LLM-aided explanations of EDA synthesis errors

Siyu Qiu
University of New South Wales
Sydney, NSW Australia
siyu.qiu1@student.unsw.edu.au

Benjamin Tan
University of Calgary
Calgary, AB Canada
benjamin.tan1@ucalgary.ca

Hammond Pearce
University of New South Wales
Sydney, NSW Australia
hammond.pearce@unsw.edu.au

2024 IEEE LLM Aided Design Workshop (LAD) | 979-8-3503-7608-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/LAD62341.2024.10691721

***Abstract*—Training new engineers in digital design is a challenge, particularly when it comes to teaching the complex electronic design automation (EDA) tooling used in this domain. Learners will typically deploy designs in the Verilog and VHDL hardware description languages to Field Programmable Gate Arrays (FPGAs) from Altera (Intel) and Xilinx (AMD) via proprietary closed-source toolchains (Quartus Prime and Vivado, respectively). These tools are complex and difficult to use—yet, as they are the tools used in industry, they are an essential first step in this space. In this work, we examine how recent advances in artificial intelligence may be leveraged to address aspects of this challenge. Specifically, we investigate if Large Language Models (LLMs), which have demonstrated text comprehension and question-answering capabilities, can be used to generate novice-friendly explanations of compile-time synthesis error messages from Quartus Prime and Vivado. To perform this study we generate 936 error message explanations using three OpenAI LLMs over 21 different buggy code samples. These are then graded for relevance and correctness, and we find that in approximately 71% of cases the LLMs give correct & complete explanations suitable for novice learners.**

***Index Terms*—EDA, CAD, AI, LLM, Bug Explanation**

I. Introduction

With increasing demand for digital devices, there is a need for more digital design practitioners. However, existing electronic design automation (EDA) tools have a considerably steep learning curve. For example, in the FPGA design space, Altera and AMD Xilinx tools are frequently used in educational settings. These tool suites are renowned for their difficulty a nd c omplexity, p articularly f or n ew u sers. Indeed, the combination of new languages, design paradigms, software tools, and hardware requirements can leave novices feeling well and truly "stumped" [1], particularly when the software provides unhelpful messages upon reaching erroneous code (e.g., Figure 1). Likewise, educators themselves can struggle with the broad knowledge base required [2]. This is a serious challenge, especially considering the worldwide shortfall in qualified c hip d esigners (in t he U S a lone, e stimates h ave a 67,000 employee shortfall by 2030 [3]).

It is therefore desirable to see if recent advancements in artificial intelligence may be able to *assist* novice digital hardware designers and accelerate their training, be it in the classroom or perhaps as part of onboarding/professional development. In particular, large language models (LLMs) have demonstrated considerable capabilities for comprehending text and program code, enabling code generation and explanation. Given that one of the most common difficulties noted when learning to program comes from understanding and overcoming compiler error messages [4], [5] we pose the question: **Can LLMs be leveraged to explain error messages from EDA tools?**

The research in this work was supported in part by Intel Corporation and in part by Woodpecker Technologies. This work does not in any way constitute an Intel endorsement of a product or supplier.

979-8-3503-7608-1/24/$31.00 ©2024 IEEE

```
41 architecture Behavioral of top1 is
42 begin
43     process (clk, rst) begin
44         if rst = '1' then
45             data_out <= (others => '0')
46         elsif rising_edge(clk) then
47             data_out <= data_in;
48         end if;
49     end process;
50 end Behavioral;
```

(a) Snippet of buggy VHDL code

```
1 ERROR: [Synth 8-2715] syntax error near elsif [path/to/
      bug_1/rtl/top1.vhd:46]
```

(b) Vivado's corresponding error message

Fig. 1. Example unhelpful error message. It does not describe the real problem (a missing semicolon), and it links to line 46, not the fault on line 45!

In this work, we thus undertake a proof-of-concept examination, tasking a series of OpenAI LLMs with generating explanations for a series of synthesis-time (i.e. compile-time) bugs commonly encountered by novice digital designers. Our synthetic dataset contains error messages from both Intel's Altera Quartus Prime and AMD's Xilinx Vivado with both VHDL- and Verilog-based designs.

