A Generalize Hardware Debugging Approach for Large Language Models Semi-Syntectic Datasets

Weimin Fu *Student Member, IEEE,* Shijie Li *Student Member, IEEE,* Yifang Zhao *Student Member, IEEE,* Kaichen Yang *Member, IEEE,* Xuan Zhang, *Senior Member, IEEE,* Yier Jin, *Senior Member, IEEE,* Xiaolong Guo, *Member, IEEE*

***Abstract*—Large Language Models (LLMs) have precipitated emerging trends towards intelligent automation. However, integrating LLMs into the hardware debug domain encounters challenges: the datasets for LLMs for hardware are often plagued by a dual dilemma – scarcity and subpar quality. Traditional hardware debug approaches that rely on experienced labor to generate detailed prompts are not cheaply scalable. Similarly, strategies that depend on existing LLMs and randomly generated prompts fail to achieve sufficient reliability. We propose a directed, semi-synthetic data synthetic method that leverages version control information and journalistic event descriptions. To produce high-quality data, this approach utilizes version control data from hardware projects combined with the 5W1H (Who, What, When, Where, Why, How) journalistic principles. It facilitates the linear scaling of dataset volumes without depending on skilled labor. We have implemented this method on a collected dataset of open-source hardware designs and fine-tuned fifteen general-purpose LLMs to enable their capability in hardware debugging tasks, thereby validating the efficacy of our approach.**

***Index Terms*—Large Language Model, Artificial Intelligence, Hardware Debug, Version Control, Electronic Design Automation**

I. Introduction

DEBUGGING in hardware design heavily relies on experienced labor, significantly limiting the adoption of automated and intelligent methodologies. Very recent mutation-based approaches [1], [2] have automated the repair of hardware design code at the register transfer level (RT-Level). However, they rely on reference models, and their ability to repair is limited to specific bugs due to their use of templates. Large Language Models (LLMs) signify a new frontier in automation, achieving a global discourse beyond academia and industry with the release of ChatGPT3.5 [3] at the end of 2022. This work utilizes LLMs to facilitate debugging in hardware design by marking the first adoption of training domain-specific LLMs in this field, similar to initiatives in other fields.

Notable other fields domain-specific LLMs examples include DoctorLM for the medical sector [4], and Disc-lawllm [5] for legal applications. In the closely related field of software, which shares similarities with hardware, we have conducted a comparative analysis of the implementation stages of LLMs. This comparison, detailed in Table I, examines pre-training, fine-tuning (specifically instruction tuning), alignment, benchmarks, and the utilization method known as prompting across software and hardware domains. In software development, LLMs participate in a mode known as *copilot*, where specially training LLMs generate predictions for potential code.

However, the hardware domain is at odds with the *copilot* mode with training. Most approaches are prompt-based but have one challenge: hallucination, where models generate plausible but factually incorrect or irrelevant responses. Conversely, if the accuracy of the generated answers is high, the hallucination problem is mitigated. Our findings indicate that hallucination is a far more prevalent issue in the hardware domain compared to the software domain. In the software domain, a typical benchmark such as Code Generation on HumanEval [6] has seen top-performing models achieve a $pass@1$ score as high as 98.2 [7]. Among general-purpose LLMs, the best performance is by Claude 3.5 Sonnet [8], scoring $90.2(pass@1)$. In contrast, the hardware domain's equivalent benchmark, VerilogEval-human [9], currently has a top score of $76.2(pass@1)$ [10], with the best performance among general LLMs being $61.5(pass@1)$ by Claude3-Haiku [11]. This stark difference highlights the more severe hallucination problem in hardware-related tasks. Reliance on closed-source LLMs, which do not provide a chance to correct hallucinations by providing domain knowledge data, suggests prompt-based works merely capitalize on OpenAI's efforts. Consequently, the reliability of generated content is compromised. For reminding hallucinations, a cautionary note is displayed on the ChatGPT below.

"ChatGPT can make mistakes. Consider checking important information."

Additionally, the OpenAI's strategies [12] in the hardware domain remain a significant concern for effectiveness.

One solution to revise hallucinations is training the LLM with domain-specific knowledge. But this leads us to another challenge: *the scarcity of data*. Without considering data quality and including the software-hardware interface, the volume of accessible textual knowledge in the hardware domain is approximately 23 billion tokens [13]. According to the Chinchilla Scaling Law [14], previous public datasets can support pretraining models up to 1 billion parameters, but only models with $70 \sim 160$ million parameters to reach saturation, which is too small for ideal intelligent performance. When considering private dataset, ChipNeMo [15] utilizes the dataset with NVIDIA proprietary code, documentation, and question-and-answer data, which indicates that the hardware domain LLM assistant is reachable with private knowledge assets. However, such developments provide negligible benefits to the broader community beyond corporate boundaries, as they are not open-sourced.

Therefore, this paper develops a method to construct a dataset that does not rely on proprietary corporate digital assets. Two common approaches for constructing LLM training datasets are using raw data and employing data synthesis. For raw data, when we revisit the three common LLM dataset sources in the hardware domain—namely, *User-generated content websites*, *Writers and publishers*, and *Open-source code*—we identify the following limitations:

- *User-generated content websites* lack substantial hardware-related information and tend to close off data access.
- *Writers and publishers* are unlikely to be reliable sources for hardware, and the opposition against using the content by LLMs.

