2025 Design, Automation & Test in Europe Conference (DATE 2025) – Focus Session

FVEval: Understanding Language Model Capabilities in Formal Verification of Digital Hardware

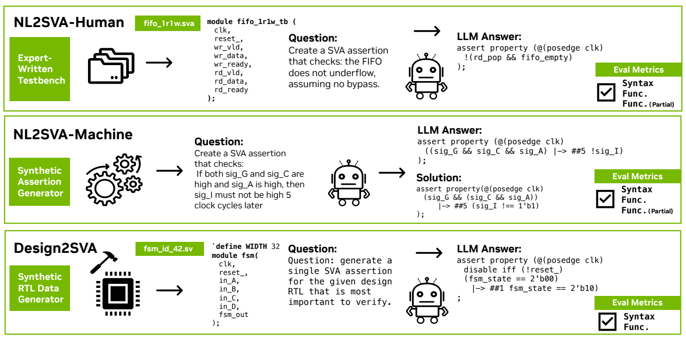
Minwoo Kang[*], Mingjie Liu[†], Ghaith Bany Hamad[†], Syed M. Suhaib[†] and Haoxing Ren[†]
[*]University of California, Berkeley, CA
[†]NVIDIA, Santa Clara, CA
mkang@cs.berkeley.edu, {mingjiel, gbanyhamad, ssuhaib, haoxingr}@nvidia.com

***Abstract*—The remarkable reasoning and code generation capabilities of large language models (LLMs) have spurred significant interest in applying LLMs to enable task automation in digital chip design. In particular, recent work has investigated early ideas of applying these models to formal verification (FV), an approach to verifying hardware implementations that can provide strong guarantees of confidence but demands significant amounts of human effort. While the value of LLM-driven automation is evident, our understanding of model performance, however, has been hindered by the lack of holistic evaluation. In response, we present FVEval, the first comprehensive benchmark and evaluation framework for characterizing LLM performance in tasks pertaining to FV. The benchmark consists of three sub-tasks that measure LLM capabilities at different levels—from the generation of SystemVerilog assertions (SVA) given natural language descriptions to reasoning about the design RTL and suggesting assertions directly without additional human input. As test instances, we present both collections of expert-written verification collateral and methodologies to scalably generate synthetic examples aligned with industrial FV workflows. A wide range of existing LLMs, both proprietary and open-source, are evaluated against FVEval, based on which we investigate where today's LLMs stand and how we might further enable their application toward improving productivity in digital FV. Our benchmark and evaluation code is available at https://github.com/NVlabs/FVEval.**

I. Introduction

Hardware verification is a critical stage in digital VLSI development that is not only challenging but also indispensable to successful chip manufacturing. To ensure that design implementations are fully verified against expected functional behaviors, industrial teams are increasingly adopting *formal* verification (FV), an approach that formulates specifications of the design as temporal logical properties and applies formal methods to mathematically prove that the said properties hold across all expected input stimuli. While FV offers distinct advantages, such as the ability to detect bugs that are elusive or hard-to-reach with simulation-based approaches, it also bears limitations: FV workflows suffer from steep engineering costs of manually crafting formal testbenches and collateral by human experts. As a result, scaling the adoption of FV and extending its use to cover a larger number of sub-systems remains a challenge.

With the recent advancements of large language models (LLMs) [1], [5], [12], [8], researchers have in turn started to apply language models automation and improving productivity in hardware verification [16], [10], [25], [26], with hopes to apply the impressive capabilities in reasoning and code generation [4], [11], [2]. to hardware domain tasks. In particular, recent work has investigated the possibility of LLMs



```
module fifo_1r1w_tb (
  clk,
  reset_,
  wr_vld,
  wr_data,
  wr_ready,
  rd_vld,
  rd_data,
  rd_ready
);
```

Question:
Create a SVA assertion that checks: the FIFO does not underflow, assuming no bypass.