Crucially, as we aim to up-skill tool users, we desire pedagogically-useful responses (i.e., not automated program repair). The LLM should assist the user but should not outright solve the issue—per the constructivism pedagogy in computer science [6], learners should "build" knowledge rather than be simply told answers outright. Moreover, the insights from our study can also lay the foundation for other LLM-based augmentation of EDA tool feedback to improve their readability/actionability and, thus, designer productivity. Our contributions include the following:

- A new open-source dataset of 21 representative synthesis-time bugs and error messages based on the authors' experiences with teaching introductory digital hardware design.
- Using these bugs, results from the first pedagogically focused evaluation of 936 LLM-generated bug explanations, finding that ≈71% have 'good' explanations.

Open-source: All synthetic bugs and generated data is provided at https://zenodo.org/doi/10.5281/zenodo.10937409.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.

II. Background and Related Work

A. Large Language Models for hardware design

LLMs, trained over large quantities of text scraped from the Internet (including millions of open-source repositories), have demonstrated considerable cross-domain expertise in lexical tasks. While early models such as OpenAI's Codex acted primarily as a kind of "smart autocomplete," more recent training methodologies such as Reinforcement Learning with Human Feedback (RLHF) can create models more capable of following user intent [7]—meaning such models may actually "follow instructions." Currently, leading commercial LLMs in this space come from OpenAI's ChatGPT family [8], which can be "prompted" to translate and debug code and provide code explanations using natural language.

Multiple works have acknowledged the potential for LLMs to work with hardware design tasks, including for authoring hardware description language (HDL) code [9]–[12], bug fixing [13], SystemVerilog Assertion generation [14], and scripting hardware tool suites [15] among others. Some works have even explored how conversational LLMs like OpenAI's ChatGPT [8] can author whole processor designs [16].

B. LLMs for training and education

Proponents of the technology argue that LLMs, when carefully utilized, can unlock new pedagogical tools and strategies. Kasneci et al. provide a comprehensive survey in this area [17], finding that, for example, ChatGPT is already being used for educational methods such as generating tests, quizzes, and flashcards [18], [19].

Given their recency, investigations on the challenges and opportunities provided by LLMs in the education domain are ongoing [20]. Particular attention has been provided on their potential impact in "CS1" introductory courses (e.g. [21], [22]) and on quality of code explanations for novices [23]–[25]. Of particular interest to us is the potential for LLMs to aid in error message explanation, where LLMs are used to help learners overcome compile and runtime errors. Taylor et al. [26] explored this application for C, where they examined over 64,000 uses of the ChatGPT-enabled C compiler DCC by over 2,500 students. They found that over 90 % of compile-time and 75 % of run-time error messages had valid explanations.

This motivates our investigation, which explores a similar use case for hardware synthesis rather than software compilation. The DCC compiler provided considerable additional context to the ChatGPT 3.5 LLM in the form of stack traces and templated error message assistance [26]. Can we find a similar level of assistance but for hardware, simply using the context available in the typical hardware EDA tool suite?

III. Digital Design Assistance via LLM prompts

A. Overview

This study explores the effectiveness of using LLMs to improve error feedback in Vivado and Quartus. We use the methodology outlined in Figure 2. Firstly, we create a corpus of representative bugs. We use these to collect synthesis error messages, then use two different prompts to request explanations from selected LLMs. Finally, we score the responses.

