A New Approach Of Hardware Verification Through Natural Language Queries

Priya Aggarwal, Samuel Katapur, Parth Bhatia, Viral Sharma, Nusrat Ali

Abstract*—Large language models (LLMs) has made a huge impact in the Artificial Intelligence (AI) world. It has shown immense potential to generate high-quality content based on natural language processing (NLP). These capabilities inspires the research community to find answers to question such as "Can we harness the power of LLMs for intelligent verification and reduce the overall verification time in Electronic Design Automation (EDA)?" In order to understand it better, we conducted an in-depth investigation of application of LLMs with respect to automated EDA verification, drawing vital insights into the prospects of LLM-powered design verification. Essentially, this paper discusses: 1) recent developments, advances and empirical results on LLM-based automated verification; 2) identify gaps in the fast-growing research literature and highlight possibilities of applying LLMs in EDA verification; 3) challenges of applying LLM in EDA verification. We also discuss feasibility for researchers on using this tool responsibly, with the focus on promoting automation into EDA verification workflows. We expect that our ideas and framework can inspire advancements in automation tools for design verification using next generation LLM technology.

I. Introduction

Companies face significant pressure to bring new products to market quickly, while maintaining high quality. This demand for speed can impact the thoroughness of the verification process. Therefore, designers are looking for reducing development time which in turn majorly depend on verification time in the development of electronic device. Moreover, the completeness of verification in each development stage is critical to meet the goals for design maturity. Time for completeness of verification is increasing exponentially with the continuous evolution of electronic designs. This means that it is becoming increasingly difficult to meet the timeline.

To overcome challenges associated with verification, there is a need to automate tasks as much as possible for an optimal verification planning and strategy. With the rise in the use of large language models in various domains, researchers have encouraged to explore LLM-based automated solutions in EDA verification workflow. This paper begins with an overview of LLMs in brief, followed by a discussion on the role of large language models in EDA verification workflow. We examine contemporary trends, explore potential uses, and address the obstacles encountered in implementing this integration. Our analysis allows for a detailed understanding of the future use of LLMs in supporting verification flow. Based on the detailed observations, the paper concludes with recommendations for researchers on using this tool responsibly, with the focus on promoting automation. We summarize our contributions as follows:

- Specifically, we first perform an in-depth investigation into current capabilities and limitations for Generative AI enabled automotive verification design, thus developing our understanding on what is available now and garnering insights on how to effectively leverage current LLMs in an EDA verification pipeline. With the detailed literature investigation, we have identified few major applications adopted so far leveraging the strengths of LLMs in EDA verification. Most LLM applications primarily concentrate on activities like code creation, testing, and debugging. Detailed description of all these applications is being provided in the next section.
- This paper also present comprehensive details about open-source datasets that can be utilized to train or fine-tune LLMs in verification design flow.
- We also highlight gaps in this rapidly growing research literature. Considering the existing gaps in the literature and emerging technical opportunities, we also highlight unresolved issues and challenges that the research community can address in future studies.
- We also aim to provide a comprehensive overview of using LLMs responsibly for the future of EDA verification, evaluating whether this integration is a groundbreaking advancement or an overestimated prospect. Basically, we explore how to make an easy use of large-scale LLM models which can be used in easy hassle-free form. Ethical and practical concerns are paramount and should be considered. Issues such as data security, privacy, and the ethical use of AI in professional settings need rigorous standards and regulatory frameworks. With the increasing integration of LLMs into EDA, it is crucial to use these tools responsibly, fully aware of their limitations and potential biases.

The rest of this paper is organized as follows. Section 2 briefly presents LLMs basics. Applications of LLMs in EDA verification is described in Section 3. Publicly available EDA dataset is presented in Section 4. In Section 5 and 6, we present challenges and future directions of using LLMs in EDA. Finally, some concluding remarks are presented in

Priya Aggarwal is working as Staff, R&D Engineer, Systems Design Group, Synopsys Inc., India.

Samuel Katapur is working as Staff, R&D Engineer, Systems Design Group, Synopsys Inc., India.

Parth Bhatia is working as R&D Engineer, Systems Design Group, Synopsys Inc., India.

Viral Sharma is working as Sr. Manager, R&D Engineering, Systems Design Group, Synopsys Inc., India.

