2026 Design, Automation & Test in Europe Conference (DATE 2026)

LiveVerilogEval: Contamination Free and Automatically Scalable Benchmark for Verilog Code Generation

Charles Young[§], Hao Yu[‡*], Dezhi Ran[§], Qingchen Zhai[†],
Tianqi Qiu[‖], Frank Qu[¶], Bangyan Wang[‡], Yuan Xie[‡], Tao Xie[§*]
[§]School of Computer Science, Peking University, Beijing, China
[‡]Hong Kong University of Science and Technology, Hong Kong, China
[†]Institute of Automation, Chinese Academy of Sciences, Beijing, China
[¶]Independent Research
[‖]Beijing Jiaotong University, Beijing, China

***Abstract*—Verilog code generation has emerged as a critical application for Large Language Models (LLMs) in Electronic Design Automation (EDA). However, existing benchmarks suffer from data contamination issues where training datasets overlap with evaluation problems, leading to artificially inflated performance. Additionally, periodically creating new benchmark problems is often too cost-prohibitive to be maintained by humans. In this paper, we propose LiveVerilogEval, a dynamic framework that automatically generates novel evaluation problems from existing RTL designs. LiveVerilogEval addresses both challenges by automatically generating mutated variants of valid Verilog designs while maintaining semantic correctness. Our experimental results demonstrate significant performance degradation across state-of-the-art LLMs when evaluated on LiveVerilogEval-enhanced benchmarks compared to traditional static benchmarks, revealing that LLM-based Verilog generation remains challenging and confirming the effectiveness of our contamination-free evaluation approach.**

I. Introduction

The integration of Large Language Models (LLMs) into Electronic Design Automation (EDA) workflows has opened new possibilities for automated hardware design. Recent advances in LLM-based Verilog code generation, exemplified by domain-specific LLMs such as Verigen [1] and RTLCoder [2], demonstrate significant potential for accelerating RTL development processes. As the study of improving generative AI quality on hardware tasks gains traction in academic research, establishing reliable evaluation methodologies has become critical for assessing model capabilities and guiding future developments.

Current approaches for evaluating LLM-based RTL generation rely on static benchmarks that follow a standardized format: natural language problem descriptions paired with reference Verilog/SystemVerilog implementations, with corresponding testbenches for functional validation. Examples include popular benchmarks such as VerilogEval [3] and RTLLM [4], which have enabled the systematic comparison of different models and approaches in this domain. However, static benchmarks are vulnerable to *data contamination*, an issue where evaluation problems are leaked into LLM training datasets, leading to artificially inflated performance metrics and compromised benchmark integrity.

Data contamination is especially harmful to RTL evaluation for two reasons. First, existing RTL benchmarks contain few problems and small designs, making complete memorization feasible. Second, the deterministic nature of hardware designs causes RTL implementations to converge to similar structural patterns, unlike software domains that allow diverse functionally equivalent solutions. A recent analysis by Wang et al. [5] confirms this threat, revealing that major RTL benchmarks have been extensively leaked into commercial LLM training data, reducing the validity of their performance evaluations.

Addressing data contamination in RTL evaluation faces a unique challenge compared to its software counterpart: a dearth of available problem sets. While software benchmarks such as LiveCodeBench [6] and LiveBench [7] can continuously obtain new problems from frequently-hosted coding competitions, RTL problems are significantly harder to find, with very few sources available for gathering complete question/answer sets. Addressing this problem by manually writing an RTL benchmark is not scalable due to its prohibitive labor requirement, as attempting to do so would require a large number of experts to dedicate a significant amount of time crafting problem descriptions, functionally correct reference implementations, and comprehensive testbenches. To avoid data contamination, these benchmarks would also have to be periodically refreshed with new problems, making this approach too expensive in practice. To reduce cost, one way is to use LLMs to automate this process. However, all current automated approaches still require a human reviewer to manually confirm whether the answer and testbenches generated by an LLM truly align with their corresponding question, hence limiting the cost-effectiveness of these approaches.

To address these challenges, this paper introduces LiveVerilogEval, an automated framework that systematically gen-

*Corresponding authors.