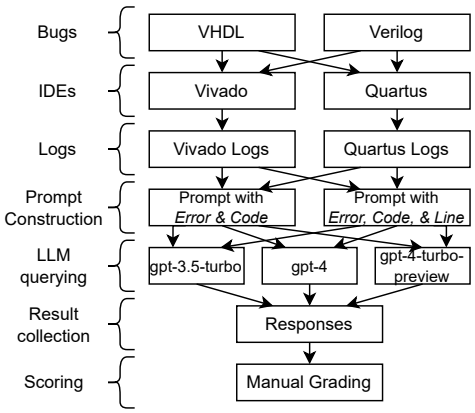

Fig. 2. Overall experimentation methodology

B. Defining a corpus of bugs

Table I's first four columns present the range of bugs used in this study (the latter four columns present results; see Section IV). The bugs were based in general on 10 distinct error categories (e.g., syntax errors, multiple driver errors, type errors and others) that the authors have frequently observed in code written by learners and novice hardware designers. Two of the VHDL bugs did not have an equivalent Verilog representation. The buggy files are short (usually less than 50 lines of comments and code)—e.g., Figure 1 depicts Bug 1.

In this work, we focus exclusively on synthesis-time bugs rather than the more complex run-time issues that could occur. The reasons for this are twofold: (1) both the Vivado and Quartus IDEs have limited capabilities in detecting run-time issues, given that they primarily rely on user-provided test-benches in simulation for this purpose. (2) Run-time issues in the novice-focused area will primarily be logic-based, which may be identified by reading simulation waveforms. A user having a simulation error thus has more information available to them than one who is stuck with an inscrutable and unchanging synthesis error message. In future, we plan to extend our study to investigate how LLMs can be leveraged for more complex debugging and training.

C. Harvesting Vivado and Quartus error log files

Quartus and Vivado can synthesize VHDL and Verilog files into bitstreams for FPGAs. When an error occurs, synthesis stops, and the error message is saved to a file—Quartus stores synthesis logs in a file like '`path/to/project/output_files/project.map.rpt`', and Vivado '`path/to/project/project_1.runs/synth_1/runme.log`'.

As these logs also include other information about the tool flow, we extract error messages using regular expressions, scanning for lines beginning with "Error:" for Quartus and "ERROR:" for Vivado. Further regular expressions can extract details about the error, such as the message, faulty file, and reported error line number.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.

TABLE I
LIST OF HDL BUGS (AND THEIR LANGUAGES) EXAMINED IN THE STUDY, WITH LLM EVALUATION RESULTS. EACH BUG HAD 24 RESPONSES IN EACH PROMPTING STRATEGY, AND THESE WERE MANUALLY GRADED.

				LLM answer: concept accurate		LLM answer: correct & complete	
Bug	**Error type**	**Language**	**Error description**	**E&C**	**EC&L**	**E&C**	**EC&L**
1	Syntax error	VHDL	Missing semicolon	71%	67%	46%	58%
2	Type error	VHDL	Can't add std_logic_vectors	100%	96%	21%	25%
3	Compilation error	VHDL	Can't write to an input ports object	100%	100%	79%	83%
4	Width mismatch	VHDL	Mismatch in the size of two std_logic_vectors	100%	100%	71%	79%
5*	Type conversion	VHDL	Can't perform two operations simultaneously in one line	100%	92%	58%	42%
6	Signal and variable	VHDL	Declaring a variable outside of a subprogram or process	100%	100%	63%	63%
7	Concurrent and sequential error	VHDL	Having both 'wait' and a sensitivity list in the same process	71%	67%	50%	38%
8	Semantic error	VHDL	Using a signal or variable that has not been declared	100%	100%	88%	88%
9	Signal Readability error	VHDL	Attempting to read from an object with the mode "out"	100%	100%	96%	96%
10*	Top Level Undefined	VHDL	Incorrect definition of the top-level module or entity	100%	100%	83%	92%
11	Case error	VHDL	Missing certain choices in a case statement	100%	100%	83%	79%
12	Singal Bit error	VHDL	Mismatch between a std_logic type and a string literal	58%	100%	29%	92%
13	Syntax error	Verilog	Missing semicolon	100%	100%	79%	92%
14	Semantic error	Verilog	Using an undeclared variable or signal	100%	100%	83%	92%
15	Wire and Reg error	Verilog	Assign a value declared as wire using non-blocking assignments	100%	100%	83%	92%
16	Blocking and non-blocking	Verilog	Mixing blocking and non-blocking assignments to the same variable	79%	71%	58%	67%
17*	Multiple Driver error	Verilog	Assigning different values to the same signal from different processes	100%	100%	67%	83%
18	Port error	Verilog	Connect a port that does not exist	100%	100%	63%	63%
19	Binary error	Verilog	Using an illegal character in a binary number representation	100%	100%	75%	100%
20	Infinite combinational loop	Verilog	Having a infinite combinational loop that cannot be resolved	100%	100%	58%	71%
21	Double-edge error	Verilog	Mismatch between operands used in condition of an always block	100%	100%	83%	83%