Nusrat Ali is working as Sr. Director, R&D Engineering, Systems Design Group, Synopsys Inc., India

*Corresponding author.

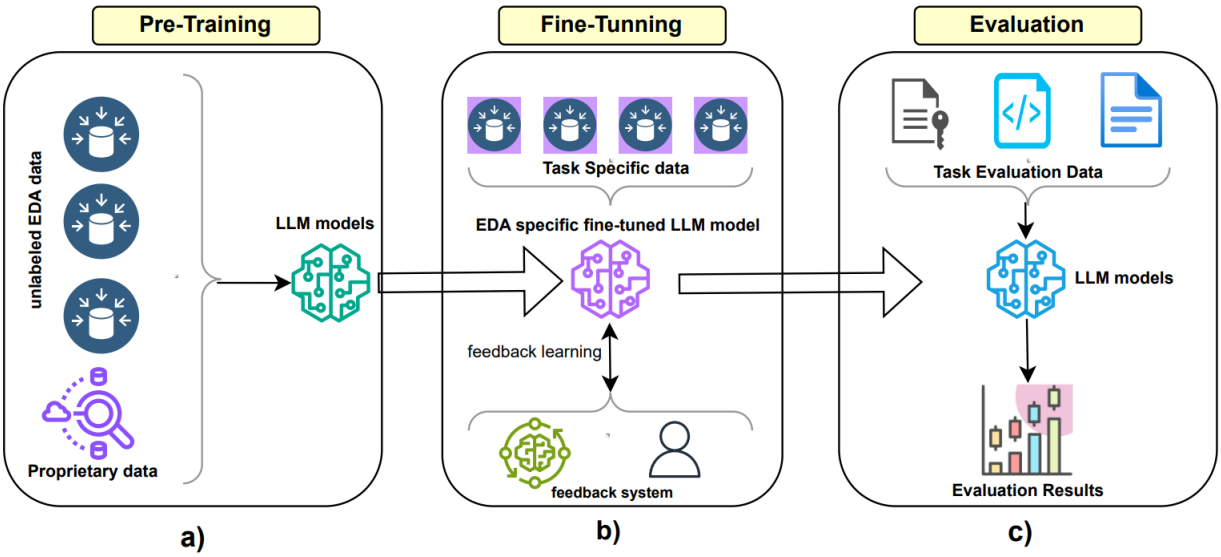

Fig. 1: The training flow for domain-adaptive pretraining and evaluation

Section 7.

II. Background

A Large language model is an AI model trained on vast datasets, enabling it to produce text that resembles human like fashion. It makes them adept at recognizing the complex text patterns and are thus able to perform various NLP tasks, such as text generation, translation, summarization, and answering questions etc. with ease. These are based on the transformer based deep neural network architecture [1]. The transformer consists of two major layers, namely – Encoder and Decoder. There can be Encoder-only models such as BERT from Google, RoBERTa from Meta, and DeBERTa from Microsoft, Encoder-Decoder models such as T5 from Google and BART from Meta and finally, Decoder-only models such as GPT (Generative Pre-trained Transformer) series developed by OpenAI in 2018, LLaMA from Meta, Claude from Anthropic, and PaLM from Google [2]. Decoder-only models are mainly used in EDA verification as it is mainly used for text generation.

LLMs are utilized in the field of EDA via three learning paradigm as show in Fig.1. More details of each of them is explained below:

- **Pre-Training (PT):** Generally, LLMs are trained on massive corpora of unlabeled domain specific data which enables us to utilize them for various tasks w.r.t. the corresponding domain. This approach is known as Domain-Adaptive Pre-Training (DAPT). For e.g. GPT-3 model is trained on 45 terabytes of text-based internet data resultant into the 175 billion parameters in the model. In February 2023, Meta introduced the open-source LLaMA, which is trained on 1.4 trillion tokens and offers various model sizes, ranging from 7 billion to 65 billion parameters.
- **Fine-tuning (FT)** involves additional training of a pre-trained LLM on specific task-related data to enhance its performance. Instruct-tuning (IT) is one such FT method which represents training pre-trained LLMs model using instruction-and-example pair data to improve their ability to handle new domain specific unseen tasks. Please note that fine-tuning on its own, without the need for pre-training, may generate exceptional LLM responses based on the provided user input prompt.
- **Evaluation (Ev)** involves evaluating LLM response using task specific evaluation dataset. Evaluation can be done via prompting LLM models which involves the process of guiding the LLM model in generating desired outputs without making any changes to the model parameters. It is done by creating specific prompts. When the prompts include a few examples or illustrations, they are referred to as few-shot. In contrast, one-shot prompts only provide one example, and zero-shot prompts do not provide any examples at all. To guide LLM more better, Retrieval-augmented generation (RAG) is one such technique that extracts queries from input prompts and retrieves relevant information from external databases [3]. This retrieved information is then used by a generative component or agent to craft responses for the language model. Additionally, the reasoning and action (ReAct) mechanism encourages LLMs to generate both reasoning processes and actions related to a task in an interwoven manner, thereby enhancing the model's dynamic problem solving capabilities [4].

