Hardware Phi-1.5B: A Large Language Model Encodes Hardware Domain Specific Knowledge

Invited Paper

Weimin Fu*, Shijie Li†, Yifang Zhao†, Haocheng Ma‖, Raj Dutta¶, Xuan Zhang§, Kaichen Yang‡, Yier Jin†, Xiaolong Guo*
*Kansas State University, {weiminf, guoxiaolong}@ksu.edu
†University of Science and Technology of China, {shijie_li, zhaoyifang}@mail.ustc.edu.cn, jinyier@ustc.edu.cn
‡Michigan Technological University kaicheny@mtu.edu
§Washington University in St. Louis xuan.zhang@wustl.edu
¶Silicon Assurance rajgautamdutta@siliconassurance.com
‖Tianjin University, hc_ma@tju.edu.cn

***Abstract*—In the rapidly evolving semiconductor industry, where research, design, verification, and manufacturing are intricately linked, the potential of Large Language Models to revolutionize hardware design and security verification is immense. The primary challenge, however, lies in the complexity of hardware-specific issues that are not adequately addressed by the natural language or software code knowledge typically acquired during the pretraining stage. Additionally, the scarcity of datasets specific to the hardware domain poses a significant hurdle in developing a foundational model. Addressing these challenges, this paper introduces *Hardware Phi-1.5B*, an innovative large language model specifically tailored for the hardware domain of the semiconductor industry. We have developed a specialized, tiered dataset—comprising small, medium, and large subsets—and focused our efforts on pre-training using the medium dataset. This approach harnesses the compact yet efficient architecture of the Phi-1.5B model. The creation of this first pre-trained, hardware domain-specific large language model marks a significant advancement, offering improved performance in hardware design and verification tasks and illustrating a promising path forward for AI applications in the semiconductor sector.**

***Index Terms*—Large Language Model; Hardware Design; Hardware Verification; Generative AI;**

I. Introduction

Knowledge and language processing are crucial in multiple critical aspects of the semiconductor industry, such as research, design, verification, and manufacturing. These aspects involve complex interactions among numerous entities, demanding high accuracy and efficiency in information exchange. Artificial intelligence (AI) has demonstrated significant potential in fields such as hardware design, verification, and routing in recent years [1]–[5]. Nevertheless, it should be noted that these AI applications do not yet fully exploit the entirety of available knowledge or information, leading to errors in their judgments based on partial data, relegating AI methodologies to a supplementary role in the hardware domain. In contrast, Large Language Models (LLMs) can extensively leverage the knowledge conveyed and utilized through natural language and code during the hardware design process. Given these capacities, LLMs possess the potential to revolutionize the fields of hardware design and security verification.

Fig. 1 illustrates the progression of stages in training LLMs from raw datasets to assistants. In the hardware domain, research endeavors are presently focused on In-Context Learning (ICL) strategies, such as hardware bug fixing and verification assistant [6]–[8]. ICL does not alter parameters in LLMs; instead, it serves more as a tactical approach than a fundamental solution for performance enhancement. Consequently, irrespective of how the initial prompts are optimized, the improvements in model performance cannot be directly attributed to ICL. Supervised Finetuning (SFT) represents another stage. Several groups attempted to employ SFT to address hardware debugging and design challenges [9]–[11], but outcomes often lack consistency. The primary challenge lies in the complexity inherent to hardware-specific issues not adequately addressed by the natural language or software knowledge acquired during the Pretrain stage. Hence, developing a base model tailored to enhance robustness

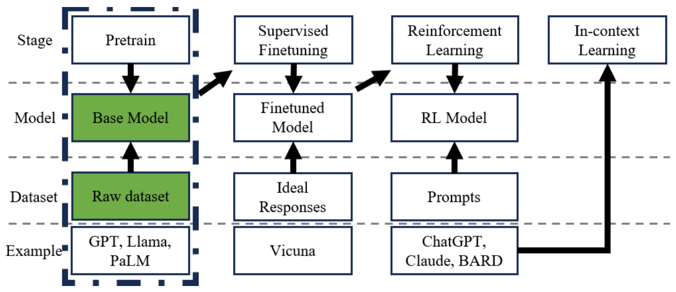


Fig. 1. Four-Stage LLM-Based Assistant Development: Pretraining with raw data for a base model; ideal response-driven supervised fine-tuning; instruction-based reinforcement learning with few-shot examples for a deployable model; culminating in user engagement via in-context learning. The green cells highlight the contributions made in this paper.