LLM Answer:

```
assert property (@(posedge clk)
  !(rd_pop && fifo_empty)
);
```

Question:
Create a SVA assertion that checks:
If both sig_G and sig_C are high and sig_A is high, then sig_I must not be high 5 clock cycles later

LLM Answer:

```
assert property (@(posedge clk)
  ((sig_G && sig_C && sig_A) |-> ##5 !sig_I)
);
```

Solution:

```
assert property(@(posedge clk)
  (sig_G && (sig_C && sig_A))
     |-> ##5 (sig_I !== 1'b1)
);
```

```
`define WIDTH 32
module fsm(
  clk,
  reset_,
  in_A,
  in_B,
  in_C,
  in_D,
  fsm_out
);
```

Question:

```
Question: generate a
single SVA assertion
for the given design
RTL that is most
important to verify.
```

LLM Answer:

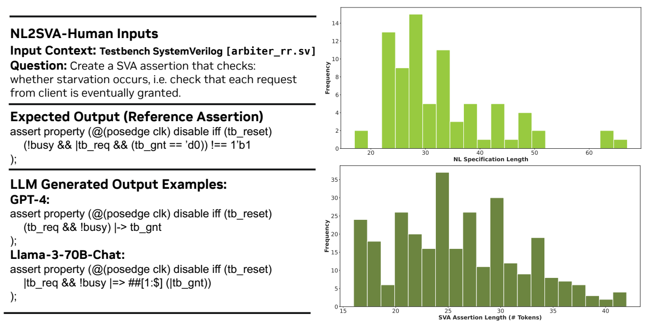
```
assert property (@(posedge clk)
  disable iff (!reset_)
  (fsm_state == 2'b00)
    |-> ##1 fsm_state == 2'b10)
;
```

Fig. 1: Overview of the FVEval benchmark and evaluation flow. FVEval consists of three sub-benchmarks NL2SVA-Human, NL2SVA-Machine, and Design2SVA that measure model capabilities in generating functionally correct implementations of assertions from NL descriptions and also directly from design under test. The evaluation framework integrates industry-standard FV tools to accurately measure LLM response correctness.

to perform tasks in formal verification such as generating formal assertions from natural language (NL) specifications [6] and from register-transfer level (RTL) implementations of the design-under-test (DUT) [21].

While prior work has demonstrated that current LLMs have the potential to generate hardware assertions, their evaluation, however, has been limited to a hand-select set of test instances with typically less than 10 or 20 problems. Due to the dearth of publicly available repositories of reliable designs and formal testbenches, it has been challenging to curate a dataset for evaluation that encompasses a variety of designs. Furthermore, prior evaluation of LLMs has considered a limited set of tasks which renders a holistic evaluation of model capabilities unviable. This altogether raises the question: how should we *comprehensively* understand and quantify current LLMs capabilities in performing tasks in hardware formal verification?

In this work, we present the first comprehensive benchmark for LLMs on diverse tasks pertaining to formal verification via SystemVerilog assertions, which we refer to as *FVEval*. As shown in Figure 1, FVEval is a collection of three sub-benchmarks: NL2SVA-Human, NL2SVA-Machine, and Design2SVA which each evaluate LLMs in different industry-relevant scenarios exercising various capabilities related to code generation and reasoning in the context of SystemVerilog and hardware formal properties. As test problem instances, we



NL2SVA-Human Inputs

Input Context: **Testbench SystemVerilog [`arbiter_rr.sv`]**

Question: Create a SVA assertion that checks:
whether starvation occurs, i.e. check that each request from client is eventually granted.

Expected Output (Reference Assertion)

```
assert property (@(posedge clk) disable iff (tb_reset)
    (!busy && |tb_req && (tb_gnt == 'd0)) !== 1'b1
);
```

LLM Generated Output Examples:

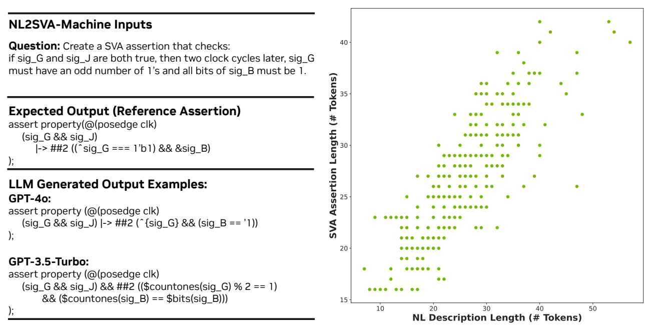
GPT-4:

```
assert property (@(posedge clk) disable iff (tb_reset)
    (tb_req && !busy) |-> tb_gnt
);
```

Llama-3-70B-Chat:

```
assert property (@(posedge clk) disable iff (tb_reset)
    |tb_req && !busy |=> ##[1:$] (|tb_gnt))
);
```

Fig. 2: (Left) Input and output examples of the NL2SVA-Human benchmark. Reference solution is the SVA assertion written by the human engineer that the LLM response is expected to match in terms of functionality. (Right) Distribution of token length for the NL specifications and the reference SVA solutions contained in the benchmark.



