INVITED: Data4AIGChip: An Automated Data Generation and Validation Flow for LLM-assisted Hardware Design

Yongan Zhang, Yonggan Fu, Zhongzhi Yu, Kevin Zhao, Cheng Wan, Chaojian Li, Yingyan (Celine) Lin
celine.lin@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, USA

Abstract

Recent breakthroughs in Large Language Models (LLMs) have spurred interest in their application to hardware design, aiming to streamline design flows and increase accessibility for non-experts. However, the effectiveness of current LLMs in hardware design is limited due to a lack of exposure to hardware-specific data during training. A significant barrier is the scarcity of high-quality, annotated hardware data available for developers to fine-tune LLMs. To address this data scarcity, we introduce a set of essential criteria and insights for the scalable generation of high-quality, annotated hardware data. Building upon these, we develop Data4AIGChip, an open-source, automated flow for hardware data generation and validation. Data4AIGChip incorporates three key innovations: (1) a novel flow for generating Pyramid of Thoughts structured data to enhance LLMs' capability in hardware code generation; (2) a Retrieval-Augmented Generation technique to improve generated data quality; and (3) an automated validation flow to ensure data integrity. We demonstrate the utility of Data4AIGChip by fine-tuning existing pre-trained LLMs with the data it generates. Our experiments show that Data4AIGChip can reliably produce high-quality data, significantly enhancing the performance of fine-tuned LLMs in hardware code generation tasks.

ACM Reference Format:
Yongan Zhang, Yonggan Fu, Zhongzhi Yu, Kevin Zhao, Cheng Wan, Chaojian Li, Yingyan (Celine) Lin. 2024. INVITED: Data4AIGChip: An Automated Data Generation and Validation Flow for LLM-assisted Hardware Design. In *Proceedings of 61st ACM/IEEE Design Automation Conference (DAC'24).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The remarkable capability of Large Language Models (LLMs) in generating high-quality content from natural language prompts has sparked growing interest in their application to hardware design [1, 4, 6–8, 14]. This interest is fueled by LLMs' potential ability to streamline design flows and enhance hardware design accessibility for non-experts. Firstly, LLMs may encapsulate the collective wisdom of various experts and hardware design communities, allowing users to tailor their design processes to specific applications and toolchain specifications through adjusted prompts. Secondly, LLMs may analyze vast data and previous design patterns to suggest optimizations, simplifying the design flow and democratizing the process for users at all levels of expertise. Thirdly, LLMs may accelerate hardware prototyping by automating design aspects, leading to faster prototype development and more dynamic, interactive environments with real-time feedback. Recognizing these promising capabilities, initiatives like *Architecture 2.0* [12] aim to transform the hardware design paradigm, significantly reducing manual design overhead and leveraging artificial intelligence to create more advanced and efficient hardware systems.

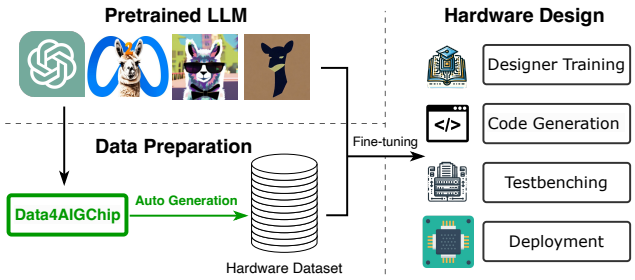


Figure 1. The proposed Data4AIGChip which is to address the data scarcity bottleneck in current LLM-assisted hardware design.

Despite the significant potential and community excitement, current state-of-the-art (SOTA) pretrained LLMs, such as OpenAI's GPT-4 [10], still struggle to produce practical hardware designs without extensive human intervention. In hardware code generation, for example, these models tend to either (1) generate non-synthesizable or non-functional code, necessitating human correction, or (2) produce overly simplistic or impractical implementations [4]. This issue primarily stems from the LLMs' limited exposure to hardware design data during pretraining. While the datasets used for pretraining LLMs are vast [15], only a small fraction comprises code, with hardware-specific code being even rarer [14]. Therefore, enriching pretrained LLMs with high-quality, hardware-specific code datasets through domain-specific fine-tuning [16] is essential for unlocking the full potential of LLM-assisted hardware design.

