Large Language Models and Hardware Design

August 16, 2024

Cynthia Shao
and Jiming Chen
Cornell University,
Computer Systems Laboratory
Research Assistant

Jordan Dotzel
Cornell University,
Computer Systems Laboratory
PhD Candidate

Zhiru Zhang
Associate Professor
Computer Systems Laboratory
School of Electrical and Computer Engineering
College of Engineering
Cornell University

Funding Provided by ELI Undergraduate Research Grant

Abstract—**Large-language models (LLMs) like GPT-4 have become indispensable tools for software development and can currently suggest software designs, produce high-quality, create test cases, and iteratively debug programs. Yet, due to a lack of open-source design and additional performance complexities, these tools have not been as explored within hardware design. This project expands the use of LLMs to hardware design to assist students and engineers throughout the process. This model should be able to produce hardware-design language (e.g. Verilog), debug common issues, and iteratively suggest coding improvements.**

Currently, LLMs are benchmarked on their ability to complete very simple designs [1]. We gauge the ability of LLMs, such as GPT-4o, Claude 3.5 Sonnet, and Gemini, to complete the labs from ECE 2300, Cornell University's course on digital logic and computer organization, and find that in most cases, multiple turns are required with the user needing to debug the code and provide feedback to the LLM. This indicates that LLMs do not exhibit the ability to reason well about hardware designs of at least moderate complexity, such as finite state machines and working across multiple modules.

In addition to testing existing closed-source LLMs, we explore fine-tuning models. Fine-tuning GPT-3.5 Turbo and GPt-4o mini on a filtered version of an existing dataset [2] benchmarked worse than the frontier GPT-4 and GPT-4o models, which are not available for fine-tuning. To achieve better results, we are creating a larger dataset which would be fine-tuned on an open-source model, such as Llama 3.

1. Introduction

1.1. Background

The rapid advancement of large-language models (LLMs) like GPT-4 has revolutionized various fields, particularly in software development. These models, based on deep learning techniques, are capable of understanding and generating human-like text by processing vast amounts of data. Trained on diverse datasets that include code, documentation, and natural language, LLMs can suggest software designs, generate high-quality code, create test cases, and iteratively debug programs, making them invaluable tools for developers. However, despite their success in software, the application of LLMs in hardware design has not been as extensively explored. This is primarily due to the lack of open-source models tailored for hardware design and the unique challenges posed by the complexities of hardware design tasks.

Understanding how LLMs work provides insight into both their potential and their current limitations in hardware design. LLMs are trained on massive datasets, which are crucial for their performance. These datasets typically consist of a wide range of text and code, enabling the models to learn the patterns, structures, and nuances of language and programming. The quality and diversity of the training data directly impact the model's ability to generalize to new tasks. In the context of hardware design, where specific knowledge of hardware description languages (HDLs) like Verilog or VHDL is required, the availability of specialized datasets becomes even more critical. Without extensive and high-quality datasets in this domain, LLMs may struggle to produce accurate and effective results.

The superiority of closed-source models like GPT-4 in many tasks, including software development, stems from their access to vast, high-quality datasets and extensive resources dedicated to their training. These models are typically trained on data that is not publicly available, which includes proprietary information and large-scale curated datasets. This enables closed-source models to perform better than open-source alternatives. However, this also makes them more expensive to train and deploy. Training an LLM involves running computations across thousands of GPUs over weeks or months, costing millions of dollars. Inference—using the model to generate outputs—is also resource-intensive, requiring significant computational power, especially for large models like GPT-4.

Fine-tuning is a process where an existing LLM, pretrained on a broad dataset, is further trained on a smaller, more specialized dataset to adapt it to a specific task or domain. For example, a general-purpose LLM might be fine-tuned on a dataset of HDL code to improve its performance in hardware design tasks. Fine-tuning allows the model to learn from more specific examples, making it more effective in the target domain. However, fine-tuning closed-source models is often not feasible due to the lack of access to the model's internal parameters. As a result, open-source models are more attractive for tasks requiring domain-specific expertise, even if they currently lag behind closed-source models in general performance.

Hardware design involves the creation and validation of digital circuits and systems, often using HDLs. Unlike software, which is primarily sequential and linear, hardware design requires a deep understanding of parallelism, timing constraints, and the physical layout of circuits. This adds layers of complexity that LLMs, which have been predominantly trained on textual and sequential data, may not naturally handle. Early attempts to apply LLMs to hardware design have primarily focused on simple tasks, such as completing small, isolated design problems. These models often struggle with more complex tasks, such as reasoning across multiple modules or handling intricate designs like finite state machines.

