SPICEPilot: Navigating SPICE Code Generation and Simulation with AI Guidance

Deepak Vungarala, Sakila Alam, Arnob Ghosh, Shaahin Angizi
Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
E-mail: {dv336, sa3229, arnob.ghosh, shaahin.angizi}@njit.edu

2024 IEEE International Conference on Rebooting Computing (ICRC) | 979-8-3315-4127-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRC64395.2024.10937006

***Abstract*—Large Language Models (LLMs) have shown great potential in automating code generation; however, their ability to generate accurate circuit-level SPICE code remains limited due to a lack of hardware-specific knowledge. In this paper, we analyze and identify the typical limitations of existing LLMs in SPICE code generation. To address these limitations, we present SPICEPilot—a novel Python-based dataset generated using PySpice, along with its accompanying framework. This marks a significant step forward in automating SPICE code generation across various circuit configurations. Our framework automates the creation of SPICE simulation scripts, introduces standardized benchmarking metrics to evaluate LLM's ability for circuit generation, and outlines a roadmap for integrating LLMs into the hardware design process. SPICEPilot is open-sourced under the permissive MIT license at https://github.com/ACADLab/SPICEPilot.git.**

***Index Terms*—SPICE, LLM-powered code generation, circuit design**

I. Introduction

The escalating complexity of modern software and the rapid advancements in hardware technologies present significant challenges in developing innovative circuit solutions. As software systems grow more intricate, the hardware that supports them must also evolve, leading to an intertwined cycle of increasing complexity in both domains. Traditional methods of circuit design and simulation struggle to keep pace with these developments, necessitating new approaches that can efficiently handle the growing demands [1], [2], [3], [4]. Recently, Large Language Models (LLMs) or Language Models (LMs) have demonstrated remarkable capabilities in generating Python code, offering potential avenues for automating aspects of software development. However, their application in hardware design remains limited due to a lack of inherent hardware domain knowledge. This gap poses a significant obstacle, as hardware design requires a deep understanding of architectural, micro-architectural, and logic levels as well as electronic components, circuit behaviors, and simulation processes that LLMs are not typically trained on [1], [3].

LLMs have recently shown promising solutions for generating digital and micro-architectural designs, e.g., MEV-LLM [5] proposes multi-expert LLM architecture for Verilog code generation. RTLLM [2], GPT4AIGChip [6], and SA-DS [7] enhance design efficiency, showcasing LLMs' ability to manage complex design tasks and broaden access to AI accelerator design. Nevertheless, the application of LLM in analog circuit design has been limited to a few works. To the best of our knowledge, AnalogCoder [3] is among the first Analog circuit generators.

The research presented here opens up a vast array of intriguing research questions that are yet to be explored. Our aim is to address the most pressing of these and propose a structured path for future works. Key questions include: (*RQ1*): How reliable are LLMs in the context of analog circuit design, and what are their foundational limitations in this domain? (*RQ2*): What steps are required to develop a specialized LLM tailored specifically for analog circuit design? (*RQ3*): How can the challenge of data scarcity be addressed in the niche field of analog circuit design? (*RQ4*): Are there methodologies that would enable LLMs to autonomously generate or enhance datasets needed for analog circuit design? (*RQ5*): How can LLMs be equipped with logical reasoning capabilities specific to circuit design to ensure effective interpretation and solution of hardware-specific problems? (*RQ6*): What new metrics and benchmarks should be established to accurately assess the performance, accuracy, and reliability of LLMs in executing hardware design tasks? These research questions form the basis of our investigation, with the ultimate goal of advancing LLM-driven hardware design solutions.

In this paper, we propose a novel framework that leverages the strengths of LLMs in Python code generation to assist in analog circuit design creation and simulation. By utilizing PySpice[1], a Python library for SPICE simulation, we generate a comprehensive dataset of Python-based SPICE codes that correspond to various transistor models and circuit configurations. This approach effectively bridges the gap between software and hardware domains, enabling the use of LLMs to facilitate SPICE simulations and accelerate the design process. Moreover, we introduce standardized benchmarking criteria to evaluate the performance and accuracy of the generated circuits. This standardization is crucial for comparing different designs and ensuring that the innovations meet the required specifications and industry standards. Our framework lays the groundwork for future research directions, highlighting the potential for integrating LLMs more deeply into hardware design workflows and paving the way for automated, intelligent circuit generation and optimization. This research addresses many of the key questions outlined earlier, as we explore the capabilities of LLM's in the analog domain. Noteworthy contributions include:

- We evaluate LLM performance in SPICE code generation, where both open-source and proprietary models are

[1]PySpice is an open source Python module which provides a Python interface to the Ngspice and Xyce circuit simulators.