** Bug 5 only an error in Vivado. Bugs 10 and 17 only an error in Quartus.*

D. Prompting LLMs for error explanations

LLMs function by providing an output "response" to an input "prompt'. For this study, we selected three of OpenAI's LLMs, `gpt-3.5-turbo`, `gpt-4`, and `gpt-4-turbo-preview`. These first take a "System" prompt to provide overall model guidance, followed by "User" prompts which can contain data. As the models evolve over time, we note our usage was on March 29, 2024.

Recall that the goal of this work is for error explanations not bug repair. We therefore base our LLM prompting strategy on that used in prior work [26], which aimed for pedagogically-focused error message explanations. As depicted in Figure 4 (in the Appendix), we used one system prompt (requesting debugging assistance but no code outputs) with two similar user prompt options. "Error & Code (E&C)" is a straightforward option which provides both the error from the log file, plus the entire faulty code file. However, given that LLMs are known to be poor at word and line counting, we also examine a second prompt, "Error, Code, & Line (EC&L),", which also reproduces the tool-localised faulty code line a second time for emphasis.

Response Generation: OpenAI's LLMs are non-deterministic, potentially giving different outputs for the same inputs. However, the GPT4 models are also more expensive to run. We decided to run the `gpt-3.5-turbo` model 10 times for each bug, and the other models just once, meaning that for each bug in Table I we prompted for LLM responses 48 times (2 IDEs $\times$ (10 iterations of `gpt-3.5-turbo` and 1 each of `gpt-4` and `gpt-4-turbo-preview`)). However, as noted in Table I, during experimentation we found that certain bugs (5, 10, and 17) were errors in only one IDE; in total we collected 936 LLM responses for grading.

E. Manual grading with pedagogically focused metrics

It is difficult to automatically judge the quality of answers by a question and answer system reliably. In this work we therefore *manually* grade each of the 936 LLM-generated explanations against a series of metrics based on those used in [26]. To avoid complexity/marker subjectivity, we only grade using binary yes/no questions, and to avoid inter-rater reliability challenges all answers were graded uniformly by the first author. Our metrics follow:

- *Concept accurate*: i.e. Does the explanation link to the right concepts and keywords?
- *No inaccuracies*: i.e. Does the explanation only contain factually correct information? An explanation may be accurate even if it is incomplete. Falsehoods can lead learners astray.
- *Relevant:* i.e. Is the explanation relevant to the problem at hand? Whether the explanation is correct or incorrect does not impact the relevance assessment.
- *Correct & complete*: i.e. Does the explanation contain everything a user needs to understand and fix the error? This is the metric we use to grade overall success.
- *Solution is provided*: i.e. Did the model provide 'too much' help? From the constructivism pedagogy, we know that learners build knowledge better by 'doing' rather than by being directly 'told' [6]. To judge this category, we answer 'Yes' if an answer was provided with code that could be copied and pasted (even if the overall answer was wrong).

Figure 3 presents two example explanations from `gpt-3.5-turbo` for Bug 1 from Figure 1 (a), one judged as 'good' and one 'bad'. The 'good' correctly identifies the error, and provides a detailed (and complete) method to fix it. However, the 'bad' misidentifies the error and suggests adding a "then" keyword incorrectly. Figures 5 and 6 in the Appendix show examples of other metric scores.