1.2. Objective

This project seeks to bridge the gap between the capabilities of LLMs in software development and their application in hardware design. The primary objective is to expand the use of LLMs to effectively support the hardware design process, making these models valuable tools for both students learning hardware design and professionals working in the field. Specifically, the project aims to create models that can not only produce HDL code but also debug and iteratively improve it, thereby streamlining the hardware design process.

Before these advancements can be made, it is essential to establish a baseline understanding of the current capabilities of existing LLMs in the context of hardware design. To achieve this, the project involves evaluating the performance of various LLMs, such as GPT-4o, Claude 3.5 Sonnet, and Gemini, by testing them on labs from an introductory hardware design course—ECE 2300 at Cornell University. These labs cover fundamental concepts in digital logic and computer organization, providing a practical measure of how well these models can handle real-world tasks.

By assessing the ability of LLMs to complete these labs, the project aims to identify specific areas where they struggle, such as reasoning about complex designs like finite state machines or working across multiple modules. Understanding these limitations will inform the subsequent steps in the project, which include developing and fine-tuning open-source LLMs specifically for hardware design.

In addition to evaluating current models, the project explores the creation of larger and more specialized datasets to fine-tune open-source models like Llama 3. The ultimate goal is to enhance the ability of LLMs to handle the complexities inherent in hardware design, making them as indispensable in hardware engineering as they have become in software development. Through this dual approach—benchmarking existing models and advancing their capabilities—the project aims to enable more efficient and effective design processes across the industry.

1.3. Summary of Report and Results

In this report, we highlight the two major parts of our research. First, we describe how we tested LLMs' abilities to complete the labs for Cornell's ECE 2300: Digital Logic and Computer Organization. We show that for simple designs, they were able to produce functional code. However, for longer, more complex designs, multiple turns were needed, with a lot of the design coming from debugging on the user end. Second, we describe benchmarking LLMs according to the VerilogEval benchmark and benchmarking fine-tuned models. We show that frontier-level closed source models outperform smaller models fine-tuned on small datasets but will continue to find datasets that benchmark better.

2. Materials and Methods

2.1. ECE 2300 Labs

As an assessment of the performance of the current close-sourced models, we first focused on evaluating ChatGPT, Claude 3.5 Sonnet, and Gemini. Below is the structure we proposed to evaluate the Cornell ECE 2300 labs.

We took advantage of ChatGPT's multimodal input in the user ChatGPT-4o interface.

Prompt: I have provided a lab handout, a directory of Verilog files to be completed, and a prelab handout mentioned in the original lab handout. Below are the [file names provided in lab structure]. Please complete [first file name]. Specifically, this involves completing [file modules within this first file]. You may create files if needed. Make sure variable names are consistent with the provided information and files. Add comments to explain your reasoning.

Each time the model produced an incorrect design or code, we identified what was incorrect and tried to lead the model to correcting the problem. For example, consider the following interaction with GPT-4o in which the LLM produces partially incorrect code and is told to fix it.

Assistant:

```
// tffp.v - T Flip-Flop with Enable and
     Preset
module tffp(
   input wire CLK,
   input wire CLR,
   input wire T,
   output reg Q
);

always @(posedge CLK) begin
   if (CLR) begin
      Q <= 1’b0;
   end else begin
      if (T) begin
         Q <= ~Q;
      end
   end
end

endmodule
```

User:

Make sure that the inputs and outputs of tffp.v are CLK, RESET, T, and Q. This is in tutorial B.

Assistant:

```
// tffp.v - T Flip-Flop with Enable and
     Preset
module tffp(
   input wire CLK,
   input wire RESET,
   input wire T,
   output reg Q
);

always @(posedge CLK) begin
   if (RESET) begin
      Q <= 1’b0;
   end else begin
```

```
      if (T) begin
         Q <= ~Q;
      end
   end
end

endmodule
```

Figure 1. An interaction between a user and GPT-4o. Completing the labs involved multiple turns. Note that the assistant (LLM) responses are abridged to only show the relevant code, as there was accompanying text and other modules as well.

In this fashion, a user could guide GPT-4o to produce correct Verilog code. Since the user response depended on what the LLM produced, we measured the number of turns required to arrive at functional code.

2.2. Evaluation Methodology

In addition to testing if LLMs could complete ECE 2300 labs, we used an open-source, domain-specific evaluation benchmark to evaluate LLMs we had API access to, mainly OpenAI's GPT models and models we fine-tuned ourselves.