Weimin Fu is with the Mike Wiegers Department of Electrical and Computer Engineering, Kansas State University, Manhattan, Kan. 66506, USA (email: weiminf@ksu.edu)

Shijie Li is with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, PRC

Yifang Zhao is with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, PRC

Kaichen Yang is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, Mich. 49931, USA

Xuan Zhang is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, Mass. 02115, USA

Yier Jin is with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230026, PRC

Xiaolong Guo is with the Mike Wiegers Department of Electrical and Computer Engineering, Kansas State University, Manhattan, Kan. 66506, USA (email: guoxiaolong@ksu.edu)

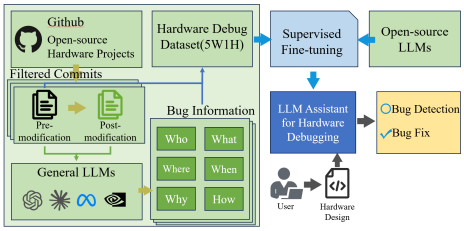


Fig. 1. Construction of a Fine-Tuning Dataset for Hardware Design Changes: From Isolated File Extraction to Natural Language Description Generation

- *Open-source code* of hardware designs do not match the software domain's quality.

As a result, directly constructing a high-quality and scalable fine-tuning dataset from raw data is impossible.

Synthetic data, which uses either human resources or artificial intelligence agent to generate data, is a viable approach. RTL-Coder [16] employs random instruction and ChatGPT for code generation. Nonetheless, challenges persist; modern LLMs are still far from becoming hardware experts, requiring detailed prompts or comprehensive tutorials. The deviation from the desired automation process suggests that synthetic methodology is not practical.

To solve the above problems, we propose a **Semi-Synthetic** approach, leveraging version control information between hardware design iterations to guide data synthesis. These narratives utilized the widely accepted 5W1H (Who, Why, What, Where, When, and How) journalism theory [17], effectively transitioning the task from hardware domain specific to reading comprehension. According to Chatbot Arena [18], modern high-performance LLMs (such as ChatGPT-4.0, Gemini 1.5 Pro Exp (0801), Claude 3.5 Sonnet, LLama 3.1 405B, and LLama 3.1 70B) are considered superior to the original version of GPT-4. GPT-4, in 2023, demonstrated proficiency by achieving a score of 169 on the GRE Verbal [19], surpassing 99% of test-takers (most of whom are fresh graduates). Therefore, any modern high-performance LLM emerges as an exemplary candidate for the data semi-synthetic. As shown in Fig. 1, the initial isolated hardware designs were from single-version releases of open-source hardware projects. Once integrated with information from the version control system, changes in each file within the projects are identified. Subsequently, an LLM agent is introduced to create a modeling method and design prompts that facilitate the generation of natural language descriptions for each identified design change. Finally, these elements are reassembled to construct a fine-tuning dataset, which is then utilized in the SFT process. Compared to directly using LLMs to solve problems based on prompts (e.g., [20]), our semi-synthetic method provides additional information for the same input (specifically, the next version of the hardware design). Introducing this additional information eliminates uncertainty [21], making our approach more reliable. The main contributions of this paper are:

1) We build an RT-level hardware debug dataset using version control information of hardware design iterations. By leveraging data from both new and old versions of hardware designs instead of the exhaustive prompts crafted by humans, creating a scalable LLM SFT dataset is possible. The Semi-Synthetic approach is universal, generalizable, and applicable to any magnitude of data collection.
2) This work employs the 5W1H narrative method to extract and utilize the version control data information. This interdisciplinary, innovative approach is inherently capable of describing code functionalities, issues, and solutions, maximizing the efficient use of all information within the dataset, and reducing the need for extensive human intervention.

TABLE I
COMPARATIVE OVERVIEW OF DOMAIN SPECIFIC LARGE LANGUAGE MODEL DEVELOPMENT AND APPLICATION ACROSS SOFTWARE AND HARDWARE DOMAINS

Stage	Software Domain	Hardware Domain
Pre-training	Codegen [22], CodeLlama [23], StarCoder [24]	HardwarePhi [13], ChipNeMo [15]
Supervised Fine-Tuning	WizardCoder [25], Octopack [26]	LLM4SecHW [27], RTLCoder [16] ChipNeMo
Alignment	RLTF [28], Pangu-coder2 [29]	RTLCoder
Prompt	Codet [30], Lever [31]	[32], [33], AssertLLM [34], DIVAS [35]
Benchmark	BigCodeBench [36], HumanEval [37],SWEBench [38]	VerilogEval [9], RTLLM [34]

3) Leveraging modern LLM reading capabilities to identify design issues by comparing versions sidesteps the challenge of hardware code analysis. The strategy maximizes the existing LLM potential by comparing different versions of information, thereby avoiding the hallucination that arises when facing challenging content.
4) We have fine-tuned 15 LLMs on this dataset to validate the feasibility. We have shown it effectively improves model performance in real-world scenarios.

Note, the 5W1H hardware debug dataset is available on Huggingface[1]. This dataset includes 6,545 entries, each structured around hardware design versions before and after bug fixes. Each entry contains detailed explanations using the 5W1H framework. The dataset is organized as follows:

- **Original**: The original version before the bug fix.
- **Modified**: The updated version after the fix.
- **Semi-Syntectic Data**: One detailed explanation with specific 5W1H question.