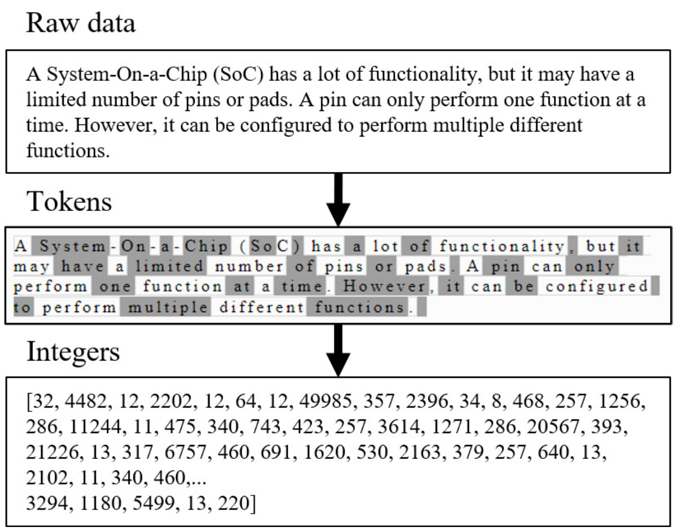
in the hardware domain is in high demand and will significantly strengthen the open-source hardware community.

However, the scarcity of datasets in the hardware domain presents the initial challenge in Pretrain stage. This scarcity is not unique to the hardware domain; datasets have become the most limited yet critical resource in developing LLMs. Leading proprietary LLM GPT-4 [12], as well as the most potent open-source LLM Llama2 [13], do not furnish datasets. However, they share methodologies or *recipes* for Pretrain dataset construction. Informed by the approaches delineated in constructing open-source datasets RedPajama [14] and the Stack [15], based on these recipes, we have crafted a Hardware Domain-Specific Dataset. This dataset has been segmented into three distinct tiers based on content volume: small, medium, and large.

In this study, we have adopted the Phi-1.5B model [16] architecture and conducted pretraining of a hardware domain-specific LLM on the medium dataset[1]. The Phi-1.5B model boasts performance that rivals that of Llama2 7B despite being only a fifth of its size, an attribute that underscores the model's efficiency and effectiveness in training. Moreover, the reduced scale of the model also implies that the training costs for subsequent task-specific fine-tuning will be significantly lower. We anticipate that this fully open-source pretrained model will be a robust foundational support for a wide array of tasks within the hardware domain, especially playing a pivotal role in addressing hardware security challenges. Through meticulously constructed hardware-specific datasets and customized pre-trained LLMs, this paper aims to precisely address the unique challenges in hardware domain tasks, achieving a deep understanding and response to the needs of this field. Our contributions are mainly reflected in the following aspects:

1) This paper conducted pretraining based on the Phi-1.5B model structure, making it more closely aligned with the needs of the hardware domain, enhancing the model's performance and stabil-

[1]Our available computational resources informed this decision.



A System-On-a-Chip (SoC) has a lot of functionality, but it may have a limited number of pins or pads. A pin can only perform one function at a time. However, it can be configured to perform multiple different functions.

A System - On - a - Chip (So C) has a lot of functionality , but it may have a limited number of pins or pads . A pin can only perform one function at a time . However , it can be configured to perform multiple different functions .

[32, 4482, 12, 2202, 12, 64, 12, 49985, 357, 2396, 34, 8, 468, 257, 1256, 286, 11244, 11, 475, 340, 743, 423, 257, 3614, 1271, 286, 20567, 393, 21226, 13, 317, 6757, 460, 691, 1620, 530, 2163, 379, 257, 640, 13, 2102, 11, 340, 460,...
3294, 1180, 5499, 13, 220]

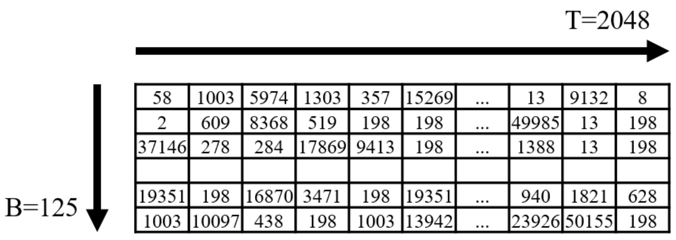
Fig. 2. Tokenization example: transform all text to a list of integers.

ity in hardware design and verification tasks. To our knowledge, it is the first pretrained hardware domain-specific LLM.

2) We created three differently sized datasets rigorously screened and optimized them to guarantee content relevance and quality, thus laying a strong foundation for model training.
3) The pre-trained model is offered openly to the community, thus supporting ongoing research, development, and innovation in both academic and industrial spheres.