NL2SVA-Machine Inputs

Question: Create a SVA assertion that checks:
if sig_G and sig_J are both true, then two clock cycles later, sig_G must have an odd number of 1's and all bits of sig_B must be 1.

Expected Output (Reference Assertion)

```
assert property(@(posedge clk)
    (sig_G && sig_J)
        |-> ##2 ((^sig_G === 1'b1) && &sig_B)
);
```

LLM Generated Output Examples:

GPT-4o:

```
assert property (@(posedge clk)
    (sig_G && sig_J) |-> ##2 (^{sig_G} && (sig_B == '1))
);
```

GPT-3.5-Turbo:

```
assert property (@(posedge clk)
    (sig_G && sig_J) && ##2 (($countones(sig_G) % 2 == 1)
          && ($countones(sig_B) == $bits(sig_B)))
);
```

SVA Assertion Length (# Tokens)
NL Description Length (# Tokens)

Fig. 3: (Left) Input and output examples of the NL2SVA-Machine benchmark. The dataset consists of naturalized descriptions of the formal-symbolic logic formulae and the corresponding SVA assertion, based on which the NL annotations were created. (Right) Distribution of lengths of NL descriptions and SVA assertions, measured in token length using the Llama3 tokenizer.

either collect human-written designs and testbenches or present methodologies to scalably generate synthetic yet realistic test cases motivated by real-world designs. Finally, we evaluate a wide range of existing LLMs, both proprietary and open-source, against FVEval and delineate the limitations of current models.

II. Overview of FVEval

In this work, we focus on assessing the viability of LLMs to be applied towards industrial hardware FV based on SystemVerilog Assertions (SVAs).

A. Assertion Generation with Real-World Testbenches

The first sub-benchmark, **NL2SVA-Human** raises the question: *Do LLMs have the capability to generate SVA assertions, given real-world, human-written testbenches and high-level specifications of design functionality?* Even though SVA assertions can be relatively short pieces of code, the outputs must accurately reflect the specification and do so in the context of the provided testbench, which entails grounding the implementation of the assertion on existing modeling code provided in the testbench.

Dataset. Our collection of test instances covers basic units of FV that engineers repeatedly encounter as part of larger subsystems, such as first-in-first-out (FIFO) queues, arbiters, hardware counters, random-access memory (RAM) units, and finite-state machines. Each testbench comes with a set of assertions and accompanying natural language description written by formal experts that address various design functionalities, with the benchmark holding a total of 79 assertions. Figure 2 summarizes the statistics about the test instances, in terms of the distribution of lengths of natural language specifications and reference SVA assertion solutions. We measure length as the number of tokens based on the tokenizer used in the Llama3 models [5]. We see a wide range in the distribution reflecting the variety of formal properties in the benchmark.

The models are given as prompt input: (1) the testbench code (SystemVerilog) with all original assertions removed but containing modeling code with internal signals (wires and state) and input/output ports of the testbench module; and (2) a *high-level* NL description of the intended assertion. An example is shown on the left of Figure 2. The model response is expected to match the human-written assertion that serves as ground-truth.

Evaluation. We evaluate model responses in terms of syntax and functional correctness. For syntax correctness, we measure whether the generated SVA assertions pass the syntax check performed by industry-standard, commercial FV tools such as Cadence Jasper. For functional correctness, we present a novel approach in measuring *exact logical equivalence* between a pair of assertions, model-generated and ground-truth. Our method utilizes a custom-implemented Jasper function that *formally* proves logical equivalence, and further, can also evaluate whether there is an implication relationship between the two assertions. We also report a relaxed metric of functional accuracy that includes such a case of *partial equivalence* and thereby capture the performance of LLMs at a finer granularity than only considering exact equivalence. Finally, as an alternative measure of functional correctness, the lexical similarity between the model-generated code against the reference solution using standard n-gram metrics such as BLEU could be considered. While we report BLEU scores, we do note that commercial FV tools are required for accurate evaluation of SVA assertions.