II. BACKGROUND

A. Large Language Model Dataset

The *intelligence* of LLMs is regarded as a compression of the training datasets [39], making these datasets the most crucial component of LLMs. Fig. 2 illustrates the four training phases of an LLM-based assistant: pre-training, supervised fine-tuning (SFT), reward modeling, and reinforcement learning. It details the algorithms, datasets, and representative tasks associated with each phase. Data for each phase are categorized into three types based on their source: *natural* for data that is unprocessed and used as-is from original corpora; *synthetic* for data that humans or AI has additionally processed; and *hardware domain*, which is specifically segregated considering our focus on the hardware field. This section introduces these datasets, organized by the training phases and data sources outlined earlier.

1) Pretrain Dataset

Natural: The datasets C4 [40] and Redpajama [41] are utilized for representing internet-based datasets. C4 specifically collects and processes internet data, focusing on deduplication, removing non-English content, and filtering out offensive language. Redpajama combines sources from Wikipedia and StackExchange, similar to C4, and is often used alongside it [42]. For code datasets, Starcoder [24] emerges as the largest, drawn from GitHub. It exclusively includes projects under MIT, BSD, and Apache licenses to minimize restrictions
Synthetic: Costs exceeding millions of dollars during the pretraining stage result in constraints on synthetic data. There are still some attempts being made. Phi1 [43] has utilized ChatGPT to cleanse the

[1]https://huggingface.co/datasets/KSU-HW-SEC/5W1HHardwareDebug

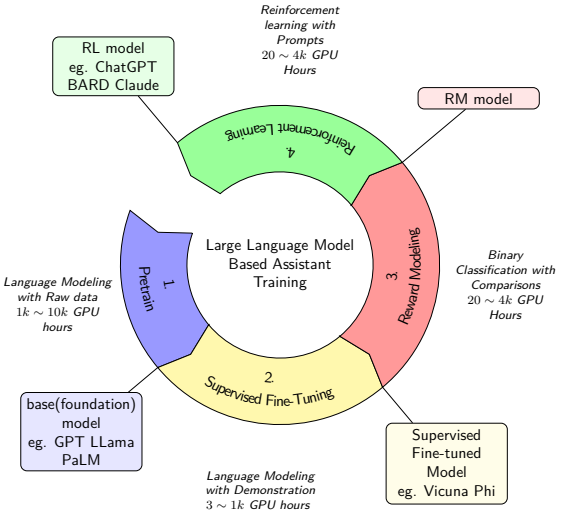


Fig. 2. Training Pipeline for LLM Assistants. In the pretrain phase, the raw textual content is used with a language modeling algorithm, creating the base model. The supervised fine-tuning phase uses ideal responses to specific prompts as a dataset, producing the SFT Model. The Reward Modeling phase introduces preference training through user selections. The final reinforcement learning phase uses prompts and preference scoring to produce the RL Model. The pretrain dataset is high-volume but low-quality, while the next stages require high-quality human or agent generation data.

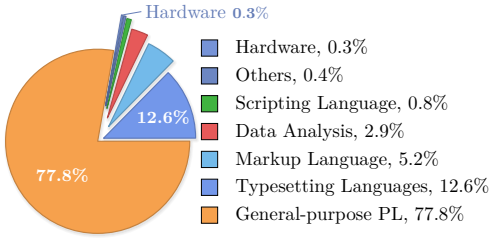


Fig. 3. Classification of code repositories on GitHub by application, reproduction of StarCoder receipt: hardware-related code constitutes 0.3%, with a potential underestimation error of 10%.

raw data. For multimodal models, LLaVA uses LLMs to generate annotations to alleviate data scarcity [44].

Hardware Domain: As demonstrated in Fig. 3, aggregating and analyzing open-source hardware projects by the StarCoder receipt, a data scarcity challenge in the hardware domain, only 1.711GB, stop hardware domain achieved positive LLMs outcomes.

As shown in Fig.4, the *HardwarePhi* dataset is the largest open-source pretrain dataset for the hardware domain by 2024 Spring, yet its scale is insufficient to meet the demands of the pre-training phase for modern LLMs; conversely, any further increase in the dataset's size would require to purchase costly assets from commercial.

2) Supervised Fine-tuning Dataset

SFT is noted for its efficiency and potential to achieve significant outcomes. The diversity of fine-tuning datasets started to emerge once it was recognized that fine-tuning could substantially enhance model performance. Alpaca [45] was the pioneer in releasing a large-scale open-source fine-tuning dataset, choosing a synthetic data approach that propelled synthetic data into the mainstream. Training AI using AI-generated data carries inherent risks, such as the potential for model collapse [46], which may introduce irreversible defects. Consequently, manually curated data retains significant value despite its higher cost.

Natural: OpenAI hired approximately 1,000 remote contractors

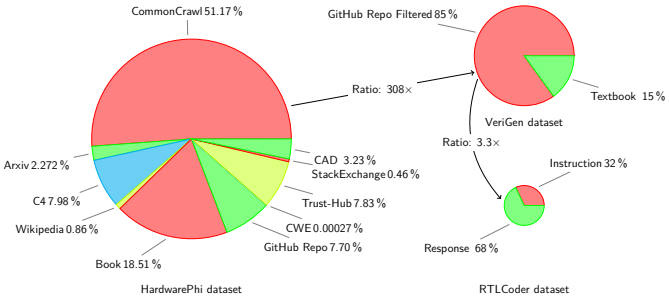


Fig. 4. Three typical datasets in the hardware domain include HardwarePhi, a pretrain dataset, and VeriGen and RTLCoder, which are fine-tuning datasets; the figure illustrates the composition and size ratios.

from Latin America and Eastern Europe from April to December 2021, with around 60% engaged in *data labeling* tasks, producing a substantial volume of images, audio clips, and other data forms to train AI tools [47]. No Robots [48] crafted a manually created dataset of 10,000 entries. OASST1 [49] is a large, high-quality, human-written conversation dataset aimed at generating more natural responses from LLMs.