TABLE I
COMPARISON OF THE SELECTED LLM-BASED HDL/HLS GENERATORS.

Property	Ours	[8]	[6]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[2]	[3]	[16]	[4]
Function	Analog ckt.	Verilog Gen.	AI Accel. Gen.	Verilog Gen.	Verilog Gen.	Verilog Gen.	Hardware Verf.	Hardware Verf.	Verilog Gen.	NA†	RTL Gen.	Spice Gen.	Schematic Desg.	Layout Desg.
Dataset	✓	✓(Verilog)	✗	NA	NA	NA	✗	✗	✓	✓	✓(Verilog)	✗	✗	✗
Output format	Python	Verilog	HLS	Verilog	Verilog	Verilog	Verilog	HDL	Verilog	Verilog	Verilog	Python	Schematic	Layout
Auto. Verif.	✓	✗	✗	✗	✗	✓	✓	✗	✓	✗	✓	✗	✗	✗
Human in Loop	Low	Medium	Medium	Medium	High	Low	Low	Low	Low	Low	Low	High	Low	Low
Fine tuning	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗

* A user interface featuring Prompt template generation for the input of LLM. † Not applicable.

analyzed to examine their ability to generate fundamental circuits;

- We propose a framework to overcome data scarcity by utilizing LLMs to generate open-source datasets for analog circuits. A reliable benchmarking metric is introduced to assess the performance of LLMs in hardware design;
- We present a comprehensive road map, outlining potential future advancements to further optimize LLMs for analog circuit design.

II. BACKGROUND

LLM for Hardware Design. LLMs show promise in generating Hardware Description Language (HDL) and High-Level Synthesis (HLS) code. VeriGen [8] and ChatEDA [17] refine hardware design workflows, automating the RTL to GDSII process with fine-tuned LLMs. AssertLLM incorporates three customized LLM and finally generate multiple system verilog assertions each performing different functionalities [13]. ChipGPT [9] and Autochip [11] integrate LLMs to generate and optimize hardware designs, with Autochip producing precise Verilog code through simulation feedback. MG-Verilog [1] created a hardware dataset with over 11,000 verilog code. Chip-Chat [10] demonstrates interactive LLMs like ChatGPT-4 in accelerating design space exploration. MEV-LLM [5] proposes multi-expert LLM architecture for Verilog code generation. RTLLM [2] and GPT4AIGChip [6] enhance design efficiency, showcasing LLMs' ability to manage complex design tasks and broaden access to AI accelerator design. In VerilogReader [18] the LLM accurately grasp the code logic and generate stimuli to reach the unexplored code branches. To the best of our knowledge, GPT4AIGChip [6] and SA-DS [7] are a few initial works focus on an extensive framework specifically aimed at the generation of domain-specific AI accelerator designs where SA-DS focus on creating a dataset in HLS and employ fine-tuning free methods such as single-shot and multi-shot inputs to LLM. LLMCompass [19] is able to describe and evaluate different hardware design. However, the *absence of prompt optimization, tailored datasets, model fine-tuning, and LLM hallucination* pose a barrier to fully harnessing the potential of LLMs in such frameworks [17], [7]. This limitation confines their application to standard LLMs without fine-tuning or In-Context Learning (ICL) [17], which are among the most promising methods for optimizing LLMs [20]. AnalogCoder [3] to our knowledge is among the first Analog circuit generator and generated the circuit through prompt engineering ICL. AmpAgent [16] is designed for multi-stage amplifier schematic design as well as process and performance porting.

SPICE. SPICE is a computer-based tool widely used by engineers for simulating and modeling electronic circuits. By performing mathematical analysis, it allows for the prediction of circuit behavior. SPICE can simulate a variety of components, from basic passive elements like resistors and capacitors to more advanced semiconductor devices like MOSFETs, making it essential for circuit design and optimization. Components are defined by their names, the nodes they are connected to, and their values, including resistors (R), capacitors (C), inductors (L), and transistors (M for MOSFETs, Q for BJTs). The netlist describes the interconnections of these components across the circuit's nodes.