978-3-9826741-1-7/DATE26/© 2026 EDAA

erates contamination-free RTL evaluation problems through the controlled mutation of existing verified designs. Instead of beginning the task of benchmark creation with a set of problems to be solved, our framework creates a set of unique RTL designs and builds questions around them. To verify that the questions truly describe their corresponding designs, we leverage the differential gap between the likelihood of an LLM to generate a functionally equivalent design given the correct question and the likelihood of the same LLM generating an equivalent design given an incorrect question. In doing so, we ensure that the generated specifications accurately describe the provided designs.

Our key contributions are as follows:

- **Automated benchmark generation:** We develop a systematic approach to automatically generate RTL problems without manual intervention, using SAT solvers for validation and LLM oracles for description generation.
- **Model Evaluation:** We evaluate multiple state-of-the-art LLMs on our benchmark, demonstrating significant performance degradation across multiple state-of-the-art LLMs when evaluated on LiveVerilogEval-enhanced benchmarks compared to traditional static benchmarks
- **Contamination Evaluation:** We evaluate LiveVerilogEval by measuring the mean similarity between the responses output by LLMs and the ground truth of the benchmarks studied. We evaluate similarity using the Jaccard similarity metric, the Levenshtein distance metric, and the Cosine similarity metric, demonstrating the drawbacks of using contamination metrics that focus on gauging LLM output distribution as a measure of contamination.

To facilitate future research with LiveVerilogEval, our experimental results, setup details, and source code are publicly available [8].

The remainder of this paper is organized as follows. Section II introduces the background and related work of Verilog code generation models. Section III details the data collection and model fine-tuning. Sections IV and V describe and answer the research questions. Section VI discusses the limitations and future work. Section VIII concludes this paper.

II. Background and Related Work

A. Background

The rapid advancements in LLMs have inspired a growing interest in applying them to hardware design, particularly for automating the generation of Register-Transfer Level (RTL) code. Models such as Verigen [1] and RTLCoder [2] have demonstrated promising capabilities in generating functional hardware descriptions from natural language specifications, leading the way in this emerging field.

The application of LLMs to RTL generation offers substantial potential benefits, including accelerated design cycles, reduced human error, and a lower barrier to entry into the field for non-experts. Furthermore, the ability to generate RTL code from high-level natural language descriptions could revolutionize the hardware design workflow, enabling rapid prototyping and iterative design exploration.

B. Yosys

Yosys [9] is a suite of widely-used open-source HDL synthesis tools. One such tool is sat, which is used to create and solve SAT problems using a provided HDL circuit model. For simplicity, we refer to this tool as Yosys in the remainder of this paper. Yosys utilizes the miniSAT library [10] to perform conflict-driven clause learning SAT resolution, which we use in our framework to determine whether two circuits are functionally equivalent. We use the tools Yosys provides to build a miter circuit for comparing designs under test (DUT) against our reference designs. During runtime, Yosys will either discover a counterexample that changes the miter circuit's signal to 1 or fail to do so, proving that the two modules are functionally equivalent.

C. Metrics for Data Contamination

Although Min-K is a more well-studied metric for measuring data contamination [11], we are unable to apply this metric for our experiment as the commercial models that we use do not expose the underlying token output distributions of already-generated samples. We instead choose to measure potential data contamination between benchmark ground-truth RTL designs and LLM-generated outputs through their textual similarity using three widely-adopted string-based metrics: Jaccard Similarity, Cosine Similarity, and Levenshtein Distance. These metrics capture different notions of overlap, ranging from set-level token reuse to character-level edit proximity. Although these metrics do not directly determine whether or not a data sample can be found within a model's training data, they are able to suggest the extent to which a corpus of samples has been leaked.

1) Jaccard Similarity: Jaccard Similarity measures the overlap between two sets of tokens. Given a generated code sample G and a reference implementation R, both tokenized into sets of lexical tokens, the Jaccard Similarity is defined as follows:

$$J(G, R) = \frac{|G \cap R|}{|G \cup R|}$$

This metric captures the proportion of shared tokens relative to the total unique tokens present. However, Jaccard similarity is insensitive to token frequency and ordering, making this metric robust to minor reformatting but blind to structural similarity.