IV. RESULTS

A. Top-line results:

Of the graded metrics, we consider the two most important categories 'Concept accurate' (i.e. the LLM linked to the

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.

TABLE II
AGGREGATED PEDAGOGICAL GRADES FOR GENERATED EXPLANATIONS GROUPED BY IDES, LANGUAGE, PROMPT STRATEGIES, AND LLMS

Measurement	Total	IDE Vivado	IDE Quartus	Lang. VHDL	Lang. Verilog	Prompt E&C	Prompt EC&L	GPT3.5-t. pass@10	GPT3.5-t. pass@1	GPT4 pass@1	GPT4-t.-p. pass@1
# responses (n)	936	456	480	528	408	468	468	780	78	78	78
Concept accurate	94.23%	92.32%	96.04%	92.05%	97.06%	93.80%	94.66%	93.33%	93.59%	98.72%	98.72%
No inaccuracies	91.03%	88.82%	93.12%	86.93%	96.32%	89.74%	92.31%	89.62%	87.18%	98.72%	97.44%
Relevant	84.18%	84.87%	83.54%	85.42%	82.60%	81.62%	86.75%	88.97%	83.33%	60.26%	60.26%
Correct & complete	71.26%	69.74%	72.71%	66.48%	77.45%	67.31%	75.21%	74.36%	69.23%	53.85%	57.69%
Solution is provided	3.31%	3.51%	3.13%	2.27%	4.66%	4.27%	2.35%	2.82%	2.56%	3.85%	7.69%

```
The bug is in the process statement of the VHDL code. The
    issue is with the missing semicolon after 'data_out <=
    (others => '0')' within the 'if rst = '1' then' block
    . The error message indicates that the VHDL syntax is
    expecting a semicolon before the 'elsif' keyword.
    Adding the missing semicolon will resolve the syntax
    error.
```

(a) 'Good' explanation—Concept accurate: **Yes**; Any inaccuracies: **No**; Relevant: **Yes**; Correct & complete: **Yes**, Solution is provided: **No**

```
The bug in the provided VHDL code is that there is a
    missing "then" keyword after the condition 'rst = '1''
    in the 'if' statement. This causes a syntax error
    when the 'elsif' statement is encountered because the
    previous 'if' statement is not properly closed.

To fix this bug, you should add the 'then' keyword after
    the condition 'rst = '1'' in the 'if' statement.
```

(b) 'Bad' explanation—Concept accurate: **No**; Any inaccuracies: **Yes**; Relevant: **No**; Correct & complete: **No**, Solution is provided: **No**

Fig. 3. Example of 'good' and 'bad' error explanations for Bug 1 (Figure 1) generated by `gpt-3.5-turbo`. Each bug is presented with graded metrics.

right error concepts) and 'Correct & complete' (i.e. the LLM provided everything the user needed to fix the problem). Results per-bug for these are presented in Table I's last four columns. This shows that some bugs were easier to explain than others—e.g. Bug 19 was correctly explained in 100% of cases. This is likely because certain bugs such as this one are conceptually simpler, and more likely to be issues in other languages as well, compared to say Bug 7, an error unique to VHDL and only explained correctly in 38% of cases.

Table II presents aggregated metrics across IDEs, Languages, Prompting strategies, and LLMs. We see that 'Correct & complete' can be thought of as a subset of the other metrics, i.e. it may be possible to be relevant and accurate but still not feature a complete answer. Overall, conceptually accurate explanations were observed in 94% of cases, with slight variations across different contexts. We saw only rare occasions where explanations featured outright mistakes (No inaccuracies in ≈91% of cases), and overall, the LLMs had correct and complete answers in ≈71% of cases.

IDEs: Quartus sees better explanations than Vivado, indicating that the information provided in Quartus's error messages may be of higher quality. When we performed an informal examination of this, we felt this to be true—for instance, Quartus's error message for Bug 1 (in Figure 1 (a)) includes the words 'missing semicolon', unlike Vivado.