Synthetic: Alpaca leveraged the GPT-3 API with input from Self-Instruct [50] to collect responses, setting a trend in using OpenAI services for data synthesis. With the release and enhancements of the ChatGPT API, projects such as GPT-4-LLM [51] and multimodal datasets like LLaVA Instruction [44] have utilized the GPT-4 API.

Hardware Domain: As shown in Fig 4, VeriGen [52] and RTL-Coder are two famous SFT datasets that have embarked on distinct paths. VeriGen, through the aggregation of hardware designs from GitHub and textbooks, has amassed a 400MB dataset and has reached its scalability limit, making expanding the dataset size challenging without incurring significant costs.

The RTLCoder has employed GPT-3.5 to generate Verilog source code from random instructions, followed by syntax verification through simulators, yielding $27k$ question-answer pairs. However, general-purpose LLMs exhibit perplexities in hardware-related tasks [53], leading to data quality concerns. An automated generation always results in similar results; the data are remarkably alike, leading to redundancy within the dataset, such as 24 4-bit counters, 11 FIFO buffers, and 42 4-to-1 multiplexers. We see that these hardware implementations in RTLCoder need more practical significance.

3) Reward Modeling and Reinforcement Learning

Reward Modeling emulates human preferences by providing rewards during training to tune LLMs. The ultimate aim of developing and enhancing LLMs is to benefit humanity by assisting people in addressing complex challenges in the real world. Meanwhile, Reinforcement Learning from Human Feedback (RLHF) occurs when a language model generates text and then receives a score or reward from a human annotator that assesses the quality of the text. While Reward Modeling and RLHF are not identical, they are often discussed together due to their shared idea of enhancing performance.

Natural: HH-RLHF [54] is a pairwise dataset designed for training reward models in reinforcement learning to enhance the harmlessness and helpfulness of LLMs. WebGPT [55] is specifically utilized to train the reward model in RLHF, focusing on web-based interactions.

Synthetic: Since this training phase requires aligning the model's outputs with human behavior, no synthetic data meets the performance standards necessary for such applications.

Hardware Domain: In contrast to typical dialogue processing, the hardware domain benefits significantly from robust automated metrics at this stage. Key metrics include PPA (Power, Performance, Area), emulator circuit synthesis reports, and performance reports based on

a fixed test bench [16], [56]. In the software domain, the approach of integrating unit tests, compilation feedback, and OS feedback from the execution process into a comprehensive reward function [57] and limiting the range of LLM generated tokens by additional prediction algorithms [58] offers an innovative and heuristic method. This strategy provides valuable inspiration for potential applications in the hardware domain, where similar techniques could be adapted to enhance the reliability and performance of hardware design processes.

B. Fine tuning technique

Fine-tuning is taking pre-trained models and further training them on smaller, specific datasets to improve performance in a particular task or domain. The techniques for fine-tuning are thus termed because they only minimally alter the model's behavior. Low-Rank Adaptation (LoRA) [59] and Llama Adapter [60] are two common fine-tuning methodologies. LoRA achieves updates through low-rank decomposition, representing weight updates using two smaller matrices called update matrices. These new matrices are trainable and can adapt to new data while maintaining a lower total number of parameters, with the original weight matrix remaining frozen. Llama-Adapter adds a set of learnable adaptation prompts before the input instruction tokens. The instruction output tokens from the input prefix guide the generation of contextually responsive prompts.

C. Limitation of Fine-tuning method on hardware domain

Suppose we have a pre-trained dataset $dataset_{pt}$ and a fine-tuning dataset $dataset_{ft}$, and the function $f_{LLM}()$ represents the training result of an LLM. Ideally, we would expect the following inequality to hold as shown in Equation 1, which suggests a preference for maximizing the benefits derived from the pre-training phase. $|*|$ shows the performance of LLM training on a specific dataset.

$$|f_{LLM}(\text{concat}(dataset_{pt}, dataset_{ft}))| < |f_{LLM}(dataset_{pt})| + |f_{LLM}(dataset_{ft})| + O(1) \quad (1)$$

Ideally, the two datasets should exhibit a certain degree of consistency, as outlined in Equation 2. This consistency ensures that the model's fine-tuning performance can benefit from the foundational knowledge gained during pre-training.

$$\text{Distribution}(F(dataset_{pt})) \cong \text{Distribution}(dataset_{pt}) \quad (2)$$

A challenge in the hardware domain is that the distribution of knowledge in the typical pre-trained datasets for general-purpose LLMs differs significantly from that in the hardware domain. This disparity leads to sub-optimal utilization of the unsupervised pre-training benefits. The significant divergence in the distribution between the pre-trained dataset (implicitly present in the pre-trained model) and the fine-tuning dataset leads to previous fine-tuning performances on raw data [27] that only partially meet expectations.

Given the mismatch between $dataset_{pt}$ and $dataset_{ft}$, we have identified two potential solutions:

1) Modify $dataset_{pt}$ to incorporate significant knowledge from the hardware domain.
2) Alter $dataset_{ft}$ to align more closely with the descriptive norms of general-purpose language models.