2) Cosine Similarity: Cosine Similarity measures the angular similarity between two token-frequency vectors. After tokenizing the generated and reference code and embedding them into bag-of-words vectors $\mathbf{v}_G$ and $\mathbf{v}_R$, the cosine similarity is computed as follows:

$$\cos(\theta) = \frac{\mathbf{v}_G \cdot \mathbf{v}_R}{\|\mathbf{v}_G\| \|\mathbf{v}_R\|}$$

Unlike Jaccard similarity, cosine similarity accounts for token frequency, making this metric more sensitive to repeated structural patterns.

3) Levenshtein Distance: Levenshtein Distance measures the minimum number of character-level insertions, deletions, and substitutions required to transform one string into another. We normalize this distance by the length of the reference implementation to obtain a scale-invariant similarity score:

$$L_{\text{norm}}(G, R) = 1 - \frac{d_{\text{lev}}(G, R)}{\max(|G|, |R|)}$$

D. Related Work

Multiple early benchmarks aimed at evaluating RTL generation using LLMs laid the groundwork for evaluating hardware code generation. One of the earliest of these benchmarks is introduced by Thakur et al. [12], and consists of 17 problems sourced from the HDLBits website [13]. These problems range in complexity and provide multiple levels of problem descriptions.

Over time, this benchmark evolves into VerilogEval, with its two versions: VerilogEval-Human and VerilogEval-Machine [3]. VerilogEval expands the range of problems to 156, incorporating more complex designs, such as finite state machines (FSMs), Karnaugh-map-based problems, and sequential waveform-based tasks. These benchmarks have become popular for comparing LLM performance in hardware design. Still, their age and the availability of their problem sets on platforms such as GitHub make them vulnerable to contamination, as many problems may have been incorporated into training datasets of LLMs over time. VerilogEval is further enhanced in VerilogEval-V2, which introduced more challenging problems and categorized model failures into different groups. VerilogEval also supported In-context learning [14], improving the model's ability to adapt during evaluation. However, as an open-source and widely-used benchmark, VerilogEval also remains visible to web-scraping algorithms that consolidate data to be used in LLM training at scale. Because of this, VerilogEval is highly susceptible to data contamination, a concern that is heightened by its long-term availability.

Another benchmark that has been widely used in LLM comparisons is RTLLM, introduced by Lu et al. [4]. This benchmark, initially consisting of 30 questions, was designed to evaluate power and timing metrics of generated RTL designs using a more conversational tone in the problem descriptions. RTLLM has since been expanded to include 20 additional problems in RTLLM-V2. However, like VerilogEval, RTLLM faces the same contamination issues due to its static, open-source nature.

Beyond these mainstream benchmarks, multiple new approaches to RTL evaluation have been proposed. RealBench, introduced by Jin et al., uses significantly larger designs based on real-world IPs, presenting highly complex problems for LLMs to solve [15]. However, despite the difficulty of these problems, current models have failed to solve any of them, making comparisons between models difficult. Similarly, Chang et al. [16] introduced a benchmark focused on multi-modal models, comparing the performance of models that use both images and text with those that rely solely on text inputs.

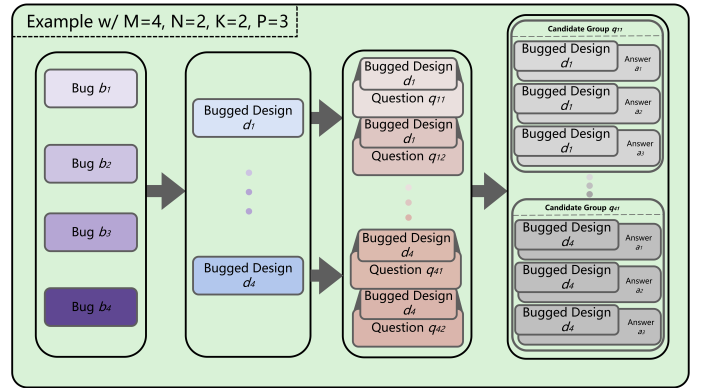

Fig. 1: An example of how LiveVerilogEval generates questions given an initial design. Four bugs are generated, and two are used to generate bugged designs. Each design is paired with a set of two generated questions, and the validity of each question is determined by a generated set of 3 answer designs.