III. HOW TALL DOES LLM STAND IN SPICE CODE GENERATION?

While LLMs demonstrate exceptional performance across various generative tasks, such as question answering, language translation, and conversational agents, these applications primarily involve natural language processing, an area in which LLMs receive extensive training. In contrast, their proficiency in managing specialized languages and tasks that are less frequently encountered during pretraining, such as generating SPICE code for hardware design, remains uncertain. Therefore, to effectively employ LLMs in automating hardware design tasks like SPICE code generation, it is essential to develop a comprehensive understanding of the capabilities and limitations of state-of-the-art LLMs. This understanding can prevent both undue optimism and unwarranted pessimism regarding their application. Our evaluation aims to provide this insight, establishing a foundation for future advancements in LLM-driven automated hardware design. To achieve this objective, we conducted an in-depth exploration and identified the common limitations of existing LLMs in the context of SPICE code generation. Ultimately, by addressing the identified shortcomings, we can reevaluate the potential of LLMs for practical automation in SPICE-based hardware design.

A. Misconception of Gate Width and Length

In SPICE code simulation, the precise definition and selection of gate lengths and widths are paramount to accurate circuit simulation and analysis. However, LLMs exhibit notable misconceptions in this area, and these misconceptions have a direct impact on circuit performance. Specifically, LLMs need a more foundational understanding of critical circuit design principles, such as the 2:1 PMOS to NMOS

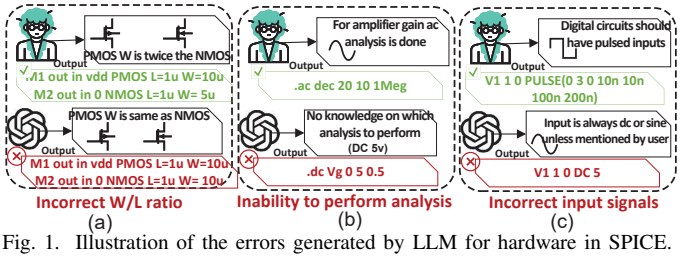

Fig. 1. Illustration of the errors generated by LLM for hardware in SPICE. (a) Incorrect W:L ratio, (b) Inability to perform circuit analysis, (c) incorrect input signals.

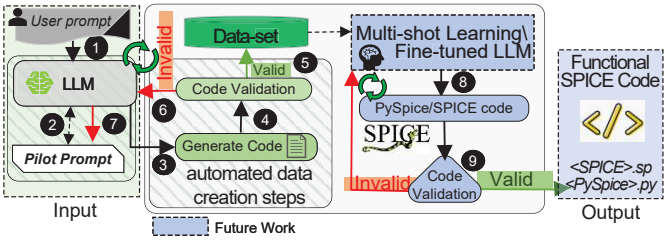

Fig. 2. SPICEPilot Framework.

gate width ratio in many standard design practices to balance the drive strength (or current-carrying capability) between the two types of transistors, as NMOS transistors generally have higher electron mobility compared to the hole mobility in PMOS transistors. Fig. 1(a) shows that a novice or expert in SPICE coding is aware of the 2:1 gate width ratio basics, and on the other hand, the LLM fails. Failure to respect design norms can result in improperly balanced circuits that deviate from intended performance characteristics. The inability of LLMs to recognize and apply these conventions suggests a gap in their comprehension of circuit domain knowledge. It concerns their efficacy in generating accurate SPICE codes for circuit designs. We reported the results of our zero-shot prompting to generate SPICE codes for basic digital circuits (i.e., inverter, NAND, NOR) in Table II analyzing LLMs' power in generating the correct W:L ratio. We observe that plain implementation of under-test LLMs fail.

B. Inability to Perform Circuit Analysis

SPICE simulations are a powerful tool for various forms of circuit analysis, such as transient, DC, and AC analysis, each tailored to reveal specific characteristics of the circuit under study. However, a significant shortfall observed with LLMs, as shown in Fig. 1(b), is their failure to autonomously select and perform the appropriate type of analysis based on the circuit's operational requirements. Effective circuit simulation often depends on the proper application of these analyses: transient analysis for time-domain response, DC analysis for steady-state behavior, and AC analysis for frequency response characteristics. LLMs have demonstrated an inability to grasp these distinctions, often leading to inappropriate or irrelevant analyses in response to user requests. This limitation highlights a deficiency in logical reasoning within the context of circuit simulation and an inadequate understanding of how these analyses contribute to evaluating circuit performance. Table II highlights that plain implementation of under-test LLMs fail to perform proper circuit analysis.