III. Applications of LLMs in EDA verification

Despite Moore's Law continuing to hold true, the complexity of integrated circuits (ICs) keeps growing with more than billion of transistors connected together. Designing these circuits is a daunting task as it necessitates verifying each individual component without any fail. Concurrently, LLMs have recently made significant strides in NLP and AI, demonstrating exceptional capabilities in managing large datasets and performing a diverse array of tasks. Researchers are now

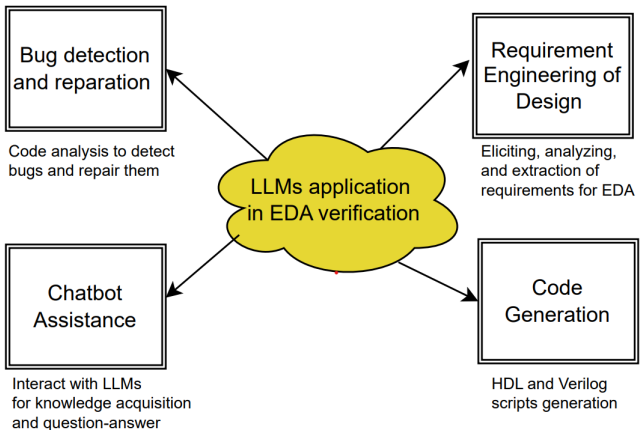

Fig. 2: Overview of selected LLMs applications for EDA

exploring the potential applications of LLMs within the EDA verification process. These can be categorized majorly into few categories as shown in Fig.2. More details about each of the categories are explained in detail below and Table I lists the most relevant state-of-the-art studies in this direction published in recent years.

Requirements Engineering of Design (REOD): Identifying potential checkers requirements and assertion requirements are an important tasks in verification cycle. There has been progress made in identification of requirements using LLM. Recently, few papers [5], [6] conducted a preliminary evaluation of ChatGPT's response on identifying natural-language requirements using open-source data sets. Although these results are only preliminary, they provide optimism that LLMs can be used as a support for efficient and effective requirements engineering. Indeed, there is thus a great opportunity to expand on this open field of research.

Likewise, LLM-assisted assertion is being generated recently in few of the works [7]–[9]. These papers primarily designed and evaluated variety of prompts for hardware assertion generation using LLM. However, there is still a need for further investigation into the level of adequacy that can be achieved through LLM-based assertion generation. Recently, Kande et al. [10] developed a pipeline to assess generated assertions by parsing them into a SystemVerilog simulator. The simulation logs are then analyzed for assertion violations and compared against a set of "golden assertions." However, the expertise of designers is required to evaluate the generated assertions accurately and determine the effectiveness of the LLM.

Code Generation (CG): Among all application areas for LLMs, code generation has been the most extensively studied in the literature. Several papers have introduced Verilog RTL code generation [11], [12] and hardware description languages (HDL) code generation [13]–[15] using LLMs. The majority of these papers involve engaging with conversational GPT-4 or GPT-3.5 by using prompts. They then refine the prompts while taking into account factors such as functional correctness, syntax checking, and evaluating the code's quality in terms of power, performance, and area optimization.

Recently, few papers fine-tuned LLM on synthetic code base datasets, demonstrating improved performance [16]–[18]. Synthetic dataset helped to evaluate generated code against benchmark dataset. Recently, Lu et al. in [12] conducted a comparative analysis of various methods for generating RTL across 29 designs using LLM. Although hallucination is a commonly recognized drawback of LLMs, it falls upon the developer to meticulously filter out any hallucinated outputs from the LLM to prevent them from entering the code base.