Despite the introduction of multiple new evaluation approaches, benchmarks that specifically address data contamination remain rare, especially in hardware design tasks. Notable exceptions include LiveBench and LiveCodeBench, which reduce contamination by periodically updating their problem sets. In a similar vein, we introduce LiveVerilogEval, a contamination-free benchmark and generation framework that autonomously generates RTL problem sets, mitigating the risk of contamination even if the original designs have leaked into LLM training data.

III. LiveVerilogEval Benchmark

A. Data Collection

LiveVerilogEval is flexible under limited data and does not require testbenches or specifications to generate valid problems; however, for fair comparison, we initialize it with datasets that include human-written questions, descriptions, and testbenches. Using 50 RTLLM-V2 and 156 VerilogEval-V2 problems, LiveVerilog generates and prunes instances to form LiveVerilogEval-V1, yielding 73 questions derived from 23 designs (15 from VerilogEval and 8 from RTLLM).

B. Validity of the Formal SAT Solver as an Oracle

Human-designed testbenches are costly, rare, and prone to mistakes. We instead determine the correctness of an answer generated by an LLM by using a formal SAT solver (Yosys) to perform an equivalence check between the answer and a reference design. We run the equivalence check for a period of 60 seconds. If the formal checker fails to prove that the two designs are not equivalent, then we mark the design as correct. To confirm the validity of our verification approach, we conduct a preliminary experiment using a set of answers generated by Deepseek-V3.1 for the full VerilogEval-V2 benchmark. We then evaluate the generated answers using both the formal equivalence and the testbenches provided by VerilogEval that are compiled and simulated using IcarusVerilog [17]. The

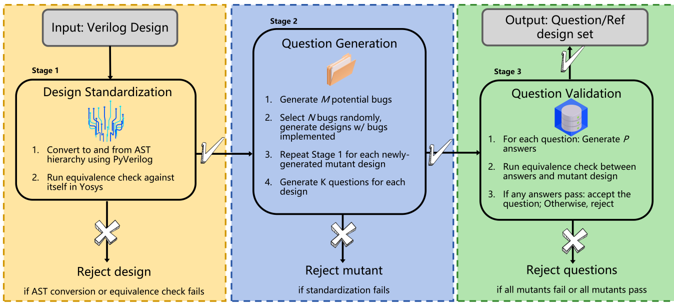

Fig. 2: The overview of the LiveVerilogEval framework, showing how questions are generated, verified, and pruned

results of the preliminary experimentation demonstrate that Yosys and the testbenches provided by VerilogEval-V2 agreed on 93.6% of evaluations. A manual inspection of the disagreements between the two approaches, where the answer fails to pass the testbench while passing the equivalency check, finds that most discrepancies can be attributed to initialization handling differences in Yosys, as Yosys does not verify that necessary signals have been initialized. On the other hand, disagreements where the answer fails the equivalence check but passes the testbench are often due to either a compilation failure by IcarusVerilog or an edge case that is not caught by the VerilogEval testbench.

C. Question Generation

Since we use LiveVerilogEval for producing LLM evaluation benchmarks instead of creating designs for commercial use, the designs in our benchmark are not necessarily practical. While such designs have little value in real-world digital systems, they still serve as valuable test cases for evaluating LLM performance. To create a challenging benchmark that does not suffer from test set contamination, the generated questions from LiveVerilogEval do not incorporate designs that are used for practical purposes. Instead, our framework proposes bugs, also denoted as *mutations*, that can potentially be applied to an initial design. These proposed mutations are then used to generate candidate designs as outlined in Figure 1.

D. Generation Pipeline

As shown in Figure 2, LiveVerilogEval is composed of three stages: design standardization **(a)**, question generation **(b)**, and question validation **(c)**.

(a) **Design Standardization:** An initial design is converted into an Abstract Syntax Tree (AST) using the PyVerilog library [18] and the resulting AST is converted back into Verilog. This conversion is done to ensure that all initial designs are syntactically correct and easily identifiable in later stages. Additionally, the conversion acts as a linter, standardizing the design formatting of all modules and designs, and filtering out duplicate designs. A design converted from the AST is then duplicated, and a formal SAT solver (Yosys) performs an equivalence check between the two known-to-be-equivalent designs. By performing an equivalency check on two designs that we know are identical, we can both confirm that the design does not incorporate any features that the formal checker does not support and prevent false negatives that stem from the limitations of the formal checker being used.