II. Background: Foundational Concepts

LLMs are not inherently capable of directly processing raw knowledge and information. To make the information intelligible to these models, we must first transform it into a sequence of integers that the model can directly interpret. This transformation process is known as **Tokenization**. As illustrated in Fig. 2, we employ the CodeGen-mono [17] tokenizer to segment the raw text into discrete units, referred to as *tokens*. Note that although a tokenizer's vocabulary may contain a vast array of words, a token does not always correspond directly to a single word. For instance, the proprietary term *SoC* depicted in the figure is split into two separate tokens: *So* and *C*. Each token is subsequently assigned a unique numerical identifier correlating to an index in the vocabulary list, generating an extensive sequence of integers. Upon analyzing our dataset, we observed that, on average, each token is equivalent to 0.74 words.



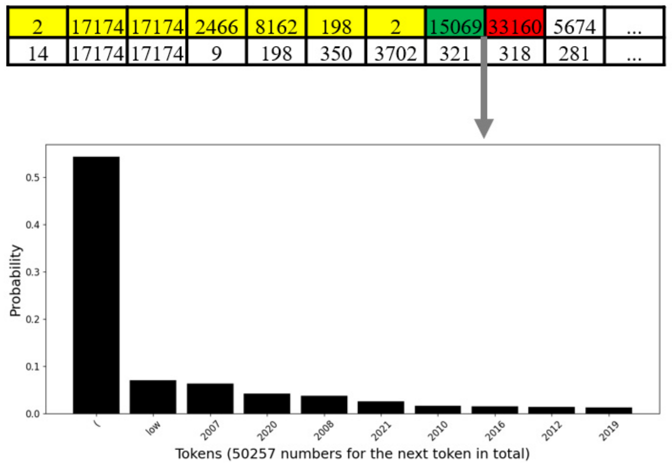
58	1003	5974	1303	357	15269	...	13	9132	8
2	609	8368	519	198	198	...	49985	13	198
37146	278	284	17869	9413	198	...	1388	13	198
19351	198	16870	3471	198	19351	...	940	1821	628
1003	10097	438	198	1003	13942	...	23926	50155	198

Fig. 3. Matrix representation of the batch structure in Hardware Phi-1.5B.

In the initial data handling phase, we encounter data characterized by elements of disparate lengths and a discrete nature. To methodologically address this heterogeneity, we used the batch structure as depicted in Fig. 3, which is organized into a matrix with dimensions (B, T). Here, B denotes the batch size—fixed at 125 for the scope of our experimental analysis—and T encapsulates the maximal context length that our model, designated as Phi-1.5B, is capable of processing, the value of which is set at 2048 tokens.

Batch Size is the number of training examples utilized in one iteration. A vast batch size may exceed the memory constraints, precipitating out-of-memory errors. Conversely, an unduly small batch size could result in excessively noisy gradient updates, thereby detrimentally affecting model performance. For comparative context, the Llama2 employs a batch size 64, correlating to a 64×4096 configuration. In contrast, the original MicroSoft Phi-1.5B utilizes a batch set at 2048×2048.

We employ a concatenation strategy to account for the inherent variability in the lengths of data items within a dataset—which is unlikely to match the specified value of T uniformly. This involves unifying disparate data lengths and affixing an end-of-sequence (EOS) token to the culmination of each data item. Subsequently, the data undergoes a normalization process, uniformly resized to conform to the dimensions (B, T). In our research, the EOS token is denoted by the symbol `<|endoftext|>`.



2	17174	17174	2466	8162	198	2	15069	33160	5674	...
14	17174	17174	9	198	350	3702	321	318	281	...

Fig. 4. Token prediction probability distribution with preferences after pretrain.

Fig. 4 illustrates the pretraining process of the LLMs, explicitly highlighting their capability to predict the next token. Within each cell of the training batch, the model is only privy to the data in the cells to the left within the same row. During the pretraining, the model learns to predict the content of the right-side cell by utilizing the context provided by the cells on the left, a paradigm known as a causal language model (CLM). The green cell represents a randomly selected token awaiting processing, the yellow area denotes the preceding text utilized for prediction, and the red cell signifies the prediction target. *Hardware Phi-1.5B* would select from the entire vocabulary list comprising $50,257$ options. Through training, the model transitions from uniform probability distribution across all possible outcomes to allocating probabilities with a clear preference toward a handful of potential choices.

III. Pretrain Dataset

A. Dataset Overview

In this study, we have developed a hardware domain-specific dataset derived from various publicly accessible sources and bifurcated it into two main categories: code and natural language. Table I delineates the primary constituents of the dataset.