Language differences: Interestingly, errors in Verilog seem to be better explained than those in VHDL—we theorize this could be because of the relative differences in training data available online (Verilog is more popular for open-source).

Prompting strategies: When comparing the prompting strategies, we see that prompts that include the specific error line (EC&L) tend to yield better responses (≈75%) compared to those that do not (≈67%)—i.e., providing the extra context and information to the language models appears to help.

LLMs: To fairly compare the LLMs, we tabulated just the first responses (i.e. 'pass@1') received by `gpt-3.5-turbo` alongside the responses by `gpt-4` and `gpt-4-turbo-preview`. Counter-intuitively, the smaller model (GPT-3.5) outperforms the two larger models in 'correct & complete', but the larger models are better at returning conceptually accurate responses without inaccuracies, although they have a greater tendency to over-help.

B. Discussion

LLMs generate their responses based on the data they have been trained over and on their ability to retain that data. For hardware as compared to software, there is much less training data online [10]. Still, when we compare our generated HDL explanations with the C error explanations from [26], we see that both works have ≈71% correct & complete, indicating that for this use case the data gap may not be significant.

Interestingly, our HDL explanations have a much lower incident rate of over-help ('Solution is provided') than in [26]—just 3.35% compared with their 48%. It is not immediately clear why this could be the case, as they also instructed GPT-3.5 to not emit answers directly. Perhaps OpenAI's LLMs 'understand' C better, and so inadvertently are aligned to give out too much help. If model size can be thought of as a proxy for 'intelligence', then this can also be observed with the larger model sizes in our work, where the GPT-4 models (which may 'understand' the code better) had a higher rate of over-help compared to the smaller GPT-3.5.

V. CONCLUSION

This work set out to examine if LLMs could explain the kinds of synthesis errors that novice users of EDA tools will encounter. Our findings suggest that they indeed can, with 18/21 explored errors seeing good explanations in a majority of the LLM responses and 71% of explanations being complete and correct overall. This work serves as a valuable proof of concept for LLM-powered techniques for improving the accessibility of EDA tools like Vivado and Quartus, and we believe that additional research in this area could significantly change how EDA tools are both learned and utilised by both novice and experienced engineers.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] S. A. Edwards, "Experiences teaching an FPGA-based embedded systems class," *ACM SIGBED Review*, vol. 2, no. 4, pp. 56–62, Oct. 2005. [Online]. Available: https://dl.acm.org/doi/10.1145/1121812.1121823

[2] S. Pasricha, "Embedded Systems Education in the 2020s: Challenges, Reflections, and Future Directions," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, ser. GLSVLSI '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 519–524. [Online]. Available: https://dl.acm.org/doi/10.1145/3526241.3530348

[3] M. A. Cherney, "U.S. will be short 67,000 chip workers by 2030, industry group says," *Reuters*, Jul. 2023. [Online]. Available: https://www.reuters.com/technology/us-will-be-short-67000-chip-workers-by-2030-industry-group-says-2023-07-25/

[4] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney, "Effective compiler error message enhancement for novice programming students," *Computer Science Education*, vol. 26, no. 2-3, pp. 148–175, Jul. 2016. [Online]. Available: https://doi.org/10.1080/08993408.2016.1225464

[5] I. Karvelas, A. Li, and B. A. Becker, "The Effects of Compilation Mechanisms and Error Message Presentation on Novice Programmer Behavior," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 759–765. [Online]. Available: https://dl.acm.org/doi/10.1145/3328778.3366882

[6] M. Ben-Ari, "Constructivism in Computer Science Education," *Journal of Computers in Mathematics and Science Teaching*, vol. 20, no. 1, pp. 45–73, 2001, publisher: Association for the Advancement of Computing in Education (AACE). [Online]. Available: https://www.learntechlib.org/primary/p/8505/

[7] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 27 730–27 744. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

[8] OpenAI, "ChatGPT: Optimizing Language Models for Dialogue," Nov. 2022. [Online]. Available: https://openai.com/blog/chatgpt/

[9] H. Pearce, B. Tan, and R. Karri, "DAVE: Deriving Automatically Verilog from English," in *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, Nov. 2020, pp. 27–32.