However, creating high-quality, hardware-specific code datasets is challenging due to the complexity and cost of acquiring labeled data that pairs hardware code implementations with natural language descriptions. The primary methods for obtaining such datasets include web scraping, manual creation, and synthetic data generation [8, 14], each with its drawbacks. Firstly, web scraping often yields limited usable data due to many online code lacking accurate descriptions, resulting in minimal viable data from open-source repositories. Secondly, manually creating descriptions for existing code or generating new code samples by human experts is time-consuming and labor-intensive, thus not scalable. Creating a single data point could take hours to days. Thirdly, while some recent studies have explored using LLMs for synthetic data generation, like auto-generating descriptions for existing code samples [8], the output is often too

DAC'24, June 23–27, 2024, San Francisco, CA

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

Table 1. Pass rates (Pass@10) of the LLaMA-2-Chat models on generating RTL code and code descriptions. Generated code is evaluated based on the standard testbench in [8] while the generated descriptions are evaluated as the ability to instruct another LLM (GPT-4 as an oracle model) to generate correct code.

	LLaMA-2 7B-Chat	LLaMA-2 13B-Chat	LLaMA-2 70B-Chat
Generated Code Pass@10 (%)	0 (Fail)	0.2	0.6
Generated Description Pass@10 (%)	30.1	31.2	47.5

high-level and simplistic for intricate hardware design. Additionally, most generated data lacks sufficient hierarchy depth, failing to leverage advancements in in-context learning for LLMs, such as the chain of thoughts approach [17].

To address the significant data scarcity issue hindering LLM-assisted hardware design, we propose Data4AIGChip, an acronym for "Data for AI (LLM) Generated Chips". As illustrated in Fig. 1, it is an automated data generation and validation flow to facilitate the fine-tuning of LLMs for accelerator code generation. Our main contributions are summarized as follows:

- We propose an essential insight for generating high-quality and scalable hardware datasets, which can effectively enhance LLM-assisted hardware code generation through model fine-tuning.
- To meet the criteria outlined above and address the challenge of data scarcity in fine-tuning LLMs for hardware accelerator design, we developed Data4AIGChip. This automated flow for hardware code data generation and validation leverages LLMs' exceptional capabilities in code description to produce high-quality datasets.
- As the first enabler of Data4AIGChip, we propose a novel flow for generating Pyramid of Thoughts (PoT) structured data, which can largely enhance the capabilities of LLM-assisted code generation, particularly in the context of complex hardware designs based on high-level user instructions.
- To further enhance the quality of the generated data, we boost the potential of the aforementioned PoT data structure by integrating Data4AIGChip with two additional enablers, including a Retrieval-Augmented Generation (RAG) technique and an automated validation flow.
- Extensive experiments show that LLMs fine-tuned with our generated dataset outperform those trained on datasets from other sources in both code implementation accuracy and the sophistication of generated hardware designs.

2 Insights for Scalable Dataset Generation

What Capabilities of LLMs Can We Leverage? A promising and scalable solution to create a dataset that enhances LLM-assisted hardware design is to incorporate LLMs themselves into the data generation pipeline. However, as observed in prior works [4, 14], existing LLMs might encounter difficulties when attempting to directly generate hardware code without any fine-tuning. Therefore, the research question is how to effectively harness the capabilities of LLMs to empower the data generation pipeline.

The key insight. In this work, the crucial insight we leverage is that existing LLMs excel at generating natural language descriptions for provided hardware code. This proficiency is essential for automating the creation of more comprehensive datasets, given the substantial amount of hardware code available on public sources without accompanying natural language descriptions.