B. Synthetic Benchmark for Stress-Testing SVA Generation

We also note that in practice, FV engineers not only repeat similar patterns of assertions for common sub-units but also implement custom assertions tailored to a specific property to be checked. **NL2SVA-Machine** attempts to quantify LLM performance the second case: *"Can LLMs flexibly handle diverse NL specifications of formal properties and accurately formulate the same formal logic in SVA syntax?"*

Dataset. To create diverse test cases of NL description and SVA assertion pairs, we follow the process of: (1) random SVA assertion generation, based on random sampling of SVA operators and symbolic signal names; (2) LLM generation of NL descriptions for each random assertion; (3) LLM as a critic to assess whether the NL description accurately reflects the temporal logic of the assertion—if this fails, re-try description generation; and (4) Human inspection to finalize the appropriateness of generated descriptions. For description generation, we run LLM inference with sampling temperature $T = 1.0$ and for assessment as a critic, we use greedy sampling with temperature $T = 0.0$. We use `gpt-4o` for NL description generation and `gpt-4-turbo` for assessment.

With a fully automated generated flow, NL2SVA-Machine consists of 300 test cases with widely varying patterns of SVA assertions and natural language usage variations to describe them. An example of a test instance and sample LLM responses are shown in Figure 3. As input context, models are solely given a symbolic text description of the logical formula, and the expected output is an SVA assertion matching the logical expression. Summaries of the distribution of randomly generated test cases are shown in Figure 3.

Evaluation. As NL2SVA-Machine also prompts models with natural language specifications and expects outputs in concrete SVA assertion implementations, we follow the same evaluation protocol and metrics as NL2SVA-Human.

C. Direct Generation of Formal Assertions from Design RTL

Finally, a more challenging but extremely helpful application of LLMs is their use in suggesting formal properties to be verified for a particular design RTL and prior to a human engineer drafting a full specification. In **Design2SVA**, we ask: *"Can LLMs craft relevant SVA assertions directly from design RTL and without human guidance?"*

Dataset. Our objective in formulating the Design2SVA benchmark is to present a set of test cases consisting of design RTL that are: (1) sufficiently correct, as in there exist formal properties that can be proven to be true; (2) relevant to real-world FV use-cases; (3) suitable as test instances for language models; and (4) varied in terms of complexity. While it is ideal to collect and curate existing RTL examples, we find that openly available SystemVerilog/Verilog repositories contain limited numbers of verified RTL designs. Of the cases that satisfy the first and second criteria, such as module instances from OpenTitan [20], these cases fail to be suitable for language model evaluation, as each module is a part of a large System-on-a-Chip (SoC) system and the context needed to resolve all sub-module dependency information is prohibitively large.

Instead, we propose a methodology to scalably generate complex and parameterized synthetic test instances that are derived from common design patterns encountered in industrial FV workflows. As shown in Figure 4, the two categories of RTL designs we generate are (1) arithmetic pipelines that resemble scenarios where we check for data integrity and forward propagation across datapaths; and (2) finite-state machines (FSMs) that commonly appear in control logic implementations, such as cache controllers, memory interfaces, etc. Each category of our designs has randomized sub-components—the arithmetic execution units in the pipeline designs and the FSM graph and state transition logic in the FSM design—and each generated RTL is parameterized such that controlled generation of test instances is possible. While a virtually limitless number of designs can be generated in this manner, we compose the Design2SVA benchmark to contain 96 test instances for each design category (pipeline and FSM) based on a controlled sweep of generator parameters.

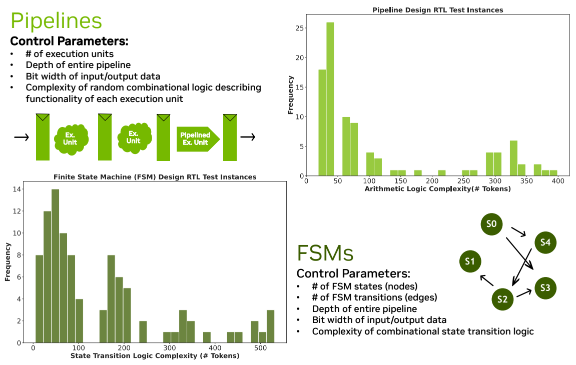


Fig. 4: Synthetic RTL Testcase generation. For each category of designs, we vary a list of control parameters to generate test design instances such that the total set of 96 cases for each category exhibit a wide distribution of difficulties. The total token length of the randomly generated arithmetic logic and FSM transition logic, correlated with complexity of designs under FV, are shown.