For the code segment, we leveraged Google BigQuery GitHub Public Datasets [18] , selecting projects that encompass hardware design source code. Our selection was concentrated on three pivotal hardware programming languages: SystemVerilog, Verilog, and VHDL. This concentration facilitated the incorporation of entire repositories containing pertinent code into our dataset. Additionally,

TABLE I
OVERVIEW OF DATASET COMPOSITION: CATEGORIES, LANGUAGES, SOURCES, AND SELECTION METHODS

Data Category	Languages	Sources	Methodology or Selection Criteria
Hardware Design Code	SystemVerilog, Verilog, VHDL	Google BigQuery Github, the Stack	Filtering GitHub repositories for hardware design code
Hardware Nature Language Knowledge	English	RedPajama CommmolCrawl, ArXiv, StackExange, Books, C4, Wikipedia), TrustHub, Cad4Assurances, CWE	Aggregating content from the hardware domain

to ensure both legal compliance and open accessibility, we meticulously selected projects under specific open-source licenses.

In parallel, we curated a comprehensive hardware security dataset, amalgamating both code and natural language content from eminent hardware security sources, including TrustHub [19], CAD for Assurance of Electronic Systems [20] , and Common Weakness Enumeration (CWE) [21] . This compilation enriches our dataset with current and vital insights into hardware security, encompassing codes, their descriptions, best practices, and security recommendations for hardware design.

For practical applications in Code Language Models, integrating natural language data is commonly recognized as beneficial for enhancing model performance [22]. Hence, we adopted and modified the Redpajama dataset construction methodology with a focused segmentation due to the relative rarity of hardware-specific content. The Redpajama dataset, an expansive corpus exceeding 1.2 trillion tokens, was segmented to enhance hardware-related discourse. The first segment includes data from specialized platforms such as Arxiv , Books, Wikipedia, and StackExchange; the second segment is derived from broader internet content via CommonCrawl and C4.

Table II provides a comprehensive overview of the rigorous processes applied to verify and cleanse our dataset, ensuring its quality and integrity. These steps are crucial for maintaining the dataset's reliability and usability in research:

- *Verification and Cleansing*: This step involves identifying syntactic errors and reviewing natural language descriptions to ensure data accuracy. It combines automated scripts for efficiency and manual reviews for precision, addressing the dual aspects of mechanical accuracy and contextual relevance.
- *Redpajama Dataset Filtering:* Here, we filter hardware security content using targeted keywords. This step is vital for maintaining the dataset's focus and relevance to the field of hardware security, ensuring that the dataset remains aligned with its intended purpose.
- *Text Processing:* This phase includes several processes: NFC normalization, short content filtering, and removing punctuation and unnecessary spaces. The use of datasketch [23], a tool known for its efficiency in handling large datasets, helps streamline this process, improving data quality and consistency.
- *De-duplication:* To enhance the dataset's utility, this step employs indexing and the MinHash technique for de-duplication, again utilizing datasketch. De-duplication is critical for eliminating redundant data, enhancing overall quality, and making the dataset more manageable and effective for users.

Each of these steps plays a vital role in refining the dataset. This meticulous process ensures that the dataset is extensive but also precise and reliable, catering to the nuanced needs of hardware security research.

B. Visual and Quantitative Analysis

Fig. 5 visually represents our dataset's construction and segmentation. The smallest dataset comprises information related to hardware security and the source code of hardware design. In this dataset, the proportion of CWE is approximately 0.0014468%, yet its significance is paramount. This CWE section contains over $70,000$ tokens. Although it represents a minor proportion of the dataset's total volume, CWE information is crucial, offering critical insights into security vulnerabilities and weaknesses relevant to hardware security. While numerically limited, these data are of high quality and professionalism, reflecting common security weaknesses and potential risks in hardware design. Hence, despite their small percentage in the overall dataset, the CWE elements are indispensable and significantly contribute to in-depth hardware security research.

TABLE II
STEPS IN DATA VERIFICATION, CLEANSING, AND PROCESSING WITH APPLIED TOOLS AND METHODS

Step	Description	Tools/Methods
Verification & Cleansing	Identifying syntactic errors; Reviewing natural language descriptions.	Automated scripts, Manual review
Redpajama Dataset Filtering	Filtering content related to hardware using keywords	Keyword filtering
Text Processing	NFC normalization; Filtering short content; Removing punctuation, spaces, etc.	datasketch
De-duplication	Indexing and de-duplication using MinHash	datasketch