TABLE II
ANALYSIS OF LLM'S ABILITY IN SPICE GENERATION.

LLM	W/L ratio		Input signal		Analysis	
	Plain	Pilot Prompt	Plain	Pilot Prompt	Plain	Pilot Prompt
GPT-4o	✗	✓	✗	✓	✗	✓
GPT-3.5	✗	✓	✗	✓	✗	✓
Gemini-Adv	✗	✓	✗	✓	✗	✓
Claude-3.5 sonnet	✗	✓	✗	✓	✗	✓
Codestral	✗	✓	✗	✓	✗	✓
Mistral Large 2	✗	✓	✗	✓	✗	✓
Mistral Nemo	✗	✓	✗	✓	✗	✓

C. Incorrect Input Signal Assignment and Device Parameter Configuration

Accurate input signal assignment and device parameter configuration are essential for circuit testing within SPICE simulations. However, as shown in Fig. 1(c), LLMs frequently fail to assign input signals in a manner consistent with the circuit's functional requirements, such as setting correct voltage levels, signal frequencies, or waveform types. Furthermore, fundamental parameters, including the device's operating temperature, should be considered or correctly set by LLMs. The temperature, which influences carrier mobility and, consequently, the overall behavior of semiconductor devices, must be carefully controlled to ensure realistic simulation results. The oversight of such basic parameters undermines the reliability of simulation outputs and reflects a need for a more nuanced understanding of device-specific considerations essential for precise SPICE simulations.

IV. SPICEPILOT

In this section, we address *RQ3*, *RQ4*, and *RQ5* posed in the introduction. The proposed SPICEPilot framework focuses on leveraging the capabilities of LLM to generate hardware SPICE code within a Python environment using PySpice, while avoiding the costly process of fine-tuning. Given the scarcity of data, the goal is to enable the LLM to reason and creatively contribute to the development of intelligent models in circuit design. This is achieved by embedding fundamental circuit logic, reasoning, and error identification mechanisms. In the SPICEPilot framework, we introduce a computation-friendly method that leverages a reference containing step-by-step guidelines for code generation. This reference minimizes trivial errors and limits the need for extensive hardware knowledge, helping to avoid common mistakes that we discussed in Section III. The reference is used to validate every output generated by the LLM, significantly reducing errors. This approach is akin to ICL[2].

[2]Dai et al. [20] suggest that ICL can be viewed as implicit fine-tuning, where ICL produces meta-gradients through forward computation similarly to explicit fine-tuning, which updates model parameters through back-propagation.

A. Framework

Fig. 2 illustrates the initial methodology designed for data augmentation in circuit generation using LLMs. The process begins with the ❶ **User Input**, where the user provides specifications for the desired circuit. This input is then processed through the *Pilot Prompt* (❷), which integrates hardware knowledge to help the LLM mitigate common errors and offer insights into PySpice modules and coding styles. Leveraging this refined prompt, the LLM generates the corresponding PySpice code (❸). The generated code undergoes a **Validation** process (❹) to ensure the netlist correctness, where a human expert reviews and corrects trivial errors such as keyword mismatches in Python with PySpice. Successfully validated code is added to the **Dataset** (❺). If the netlist is improperly constructed, the code is deemed invalid, and additional comments detailing the errors are sent back to the LLM (❻), prompting a revision. This iterative cycle (❸→❹→❻→❸) continues until valid code is produced (❺). Concurrently, the identified errors contribute to the ongoing optimization of the *Pilot Prompt* (❼), which is updated either manually by a human expert or automatically via scripting based on the errors encountered. The valid PySpice in step ❸ also allows us to generate the SPICE netlist, which is processed to extract key parameters such as the number of transistors after code verification step ❹. From the analysis, we obtain metrics such as gain while maintaining associated metadata as data points. The curated dataset will then be effectively utilized in our future works to enhance the LLM through ICL or fine-tuning, aiming to generate more robust designs in PySpice and SPICE (step ❽). The enhanced model continues to produce code that is subjected to functional validation, adhering to the *valid* and *invalid* classifications (step ❾). In cases of invalid code generation, the model receives specific error feedback to facilitate continuous improvement. The ultimate goal of this methodology is to iteratively refine the LLM's capability to generate accurate and functional circuit designs based on user inputs, leveraging continuous validation and prompt optimization to enhance performance and reliability. Ultimately, the framework outputs the functional SPICE (.sp) and PySpice (.py) codes.