Evaluation. For each generated RTL test instance, we also generate an accompanying formal testbench, based on which language models generate suggestions of assertions. The FVEval evaluation flow then re-formats the testbench with the model-generated assertions and supplies the SystemVerilog to the commercial tool backend for evaluation. The metrics of evaluation are similar to the NL2SVA benchmarks: (1) Syntax—we measure the LLM generated assertion is first syntactically correct; (2) Functionality—we use the results of formal proofs, *i.e.* whether the assertions are proven with model checkers and other formal engines in industrial tools, as an indication of functional correctness.

Unlike in NL2SVA-Human and NLSVA-Machine, the Design2SVA benchmark subjects LLMs to generate relevant and correct SVA assertions given a design under verification, and as such, there are numerous valid completions that are functionally different from each other. To quantify model capability in this setting, we consider the *pass@k* metric, where a problem instance is considered solved if any of the k attempted solutions are correct.

III. Results and Analysis

In this section, we report the results of evaluating a suite of current state-of-the-art language models, trained on considerable sizes of text corpora pertaining to diverse programming languages.

A. Experiment Setup

A wide range of LLMs, both proprietary and open-source (i.e. models with weights publicly available) are considered. We consider the most recent OpenAI model `gpt-4o-0513` [1] and Google Gemini models [8] `gemini-1.5-pro-001` and `gemini-1.5-flash-001`. All results of the proprietary models are based on accessing their respective endpoint services in May 2024. For open-source models, we evaluate the models from the Llama 3 and Llama 3.1 family [5] with 8B and 70B parameters; we also consider `mixtral-8x22b-instruct-v1`, a sparse Mixture-of-Experts model with approximately 39B active

Model	Syntax	Func.	Partial Func.	BLEU
gpt-4o	<u>0.911</u>	**0.456**	**0.582**	0.503
gemini-1.5-pro	0.810	0.253	0.380	0.484
gemini-1.5-flash	**0.949**	<u>0.380</u>	<u>0.557</u>	0.518
Mixtral-8x22b	0.823	0.190	0.278	0.450
Llama-3.1-70b	0.861	0.291	0.354	0.464
Llama-3-70b	0.899	0.291	0.506	0.464
Llama-3.1-8b	0.835	0.203	0.304	0.525
Llama-3-8b	0.747	0.063	0.215	0.491

TABLE I: Evaluation results for the NL2SVA-Human Benchmark. Boldface and underlined results indicate highest and second highest metric values, respectively. We see that models that proprietary models with generally advanced capabilities for code generation and logical reasoning achieve the best performance across all metrics. Notably, all models show a significant gap between full and *partial* equivalence of assertion functionality—this highlights the value of fine-grained measurement of functional correctness that captures how close LLM-generated assertions are similar to the human-written reference solution.

parameters [12]. Note that all models are instruction fine-tuned models, as our tasks heavily require instruction-following capabilities. Evaluation of open-source models was performed via vLLM [15], a high-throughput LLM inference framework.

B. *NL2SVA-Human Results*

We consider zero-shot model inference where (1) the problem testbench and (2) NL descriptions of the intended SystemVerilog assertion is given as model input. Evaluation results from running model inference with greedy decoding are summarized in Table I. We observe that the models reported with highest general capabilities in code generation and logical reasoning, such as `gpt-4o` and `gemini` models, show the best performance across all metrics.

LLMs can generate syntactically correct SVA code but are still prone to hallucinations. Even the most capable models considered, we observe that models are still prone to hallucinating subsets of SVA syntax, e.g. `s_eventually`, `strong`, and other particular operators. Such hallucination is also observed in the most capable models including `gpt-4o`, `gemini-1.5-pro`, and `gemini-1.5-flash`.

Partial functional equivalence as a correctness metric reveals further understanding of model capabilities. Overall, there is a notable gap between the accuracy measured with exact functional equivalence and that with partial equivalence. Such a gap can be attributed to the innate ambiguity in the high-level NL descriptions—while full functional equivalence necessitates correctness, those that imply the ground-truth solution and vice-versa are also possibly valid solutions given the input descriptions. NL2SVA-Human reflects scenarios in the industrial FV workflows, where engineers wish to benefit from LLM-based auto-generation of SVA assertions from possibly ambiguous NL specifications. Partially equivalent assertions are often reasonable responses to the natural language input, and taking into account both the full and partial functional equivalence of LLM-generated solutions gives a richer picture of how