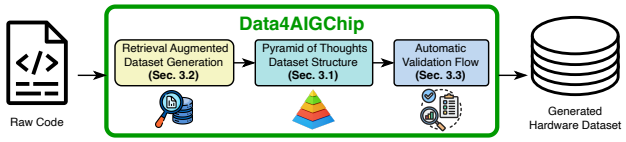


Figure 2. The overall flow of the proposed Data4AIGChip.

Validating the insight. To analyze the uneven proficiency between directly generating hardware code and generating corresponding natural language descriptions, we design a simple experiment to benchmark the two capabilities of LLMs, as elaborated in Tab. 1. Specifically, we employ the pretrained LLaMA2-Chat [15] models for this purpose. The evaluation comprises two aspects: (1) generating RTL code from descriptions using the VerilogEval benchmark [8], and (2) conversely, creating descriptions for RTL code based on the same benchmark. We assess the generated code using pre-built testbenches [8]. To evaluate the generated descriptions from LLaMA2-7B-Chat, we utilize a more advanced model GPT-4 to generate code from them. If the output code from GPT-4 passes the RTL testbenches, the description is considered successful. Our evaluation metric is pass@10 [8]. We can observe that the results presented in Tab. 1 consistently demonstrate an astonishingly higher pass rate for generated descriptions compared to code samples (up to 46.9% higher), underscoring LLMs' stronger ability to generate descriptions for hardware code than to generate the code itself.

Leveraging this insight, we can utilize LLMs to generate datasets containing both hardware code and descriptions. Subsequently, these datasets can be employed to fine-tune LLMs further, enhancing their ability to generate hardware code, and thereby facilitating LLM-assisted hardware design.

3 Data4AIGChip: An Automated Dataset Generation and Validation Flow

In this section, we introduce Data4AIGChip, an automated flow for generating and validating datasets for LLM-assisted hardware design. The flow is crafted to harness the insights in Sec. 2 to generate high-quality hardware datasets in a scalable manner.

Framework overview. Data4AIGChip comprises three primary components, as depicted in Fig. 2. Our framework takes raw code collected from public sources as input and generates high-quality hardware datasets with accurate code-description pairs as output. To achieve this objective, we develop three key enablers to complete this pipeline. Specifically, we first process the input raw code using an RAG technique. This technique accurately generates a set of natural language descriptions with varying levels of detail corresponding to the input raw code. These descriptions are then organized within the proposed PoT data generation flow, encompassing descriptions at various levels of detail. Finally, the PoT data structure undergoes an automated validation flow, filtering out low-quality code-description pairs. Further elaboration on each of these enablers is provided in the subsequent subsections.

3.1 Enabler-1: Pyramid of Thoughts Dataset Generation Flow

To harmonize design generation accuracy with ease of use, the PoT data structure is introduced. As shown in Fig. 3, it organizes hardware code descriptions at various levels of detail, ranging from high-level summaries to line-by-line comments. The PoT dataset utilizes high-level descriptions to offer users user-friendly interfaces for their queries to match with, while the paired detailed descriptions

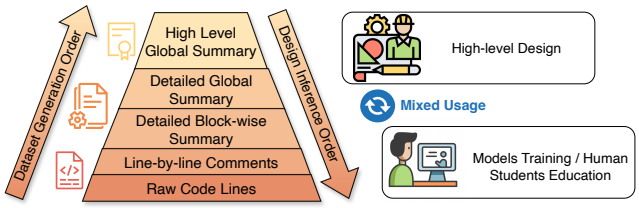


Figure 3. Illustrating the proposed PoT dataset generation flow.

and line-by-line comments can provide better supervision during either fine-tuning or in-context learning. Specifically, for the generation of PoT structured data, we develop a bottom-up approach to gradually generate natural language descriptions at varying levels of detail from the input raw code. This approach serves as a foundational version of our RAG technique introduced in Sec. 3.2. To begin, we generate detailed, line-by-line comments on the input raw code. These comments are then summarized by LLMs into a more abstract block summary, which is further consolidated into a comprehensive global summary. Lastly, we prompt LLMs to abstract a high-level global summary regarding the code sample's functionality from the detailed one.