Bugs Detection and Reparation (BDR): Bugs' are inevitable when writing large quantities of HDL code. Addressing these issues is time-consuming; therefore, automated tools are necessary for their identification and correction. While, there has been few works in this direction utilizing LLMs [19]–[23], the state of the art is still less mature and requires further exploration.

In [19], the author presents a debugging framework that integrates the RAG and ReAct prompting mechanisms, allowing LLMs to serve as agents for interactively debugging code based on feedback. This iterative code refinement process is effective at correcting basic implementation errors but still falls short in terms of advanced reasoning and problem-solving capabilities for more complex issues.

Ahmad et al. [20] investigate the application of LLMs in the automatic repair of security bugs. Their evaluation criteria for repairs include both functionality and security improvements. The authors leverage domain-specific knowledge of bugs and provide detailed fixing instructions to formulate prompts. However, the performance of these LLM-based fixes is affected by the restricted use of generative instructions and the absence of an exhaustive functionality-versus-security assessment.

The HDLDebugger technique described in [21] highlights the shortcomings of [19] in adhering to industry standards. Consequently, HDLDebugger implements a fine-tuning process to improve debugging functionalities. This involves supervising the fine-tuning of the LLM using optimized HDL documents and RAG from search engines, supplemented with generated insights for problematic code. Nevertheless, managing complex scenarios remains difficult due to the limited capabilities of LLMs and the high expense associated with data collection.

LLM4SECHW [22] fine-tunes LLMs using a dataset specifically curated for debugging, which is derived from version control data of open-source designs and their corresponding remediation steps. This dataset tackles the shortage of functional hardware designs and facilitates the fine-tuning of LLMs to identify and fix hardware bugs. However, its modest performance suggests that an even larger dataset is necessary to enhance its effectiveness.

Pearce et al. [23] investigate the use of LLMs for bug repair in a zero-shot context, showing that black-box LLMs can generate fixes with well-crafted prompts without additional training. However, their method is currently ineffective for real-world scenarios, rendering it unreliable for critical repairs.

Chatbot Assistance (CA): LLMs have been trained on large amounts of data and is able to generate text in natural language given on any user prompt. They are predominantly

TABLE I: **Summary of state-of-art papers used for the LLM applications in EDA.**
Abbreviations: BDR- Bug Detection and Repairation, CA- Chatbot Assistance, CG- Code Generation, FT- Fine-tuning, REOD- Requirements Engineering of Design.

Reference	**Task Type**	**LLM Models**	**Training**	**Code**	**Data**
Fang et al. [7]	REOD	GPT-4 turbo	×	×	Custom (open source)
Kande et al. [10]	REOD	Code-davinci-002	×	×	Custom
Luitel et al. [6]	REOD	GPT-3.5-turbo	×	×	Public
Mali et al. [8]	REOD	GPT-4-turbo	×	✓	Custom (open source)
Sun et al. [9]	REOD	GPT-4	×	×	Private
Zhang et al. [5]	REOD	GPT-3.5-turbo	×	×	Public
Chang et al. [11]	CG	GPT-3.5	×	×	Private
Fu et al. [14]	CG	GPT-4 with LLM-friendly hardware template and demo-augmented prompt generator	✓ (FT)	✓	Public
Liu et al. [17]	CG	CodeGen	×	✓	Custom (open source)
Liu et al. [18]	CG	Mistral-7B-v0.1 & DeepSeek-Coder-6.7b	✓ (FT)	✓	Custom (open source)
Lu et al. [12]	CG	GPT-4 with self planning (new prompt engineering technique)	×	✓	Custom (open source)
Thakur et al. [16]	CG	CodeGen-16B	✓ (FT)	✓	Public
Thakur et al. [13]	CG	GPT-4, GPT3.5, Claude 2, and Code Llama with feedback loop	×	✓	Private
Wan et al. [15]	CG	GPT-4	×	✓	Custom (open source)
Ahmad et al. [20]	BDR	Codex and CodeGen-16B-multi	×	×	Public
Fu et al. [22]	BDR	LLama2	✓ (FT)	×	Public
Pearce et al. [23]	BDR	GPT-2	✓ (FT)	×	Custom
Tsai et al. [19]	BDR	GPT-3.5-turbo-16k-0613 with RAG and ReAct	×	✓	Custom (open source)
Yao et al. [21]	BDR	GPT-4 with finetuning and RAG	✓ (FT)	×	Public
Blocklove et al. [24]	CA	ChatGPT-4	✓ (FT)	✓	Custom (open source)
Liu et al. [25]	CA	Llama2	×	×	Private
Wu et al. [26]	CA	Llama2	✓ (FT)	✓	Custom (opens source)