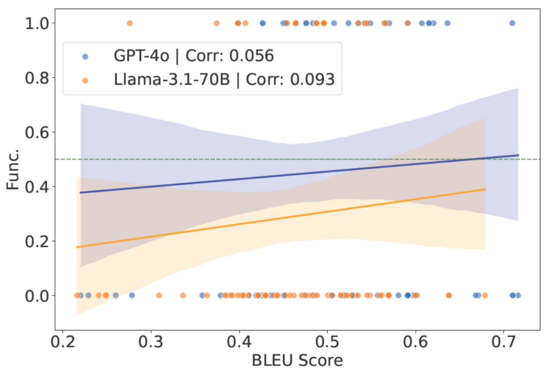


Fig. 5: Analysis on the correlation between functionality correctness measures based on formal equivalence versus BLEU scores as approximate measures of assertion similarity. We see little correlation between the two metrics, with coefficients 0.0563 and 0.0926 respectively, indicating that BLEU scores are insufficient to fully capture the equivalence of a LLM-generated assertion against a reference solution.

these models would be applicable in scenarios where engineers provide high-level sketches of desired assertion functionality.

BLEU scores are not indicative of formal, functional correctness for NL2SVA-Human. In Figure 5, we see that BLEU scores, measuring similarity between LLM outputs against ground-truth assertion implementations, are not correlated with formal equivalence based measures of correctness. While BLEU scores or other lexical measures capture surface-level similarities between outputs, we find that for a complete analysis of LLM capabilities, it is necessary to analyze formal equivalence between assertions, which is implemented as part of FVEval evaluation framework.

C. *NL2SVA-Machine Results*

A further task to quantify model capability in handling diverse SystemVerilog operator syntax and temporal logic reasoning based on such grammar, NL2SVA-Machine evaluation examines whether LLMs can accurately reason about the temporal logic expressed in natural language and translate into SVA syntax. Table II summarizes model performances in zero-shot and with 3-shot inference settings, where the hand-crafted in-context examples are fixed across test questions.

LLMs achieve high accuracy in terms of syntax but struggle with generating SVA matching the assertion specification expressed as natural language input. With in-context learning, models with high parameter count reach close to perfect accuracy in terms of syntax; however, the accuracy on functionality both exact (full equivalence) and relaxed (partial equivalence) does not improve significantly. The gap between syntax and functional correctness in state-of-the-art models indicate the limitations of these models in accurately identifying the propositional and modal temporal logic expressed in natural language and translating into SVA implementation.

In-context examples can aid models to generate functionally correct implementations. With three handcrafted in-context examples, the same model inference under greedy decoding can generate functionally correct SVA

Model	0-shot				3-shot			
	`Syntax`	`Func.`	`Partial Func.`	BLEU	`Syntax`	`Func.`	`Partial Func.`	BLEU
gpt-4o	**0.927**	**0.430**	**0.540**	0.622	**0.937**	**0.467**	**0.570**	0.660
gemini-1.5-pro	0.467	0.137	0.203	0.510	0.880	0.417	0.517	0.622
gemini-1.5-flash	0.783	0.377	0.470	0.574	0.837	0.397	0.480	0.613
Mixtral-8x22b	0.913	0.327	0.500	0.546	0.880	0.430	0.523	0.599
Llama-3.1-70b	0.887	0.303	0.397	0.595	0.920	0.457	0.567	0.651
Llama-3-70b	0.863	0.330	0.430	0.570	0.860	0.380	0.503	0.621
Llama-3.1-8b	0.813	0.320	0.520	0.533	0.840	0.267	0.370	0.582
Llama-3-8b	0.673	0.187	0.320	0.493	0.827	0.240	0.397	0.552

TABLE II: Evaluation results for the NL2SVA-Machine Benchmark. Boldface and underlined results indicate highest and second highest metric values, respectively. While even the smaller models achieve moderately high syntax correctness (>80%) with in-context examples, we see limitations in functional correctness and again a gap between full and partial correctness of model-generated assertions.

implementations for the case of `gpt-4o`. However, in some cases, the fixed in-context examples may serve as distractors and negatively impact generation, especially for models with smaller parameter count such as `Llama-3.1-8B`. We posit that further improvements schemes of generating or retrieving appropriate in-context examples could therefore improve LLM capabilities in generating SystemVerilog assertions.