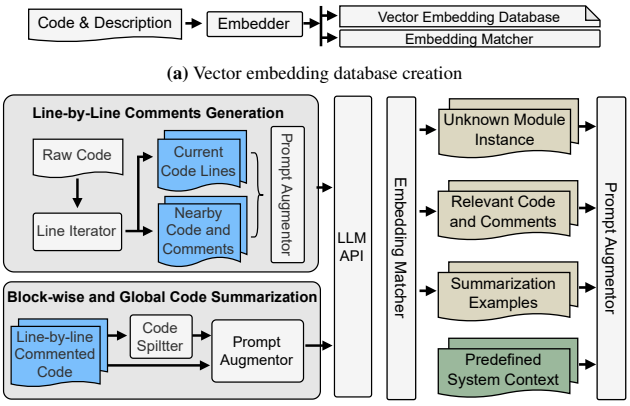
This bottom-up approach offers two significant advantages. First, it ensures comprehensive coverage across various levels of detail, which is crucial for effective LLM-assisted hardware design. Second, our approach addresses the challenge of generating high-level descriptions directly from hardware code, a task that can be problematic due to LLMs' limited ability to interpret and relate various hardware modules. More specifically, in our approach, the generation of line-by-line comments primarily entails understanding the local semantics of the code, an area where LLMs demonstrate greater proficiency. These comments act as intermediaries, translating complex code elements into the natural language domain, thus ensuring the accurate construction of high-level descriptions.

3.2 Enabler-2: Retrieval-Augmented Generation

To further enhance the accuracy of the generated descriptions at various levels of detail and leveraging the insights in Sec. 2, it is essential to supply LLMs with ample contextual information. To do so, as depicted in Fig. 4, we introduce the RAG technique to enhance the capabilities of LLMs for the PoT creation process, as described in Sec. 3.1, by providing few-shot examples as additional prompts.

RAG-aided dataset generation pipeline. Initially, all code samples, along with any existing descriptions, are converted into vector embeddings, following [5], which are stored in a database. As new descriptions are generated, they are also added to this database. Next, in the generation process for each code sample, we conduct retrieval from the database, and the retrieved codes and descriptions serve as few-shot examples appended to the prompts for LLMs. These are then used to generate each level of the PoT structures, following the creation process detailed in Sec. 3.1. Specifically, retrieval is based on the similarity between the vector embeddings of the new content and those of the existing database entries.

The advantage of our RAG-based approach is to ensure that the LLM employed in PoT structured dataset generation is well-informed, thereby resulting in more precise and contextually accurate generation of code descriptions. This approach is particularly useful (1) when generating line-by-line comments for code with unknown module instantiations and (2) when summarizing code blocks with similar descriptions stored in the database.



(a) Vector embedding database creation

(b) PoT aligned description generation **(c)** Code and description retrieval

Figure 4. Illustrating the overall flow of the proposed Retrieval-Augmented Generation.

3.3 Enabler-3: The Automated Validation Flow

To ensure the quality of our generated dataset, we implement an automated validation flow to filter out low-quality data. Specifically, this flow utilizes the generated descriptions to recreate the original code using another LLM (e.g., GPT-4), as adapted from [8]. We consider the code-description pair as valid as long as the regenerated code successfully compiles within a certain number of generation trials using any level of the descriptions. In cases where the code-description pair fails this criterion, it is filtered out from the dataset.

4 Experimental Results

4.1 Experiment Setup

Dataset generation. Raw RTL code is sourced and preprocessed from BigQuery, in line with [8, 14]. The primary model for generating descriptions is LLaMA2-70B-Chat. GPT-3.5-turbo serves as an automated backup for scenarios where the maximum token limit is exceeded. We perform ten completion trials for each description, aiming to include as many valid descriptions as possible while avoiding excessive time on invalid entries. This process produces ~11k valid code-description pairs, as detailed in Tab. 2.