The specific benchmark we used was VerilogEval [1], which is comprised of 156 (out of 182) problems from HDLBits, a website where users can practice their hardware design with a collection of simple Verilog problems. In recent literature so far, VerilogEval has been a reliable benchmark for LLM-generated Verilog code. The modules and problem descriptions being relatively simple, flagship LLM models have been performing consistently higher than 60% accuracy.

Firstly, using OpenAI's API, we ran the following models through VerilogEval: GPT-4o, GPT-4, GPT-4 mini, GPT-3.5-turbo January, GPT-3.5-turbo November, and our fine-tuned models. These models are all the current OpenAI models open for API use.

VerilogEval doesn't include a script to get responses from the models, so we created that ourselves, using the OpenAI API and various jsonl (a formatted text file) packages in Python to make asynchronous calls to the API and fetch the responses to the detailed descriptions. The API calls are structured as follows:

We begin with a system prompt that is concatenated with a detailed prompt provided by VerilogEval.

System Prompt: You only complete chats with syntax correct Verilog code. End the Verilog module code completion with "endmodule". Do not include module, input and output definitions.

Description Prompt: (example from VerilogEval) Create 8 D flip-flops with active high asynchronous reset. The output should be reset to 0. All DFFs should be triggered by the positive edge of clk.

```
top_module(input clk, input [7:0] d,
    input areset, output reg [7:0] q);
```

We wrote a script to combine the provided VerilogEval descriptions and the provided top module code headers. We decided to provide the top module headers because the LLM has trouble sticking to the exact variable names when prompted to create code for a module, and the evaluation function of VerilogEval has no flexibility when running the LLM-produced code.

We edited the evaluation script to provide an entire csv that contained all the error messages, how many test cases the LLM produced code passed, and out of all 156 problems, and at what rate the LLM passes. In the literature, papers use a pass@k metric, where they let the LLM take multiple attempts (k attempts) and take the attempt that passes. In this way, pass@k should yield higher results. For lack of time and resources, we only produced a pass@1 result, which is something we are looking to improve in the future.

2.3. Dataset Creation

We loosely follow the guidelines of the MG-Verilog dataset and their definition of what makes a good dataset. One novel idea that we believed would alter LLM fine-tuning performance would be adding waveform and error-handling knowledge to the dataset. Waveforms are graphs engineers use to debug and view the entirety of the variable outputs and inputs, and we believe that LLMs using this knowledge may be able to generate less erroneous Verilog code. This work is in progress, and we have not yet developed a concrete method.

3. Results

In each lab, there were parts (A, B, etc.), and each part had modules that needed to be completed, e.g. tffp. Each module would be contained in a .v file which would compile with the other Verilog files in the project to produce a working project.

Lab	Module	GPT-4o	Sonnet	Gemini
Pre-Lab 2A	tffp.v	1		
	treg4bit	1		
Lab 2A	tcounter.v	1		
	lab2	1		
Lab 2B	lab2	6		
Pre-Lab 3A	address_generator	1	1	
	prandom	1	1	
	countdown	1	1	
Lab 3A	lab3	18	1	> 9
Lab 3B	lab3 (False Start)	8	> 3	
	lab3 (Mult. Rounds)	28		
Lab 4A	Test Cases	5		
	adder	1		
	shifter	1		
	logical	1		
	control	1		
	alu	2		
Lab 4B	decoder	> 13		

TABLE 1. Number of turns on labs by module. “>” indicates was not incomplete. Blank cells were not attempted.

Below are the benchmarking and evaluation metrics that we produced. To reproduce our VerilogEval benchmark results, please look at our open-source projects on GitHub.

Models	Machine	Human
	Pass@1	Pass@1
GPT-4o	70.6	60.3
GPT-4	67.1	50.6
GPT-3.5-turbo-0125	52.5	32.1
GPT-3.5-turbo-1106	56.6	29.5
GPT-3.5-turbo fine-tuning[1]	45.5	29.4
GPT-4o-mini-2024-07-18	65.0	53.2
GPT-4o-mini fine-tuning[2]	58.0	37.2

TABLE 2. VerilogEval benchmarks on various OpenAI models

4. Discussion

4.1. Labs

When completing the labs, we noticed very easy completion of short (just a few lines) designs, such as the T-flip-flop from lab 2 or the adder in lab 4. This makes sense, as the models used are likely to have been trained on many examples of such common designs. However, when it came to designs such as the finite state machines (FSMs) in lab 3, the LLMs generally struggled. While these designs are not considered super complex from a hardware perspective, they have more parts to them that the LLM likely has not seen before, which is why we had to constantly give feedback about how to correct its implementation.