trained on general language data. For LLMs to be used in EDA, they must evolve to comprehend the specialized syntax and semantics of electronic design, a process that requires extensive domain-specific data and training. Recently, researcher has trained ChipChat [24] LLM model specifically on EDA Verilog processor design dataset. Compared to conventional ChatGPT-4, this model has the potential to exhibit impressive performance in terms of its capability to generate code of relatively high quality.

Recently in ChatEDA [26], the authors employed instruct-tuning to train the AutoMage model, which decomposes complex EDA tasks into sub-tasks and generates corresponding EDA scripts for RTL-to-layout flows. Although these scripts can be executed directly, their performance depends on the specific tools setup, making generalization challenging. AutoMage, a large language model fine-tuned with EDA expert knowledge, acts as the central processing unit of ChatEDA.

In a separate study on ChipNeMo [25], the author employed DAPT techniques to enhance the performance of LLMs in three particular applications: an engineering assistant chatbot, script generation for EDA tools, and bug summarization and analysis. The approach incorporates RAG to reduce errors and applies domain-specific fine-tuning to boost accuracy. The resulting DAPT models have demonstrated superior performance compared to standard LLMs in domain-specific benchmarks and expert evaluations.

TABLE II: **Public Imaging Datasets used for LLM applications into EDA**
Abbreviations: BDR- Bug Detection and Repairation, CA- Chatbot Assistance, CG- Code Generation, REOD- Requirements Engineering of Design.

Reference	Task Type	Links
Fang (2024) [7]	REOD (Assertion Generation)	https://github.com/hkust-zhiyao/AssertLLM
Mali (2024) [8]	REOD (Assertion Generation)	https://github.com/karthikmaddala/ChIRAAG
Dehaerne (2023) [27]	CG	https://github.com/99EnriqueD/verilog_autocompletion
Lu (2024) [12]	CG	https://github.com/hkust-zhiyao/RTLLM
Liu (2024) [18]	CG	https://github.com/hkust-zhiyao/RTL-Coder
Liu (2024) [17]	CG	https://github.com/NVlabs/verilog-eval
Thakur (2023) [16]	CG	https://github.com/shailja-thakur/VGen
Thakur (2023) [13]	CG	https://zenodo.org/records/10160723
Wan (2024) [15]	CG & BDR	https://github.com/UIUC-ChenLab/Chrysalis-HLS
Blocklove (2023) [24]	CA	https://zenodo.org/records/7953725
Wu (2024) [28]	CG & CA	https://github.com/OpenROAD-Assistant/EDA-Corpus
Ahmad (2023) [20]	BDR	https://github.com/hammad-a/verilog_repair
Tsai (2023) [19]	BDR	https://github.com/NVlabs/RTLFixer
Wu (2024) [26]	CA	https://github.com/wuhy68/ChatEDAv1

IV. Publicly available EDA datasets for LLM applications

In all, about 14 public datasets have been referred to and used by researchers to validate the LLM performance on EDA tasks. The details are listed in Table II. Some of these datasets belong to requirements engineering of design, bug detection and chatbot assistance, while majority of them belong to code generation. The reason for this is using the latter type of datasets are more easy to find from internet sources. We briefly discuss some of these datasets and provide the download links for the dataset in Table II for ease of the readers and further research.

Requirements Engineering of Design (REOD): We have included two datasets for requirement engineering of design. Both of these dataset includes assertions generation. Dataset provided by Fang (2024) [7] consists of complete natural language specifications for the 20 designs along with their functionally correct RTL designs implemented according to the specifications. This dataset facilitated the demonstration that the generated assertions are accurate both in syntax and functionality, aiding in the verification of any proposed algorithm. In another work by Mali (2024) [8], authors open sourced six design specification from OpenTitan [1] and related files for each design. Primarily, unstructured design specifications are converted into a structured format, and their corresponding SystemVerilog assertion files are made open-source for assessing LLM responses.