D. Design2SVA Results

Finally, Design2SVA tests whether the models can produce plausible assertions directly from the design RTL. Besides the logical reasoning capabilities required in the NL2SVA-Human and NL2SVA-Machine tasks, Design2SVA further requires understanding the semantics of the given RTL implementation and also what concrete implementation of a formal property would be relevant for the design. The synthetic RTL test cases (pipeline and FSM examples) are generated from simple more complex examples, with the most complex instances taking greater than 16K tokens when provided as context to all of the LLMs considered. As such, we do not consider the Llama-3 family models and older OpenAI models with less than a 32K context window for this task.

Table III shows the `pass@k` values for syntax and functional correctness, as defined. The reported `pass@k` are based on the unbiased estimator of `pass@k` as described in [4] and widely used in the literature. As we generate multiple samples of the LLM output, we use nucleus sampling with top $p=0.95$ and temperature $t=0.8$ as in [18].

The results show that despite most models achieve near-perfect `pass@5` for syntax correctness, the observed functional correctness—measured by whether the model has generated an assertion that can be proven—varies significantly by model. The latest proprietary models such as `gemini-1.5-pro` and `gpt-4o` achieve considerably high `pass@5` for functional correctness for the `FSM` test instances, while open source models achieve a lower `pass@k` for most cases. Between the open-source Llama 3.1 models, we observe a significant performance gap between the larger- and smaller-scale models.

In general, we find that LLMs are of identifying plausible candidates of formal properties *without any human guidance or specification* and generating correct SystemVerilog assertions that are *proven by commercial tools*. With the success rate of `pass@k` shown in Table III, we anticipate that LLMs will be used to "draft" formal testbenches, providing a list of possible assertions to be generated for the DUT, from which the user engineer may take any assertions directly or modify as needed.

IV. Related Work

LLMs for Chip Design and Formal Verification. Researchers have applied language models, and more broadly deep learning to various problems in chip design [22], [3], [7], [27], [14]. Recent work fine-tune open-source language models such as CodeGen [19] and LLaMA on chip design related data and show that fine-tuned models out-perform both open-source and proprietary models in Verilog code generation [24], [18] and EDA tool script generation [10]. Prior work on applying LLMs to hardware formal verification can be broadly categorized into two groups: one line of work takes micro-architectural descriptions of the design-under-test (DUT) and prompts LLMs to generate a plausible formal testbench [21], [23], [13], [9]. Another line of work instead have LLMs consume architectural specification documents, describing high-level specifications that have not yet been formulated as RTL implementations, and have the models create an entire testbench [6]. However, existing evaluations of LLMs on FV [21], [6], [17], [23] are limited to less than $\sim$20 cases of test instances and have considered a limited variety of task settings. In contrast, FVEval expands both the scope and scale of test instances, combining collected human-written examples from industrial experts as well as verified synthetic instances generated from parameterized templates, and considers three sub-tasks each exercising varying aspects of LLM capabilities for real-world applications in FV.

V. Conclusion

In this work, we present FVEval—a first comprehensive benchmark for evaluating language models in the context of hardware formal verification. We demonstrate three sub-tasks that are derived from industrial FV use cases and each examine different aspects of LLM capabilities necessary for their application in this domain. FVEval defines evaluation metrics for each task with end-to-end automated evaluation made possible with a industry-standard formal verification tool. In particular, the custom-implemented formal equivalence checking between model generated assertions against ground-truth solution offers, for the first time, a concrete metric for evaluating LLMs in tasks generating formal properties in SVA syntax.

Model	Pipeline				FSM			
	Syntax@1	Syntax@5	Func.@1	Func.@5	Syntax@1	Syntax@5	Func.@1	Func.@5
gpt-4o	0.802	1.000	0.104	0.427	0.993	1.000	0.373	0.900
gemini-1.5-pro	0.665	1.000	**0.175**	0.500	0.950	1.000	**0.427**	**0.906**
gemini-1.5-flash	**0.969**	1.000	0.025	0.125	**0.996**	1.000	0.079	0.281
Mixtral-8x22b	0.867	1.000	0.119	0.472	0.974	1.000	0.054	0.167
Llama-3.1-70b	0.960	1.000	0.167	**0.615**	0.940	1.000	0.231	0.719
Llama-3.1-8b	0.904	1.000	0.150	0.552	0.906	1.000	0.121	0.521

TABLE III: Evaluation results for the Design2SVA Benchmark. Boldface and underlined results indicate highest and second highest metric values, respectively.