(b) **Question Generation:** First, the standardized design is used as input to an LLM, which generates a set of M potential mutations. These mutations represent plausible design variations, such as logical errors, timing discrepancies, or structural changes. The mutation generation step is crucial for introducing realistic variations into the design; These variations are later described in the questions.
Next, we select a subset of N mutations ($N \leq M$) from the generated pool. For each of the N mutations, we combine the mutation with the standardized design and create a prompt, which we then input into the LLM to generate a design variant based on the mutation. This step is repeated for each of the N selected mutations.
The rationale for selecting only a subset of mutations is to ensure diversity in the pool, including less commonly proposed mutations and avoiding redundant variants.
After the N design variants are generated, we standardize each one using the same approach as in Stage (a) to ensure consistency and remove any invalid syntax or unsupported features. Then, we prompt the LLM to generate K candidate question prompts for each of the N standardized mutants. Generating multiple prompts increases the likelihood of obtaining valid questions for each mutant, accounting for the nondeterministic nature of LLMs.

(c) **Question Validation:** For each of the K generated questions, we prompt the LLM to produce P candidate designs (where $P > 1$). We provide the LLM with the question as well as contextual information, such as the original reference design and the applied mutation.
Each of the P candidate answers is then checked for equivalence with the mutant design using a formal SAT solver. If the formal solver detects that an answer is not equivalent to the mutant design within 60 seconds, the answer is rejected. This process is repeated for all K questions.
If, for any of the K questions, all P candidate answers are determined to be equivalent to the mutant design, the entire question is rejected. Doing so ensures that only questions that generate truly distinct answers are kept for further use.
This validation process is repeated for each mutant design, ensuring that only valid and diverse questions are retained in the final set.

E. Generation setup

When generating our initial benchmark, we set $M = 10$, $N = 4$, $K = 3$, and $P = 10$ as our generation parameters. We primarily use DeepSeek-V3.1 non-thinking mode for any step that prompts an LLM, except the initial mutation generation, where we use GPT-4o-mini, and the design-variant generation,

TABLE I: Data contamination of benchmarks on nine models measured by mean Levenshtein distance, Jaccard similarity, and cosine similarity (BoW) for each model/benchmark

Benchmark	Model	Metrics		
		Lev. (%)	Jac. (%)	Cos. (%)
VerilogEval-V2	Deepseek-V3.1	55.10	57.86	71.14
	Claude-3.5-Haiku	50.21	52.12	67.36
	Claude-3.7-Sonnet	50.64	50.34	66.90
	Claude-4-Sonnet	51.91	52.79	73.77
	GPT-3.5-Turbo	50.22	51.67	67.58
	GPT-4	50.71	52.78	69.32
	GPT-4o-mini	53.70	54.30	65.56
	GPT-4o	54.25	51.86	69.40
	GPT-5	52.62	52.89	72.77
RTLLM-V2	Deepseek-V3.1	48.69	40.96	69.42
	Claude-3.5-Haiku	44.90	37.48	67.04
	Claude-3.7-Sonnet	46.04	38.58	67.87
	Claude-4-Sonnet	47.09	40.14	69.54
	GPT-3.5-Turbo	45.67	38.55	66.44
	GPT-4	46.19	39.39	64.26
	GPT-4o-mini	46.45	39.75	67.51
	GPT-4o	45.74	39.65	67.76
	GPT-5	45.95	39.92	66.15
LiveVerilogEval	Deepseek-V3.1	39.08	30.18	49.87
	Claude-3.5-Haiku	36.38	28.99	46.53
	Claude-3.7-Sonnet	37.07	29.32	48.51
	Claude-4-Sonnet	37.43	30.20	49.20
	GPT-3.5-Turbo	35.75	27.82	44.84
	GPT-4	36.46	28.55	47.31
	GPT-4o-mini	36.67	29.42	47.66
	GPT-4o	36.96	28.94	47.30
	GPT-5	36.83	29.36	51.65

where we use GPT-5. For the design-variant generation, we use GPT-5 to generate new variants only in cases where no design yields a valid question for DeepSeek.