[10] 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 *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Apr. 2023, pp. 1–6, iSSN: 1558-1101. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10137086

[11] 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*, Feb. 2024, just Accepted. [Online]. Available: https://dl.acm.org/doi/10.1145/3643681

[12] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Invited Paper: VerilogEval: Evaluating Large Language Models for Verilog Code Generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Oct. 2023, pp. 1–8, iSSN: 1558-2434. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10323812

[13] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "On Hardware Security Bug Code Fixes By Prompting Large Language Models," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2024, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10462177

[14] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "(Security) Assertions by Large Language Models," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2024, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10458667

[15] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, B. Khailany, K. Kunal, X. Li, H. Liu, S. Oberman, S. Omar, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, K. Xu, and H. Ren, "ChipNeMo: Domain-Adapted LLMs for Chip Design," Nov. 2023, arXiv:2311.00176 [cs]. [Online]. Available: http://arxiv.org/abs/2311.00176

[16] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, Sep. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10299874

[17] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, and G. Kasneci, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, Apr. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1041608023000195

[18] R. Dijkstra, Z. Genç, S. Kayal, J. Kamps, and others, "Reading Comprehension Quiz Generation using Generative Pre-trained Transformers," 2022. [Online]. Available: https://e.humanities.uva.nl/publications/2022/dijk_read22.pdf

[19] E. Gabajiwala, P. Mehta, R. Singh, and R. Koshy, "Quiz Maker: Automatic Quiz Generation from Text Using NLP," in *Futuristic Trends in Networks and Computing Technologies*, ser. Lecture Notes in Electrical Engineering, P. K. Singh, S. T. Wierzchoń, J. K. Chhabra, and S. Tanwar, Eds. Singapore: Springer Nature, 2022, pp. 523–533.

[20] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, and W. Lam, "ChatGPT and Software Testing Education: Promises & Perils," in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2023, pp. 4130–4137, iSSN: 2159-4848. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10132255

[21] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, pp. 500–506. [Online]. Available: https://dl.acm.org/doi/10.1145/3545945.3569759

[22] P. Denny, V. Kumar, and N. Giacaman, "Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, pp. 1136–1142. [Online]. Available: https://dl.acm.org/doi/10.1145/3545945.3569823

[23] S. MacNeil, A. Tran, J. Leinonen, P. Denny, J. Kim, A. Hellas, S. Bernstein, and S. Sarsa, "Automatically Generating CS Learning Materials with Large Language Models," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, Mar. 2023, pp. 1176–1176. [Online]. Available: https://doi.org/10.1145/3545947.3569630

[24] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, and J. Leinonen, "Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, pp. 931–937. [Online]. Available: https://dl.acm.org/doi/10.1145/3545945.3569785

[25] S. MacNeil, A. Tran, D. Mogil, S. Bernstein, E. Ross, and Z. Huang, "Generating Diverse Code Explanations using the GPT-3 Large Language Model," in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, ser. ICER '22, vol. 2. New York, NY, USA: Association for Computing Machinery, Aug. 2022, pp. 37–39. [Online]. Available: https://dl.acm.org/doi/10.1145/3501709.3544280

[26] A. Taylor, A. Vassar, J. Renzella, and H. Pearce, "dcc –help: Transforming the Role of the Compiler by Generating Context-Aware Error Explanations with Large Language Models," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.

Computing Machinery, Mar. 2024, pp. 1314–1320. [Online]. Available: https://dl.acm.org/doi/10.1145/3626252.3630822

APPENDIX

Figure 4 shows the system and user prompt templates used to generate the error explanations in this study. Prompting was the same for all 3 OpenAI models.

```
You are a helpful assistant which debugs RTL and HDL code
    in Verilog and VHDL. Do not provide code in your
    answer. Explain what has gone wrong and why a bug is
    occurring, but do not attempt to fix the bug yourself.
```

(a) System prompt.

```
Error message: %s

Full code file:```%s```

What is the bug and why is it occurring?
```

(b) Error & Code (E&C) prompt template.

```
Error message: %s

Error line:```%s```

Full code file:```%s```

What is the bug and why is it occurring?
```

(c) Error, Code, & Line (EC&L) prompt template.