Considering the discussion in Section II-A1, implementing the first option using open-source knowledge in the hardware domain is impractical. Therefore, we opted for a natural language style description approach, aligning $dataset_{ft}$ more closely with the data format used in general LLM pre-training.

D. Version Control, Git and GitHub

In data management, version control is crucial in transient and fluctuating data. Among the available tools, Git is the leading choice for managing code and text-based content [61]. GitHub, as an augmentation of the Git, offers an online interface for developers to collaborate on and contribute to projects. It boasts numerous features, such as Commits, Pull Requests (PRs), and Issues, as well as streamlining code versioning, review, and collaboration.

A commit in Git delineates changes made to the files within a repository. Each commit has a unique identifier, usually a hash, along with metadata that includes the author, date, and a message explaining the reason for the change. A commit message succinctly conveys the purpose of the modification, its justification, and potential implications. PRs, on the other hand, enable developers to propose code alterations for integration into another branch. A typical PR encompasses a title, description, multiple commit details intended for merging into the main project, and discussions within the team. Should a PR aim to address a specific challenge or task, it frequently associates with a corresponding Issue.

Issues are used to track and manage bugs, feature requests, tasks, and other relevant concerns in a project. Often initiated by users, these concerns are communicated through feedback. An issue typically includes tags such as *bug*, *enhancement*, and *help wanted*, which help team members quickly identify and address specific concerns.

E. 5W1H problem-solving method

In journalism, the 5H1H is a checklist used to ensure that the first paragraph contains all the essential points of a story. The *5W+1H* term abbreviates six key questions: who, why, what, where, when, and how. This framework encapsulates the primary elements most individuals seek in news reporting, providing a comprehensive approach to gathering and presenting information. In the book *Just So Stories*, British author Rudyard Kipling [62] introduced the *5W+1H* model in 1902. Journalists subsequently adopted this news reporting model, which has become widely utilized. From the perspective of a journalist, to effectively convey a story, it is essential to provide the reader with basic information about six key questions [17], [63]:

1) Who performed the actions reported or experienced the outcomes? (Actor)
2) Why did these actions occur? (Motive)
3) What are the actions, and what are their consequences? (Content)
4) Where did the actions take place? (Location)
5) When did the actions occur? (Time)
6) How are these actions interconnected? (Causality)

III. Workflow

Fig. 1 illustrates the overall workflow of our study. The subsequent sections provide a detailed explanation of each critical step involved in this process.

In Section IV, we begin by discussing the rationale for utilizing version control data as fine-tuning material for Chatbot LLMs, which forms the foundational basis of our approach (see Subsection IV-A). Following this, we outline the criteria for selecting target hardware designs, which not only define the scope of our methodology but also leverage the reliability of data sourced from reputable open-source projects to ensure the validity of our raw data (see Subsection IV-B). We then describe the design of our web crawler, emphasizing how we effectively gather training data and ensure that the collected data is pertinent to the themes of bug detection and fixes (see Subsection IV-C).

In Section V, we first introduce a modeling approach based on the 5W1H framework and discuss how we enhance data using LLMs. This method offers a straightforward, general, and effective modeling strategy that contrasts with the increasingly detailed prompt engineering trends, representing an innovative approach grounded in our rigorous analysis of hardware debugging (see Subsection V-A). Subsequently, we delve into constructing the semi-synthetic dataset, categorized into short code, long code, and documentation. For each

category, we employ distinct modeling strategies to address the context length limitations of LLMs (see Subsection V-B). Subsequently, we demonstrate transforming the raw materials into a fine-tuned dataset. This step involves the final restructuring of the dataset, wherein the data is reorganized into a QA format suitable for fine-tuning the model. (see Subsection V-C).Next, we discuss our model selection process, which spans from smaller models (7B) to larger models (70B), encompassing both general-purpose LLMs and code-specific LLMs (primarily for software), as well as an alternative MoE-based approach (Mixtral 8×7B) (see Subsection V-D). Finally, to facilitate the reader's understanding of subsequent evaluations, we briefly introduce the concept of $pass@k$ in Subsection V-E.

IV. Collect Raw Data: From Isolated to Sequential

In this section, we describe the methodology employed for collecting raw data. Unlike the StarCoder Recipe discussed in Section II-A1, our approach incorporates a version control system, adding a temporal dimension to the data. This modification transforms each data element from an isolated, static hardware design into one that evolves over time. Consequently, this change naturally captures information about hardware modifications, providing a solid foundation for our subsequent modeling efforts.

Fig.5 articulates the framework for constructing the target dataset. This section introduces the lower part of the graph, which is the basis of the whole methodology. We first explore the rationale behind utilizing version control information for fine-tuning LLMs, specifically those operating in chatbot mode. Subsequently, we discuss our selection of hardware projects, followed by a detailed explanation of our web crawling algorithm designed to retrieve all pertinent version control information.

A. Why Version Control Information Can Serve as a Training Dataset for LLMs Operating in Chatbot Modes

The LLM-based Chatbot assistant consistently strives to answer specific questions. The combination of red and orange arrows on the right side of Fig. 5 illustrates a typical real-world scenario: when a user inputs information, the expected response is either a reply or a modification to the input. Although users may not consciously iterate on a specific version, this process mirrors the version control.

The primary purpose of a Version Control System is to log all modifications. Version Control serves as a database that captures snapshots of a project at any given moment and offers a precise explanation of the project's evolution. As Fig. 5 illustrates, different versions are represented by teal circles at the bottom, while black arrows indicate changes. These variations between versions may arise for many reasons, such as feature additions, bug fixes, and performance enhancements. These modifications are directly related to content from previous versions, and more precisely, they often address deficiencies in earlier designs.

