanthology.org/2024.emnlp-main.1118
- [37] Hugo Touvron, Louis Martin, et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. CoRR abs/2307.09288 (2023), 77. https://doi.org/10.48550/ ARXIV.2307.09288 arXiv:2307.09288
- [38] Yun-Da Tsai, Mingjie Liu, et al. 2023. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models. CoRR abs/2311.16543 (2023), 7. https://doi.org/10.48550/ARXIV.2311.16543 arXiv:2311.16543
- [39] Ning Wang, Bingkun Yao, et al. 2024. Large Language Model for Verilog Generation with Golden Code Feedback. CoRR abs/2407.18271 (2024), 9. https: //doi.org/10.48550/ARXIV.2407.18271 arXiv:2407.18271
- [40] Ruiqi Wang, Jiyu Guo, et al. 2025. Can LLMs Replace Human Evaluators? An Empirical Study of LLM-as-a-Judge in Software Engineering. arXiv:cs.SE/2502.06193 https://arxiv.org/abs/2502.06193
- [41] Ke Xu, Jialin Sun, et al. 2024. MEIC: Re-thinking RTL Debug Automation using LLMs. CoRR abs/2405.06840 (2024), 11. https://doi.org/10.48550/ARXIV.2405. 06840 arXiv:2405.06840
- [42] An Yang, Baosong Yang, et al. 2024. Qwen2.5 Technical Report. CoRR abs/2412.15115 (2024), 26. https://doi.org/10.48550/ARXIV.2412.15115 arXiv:2412.15115
- [43] John Yang, Carlos E. Jimenez, et al. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, December 10 - 15, 2024, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, et al. (Eds.). Curran Associates, Inc., Vancouver,BC, Canada, 125. http://papers.nips.cc/paper_files/paper/2024/ hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html
- [44] Xufeng Yao, Yiwen Wang, et al. 2024. RTLRewriter: Methodologies for Large Models aided RTL Code Optimization. *CoRR* abs/2409.11414 (2024), 10. https: //doi.org/10.48550/ARXIV.2409.11414 arXiv:2409.11414
- [45] Yongan Zhang, Zhongzhi Yu, et al. 2024. MG-Verilog: Multi-grained Dataset Towards Enhanced LLM-assisted Verilog Generation. *CoRR* abs/2407.01910 (2024), 5. https://doi.org/10.48550/ARXIV.2407.01910 arXiv:2407.01910
- [46] Yang Zhao, Di Huang, et al. 2024. CodeV: Empowering LLMs for Verilog Generation through Multi-Level Summarization. CoRR abs/2407.10424 (2024), 16. https://doi.org/10.48550/ARXIV.2407.10424 arXiv:2407.10424
- [47] Yujie Zhao, Hejia Zhang, et al. 2024. MAGE: A Multi-Agent Engine for Automated RTL Code Generation. CoRR abs/2412.07822 (2024), 7. https://doi.org/10.48550/ ARXIV.2412.07822 arXiv:2412.07822
- [48] Lianmin Zheng, Wei-Lin Chiang, et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, December 10 - 16, 2023, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, et al. (Eds.). Curran Associates, Inc., New Orleans, LA, USA, 29. http://papers.nips.cc/paper_files/paper/2023/hash/ 91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html