Code Generation (CG): For code generation task, there are number of open-source dataset available for fruitful research. By providing these dataset, authors aim to facilitate LLM-focused research within the code generation in EDA domain. The authors in [12] have created a repository that houses 29 distinct designs. Each design package comprises three key components: a design description file (which provides a natural language explanation of the design), a testbench file (containing the testbench code used for simulating and testing the design), and a designer RTL file (featuring the Verilog code that has been verified for functional correctness). Additionally, the repository includes the Verilog code outputs generated by LLM for each design, which are also made available as open-source.

The authors in [28] have released an open-source dataset comprising 1,000 samples, each of which includes question prompts with answers and code prompts along with their corresponding Verilog scripts. Meanwhile, the authors in [27] provided two separate datasets: one focused on creating and curating Verilog code, and another designed for training and evaluating models that have been trained on Verilog.

In the paper [18], GPT is employed to generate a dataset comprising 27,000 samples, where each sample includes a pair consisting of a design description instruction and the corresponding reference code. Their dataset generation process involves three stages: 1) preparing RTL domain keywords, 2) generating instructions, and 3) generating reference code. To ensure GPT produces the desired outputs at each stage, authors also designed several general prompt templates. The dataset in [17] contains 156 problems sourced from the Verilog instructional website HDLBits. Meanwile, the authors in [29], collected Verilog datasets from GitHub and Verilog textbooks and made it open-source.

The Chrysalis dataset [15] is an extensive compilation tailored for high-level synthesis. It distinctively encompasses both correct source codes and intentionally injected buggy codes, making it an essential resource for training and en-

[1]https://opentitan.org/

hancing LLMs in code debugging. This dataset originates from more than 1000 function-level designs, gathered from 12 diverse open-source HLS benchmark suites. Each design undergoes a systematic injection of up to 40 different combinations of bugs, greatly broadening the range of HLS error models currently accessible.

Bug Detection and Repairation (BDR): Unlike code generation dataset, we could find fewer repositories for bug's detection in code and repair them. We have included two datasets for this task. The dataset compiled by Ahmad (2023) [20] originates from three distinct sources: CWE descriptions available on the MITRE website[2], the OpenTitan System on Chip[3], and the Hack@DAC 2021 SoC[4]. This dataset serves as an excellent benchmark for investigating the potential of using LLMs to address hardware security bugs. Meanwhile, dataset in [19] includes a total of 212 samples, encompassing both flawed (containing syntax errors) and original code implementations.

Chatbot Assistance (CA): For chatbot assistance task, ChatEDA benchmark [26] is one of the dataset comprising of 50 distinct tasks to evaluate the performance of LLMs in automating the EDA flow. In a different study [24], the authors introduced a dataset that investigates the open-ended process of exploring potential applications of Large Language Models (LLMs) in hardware design. The objective in this task was to generate Verilog code for a complete, although small, processor design.

V. Unique challenges in LLM applications to EDA

In previous sections, we discussed several works on LLM applications to EDA. While the LLM's performance appears promising, some issues need to be resolved. We now present a discussion on some of the challenges and gaps in this area. Addressing these challenges requires a concerted effort to combine advancements in AI with deep domain expertise in EDA, ensuring that the resulting solutions are both technically robust and practically viable.

A. Data Privacy and Security

Fine-tuning pretrained LLMs with domain-specific data indeed produce high-quality LLM responses. However, EDA projects often involve proprietary and sensitive information. Thus, to achieve high-quality LLM responses, vendors may have to fine-tune these models using valuable IP designs. However, ensuring that LLMs can be used in a secure manner that protects intellectual property (IP) and complies with industry regulations is a significant challenge. Moreover, The field of EDA is continuously evolving with advancements in semiconductor technology, new design paradigms, and emerging methodologies. LLMs need to be continuously updated to keep pace with these changes.

[2]https://cwe.mitre.org/data/definitions/1194.html
[3]https://opentitan.org/book/hw/
[4]https://hackatevent.org/hackdac21/

B. LLMs are compute intensive

The extensive size of LLMs, which can include billions or trillions of parameters, requires considerable computational power and memory resources for both training and inference. This not only increases energy usage but also presents a significant challenge for time-critical or real-time applications that utilize LLMs. Given high computation extensive, steps to practical adaptation of LLMs in EDA is still unclear. Moreover, the EDA domain relies heavily on a suite of specialized software tools. Integrating LLMs into these existing workflows without causing disruptions or requiring extensive engineering is a non-trivial task.