Our approach is inspired by the usage patterns of the Chatbot assistant, contrasting with the copilot pattern that often attempts to suggest potential code. Given the similarities in working patterns (user interaction versus version control information), information derived from version control can be utilized as training datasets.

B. Target Hardware Project Selection

The initial step is raw data collection. In this phase, our approach diverges distinctly from other works in the field. Unlike Verigen mentioned in Section II-A2, which utilizes various independent code segments and textbook examples, our criteria for including open-source projects in our dataset are stringent. We filtered the open-source hardware projects based on the following conditions: **1) active, with evidence of updates and iterations, 2) popular, widely used with substantial user feedback, and 3) pragmatically significant, the projects must aim to solve specific, real-world problems.** This

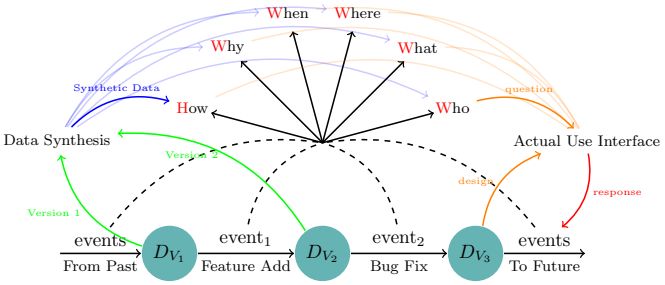


Fig. 5. Methodology for Data Collection: Extracting Version Control Information, Applying 5W1H Journalistic Techniques, and Constructing Datasets for Supervised Fine-Tuning of LLMs. The paths for synthetic data and questions are represented in shades of blue and orange, in which transparent and original functions are equivalent. This indicates only one path is activated per time.

selection process enhances the dataset's quality and improves the performance of the resulting trained LLM.

After careful selection, we have chosen to include the 20 projects in our study. These projects, written in Verilog and SystemVerilog, originate from various organizations and encompass a wide range of components, including SoC, MCU, core, cache, IC, FPGA, APU, wireless module, and GPU. In the following sections, we will introduce some of the typical examples. SoC projects include ibex [64] and OpenTitan [65] from lowRISC. Ibex is a production-quality open-source 32-bit RISC-V CPU core written in SystemVerilog and licensed under Apache 2.0. OpenTitan is an open-source silicon Root of Trust project that enhances transparency and security. It is written in SystemVerilog and licensed under Apache 2.1.

The OpenHW Group contributes to several projects across different domains. The core-v-mcu project [66] is a RISC-V MCU that provides efficient processing capabilities. It is written in SystemVerilog and licensed under the Solderpad Hardware License. Their core architectures, including cv32e40p, cv32e40s, cv32e40x, cv32e41p,) [67] and cve2 [68], focus on RISC-V cores, written in SystemVerilog and utilizing either the Solderpad Hardware License or Apache License. Additionally, cv hpdcache [69] is a high-performance, multi-requester, out-of-order L1 Dcache for RISC-V cores and accelerators, written in SystemVerilog and licensed under the Solderpad Hardware License. The core v polara apu project [70] has 4 RISC-V vector cores connected using the OpenPiton platform, written in SystemVerilog and licensed under the Solderpad Hardware License.

LowRISC's CPU designs include CVA5 [71] and CVA6 [72]. CVA5 is a 32-bit RISC-V processor written in SystemVerilog and licensed under Apache 2.0, while CVA6 is a 64-bit RISC-V processor with a 6-stage pipeline, written in SystemVerilog and licensed under the BSD 3-Clause license. The cvw project from the OpenHW Group is another CPU design written in SystemVerilog.

Additionally, there are projects in other domains: Neopixel TX Core by asm2750 is an IC designed for controlling Neopixel LED strips, written in Verilog and licensed under Apache 2.0; all spark cube by chadharrington is an LED control project comprising a cube with 4096 LED lights, written in Verilog and licensed under MIT; miaow [73] by the Vertical Research Group is a GPU implementing AMD's Southern Islands GPU ISA, written in Verilog and licensed under BSD 3-Clause.

C. Web Crawler design

We developed a web crawler to facilitate version control information collection. A mere git clone operation cannot acquire comprehensive version control information for these projects. Instead, a more sophisticated approach is necessary to extract each project's version history and metadata. Therefore, we employ the GitHub REST API

Algorithm 1: Iterate through GitHub repositories to collect HDL modification

```
Data: GitRepo = ["Repo1", "Repo2", ...]
Result: pre and post-modification of each file
for each repo in GitRepo do
    SolvedIssue = ClosedIssue[label = Bug];
    PR = PR linked to SolvedIssue;
    commits = Get merge PR for the repo;
    for each commit in commits do
        for each file in Get modified HDL files(commit) do
            Get preContent, postContent;
            dataset.Add(preContent, postContent);
        end
    end
end
```

to retrieve detailed information on commits, issues, and Pull Requests (PRs) from repositories. We filtered issues labeled as "bug" or with similar descriptions and then traced them forward to identify the merged PR associated with these issues. From these PRs, we extracted the commits and the corresponding file modifications from these commits. The specifics of our crawling algorithm are presented as pseudocode in Algorithm 1.