Fine-tuning and inference. CodeLLaMA-7B-Instruct is chosen as the primary model for hardware code generation for its superior coding performance and small model size. The fine-tuning approach is based on QLoRA [3], using its default training settings to demonstrate our delivered dataset's effectiveness. The fine-tuned model is evaluated using 143 Verilog coding questions from the benchmark in [8], excluded from the training set.

Hardware evaluation and metrics. The validity of each generated design is tested by compiling it and checking against its RTL simulation results in pre-defined testbench cases. We employ unbiased pass@1, pass@5, and pass@10 metrics, calculated from 20 generation runs, as established in [8].

4.2 Assessing Fine-tuned Models on Data4AIGChip's Delivered PoT Structured Dataset

Setup. We evaluate the effectiveness of our PoT structured dataset in enhancing RTL code generation. We fine-tune the CodeLLaMA-7B-Instruct model on this dataset format and compare its performance with models trained on standard baseline datasets, in terms of the code generation pass rates, i.e., pass@1, pass@5, and pass@10. The baseline dataset settings include training solely on three specific

Table 2. Dataset statistics and features of the proposed Data4AIGChip, where the dataset samples are hierarchically organized into the proposed PoT structure containing varying levels of code description detail levels, aiming to strike a balance between dataset user-friendliness and accurate representation of the corresponding hardware design.

Works	Number of Designs	High-level Global Summaries	Detailed Global Summaries	Block Summaries	Line-by-line Comments
Thakur et al. [13]	17	✔	✘	✘	✘
Chip-Chat [1]	8	✔	✘	✘	✘
Chip-GPT [2]	8	✔	✘	✘	✘
RTLLM [9]	30	✔	✘	✘	✘
VerilogEval [8]	8502	Partial	✔	✘	✘
Data4AIGChip	11144	✔	✔	✔	✔

types of descriptions: (1) high-level global summary, (2) detailed global summary, and (3) detailed block summaries. In contrast, our approach involves training on a combined dataset that mixes all three description types. This method aims to fully leverage the unique structure of the PoT dataset. **Note that** for ensuring user-friendly interactions with fine-tuned LLMs, the fine-tuned LLMs are tested *solely using the high-level global summary*.

Observations and analysis. The results are detailed in Tab. 3. We can observe that (1) The model trained on the PoT dataset shows a consistent improvement over baseline settings, across the pass rates metrics on the VerilogEval benchmark. For example, the model trained on our PoT dataset consistently outperforms VeriogEval [8] across different generation trials, e.g., a 3.4% improvement in the pass@10 rate. It is worth noting that our method can achieve larger improvements under more generation trials; (2) The model trained on block summaries exhibits the lowest performance. This is attributed to the significant disparity between the detailed training data (finer-grained block summaries) and the more general evaluation data (high-level summaries); (3) Interestingly, the model trained on detailed global summaries outperforms the one trained on high-level global summaries, despite being evaluated on the latter. This is likely because detailed global summaries provide richer information, aiding the model in understanding code structure and thus generating more accurate code. The gap in the detail level between detailed global summaries and high-level summaries is smaller compared to that between block summaries and high-level summaries, which is why the detailed summaries' more comprehensive supervision is not overshadowed by the detail level gap.

5 Related Work

LLMs, in-context learning, and fine-tuning. A key ability of existing LLMs is in-context learning, where incorporating just a few relevant input-output pairs can significantly enhance task performance [11]. However, applying in-context learning to hardware code generation often produces incomplete and erroneous code, as LLMs lack domain-specific reasoning in hardware [4]. This issue necessitates further fine-tuning with a meticulously crafted dataset.