C. Precision and Accuracy

EDA tasks often demand a high degree of precision and accuracy. Mistakes can lead to costly errors in hardware design, making it critical for any AI-driven solution to meet stringent correctness standards. It requires deep expertise and a nuanced understanding of both hardware and software constraints, which can be difficult for LLMs to grasp-fully.

D. Customization and Adaptability

Different projects may have unique requirements and constraints. Developing LLMs that are adaptable and can be customized to specific needs without extensive retraining is a challenge. Moreover, EDA is a highly technical field that involves the design and verification of complex electronic systems. Understanding the intricate details of digital design, algorithms, computer architecture, and related technical concepts can be challenging for someone without a strong background in electrical engineering or computer science.

E. LLMs hallucinates

Since ChatGPT was released, LLMs have drawn much attention from both academia and industry. The wide applications of LLMs have led to a notable increase in content on the Internet that is not authored by humans but rather generated by these language models. However, as LLMs may hallucinate and generate non-factual texts, the increasing number of LLM-generated contents also brings worries that these contents may provide fictitious information for users across EDA. It is challenging but necessary to consider how to build EDA LLM systems free from hallucinations. Additionally, due to the nature of an LLM being a probabilistic inference engine, it is likely that it will generate similar responses for the same prompt across multiple instances. While utilizing a conversational LLM for hardware device design and implementation can have advantages, it is evident that there is still room for improvement in this technology. The ChatGPT LLM made errors in both the specification and implementation, necessitating intervention from an experienced hardware designer. This suggests that the model would unlikely be able to generate designs without assistance, particularly in a zero-shot setting.

Moreover, we identified shortcomings when attempting to utilize the model for generating verification code.

F. Prompting Limitations

Many AI models, such as GPT-3 and GPT-4 by OpenAI, have a limited context window. This means they can only consider a certain number of tokens (words, punctuation, etc.) at a time when generating responses. For example, GPT-3 has a maximum context length of around 2048 tokens. Larger designs, particularly in fields like software architecture, product design, or detailed project planning, often require more extensive descriptions, specifications, and discussions than can fit within this limit. This can lead to incomplete or inaccurate outputs because the model might miss out on critical details.

VI. Opportunities and scope for future work

Based on the literature review presented above, we provide some suggestions for future researchers. Some of these recommendations are evident from the preceding discussion, while others pertain to the current situations in the LLM era. when a human is paired with ChatGPT-4, the language model seems to be a 'force multiplier', allowing for rapid design space exploration and iteration. In our case study, we illustrated how it contributed to the design and implementation of an innovative processor. Overall, we found that ChatGPT4 is capable of generating functionally accurate code. This ability can significantly reduce the time designers spend on creating standard modules, thereby enhancing developer efficiency. Future research could encompass a more extensive user study to explore this potential further and the creation of conversational LLMs tailored to hardware design to enhance the outcomes.

A. Availability of Structured and Authentic Dataset

During the study of literature, it is observed that a one-to-one performance comparison between two reference papers cannot be undertaken due to lack of uniformity in the datasets and the performance metrics used. This necessitates the creation of one or more big, authentic, publicly available quality datasets that can be used, compared, or evaluated against by future researchers. For ease of research, we are presenting the ten most popularly used datasets in Table II. It is worth noting that the current public datasets have a limited number of datasets for the fine-tuning and testing of LLMs.

B. Need for ablation study

Ablation studies in the context of LLM applications in EDA involve systematically removing or modifying certain components or features of a language model to assess their impact on its performance. The ablation studies are important for understanding the contribution of individual components By removing or modifying specific components of a language model, ablation studies can help identify which components are most important for its performance. This can provide insights into how the model processes and understands language, which can inform the development of more effective language models in the future. Ablation study is also useful for Evaluating the effectiveness of different architectures. It can be used to compare the performance of different language model architectures. By systematically removing or modifying components of each architecture, researchers can evaluate their relative effectiveness and identify areas for improvement.