IV. Evaluation

A. Benchmarks

In addition to LiveVerilogEval-V1.0, we include the data contamination and performance results of VerilogEval-V2 and RTLLM-V2, the benchmarks whose problems we use as seed data for generating LiveVerilogEval-V1.0.

B. Models

We comprehensively evaluate the performance of the following state-of-the-art LLMs on our benchmarks: DeepSeek-V3.1 non-thinking mode (deepseek), GPT-5 (OpenAI), GPT-4o-mini, GPT-4, GPT-3.5-Turbo, Claude 3.5 Haiku (claude-3-5-haiku-20241022), Claude 3.7 Sonnet (claude-3-7-sonnet-20250219), and Claude 4 Sonnet (claude-sonnet-4-20250514).

C. Evaluation Metrics

We adopt the same Pass@k evaluation metric as VerilogEval [3] with $k = 1, 5$ for evaluating model effectiveness on benchmarks. To evaluate the extent to which different LLMs suffer from data contamination, we compare the ground truths of our benchmark and the baseline benchmarks with LLM-generated solutions using Levenshtein distance, Jaccard similarity, and cosine similarity metrics.

V. Experimental Results

A. RQ1: To what extent do existing Verilog benchmarks exhibit data contamination compared to LiveVerilogEval?

We use three similarity metrics to measure the data contamination of VerilogEval-V2, RTLLM-V2, and LiveVerilogEval and show the measured levels and results of data contamination in Table I.

We note that the mean similarity scores from all nine models on both VerilogEval-V2 and RTLLM-V2 are significantly higher than the same models on LiveVerilogEval. Specifically, compared to VerilogEval-V2, all similarity metrics in LiveVerilogEval are at least 12% lower, and in some cases over 20% lower. Similarly, when compared to RTLLM-V2, all similarity metrics on LiveVerilogEval are at least 8% lower, with some reductions exceeding 20%.

Since the questions and reference modules found in LiveVerilogEval have never been open-sourced and cannot be found online, we consider the score by LiveVerilogEval the baseline for indicating the absence of data contamination due to the high likelihood that the data from the benchmark has not been leaked. Hence, the higher scores obtained by the other benchmarks suggest that more portions of these benchmark problem sets have likely leaked into LLM training datasets.

B. RQ2: How effective is LiveVerilogEval in helping evaluate the real performance of LLMs in RTL generation?

Table II shows a comparison of the performance of nine commercial LLMs on VerilogEval-V2, RTLLM-V2, and LiveVerilogEval-V1.0.

The "VerilogEval-V2 (All)" and "RTLLM-V2 (All)" rows show the results of evaluating the entirety of the two benchmarks. The "VerilogEval-V2 (15 problems)" and "RTLLM-V2 (8 problems)" rows show the results of evaluating the benchmark on the subset of problems that are used as seed data for generating problems in LiveVerilogEval. The "LiveVerilogEval (VerilogEval)" and "LiveVerilogEval (RTLLM)" rows show the evaluation results on the subsets of the benchmark being generated from the initial datasets of VerilogEval and RTLLM, respectively.

Based on the results shown in the "VerilogEval-V2 (15 problems)" and "LiveVerilogEval (VerilogEval)" rows, we find that the performances of different models are inconsistent. Specifically, we note that the performance of Claude 3.7-Sonnet relative to other models is significantly higher on the subset benchmark LiveVerilogEval (VerilogEval), while the performance of GPT-5 remains the highest in both benchmarks, suggesting that the performance of these two models tends to be more robust and is likely to be a product of problem memorization. On the other hand, the performance of GPT-4 relative to the other models is signficantly lower on LiveVerilogEval (VerilogEval) compared to VerilogEval-V2 (15 problems), suggesting that GPT-4's performance is significantly influenced by benchmark problem sets leaked to GPT-4's training data.