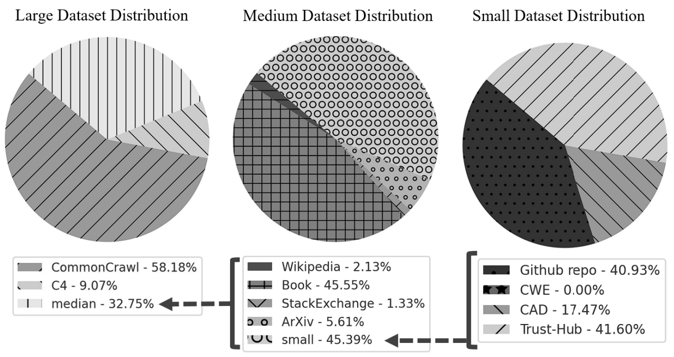


Fig. 5. Visual Representation of Dataset Construction and Segmentation

TABLE III
BREAKDOWN OF DATASETS BY SIZE IN TOKENS

Dataset Name	Dataset Size (Tokens)
Smalt	$4,838,384,488$
Medium	$10,382,663,651$
Large	$22,616,170,041$

Expanding to the second dataset, the medium dataset evolves from the smaller dataset and incorporates information from ArXiv, StackExchange, Books, and Wikipedia sources. This dataset provides a broader perspective and information base, supporting a more comprehensive analysis.

Finally, the large dataset further extends the scope, encompassing all contents of the medium dataset and a vast array of data from C4 and CommonCrawl. The diversity and scale of this dataset offer abundant resources.

To further quantify, Table III presents a detailed breakdown of the three datasets in size, measured by token counts, facilitating a direct comparison between the small, medium, and large datasets. This information is critical for users in selecting the most appropriate subset for their research needs. As an open-source dataset, this flexibility empowers users to select a subset that aligns with their computational resources and specific requirements.

IV. HARDWARE PHI-1.5B MODEL PRETRAIN

A. Model Architecture

The architecture of Hardware Phi-1.5B strictly adheres to the design principles of the original Phi-1.5B and its variants. It comprises a Transformer structure [24] with 24 layers, 32 heads, and a dimension of 64 for each head, resulting in a context length of $2,048$. We incorporated the *Flash Attention 2* [25] during the training phase to expedite the training process. To ensure optimal compatibility with tools that utilize LLMs, we opted to follow the Phi-1.5B style and employ the codegen-moni tokenizer.

B. Model Training Methodology

The key setups in pretraining Hardware Phi-1.5B are listed below.

1) Initialization Strategy: The model's training is initiated with a state of random weights. We meticulously initialized the weights of the linear and embedding layers by employing a normal distribution with a mean of 0 and a standard deviation of 0.02. This approach was adopted to prevent the extremes of weight magnitude, thus averting the well-known issues of gradient disappearance or explosion. Additionally, we set the biases in linear layers to zero, fostering a neutral starting point that prevents any early bias toward specific outcomes.

2) Training Configurations: Our training configuration was standardized with a fixed learning rate of $2e-4$ and a weight decay factor of 0.1, mirroring the training regimen of the Phi-1.5B model.

3) Optimizer Settings: The Adam optimizer [26], equipped with beta momentum values of 0.9 and 0.98 and an epsilon value of $1e{-}7$, was the chosen algorithm for its reliable performance in similar tasks.

4) Efficiency Strategies: To optimize memory usage and training efficiency, we adopted fp16 mixed precision training. Additionally, we utilized the Fully Sharded Data Parallel [27], enabling the distribution of the model's parameters across all available GPU resources. This was complemented by an effective communication strategy aimed at reducing training overheads. For further memory optimization, the transformer blocks were integrated with the `auto_wrap_policy`, and to strike a balance between memory use and computation speed, we enabled activation checkpointing.

5) Evaluation Framework: Given the substantial size of our dataset, we set a training termination criterion at $750,000$ iterations and $30,000$ steps. To monitor the model's performance progression and mitigate the potential of overfitting, we instituted a checkpoint mechanism that allowed the model's state to be saved and evaluated at every $1,000$ step, ensuring we could capture performance metrics systematically throughout the training process.

V. Experiment and Result

Our training platform is constructed on a server operating with Ubuntu 20.04.6 LTS, equipped with an Intel(R) Xeon(R) Silver 4314 CPU (2.40 GHz, 64 cores), 251 GB of memory, and dual NVIDIA A100 80 GB graphics processing cards. This configuration not only provides substantial computational power but also ensures ample memory when dealing with large models. The CUDA 12.2 and PyTorch 2.1.0 versions selected for our work fully exploit the hardware potential, optimizing the model training process. Supported by this high-performance hardware, our platform can achieve a throughput of 1.07 batches per second while maintaining approximately $100T$ floating-point operations per second (flops/sec) and a token processing speed of around $11k$ per second.