Following Algorithm 1, we generate two versions of code, namely preContent and postContent, for each modification made to the hardware design programming codes (.v, .verilog, .vlg, .vh, .sv, .svh extensions) or documents (.txt, .md extensions) within the repository. We systematically crawl through all the commits within the project repository for each hardware project. We gather the files affected by the modifications made during each commit. Utilizing the commit's SHA hash and the filename, we precisely access the content of the files before and after the changes, referred to as preContent and postContent, respectively. Algorithm 1 allows us to capture the modifications between any consecutive versions and store them as paired data entries within the raw dataset.

After the crawl, we obtained 967 code and 125 documentation pairs, which can be utilized for fine-tuning directly. Compared to prior work [27], this represents a significant increase in raw data collected. By introducing patches as a comparative element alongside long code and documentation, we have broadened our coverage to encompass more general scenarios within hardware design projects. Furthermore, the rigorous selection of open-source hardware projects has resulted in a higher-quality dataset than previous work.

V. Semi-Syntectic and Fine tuning Datasets

Following the discussions in Section II-C, it is essential to transform raw data into descriptions that approximate natural language. We focus on code changes designated as *events* rather than contrasting code with code. These events encapsulate the deficiencies of the old version and the enhancements made in the new version, reflecting the intellectual effort and time invested by project maintainers.

In Equation 3, issues and patches are defined to minimize human involvement in this process. Within projects documented by version control systems, for any two consecutive versions i and j of the same file, we leverage the variations between Design Version$_i(D_{v_i})$ and Design Version$_j(D_{v_j})$ to strategically identify potential issues $issue_i$ inherent in D_{v_i}, and solution is $patch_j$.

$$\begin{aligned} issue_i &= \text{Code}_{D_{V_i}} \setminus \text{Code}_{D_{V_j}} \\ patch_j &= \text{Code}_{D_{V_j}} \setminus \text{Code}_{D_{V_i}} \end{aligned} \tag{3}$$

An issue refers to code that existed in the old version but was removed in the new version; conversely, a patch denotes code added in the new version relative to the old one. The description of such events can be effectively articulated using the 5W1H method. The descriptions derived from the raw data set are referred to as a **semi-synthetic** dataset because they are produced from both new and old version data, employing a fixed event journalism description method, and generated by an LLM, but are constrained to fixed content.

A. 5W1H in hardware domain

We redefine the 5W1H framework specific to the hardware domain as follows:

1) **Who**: To whom does the module for the code belong?
2) **What**: What is the modification addressing the bug?
3) **Where**: Where is the issue located?
4) **Why**: Why is the modification necessary?
5) **When**: When was this specific change prompted?
6) **How**: How was the modification executed?

B. Semi-Synthetic Dataset

Fig. 5 depicts our approach to directed semi-synthetic data generation. We feed the collected raw data and these defined questions into a general-purpose LLM to extract answers. This process yields a four-part metadata group comprising two versions of hardware design, the questions, and responses from the LLM agent.

Due to the inherent limitations of transformer models, an LLM can only effectively utilize data within a specified content window. This necessitates merging the input within this content window. Although recent advancements have significantly increased the allowable content window length, the scale of hardware designs often surpasses these limits. More critically, our method, which requires input from two versions of hardware design simultaneously, necessitates a trade-off in the overall design length. Consequently, we have devised three distinct generation templates in below to mitigate the issues arising from the doubled spatial occupancy.

1) Short Code

When a hardware design's tokenized length is less than 2048 tokens, we classify it as *shortcode*. We use the following template for such cases: D_{V_i}, $D_{V_{i+1}}$ combined with the 5W1H questions.
Example: In a hardware modification, D_{V_1} is updated to D_{V_2}. Why is the modification necessary?

2) Long Code

For designs exceeding 2048 tokens post tokenization, we classify them as *long code*. We employ the following template: D_{V_i}, $patch_{\Delta i \to i+1}$, combined with the 5W1H questions. Although theoretically, no information is lost for this change, using a patch inherently raises the expertise required of the user, representing a trade-off in performance.
Example: In a hardware modification, D_{V_1} receives an update $patch_{\Delta i \to i+1}$. What is the modification addressing the bug?

3) Documentation

We apply the same approach to documentation as the long code.

Our semi-synthetic dataset comprises 1091 entries detailing hardware design changes. This includes 156 instances classified under long code and 125 instances of hardware documentation.

Table II illustrates a typical example of automated generation used in our study. We acquired pairs of old and new version design codes from the raw dataset from Section IV-C. These code pairs originate from the OpenHW Group's Core-V microcontroller project, specifically from the RTL source code simulation-related portion, identified as a2_bootrom.sv. This dataset includes six Research Questions (RQs) corresponding to the 5W1H method discussed in Section V-A.