B. Dataset Generation and Benchmarking

The data points currently are stored with each point representing both the Pyspice model and its corresponding SPICE representation. This dual representation enables our dataset to be utilized to generate SPICE models using conventional methods or for rapid simulations within a Python environment. To address *RQ6*, we aim to establish a baseline evaluation for the community. Our selection criteria for benchmarking are based on the transistor count within the circuit. This benchmark consists of both Digital and Analog circuits as depicted in Table III. We classify circuits as follows: those with a transistor count of 10 or fewer are categorized as "easy"; those ranging from 11 to 25 as "medium"; circuits with 26 to 45 transistors are deemed "hard"; and circuits exceeding 45 transistors are classified as "extreme". This initial benchmarking framework

TABLE III
BENCHMARK DESCRIPTIONS INCLUDE SELECTED DIGITAL AND ANALOG CIRCUIT DESIGN TASKS. THE TASKS ARE CATEGORIZED BY DIFFICULTY LEVELS—EASY, MEDIUM, HARD, AND EXTREME USING DIFFERENT BACKGROUND COLORS FOR DISTINCTION.

ID	Name	#T	Explanation	ID	Name	#T	Explanation
5	SR Latch	8	Two CMOS NOR gates; each NOR uses 4 T	3	Operational Amplifier	30	Common-source op-amp
6	Buffer	4	Two inverters in series; each inverter uses 2 T	9	Voltage Regulator	25	with Overcurrent Protection
18	Half-Adder	12	XOR gate: 8 T, AND gate: 4 T	17	Switched-Capacitor Filter	45	20 T Op-Amp 12 T switches 8 T Ctrl and clock 5 T biasing
9	CMOS Multiplexer (4:1)	20	Transmission gates and control logic	7	Delta-Sigma Modulator	60	20 T integrator 10 T comparator 15 T DAC 10 T for clock 5 T for biasing
20	Flash Analog-to-Digital Converter (ADC)	35	For 3-bit resolution; 7 comparators at 5 T each	12	4-bit Synchronous Counter	88	4 D-flip-flops × 22 T
29	3-bit Ripple Carry Adder	60	3 full adders; each uses 20 T	15	Successive Approximation Register (SAR) ADC	80	10 T differential compar. 20 T resistor-ladder DAC 30 T SAR Logic 20 T Ctrl and clock

can be further refined and expanded by considering additional factors, such as the number of nodes, which would enhance the understanding of circuit complexity and improve overall benchmarking for the community.

Previous studies in this domain, such as [3], have discussed benchmarking with a limited set of circuits. For our benchmark, we conducted a meticulous search and selected 60 unique circuits, which is 150% larger than the Analogcoder benchmark [3], over 7.5× the number of circuits included in the ChipChat benchmark [10], and offers 250% more circuits compared to the VeriGen benchmark [8].

V. EXPERIMENTAL ANALYSIS

This section presents the SPICEPilot implementation and evaluation in two key aspects: (i) Demonstrating the framework's ability to generate results that surpass current standards, and (ii) Establishing a comprehensive benchmark for robust evaluation. The above experiments are conducted to critically evaluate and gain insights necessary for establishing more concrete standardization for the LLMs. The identified capabilities and limitations are further discussed in Section VI. We extensively evaluate the capability of LLMs in circuit design, including CodeLlama-70B-Instruct [21], Wizardcoder-33B-V1.1 [22], Llama3-70B [23], GPT-3.5 [24], and GPT-4o. CodeLlama and WizardCoder are code generation LLMs, fine-tuned on Llama2 [25] and StarCoder [26], respectively. Llama-3 is the newest open-source general LLM. WizardCoder and Llama-3 are LLMs that outperformed GPT-3.5 on the HumanEval [27] coding tasks [28]. We adopt the 'Pass@k' metric [29] (k=1, 5) as our main evaluation standard, a widely used approach in code generation tasks [21], [22]. This metric quantifies the proportion of correct generations within k independent attempts, where higher values denote better performance. We conduct n trials ($n \geq k$) and compute Pass@k using the formula $1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}$, where c denotes the number of successful attempts.