C. Standardizing Specification Writing

The quality of the generated response hinges on the input specification and the performance of the LLMs. One of the strengths of LLMs is their ability to pinpoint areas in natural language specifications that lack adequate detail for generating assertions and tests. This can assist architects by offering more thorough information. LLMs are also capable of automatically creating detailed documentation from design files, which can be advantageous for cross-functional teams. Moreover, an LLM-based query system can enable designers to swiftly locate information within extensive repositories of design data and documentation.

D. Need for Cognitive Design Assistants

Creating more advanced cognitive assistants that not only answer questions but also anticipate design issues and provide proactive solutions. Additionally, there is potential for integrating large language models with other AI models, such as reinforcement learning, to tackle more complex electronic design automation tasks across various domains.

E. Optimization

There remains significant potential for enhancing the performance of LLMs, particularly in striking a balance between security and functionality. One strategy involves treating generative design as a search space that prioritizes functional correctness. Furthermore, when selecting and integrating modules derived from this space, it is crucial to also account for security considerations.

VII. Conclusion

This study examines LLM-assisted EDA verification methodology for reducing verification time. Our analysis reveals that the promise of LLMs is undoubtedly offering new paths to design automation, bug detection, and overall quality improvements. However, the path to full commercialization is full of challenges and uncertainties. There are technical, proprietary, and practical challenges in incorporating LLMs into EDA. The future of this integration hinges on several factors, including the development of more specialized electronic design languages tailored for LLMs, which in turn requires vast amounts of training data to achieve optimal results. Additionally, addressing issues such as data security and privacy is crucial to mitigate the risk of prompt leakage vulnerabilities in LLMs. It is essential to use these tools responsibly, with a clear understanding of their limitations and potential biases. The future of EDA verification lies in developing AI-assisted verification tools that integrate into the existing verification workflow, offering interactive learning experiences and personalized feedback.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of LLMs in practice: A survey on chatGPT and beyond," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 6, apr 2024.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

[5] J. Zhang, Y. Chen, C. Liu, N. Niu, and Y. Wang, "Empirical evaluation of ChatGPT on requirements information retrieval under zero-shot setting," in *2023 International Conference on Intelligent Computing and Next Generation Networks*, 2023, pp. 1–6.

[6] D. Luitel, S. Hassani, and M. Sabetzadeh, "Improving requirements completeness: Automated assistance through large language models," *Requirements Engineering*, pp. 1–23, 2024.

[7] 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 preprint arXiv:2402.00386*, 2024.

[8] B. Mali, K. Maddala, S. Reddy, V. Gupta, C. Karfa, and R. Karri, "ChIRAAG: ChatGPT informed rapid and automated assertion generation," *arXiv preprint arXiv:2402.00093*, 2024.

[9] C. Sun, C. Hahn, and C. Trippel, "Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions," in *First International Workshop on Deep Learning-aided Verification*, 2023.

[10] 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*, 2024.

[11] 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.

[12] 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.

[13] 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.

[14] 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)*, 2023, pp. 1–9.

[15] L. J. Wan, Y. Huang, Y. Li, H. Ye, J. Wang, X. Zhang, and D. Chen, "Software/hardware co-design for llm and its application for design verification," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 435–441.

[16] 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*, 2023.

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

[18] 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," *arXiv preprint arXiv:2312.08617*, 2023.

[19] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language models," *arXiv preprint arXiv:2311.16543*, 2023.

[20] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," *arXiv preprint arXiv:2302.01215*, 2023.

[21] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[22] W. Fu, K. Yang, R. G. Dutta, X. Guo, and G. Qu, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2023, pp. 1–6.

[23] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2339–2356.

[24] 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, Sep. 2023.

[25] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[26] 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.

[27] E. Dehaerne, B. Dey, S. Halder, and S. De Gendt, "A deep learning framework for verilog autocompletion towards design and verification automation," *arXiv preprint arXiv:2304.13840*, 2023.

[28] B.-Y. Wu, U. Sharma, S. R. D. Kankipati, A. Yadav, B. K. George, S. R. Guntupalli, A. Rovinski, and V. A. Chhabria, "EDA corpus: A large language model dataset for enhanced interaction with OpenROAD," IEEE, New York, NY, June 2024.

[29] 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)*. IEEE, 2023, pp. 1–6.