TABLE II: Comparison between the performance of 9 commercial LLMs on the VerilogEval-V2 [14] and RTLLM-V2 [4] benchmarks

Benchmark	Model	Pass@K	
		Pass@1	Pass@5
VerilogEval-V2 (All)	DeepSeek-V3.1	62.3	70.0
	Claude-3.5-Haiku	57.6	69.2
	Claude-3.7-Sonnet	76.2	82.0
	Claude-4-Sonnet	72.4	80.1
	GPT-3.5-Turbo	33.9	51.2
	GPT-4	58.9	71.7
	GPT-4o-mini	50.0	58.3
	GPT-4o	56.4	73.7
	GPT-5	83.3	91.6
RTLLM-V2 (All)	DeepSeek-V3.1	56.0	62.0
	Claude-3.5-Haiku	48.0	54.0
	Claude-3.7-Sonnet	58.0	60.0
	Claude-4-Sonnet	58.0	62.0
	GPT-3.5-Turbo	31.2	39.6
	GPT-4	38.8	46.9
	GPT-4o-mini	40.8	53.1
	GPT-4o	51.0	59.2
	GPT-5	54.0	62.0
VerilogEval-V2 (15 problems)	DeepSeek-V3.1	73.3	86.6
	Claude-3.5-Haiku	53.3	73.3
	Claude-3.7-Sonnet	80.0	86.6
	Claude-4-Sonnet	86.6	93.3
	GPT-3.5-Turbo	6.6	40.0
	GPT-4	60.0	80.0
	GPT-4o-mini	46.6	60.0
	GPT-4o	86.6	93.3
	GPT-5	93.3	100.0
RTLLM-V2 (8 problems)	DeepSeek-V3.1	37.5	50.0
	Claude-3.5-Haiku	50.0	50.0
	Claude-3.7-Sonnet	50.0	50.0
	Claude-4-Sonnet	50.0	50.0
	GPT-3.5-Turbo	25.0	25.0
	GPT-4	37.5	37.5
	GPT-4o-mini	37.5	37.5
	GPT-4o	37.5	37.5
	GPT-5	50.0	50.0
LiveVerilogEval (VerilogEval)	DeepSeek-V3.1	70.9	74.5
	Claude-3.5-Haiku	67.1	70.9
	Claude-3.7-Sonnet	72.6	78.2
	Claude-4-Sonnet	70.9	74.5
	GPT-3.5-Turbo	40.0	63.6
	GPT-4	54.4	70.9
	GPT-4o-mini	58.2	69.1
	GPT-4o	70.9	83.6
	GPT-5	72.6	78.1
LiveVerilogEval (RTLLM)	DeepSeek-V3.1	66.7	72.2
	Claude-3.5-Haiku	44.2	55.5
	Claude-3.7-Sonnet	38.9	55.5
	Claude-4-Sonnet	66.7	72.2
	GPT-3.5-Turbo	33.3	38.9
	GPT-4	27.5	44.2
	GPT-4o-mini	44.4	50.0
	GPT-4o	38.9	50.0
	GPT-5	60.8	60.8

VI. Limitations and Future Work

Despite being able to produce a contamination-free benchmark, LiveVerilogEval has multiple limitations.

First, since LiveVerilogEval generates candidate answers to confirm the validity of the question oracle, the questions produced by LiveVerilogEval are biased to be solvable by an existing LLM. We address this issue by providing the LLM with additional context when generating answers, such as the initial design and the implemented mutation. However, for a generated problem to be incorporated into LiveVerilogEval, the problem must have been solved by an LLM. Due to this limitation, LiveVerilogEval acts more as a comparative benchmark, which contrasts the overall performance differences between models, rather than a benchmark that evaluates the models' understanding of novel concepts or large designs. Second, the complexity of the benchmark questions heavily depends on the amount of time allocated to the SAT solver when confirming functional equivalence. Because we set a 60-second timeout, we disqualify a large number of difficult questions, as we are not able to confirm, with certainty, the existence of a valid answer.

In future work, we plan to expand the initial datasets used to generate LiveVerilogEval to open-source designs that are not already incorporated into a benchmark. In addition to regular updates to LiveVerilogEval's problem set, we seek to further refine the generation process of our framework's questions by incorporating multiple mutations into one mutant design and improving our question accuracy oracle's ability to differentiate between valid and invalid questions.