In our experiment, we utilize the setup illustrated in Fig. 2, employing Claude-3.5 Sonnet as the backbone LLM. The *Pilot prompt* is provided as an initial reference for learning. Subsequently, we prompt the LLM to generate solutions for each task in the Analogcoder Benchmark (ACB) [3]. Since

TABLE IV
PERFORMANCE COMPARISON ACROSS SPICEPILOT TO ACB FOR DIFFERENT MODELS.

	CodeLlama-70B		WizardCoder-33B		Llama3-70B		GPT3.5		GPT4		AnalogCoder		SPICEPilot (w/ Claude)	
Task Level	Pass@1	Pass@5	Pass@1	Pass@5	Pass@1	Pass@5	Pass@1	Pass@5	Pass@1	Pass@5	Pass@1	Pass@5	Pass@1	Pass@5
Easy (1-8)	9.15	36.7	21.2	50.7	75	92.5	54.6	82.2	100	100	100	100	**100**	**100**
Medium (9-13)	0	0	0	0	16.7	20	15.3	36.5	82.7	91.5	22.7	91.4	**91.7**	**94.3**
Hard (14-24)	0	0	0	0	0.3	3.03	0	0	7.9	14.2	33.9	60.3	**47.7**	**62.3**
Avg	3	12	7	16.9	30.6	38.5	23.3	39.6	63.5	68.6	52.2	83.9	**79.8**	**85.5**
# Solved	7	7	7	7	11	11	6	6	10	10	20	20	**21**	**24**

TABLE V
SPICEPILOT PERFORMANCE ON OUR ENHANCED BENCHMARK.

	SPICEPilot (w/ GPT-4o)		SPICEPilot (w/ Claude)	
Task Level	Pass@1	Pass@3	Pass@1	Pass@3
Easy (10)	100.0	100.0	90.0	100.0
Medium (10)	90.0	100.0	80.0	100.0
Hard (5)	20.0	40.0	20.0	60.0
Avg	80.0	88.0	72.0	92.0
# Solved	20	22	18	23

our framework emphasizes prompt engineering, we adapted the ACB prompts to include more detailed verbal descriptions of the circuit design, as outlined in Table III. Table IV demonstrates the superior performance of our framework, with notable improvement of 52.90% in Pass@1 scores and improvement of 1.91% at pass@5 and generating all 24 circuits in the benchmark, validating the efficacy of fine-tuning free approach with our prompt-engineered to augment the data in enhancing circuit design automation

In the second experiment, we employed two closed-source models, integrating the *Pilot prompt* to infer circuits from their internal knowledge. We randomly selected 10 circuits from our versatile benchmarking in varying levels of complexity, ranging from simple to challenging, to evaluate the performance of the LLM. The results in Table V indicate a high pass ratio, with the models successfully generating circuits with different transistor counts. The initial percentage for hard circuits in benchmarking V is low, and our observation revealed it is due to various factors, such as module definition in Python, which is not supported by Spice, and the tuning of the circuit, which can be later resolved by either using Chain-of-Thought (CoT) or simple repetitive asking. Additionally, our framework automates waveform generation in PySpice and SPICE code generation, facilitating easy validation of the circuits. The model outputs are further subjected to verification, ensuring the correct creation of circuit netlists. It is important to note that numerous parameters must be fine-tuned for functional verification of analog circuits, including the adjustment of resistors and coupling capacitors, which play a critical role in analog domain performance.

VI. DISCUSSIONS AND FUTURE WORKS

The research delves into the functionality of LLMs, revealing that while the framework generates accurate netlists in our experiments, achieving functional efficiency and meeting key parameters such as gain requires further knowledge instillation. This need arises due to the LLM's limited circuit-specific intelligence. To address this, we plan to enhance the LLM by providing it with high-definition circuit knowledge, enabling it to better integrate and resonate with detailed circuit behavior and design requirements. Figure 2 presents the proposed framework aimed at addressing the significant data bottleneck in circuit generation through automated data augmentation, and extension of its applicability. This framework offers the community an initial methodology to streamline data generation processes. The approach can be extended to leverage multi-modal LLM by incorporating circuit images as inputs. Utilizing Python packages, the framework can be enhanced to draw the drawing of circuit and waveform representations for better decoding, thereby facilitating reasoning based on visual inputs. The use of Python also allows for the implementation of class functions to construct circuits that can be integrated into more extensive design projects seamlessly. Additionally, the SPICE code generation dataset can be enhanced by incorporating detailed descriptions, similar to the methodology proposed in [1]. This enhancement assists LLMs in generating more sophisticated and accurate SPICE models, thereby advancing the capabilities of SPICE generation in LLMs.