LLM-assisted hardware design. LLMs have been used in hardware design for different tasks [2, 4, 8, 14]. However, their performance is limited by insufficient exposure to hardware data during pretraining [2, 4]. Some studies [8, 9, 14] have attempted to address this by providing more hardware code samples and fine-tuning the LLMs, but the datasets used are either too small [9] or can be simplistic [8, 14]. The key issue here is the lack of a comprehensive and scalable approach to creating high-quality hardware code datasets that encompass a wide range of complexities and precise natural language descriptions at different levels of detail. Data4AIGChip addresses this issue by providing an automated flow for generating and validating diverse, high-quality hardware code datasets.

Table 3. Pass rates of CodeLLaMA-7B-Instruct models fine-tuned using different data formats while evaluated on high-level global code summaries for enhanced user-friendliness.

Fine-tuning Data Format	PoT	High Level Global Summaries	Detailed Global Summaries	Block Summaries
Pass@1 (%)	45.2 (↑ 0.4 ~↑ 4.9)	42.4	44.8	40.3
Pass@5 (%)	52.2 (↑ 2.3 ~↑ 8.2)	48.1	49.9	44.0
Pass@10 (%)	55.2 (↑ 3.4 ~↑ 9.6)	49.7	51.8	45.6

6 Conclusion

This work contributes to LLM-assisted hardware design through the development of Data4AIGChip. By integrating a novel Pyramid of Thoughts data structure, a Retrieval-Augmented Generation technique, and an automated validation flow, Data4AIGChip provides an effective and scalable framework for automated hardware data generation and validation. Various experiments validate the effectiveness of our techniques and the benefits of datasets generated by Data4AIGChip in enhancing the performance of LLM models for hardware code generation. As the demand for efficient and innovative hardware design grows, tools like Data4AIGChip can pave the way for more accessible LLM-assisted hardware design.

7 Acknowledgments

The work is supported by the National Science Foundation (NSF) through the CCRI program (Award number: 2016727), an NSF CAREER award (Award number: 2345577), and CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] Jason Blocklove et al. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. *arXiv* (2023).
[2] Kaiyan Chang et al. 2023. ChipGPT: How far are we from natural language hardware design. *arXiv* (2023).
[3] Tim Dettmers et al. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv* (2023).
[4] Yonggan Fu et al. 2023. GPT4AIGChip: Towards Next-Generation AI Accelerator Design Automation via Large Language Models. *arXiv* (2023).
[5] Ryan Greene et al. [n. d.]. New and improved embedding model. https://openai.com/blog/new-and-improved-embedding-model. (Accessed on 11/20/2023).
[6] Zhuolun He et al. 2023. ChatEDA: A Large Language Model Powered Autonomous Agent for EDA. *arXiv* (2023).
[7] Mingjie Liu et al. 2023. Chipnemo: Domain-adapted llms for chip design. *arXiv* (2023).
[8] Mingjie Liu et al. 2023. VerilogEval: Evaluating Large Language Models for Verilog Code Generation. *arXiv* (2023).
[9] Yao Lu et al. 2023. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. *arXiv* (2023).
[10] OpenAI. 2023. GPT-4 Technical Report. *arXiv* (2023).
[11] Yasaman Razeghi et al. 2022. Impact of pretraining term frequencies on few-shot reasoning. *arXiv* (2022).
[12] Vijay Janapa Reddi et al. 2023. Architecture 2.0: Challenges and Opportunities. *DAC* (2023).
[13] Shailja Thakur et al. 2022. Benchmarking Large Language Models for Automated Verilog RTL Code Generation. *arXiv preprint arXiv:2212.11140* (2022).
[14] Shailja Thakur et al. 2023. VeriGen: A Large Language Model for Verilog Code Generation. *arXiv* (2023).
[15] Hugo Touvron et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv* (2023).
[16] Jason Wei et al. 2021. Finetuned language models are zero-shot learners. *arXiv* (2021).
[17] Jason Wei et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS* (2022).