VII. Conclusion

In this paper, we have introduced LiveVerilogEval, a dynamic generation framework to generate novel problems for reliably evaluating LLM-based Verilog generation. Unlike traditional benchmarks that require manually-written descriptions and testbenches, our framework generates challenging RTL design problems directly from valid reference designs, ensuring contamination-free evaluations. We have applied our framework to the VerilogEval-V2 and RTLLM-V2 benchmarks. Our experimental results reveal that LLM-based Verilog generation remains challenging, confirming the effectiveness of LiveVerilogEval for contamination-free evaluation.

VIII. Acknowledgements

Tao Xie is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China; Fudan University Institute of Systems for Advanced Computing, Shanghai, China; Beijing Tongming Lake Information Technology Application Innovation Center, Beijing, China; Shanghai Institute of Systems for Open Computing, Shanghai, China. This work was partially supported by National Natural Science Foundation of China under Grant No. 623B2006, 92464301 and U25A6023. This research was also partially conducted by ACCESS AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government. Additional support was provided by Research Grants Council of Hong Kong SAR (16213824 & 16212825).

REFERENCES

[1] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "VeriGen: A large language model for Verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[2] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "RTLCoder: Outperforming GPT-3.5 in design RTL generation with our open-source dataset and lightweight solution," in *2024 IEEE LLM Aided Design Workshop*, 2024, pp. 1–5.

[3] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating large language models for Verilog code generation," in *IEEE/ACM International Conference on Computer Aided Design*, 2023, pp. 1–8.

[4] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design RTL generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference*, 2024, pp. 722–727.

[5] Z. Wang, M. Shao, J. Bhandari, L. Mankali, R. Karri, O. Sinanoglu, M. Shafique, and J. Knechtel, "VeriContaminated: Assessing llm-driven Verilog coding for data contamination," *arXiv preprint arXiv:2503.13572*, 2025.

[6] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, "LiveCodeBench: Holistic and contamination free evaluation of large language models for code," Jun. 2024.

[7] C. White, S. Dooley, M. Roberts, A. Pal, B. Feuer, S. Jain, R. Shwartz-Ziv, N. Jain, K. Saifullah, S. Dey, Shubh-Agrawal, S. S. Sandha, S. Naidu, C. Hegde, Y. LeCun, T. Goldstein, W. Neiswanger, and M. Goldblum, "LiveBench: A challenging, contamination-limited LLM benchmark," Apr. 2025.

[8] https://github.com/agctute/liveverilogeval, 2025.

[9] C. Wolf. YosysHQ. [Online]. Available: https://github.com/YosysHQ/yosys

[10] N. Eén and N. Sörensson.

[11] W. Shi, A. Ajith, M. Xia, Y. Huang, D. Liu, T. Blevins, D. Chen, and L. Zettlemoyer, "Detecting pretraining data from large language models," in *International Conference on Learning Representations, Poster*.

[12] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated Verilog RTL code generation," in *Design, Automation and Test in Europe Conference Exhibition*, 2023, pp. 1–6.

[13] (2025) HDLBits. [Online]. Available: https://hdlbits.01xz.net

[14] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, "Revisiting VerilogEval: A year of improvements in large-language models for hardware code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 91, pp. 1–20, 2025.

[15] P. Jin, D. Huang, C. Li, S. Cheng, Y. Zhao, X. Zheng, J. Zhu, S. Xing, B. Dou, R. Zhang *et al.*, "RealBench: Benchmarking Verilog generation models with real-world IP designs," *arXiv preprint arXiv:2507.16200*, 2025.

[16] K. Chang, Z. Chen, Y. Zhou, W. Zhu, K. Wang, H. Xu, C. Li, M. Wang, S. Liang, H. Li *et al.*, "Natural language is not enough: Benchmarking multi-modal generative AI for Verilog generation," in *43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[17] S. Williams. (2025) The ICARUS Verilog compilation system. [Online]. Available: https://github.com/steveicarus/iverilog

[18] S. Takamaeda-Yamazaki, "PyVerilog: A Python-based hardware design processing toolkit for Verilog HDL," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2015, pp. 451–460.