In this experiment, we have designated $30k$ training steps, culminating in a total training duration of 8 days, 2 hours, 43 minutes, and 22 seconds. This training cycle was determined post-evaluation of the anticipated model complexity and the requisite time for convergence. Additionally, considering the power consumption of GPU devices and based on previous studies [28] , we have computed the energy efficiency during the training process. We estimate that this training has produced approximately 90kg of carbon dioxide equivalent greenhouse gas emissions.

Fig. 6 presents the variation in loss and perplexity on the validation dataset. Loss is an indicator that measures the discrepancy between model predictions and actual values, with Mean Squared Error (MSE) and Cross-Entropy Loss being commonly used metrics [29]. A high loss indicates a greater disparity between the model predictions and actual values, hence a lower model performance. Perplexity [30], particularly within natural language processing and language models, indicates model performance. It is the exponentiation of the cross-entropy loss and offers an average branching factor per word, that is, the model's uncertainty about the next word given the preceding ones. In our training context, a lower perplexity suggests a more precise understanding of the data by the model.

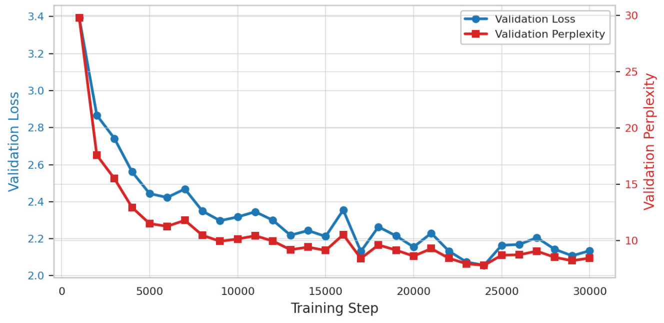


Fig. 6. Validation Loss and Perplexity During Training

A System-On-a-Chip (SoC) has a lot of functionality, but it may have a limited number of pins or pads. A pin can only perform one function at a time. However, it can be configured to perform multiple different functions. This technique is called pin multiplexing.

Fig. 7. Training Data: CWE-1189: Improper Isolation of Shared Resources on System-on-a-Chip (SoC)

Initially, models exhibit high loss and perplexity, reflecting their preliminary adaptation to the data during the learning process. As training progresses, these metrics generally decrease, signifying the model's advancements in learning. As in Fig. 7, we have conducted text continuation experiments on the CWE-1189 [31] security vulnerability description. In our experiments, we generated text using models at different training stages—the initial model, the model after $10k$ steps, the model after $20k$ steps, and the final model at $30k$ steps training.

In Fig. 8, the initial model's generated text lacked coherence and logic, indicating that at the onset of training, the model's understanding of language structure had yet to be established. This stage's model output is typically a meaningless string of words, reflecting the model's randomness and immaturity.

After $10k$ training steps, the model demonstrated correct word spelling as in Fig. 9, signifying that it had begun to capture the basic language rules, such as word boundaries and fundamental grammatical structures. However, the generation of longer sentences remained unnatural, suggesting an insufficient grasp of complex structures.

Following $20k$ training steps, as shown in Fig. 10, the model's generated text began to show readability, albeit with occurrences of looping and repetition. This may be attributed to the model finding a "comfort zone" in certain local minima and tending to repeat patterns discovered in the training data rather than generating novel content.

tributePAfort MoorFontSizeestone Karl Norris intens Norris informative refinement kg 278 impulses Yugoslavinnamon Buddha Performance Divine clinch interacts Performance Scientology alertinsulaes inex Harding Gene oral Norris heinous risk Kremlin Harding546 crossings

Fig. 8. Generated sample at initialization.

> if 0 = Pin multiplexing else do you need to change those pin numbers to the least expensive chips? A: The pin number of the SoC has to be the same in both the input and output pins, with no extra pins on the chip connected directly. There are several sources of noise, either from the FPGA or a FPGA or other components like these may be bad quality.

Fig. 9. Generated sample at $10k$ step, $250k$ iterations of training.

> On-board: I/O multiplexing I/O multiplexing For example: I/O multiplexing I/O multiplexing I/O multiplexed in between As mentioned in the link, a physical system-on-a-chip (SoC), also known as a Chip-on-a-Chip (SoC), offers pinout, multiplexer and multiplexer connected in a single interconnect.

Fig. 10. Generated sample at $20k$ steps, $500k$ iterations of training.

Finally, the model post $30k$ training steps exhibited a more mature understanding of language, integrating natural language capability and domain-specific knowledge in text generation in Fig. 11. Nonetheless, due to the lack of task-specific fine-tuning, the model might not adapt well to specific contexts or answer complex queries. This highlights the model's limitations as a base model: it is adept at predicting the next token in text and, hence, tends to complete a document, but for more advanced tasks, further optimization and adaptation may be necessary.