Additionally, when code becomes more complex and the LLM has to reason about more things, it can tend to blatantly defy instructions, e.g. not knowing the name of a variable no matter how the user asks, requiring the user to directly tell it. This makes sense considering large language models have been known to have unfaithful lines of reasoning at times [3]. On a related note, when creating test cases for the ALU in lab 4, several turns were needed because the LLM produced incorrect test cases. This is because LLMs cannot actually perform arithmetic but rather produce outputs probabilistically.

In general, we noticed that LLMs could produce functional code when the designs were simple. For complex designs, they must be trained on many examples of such designs, which may be a difficult task since hardware design is lower level than software design. Alternatively, success may also be found through LLMs which have better reasoning skills. For instance, Claude 3.5 Sonnet performed exceptionally well in creating the 5-state FSM for lab 3 although faced struggles later on, but these results are promising regarding reasoning-based LLMs.

4.2. Fine-Tuning

From Table 2 results, the newest close-sourced models perform the best out of all models tested, with GPT-4o, GPT-4, and GPT-4o mini being the best, ranked respectively. The fine-tuned models perform worse than their parent models, which leads us to think that the OpenAI fine-tuning via default parameters may result in worse performance. This result is quite a surprise, but not out of the realm for LLM research, as recent literature has seen an increase in hallucinations in fine-tuned models [4].

Because we do not know the datasets the closed-sourced models are trained on, we may cause hallucinations by introducing new knowledge that does not corroborate existing knowledge. Using open-sourced models whose training datasets are public may solve this problem.

Further analysis and changing of parameters are needed to see if fine-tuning a model results in better performance within the close-sourced realm. The introduction of a new dataset may provide better results in fine-tuning.

If fine-tuning leads to worse performance overall, we may need to resort to training open-sourced models from scratch, which will take excessive resources and extensive research methods.

Another factor that may have worsened the fine-tuned model's performance is the limited amount of tokens. As mentioned in 1, we used the shortest 100 prompts and answers for the 3.5-turbo fine-tuned model. While the human benchmark didn't drop by too much, the machine dropped by around 11%, while the performance loss for the GPT-4o mini fine-tuned versus non-fine-tuned loss was only 7%.

5. Conclusion

We explored the powers of close-sourced models OpenAI's ChatGPT and Anthropic's Claude as a hardware chatbot assistant through multi-turn prompting. Additionally, we benchmarked both fine-tuned and close-sourced models using datasets from the literature. Recent close-sourced models performed better than expected, having no issues with Verilog syntax issues. New multi-modal prompting performed worse than plain-text context, which is to be expected since it is a relatively new feature. Our future work will be focused on creating a large Verilog dataset, featuring data that is error-free and compiles. We plan on using such a dataset to fine-tune an open-source model and achieve results on par with GPT-4o. We will also focus on seeing how waveforms affect fine-tuned model performance and overall user experience.

Acknowledgments

The authors would like to thank PhD candidate Jordan Dotzel and Professor Zhiru Zhang for their amazing guidance and advice. Additionally, the authors would like to thank the Computer Systems Laboratory and the fellow undergraduate research group led by PhD Candidates Andrew Butt and Yixiao Du and undergraduates Isabella Frank, Stanley Shen, Steven Yu, Tony Mao, and Anjelica Bian.

References

[1] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," 2023.

[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," 2024.

[3] T. Lanham, A. Chen, A. Radhakrishnan, B. Steiner, C. Denison, D. Hernandez, D. Li, E. Durmus, E. Hubinger, J. Kernion, K. Lukošiūtė, K. Nguyen, N. Cheng, N. Joseph, N. Schiefer, O. Rausch, R. Larson, S. McCandlish, S. Kundu, S. Kadavath, S. Yang, T. Henighan, T. Maxwell, T. Telleen-Lawton, T. Hume, Z. Hatfield-Dodds, J. Kaplan, J. Brauner, S. R. Bowman, and E. Perez, "Measuring faithfulness in chain-of-thought reasoning," 2023.

[4] Z. Gekhman, G. Yona, R. Aharoni, M. Eyal, A. Feder, R. Reichart, and J. Herzig, "Does fine-tuning llms on new knowledge encourage hallucinations?," 2024.