The current state of FVEval is only the first step towards understanding and thereby improving model capabilities for hardware FV. We expect future work to encompass further variety in the designs and testbenches, such as considering synthetic data generation with different styles of design modules besides the arithmetic pipeline and FSMs considered in this work. We also anticipate methodologies that will improve LLM performance on our task, e.g. carefully chosen prompting schemes and inference-time methods, as well as ideas to incorporate tool-feedback or external symbolic reasoning tools as part of a LLM-agentic framework.

Acknowledgements

Authors deeply thank Vigyan Singhal for technical guidance and support. We also thank Cadence Design Systems for the implementation of the formal assertion-to-assertion equivalence checking in Jasper and contributions to the overall project, including helpful discussions.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program Synthesis with Large Language Models. *ArXiv*, abs/2108.07732, 2021.

[3] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond A. Pearce. Chip-chat: Challenges and opportunities in conversational hardware design. *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2023.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code. *ArXiv*, abs/2107.03374, 2021.

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

[6] Wenji Fang, Mengming Li, Min Li, Zhiyuan Yan, Shang Liu, Hongce Zhang, and Zhiyao Xie. Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms. *ArXiv*, abs/2402.00386, 2024.

[7] Steven J. Frederiksen, John J. Aromando, and Michael S. Hsiao. Automated assertion generation from natural language specifications. *2020 IEEE International Test Conference (ITC)*, pages 1–5, 2020.

[8] Gemini Team Google. Gemini: A family of highly capable multimodal models. *ArXiv*, abs/2312.11805, 2023.

[9] Muhammad Hassan, Sallar Ahmadi-Pour, Khushboo Qayyum, Chandan Kumar Jha, and Rolf Drechsler. Llm-guided formal verification coupled with mutation testing. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2. IEEE, 2024.

[10] Zhuolun He, Haoyuan Wu, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2023.

[11] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021.

[12] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, et al. Mixtral of experts. *ArXiv*, abs/2401.04088, 2024.

[13] Rahul Kande, Hammond A. Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, Jeyavijayan Rajendran Texas AM University, University of New South Wales, University of Calgary, and New York University. Llm-assisted generation of hardware assertions. *ArXiv*, abs/2306.14027, 2023.

[14] Minwoo Kang, Azade Nova, Eshan Singh, Geetheeka Sharron Bathini, and Yuriy Viktorov. LFPS: Learned Formal Proof Strengthening for Efficient Hardware Verification. *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[15] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[16] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, et al. Chipnemo: Domain-adapted llms for chip design. *ArXiv*, abs/2311.00176, 2023.

[17] Mingjie Liu, Minwoo Kang, Ghaith Bany Hamad, Syed Suhaib, and Haoxing Ren. Domain-Adapted LLMs for VLSI Design and Verification: A Case Study on Formal Verification. In *2024 IEEE 42nd VLSI Test Symposium (VTS)*, pages 1–4. IEEE, 2024.

[18] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[19] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *ICLR*, 2023.

[20] OpenTitan. Opentitan. https://github.com/lowRISC/opentitan, 2024.

[21] Marcelo Orenes-Vera, Margaret Martonosi, and David Wentzlaff. Using llms to facilitate formal verification of rtl. *ArXiv*, abs/2309.09437, 2023.

[22] Hammond A. Pearce, Benjamin Tan, and Ramesh Karri. DAVE: Deriving Automatically Verilog from English. *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, pages 27–32, 2020.

[23] Chuyue Sun, Christopher Hahn, and Caroline Trippel. Towards Improving Verification Productivity with Circuit-Aware Translation of Natural Language to SystemVerilog Assertions. In *First International Workshop on Deep Learning-aided Verification*, 2023.

[24] Shailja Thakur, Baleegh Ahmad, Hammond A. Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ArXiv*, abs/2308.00708, 2023.

[25] Kiran Thorat, Jiahui Zhao, Yaotian Liu, Hongwu Peng, Xi Xie, Bin Lei, Jeff Zhang, and Caiwen Ding. Advanced Large Language Model (LLM)-Driven Verilog Development: Enhancing Power, Performance, and Area Optimization in Code Synthesis. *ArXiv*, abs/2312.01022, 2023.

[26] YunDa Tsai, Mingjie Liu, and Haoxing Ren. RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models. *ArXiv*, abs/2311.16543, 2023.

[27] Junchen Zhao and Ian G. Harris. Automatic assertion generation from natural language specifications using subtree analysis. *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 598–601, 2019.