Fig. 4. LLM prompts

Figure 5 shows an explanation with 'solution is provided'.

```
The error message is indicating that there's syntax error
    near the 'elsif' keyword.

By looking at your VHDL code, the problem is that you are
    missing a semicolon at the end of the line after the
    assignment operation when the rst is '1'. In VHDL,
    every statement ends with a semicolon.

The line of the code:
```
if rst = '1' then
    data_out <= (others => '0')   --Missing semicolon
        here
```

Should be corrected with a semicolon at the end as:
```
if rst = '1' then
    data_out <= (others => '0');   --Add semicolon here
```

Your synthesizer (the tool that's processing your VHDL code
    ) is seeing the 'elsif' as being part of the previous
    line (because there was no semicolon to end it) and is
     confused as this doesn't make sense and therefore
    throws an error. This is why the error message is
    suggesting there may be a problem near the 'elsif'.

After adding the semicolon at the end of the line in
    question, the issue should be resolved.
```

Scores—Concept accurate: **Yes**; Any inaccuracies: **No**; Relevant: **Yes**;
Correct & complete: **Yes**, Solution is provided: **Yes**

Fig. 5. A 'good' error explanation for Bug 1 (Figure 1) generated by `gpt-4`. However, this explanation is flagged 'Solution is provided' as it contains a direct copy/paste solution on Lines 12-15—i.e., the explanation helps 'too much' according to constructivism in pedagogy.

Figure 6 illustrates the difference between an explanation flagged as 'relevant' versus one flagged as 'concept accurate', i.e. the two metrics do not necessarily imply each other.

```
The error message "syntax error near elsif" is indicating
    that there's a syntax issue near the "elsif" keyword
    in your code.

In this case, the error is triggered by the misuse of the "
    rising_edge" function in VHDL. The function expects a
    signal as its argument and checks whether there's a
    rising edge (low to high transition) on that signal.
    It's typically used with a clock signal inside a
    process.

However, the issue here is that the "elsif" keyword is
    being used in conjunction with the "rising_edge"
    function. In VHDL, you don't typically use "elsif"
    with "rising_edge". Instead, the usual construct would
     be "if rising_edge(clk)" directly following the
    process declaration.

It's likely that the "elsif" is intended to make the
    data_out assignment happen based on a rising edge
    event on the clock, but after a reset. The correct
    usage would be to nest "if rising_edge(clk)" inside
    the "if rst = '1'" condition, as the first to be
    processed.

Please remember that you should place the reset and clock
    conditions under separate if conditions to avoid
    ambiguities in the descriptions.
```

(a) 'Bad' explanation by `gpt-4` which is relevant but with inaccurate concepts, linking to issues with clock edges instead of the actual syntax fault

Scores—Concept accurate: **No**; Any inaccuracies: **Yes**; Relevant: **Yes**;
Correct & complete: **No**, Solution is provided: **No**

```
The bug in the code is that the 'elsif' statement is
    missing a semicolon at the end of the assignment in
    the process. This is causing a syntax error because
    the VHDL syntax expects a semicolon after each
    statement within the 'if-elsif' structure.
```

(b) 'Bad' explanation by `gpt-3.5-turbo` which has the right concepts but with irrelevant info. (the elsif has a semicolon on its later clause).

Scores—Concept accurate: **Yes**; Any inaccuracies: **Yes**; Relevant: **No**;
Correct & complete: **No**, Solution is provided: **No**

Fig. 6. Demonstrating the difference between 'relevance' and 'concept accurate' with two 'bad' error explanations for Bug 1 (Figure 1)

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on May 16,2025 at 03:35:18 UTC from IEEE Xplore. Restrictions apply.