VII. CONCLUSION

This paper presents SPICEPilot, an innovative approach to bridging the gap between software automation and hardware design in the realm of analog and digital circuits. By leveraging LLMs and PySpice, SPICEPilot automates the generation of SPICE code and introduces a reliable framework for benchmarking circuit performance. Our evaluation of both open-source and proprietary LLMs underscores the current limitations and future potential of AI-driven code generation in hardware design. Moreover, our proposed framework offers a solution to data scarcity through the generation of open-source datasets, paving the way for further advancements in the field. This work lays the foundation for future research aimed at optimizing LLMs for analog circuit applications, accelerating innovation in circuit design and automation

ACKNOWLEDGMENT

This work is supported in part by Semiconductor Research Corporation (SRC) and the National Science Foundation (NSF) under grant no. 2216772, 2228028.

REFERENCES

[1] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "MG-Verilog: Multi-grained Dataset Towards Enhanced LLM-assisted Verilog Generation," *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024.

[2] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[3] Y. Lai, S. Lee, G. Chen, S. Poddar, M. Hu, D. Z. Pan, and P. Luo, "Analogcoder: Analog circuit design via training-free code generation," *arXiv preprint arXiv:2405.14918*, 2024.

[4] B. Liu, H. Zhang, X. Gao, Z. Kong, X. Tang, Y. Lin, R. Wang, and R. Huang, "Layoutcopilot: An llm-powered multi-agent collaborative framework for interactive analog layout design," *arXiv preprint arXiv:2406.18873*, 2024.

[5] B. Nadimi and H. Zheng, "A multi-expert large language model architecture for verilog code generation," *arXiv preprint arXiv:2404.08029*, 2024.

[6] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. C. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[7] D. Vungarala, M. Nazzal, M. Morsali, C. Zhang, A. Ghosh, A. Khreishah, and S. Angizi, "Sa-ds: A dataset for large language model-driven ai accelerator design generation," *arXiv e-prints*, pp. arXiv–2404, 2024.

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

[9] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[10] 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)*. IEEE, 2023, pp. 1–6.

[11] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[12] R. Ma, Y. Yang, Z. Liu, J. Zhang, M. Li, J. Huang, and G. Luo, "Verilogreader: Llm-aided hardware test generation," *arXiv:2406.04373v1*, 2024.

[13] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv:2402.00386v1*, 2024.

[14] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," *arXiv:2309.07544v2*, 2024.

[15] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," *arXiv preprint arXiv:2407.01910*, 2024.

[16] C. Liu, W. Chen, A. Peng, Y. Du, L. Du, and J. Yang, "Ampagent: An llm-based multi-agent system for multi-stage amplifier schematic design from literature for process and performance porting," *arXiv preprint arXiv:2409.14739*, 2024.

[17] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[18] R. Ma, Y. Yang, Z. Liu, J. Zhang, M. Li, J. Huang, and G. Luo, "VerilogReader: LLM-Aided Hardware Test Generation," *School of Computer Science, Peking University; School of Information, Renmin University of China; Noah's Ark Lab, Huawei*, 2024.

[19] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1080–1096.

[20] D. Dai, Y. Sun, L. Dong, Y. Hao, S. Ma, Z. Sui, and F. Wei, "Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers," *arXiv preprint arXiv:2212.10559*, 2022.

[21] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[22] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," *arXiv preprint arXiv:2306.08568*, 2023.

[23] AI@Meta, "Llama 3 model card," 2024.

[24] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in neural information processing systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[25] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[26] R. Li, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, L. Jia, J. Chim, Q. Liu *et al.*, "Starcoder: May the source be with you!" *Transactions on Machine Learning Research (TMLR)*, 2023.

[27] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[28] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2024.

[29] S. Kulal, P. Pasupat, K. Chandra, M. Lee, O. Padon, A. Aiken, and P. S. Liang, "Spoc: Search-based pseudocode to code," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.