Through these phases, we can observe the model's progression in understanding and generating language. Nevertheless, to become a practical assistant, the model requires further training and customized fine-tuning. Future work will focus on enhancing the model's performance while reducing its training process's environmental impact.

VI. Related Work

Compared to the singular pursuit of developing Large Language Models for the goal of achieving General Artificial Intelligence, an increasing body of research is focusing on constructing specialized, domain-specific datasets and training LLMs to nurture systems that demonstrate expert-level proficiency within specific domains.

In the medical field, due to the highly complex nature of the expertise required and the constraints imposed by privacy regulations, general LLM typically fails to provide sufficient comprehensive coverage. As a result, researchers are turning to strategies involving complete pretraining, supervised finetuning, and reinforcement learning approaches. K. Singhal has developed MultiMedQA, a composite benchmark integrating six existing medical question-answering datasets, and a new online search medical question dataset, HealthSearchQA. Utilizing this benchmark, Google has further trained PaLM and its variant, FLAN-PaLM [32]. L. Y. Jiang has adopted a BERT-based pretraining and finetuning to develop NYUTron, designed to offer guidance at clinical care points [33].

In the realm of hardware security, research approaches vary depending on the perspectives of researchers. Some contend that existing commercial general-purpose LLMs, such as ChatGPT, are sufficient to support formal verification tasks, relying on OpenAI's continuous enhancements of ChatGPT to improve the performance of their tools. For instance, M. Orenes-Vera has attempted to use ChatGPT for RTL formal verification [34]. M. Chen, on the other hand, has employed codex, ChatGPT, and a Codegen variant fine-tuned for Verilog [10] to generate assertions directly [35]. However, other researchers argue that proprietary knowledge in the hardware domain necessitates custom dataset training for models. They believe that their research outcomes will be more pronounced as access to more domain-specific data and computational resources becomes abundant. For example, S. Thakur has finetuned for Verilog generation [10], and W. Fu has also finetuned an LLM for hardware debugging based on version control information [9].

On the other hand, there can be significant variances in the usage of specialized terminology between different domains, even evident in the software and hardware fields. For example, *Port* typically refers to a communication interface in software engineering, whereas it denotes a physical interface on electronic devices in hardware design. *Cache* in software signifies a temporary storage area to expedite data access. At the same time, in hardware, it might refer to a small-capacity, high-speed storage located between the CPU and main memory. Moreover, *Pipeline* can represent a sequence of processing steps in software development, whereas, in hardware design, it indicates a specific technique for parallel data processing. Terms like *Bus*, *Driver*, *Register*, and *Core* also possess dual meanings; their conflation can lead to confusion and impair the model's understanding and predictions.

In light of this, our research endeavors are concentrated on the development and use of datasets tailored explicitly for the hardware domain for pretraining, and aiming to construct a base model that paves the way for breakthroughs in the meticulous finetuning of domain-specific models. We anticipate that this approach will significantly enhance the model's performance in comprehending and handling the aforementioned complex terminologies, surpassing the capabilities of existing general code language models (such as CodeLlama [22]) and natural language models (such as BERT [36] and GPT-2 [37]).

> In the context of embedded-edge technologies, in the processor, an edge device is an integrated hardware device. An edge device is an embedded computation device that provides the capability to operate at all the levels of the chip. In the context of processors, it is more common that an embedded computation device is integrated and integrated into an embedded system. In the above example, for an embedded computation device, it is referred to the processor as computing core. It also refers to the SoC device when it is the peripheral, embedded system, or embedded system controller. On the other hand, an embedded system is an embedded systems management device that provides the capability to communicate with the

Fig. 11. Generated sample at $30k$ steps, $750k$ iterations of training.

VII. Conclusion and Future Work

In this study, we have explored the potential of Large Language Models advancing hardware design, Electronic Design Automation, and hardware security. Recognizing that hardware design shares certain formal similarities with natural language and software design yet diverges fundamentally in its core complexities, we have focused on developing a specialized LLM for the hardware domain, establishing a robust foundational dataset. This endeavor not only breaks new ground in conventional approaches to the hardware domain but also paves the way for novel perspectives in future research and applications.

Moving forward, we aim to continue advancing this project, focusing on pre-training the base model while maintaining and updating our dataset to ensure its relevance and contemporaneity. We eagerly anticipate the fine-tuning and application of this model in specific areas within the hardware design domain, particularly in addressing distinct design challenges and security issues. We believe our work will offer new insights and solutions for research and practice in hardware design, verification, and security, potentially catalyzing transformative changes in these fields.

Acknowledgments

Portions of this work were supported by the National Science Foundation (CCF-2019310, First Award Program of ARISE in EPSCoR 2148878).

References

[1] G. Hains, A. Jakobsson, and Y. Khmelevsky, "Towards formal methods and software engineering for deep learning: security, safety and productivity for dl systems development," in 2018 Annual IEEE international systems conference (syscon). IEEE, 2018, pp. 1–5.

[2] Y. Lai, J. Liu, Z. Tang, B. Wang, J. Hao, and P. Luo, "Chipformer: Transferable chip placement via offline decision transformer," arXiv preprint arXiv:2306.14744, 2023.

[3] R. Cheng and J. Yan, "On joint learning for solving placement and routing in chip design," Advances in Neural Information Processing Systems, vol. 34, pp. 16 508–16 519, 2021.

[4] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "A graph placement methodology for fast chip design," Nature, vol. 594, no. 7862, pp. 207–212, 2021.

[5] S. Guadarrama, S. Yue, T. Boyd, J. W. Jiang, E. Songhori, T. Tam, and A. Mirhoseini, "Circuit Training: An open-source framework for generating chip floor plans with distributed deep reinforcement learning." https://github.com/google_research/circuit_training, 2021, [Online; accessed 21-December-2021]. [Online]. Available: https://github.com/google_research/circuit_training

[6] 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.

[7] Z. Zhang, G. Chadwick, H. McNally, Y. Zhao, and R. Mullins, "Llm4dv: Using large language models for hardware test stimuli generation," arXiv preprint arXiv:2310.04535, 2023.

[8] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," 2023.

[9] W. Fu, K. Yang, R. G. Dutta, X. Guo, and G. Qu, "LLM4SecHW: Leveraging domain-specific large language model for hardware debugging," Asian Hardware Oriented Security and Trust (AsianHOST), 2023.

[10] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2023, pp. 1–6.

[11] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," arXiv e-prints, pp. arXiv–2308, 2023.

[12] OpenAI, "Gpt-4 technical report," 2023.

[13] 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.

[14] T. Computer, "Redpajama: an open dataset for training large language models," 2023. [Online]. Available: https://github.com/togethercomputer/RedPajama-Data

[15] D. Kocetkov, R. Li, L. Ben Allal, J. Li, C. Mou, C. Muñoz Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, and H. de Vries, "The stack: 3 tb of permissively licensed source code," Preprint, 2022.

[16] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need ii: **phi-1.5** technical report," arXiv preprint arXiv:2309.05463, 2023.

[17] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," arXiv preprint arXiv:2203.13474, 2022.

[18] F. Hoffa. Github on bigquery: Analyze all the open source code. [Online]. Available: https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code

[19] "Trust-hub: A resource for hardware security and trust." [Online]. Available: https://trust-hub.org

[20] "Cad for assurance of electronic systems," an initiative to assemble and share information on CAD for trust/assurance activities in academia and industry. [Online]. Available: https://cadforassurance.org

[21] "Common weakness enumeration," 2022, https://cwe.mitre.org/.

[22] 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.

[23] E. Zhu, "datasketch: Big data looks small," GitHub, GitHub repository: https://github.com/ekzhu/datasketch, 2023, a library providing probabilistic data structures for processing and searching very large amounts of data efficiently. Version 1.6.4. [Online]. Available: https://ekzhu.github.io/datasketch

[24] 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.

[25] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," Advances in Neural Information Processing Systems, vol. 35, pp. 16 344–16 359, 2022.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[27] Y. Xu, H. Lee, D. Chen, H. Choi, B. Hechtman, and S. Wang, "Automatic cross-replica sharding of weight update in data-parallel training," 2020.

[28] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in Proceedings of the 2021 ACM conference on fairness, accountability, and transparency, 2021, pp. 610–623.

[29] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[30] C. Manning and H. Schutze, Foundations of statistical natural language processing. MIT press, 1999.

[31] Common Weakness Enumeration (CWE). Cwe-1189: Improper isolation of shared resources on system-on-a-chip (soc) (4.13). MITRE. [Online]. Available: https://cwe.mitre.org/data/definitions/1189.html

[32] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl et al., "Large language models encode clinical knowledge," arXiv preprint arXiv:2212.13138, 2022.

[33] L. Y. Jiang, X. C. Liu, N. P. Nejatian, M. Nasir-Moin, D. Wang, A. Abidin, K. Eaton, H. A. Riina, I. Laufer, P. Punjabi et al., "Health system-scale language models are all-purpose prediction engines," Nature, pp. 1–6, 2023.

[34] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "From rtl to sva: Llm-assisted generation of formal verification testbenches," arXiv preprint arXiv:2309.09437, 2023.

[35] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," arXiv preprint arXiv:2306.14027, 2023.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.