We label the old and new source codes and pose specific questions to an LLM agent based on these versions. The questions are designed to elucidate the modifications and their implications, utilizing the 5W1H framework. The responses generated by the general LLM

TABLE II
CLAUDE 3 SONNET TRANSFORMING VERSION CONTROL INFORMATION INTO SEMI-SYNTHETIC HARDWARE DEBUG DATA USING THE 5W1H APPROACH

Old version design code

```
module a2_bootrom #(
    parameter ADDR_WIDTH = 11,
    parameter DATA_WIDTH = 32
) (
    input  logic                  CLK,
    input  logic                  CEN,
    input  logic [ADDR_WIDTH-1:0] A,
    output logic [DATA_WIDTH-1:0] Q
);
    logic [31:0]   value[(2**ADDR_WIDTH)-1:0];

    initial begin
        $readmemh("mem_init/boot.mem", value);
    end

    always @(posedge CLK) begin
        Q <= value[A];
    end
endmodule
```

New Version design code

```
module a2_bootrom #(
    parameter ADDR_WIDTH = 11,
    parameter DATA_WIDTH = 32
) (
    input  logic                  CLK,
    input  logic                  CEN,
    input  logic [ADDR_WIDTH-1:0] A,
    output logic [DATA_WIDTH-1:0] Q
);
    logic [31:0]   value[(2**ADDR_WIDTH)-1:0];
    logic [31:0] read_data;

    initial begin
        $readmemh("mem_init/boot.mem", value);
    end

    always @(posedge CLK) begin
        if (CEN == 0) begin
            read_data <= value[A];
            Q <= value[A];
        end else begin
            Q <= read_data;
        end
    end
endmodule
```

***Source**: OpenHW Group / CORE-V MCU: rtl/simulation/a2_bootrom.sv*

RQ1: What is the function of Hardware design code?

Response:

The original hardware design code defines a module for a boot ROM (Read-Only Memory), which is a type of memory that stores firmware. When the system is powered on, the CPU may read this boot ROM to load and execute initial boot or system initialization code. The module is parameterized to allow for different address and data widths, making it flexible for various memory sizes. The ROM content is loaded from a file (`mem_init/boot.mem`) at simulation time, and the data is output (`Q`) based on the input address (`A`) on every positive edge of the clock (`CLK`).

RQ2: What bug was fixed in Hardware design code?

Response:

The bug fixed in the modified code addresses a situation where the ROM output (`Q`) is updated on every positive clock edge, regardless of whether the chip enable signal (`CEN`) is active or not. In the original code, the absence of checking the `CEN` signal means the ROM is always enabled, potentially leading to unintended reads and conflicts with other memory devices or operations.

RQ3:Where are the statements and variables related to the bug? Which modules, circuits, interfaces or signals are involved in the bug?

Response:

Modules/Circuits/Interfaces/Signals Involved: The bug involves the `always @(posedge CLK)` block, specifically how it interacts with the `CEN` (Chip Enable) input signal, and the `Q` output signal. The `value` memory array and the address input `A` are also involved as they are part of the data read process.

RQ4:When was the bug introduced((requirements analysis stage, design stage, coding stage, etc.))? What impact will it have on the entire hardware?

When Introduced: This bug was likely introduced during the coding stage. The initial design and requirements analysis might have correctly identified the need for a chip enable mechanism, but the implementation failed to account for it correctly.
Impact: Without considering the `CEN` signal, the ROM could interfere with other components on the bus by driving the `Q` output when not selected. This could cause data corruption or erroneous system behavior, especially in systems designed to switch between multiple memory devices.

RQ5: How to fix the bug?

Response:

The bug is fixed by adding a conditional check for the `CEN` signal within the `always @(posedge CLK)` block. If `CEN` is low (active low convention), the ROM is enabled, and the read operation proceeds, storing the read data in a temporary variable (`read_data`) before assigning it to `Q `. If `CEN` is not active, the output `Q` retains its previous value, effectively disabling the ROM from interfering with other operations on the bus.

RQ6: Why did the modification occur?

Response:

The modification was necessary to ensure that the ROM only responds when explicitly enabled by the `CEN` signal. This change makes the module more robust and compatible with systems that multiplex access to various memory devices, preventing the ROM from outputting data when it is not selected. This adheres to common hardware design practices where memory devices are controlled by enable signals to manage their interaction with a shared bus.

agent are illustrated in Table II. This structured approach enables a systematic analysis of the code evolution within the project, leveraging the capabilities of advanced language models to interpret and describe changes in an understandable format.

C. LLM Fine-tuning Dataset

We propose transforming our semi-synthetic dataset into a fine-tuning dataset by reorganizing the semi-synthetic dataset introduced in Section V-B. This transformation is detailed as Equation 4.

$$\left.\begin{matrix} D_{V_i} \\ \text{5W1H Question} \end{matrix}\right\} \text{User-interested Question}$$
$$\text{Synthetic data} \rightarrow \text{Theoretical Response} \tag{4}$$

For the semi-synthetic dataset mentioned in Section V-B, each data point consists of 1) a pair of consecutive hardware design versions, 2) a fixed question, and 3) the corresponding model-generated response. This format mapped to a new format that aligns more directly with practical application needs as shown in Equation 5.

$$(DV_i, DV_{i+1}, \text{Fixed Question}, \text{Response}) \rightarrow$$
$$(\text{User-interested Question}, \text{Theoretical Response}) \tag{5}$$

Through this map, we generate a dataset that can be utilized for fine-tuning LLMs, thus aligning synthetic generation with practical operational demands. Following the 5W1H model, each entry in the semi-synthetic dataset has been subdivided into six distinct elements, resulting in 6546 data points. Regarding the reliability of the generation task, we observed a single-generation accuracy of 99.7%. Out of the $1,092$ pairs of hardware instances, only four instances resulted in slightly inaccurate responses. However, in these four cases, when the conversation was continued, the LLM provided correct answers in the subsequent round of responses. As a result, all data in the final dataset are accurate. To further ensure accuracy and mitigate the risk of undetected hallucinations, we employ cross-validation using the commercial LLM Claude 3 Sonnet and the open-source LLaMA 2 70B. We conducted multiple generations (Claude twice and LLaMA 2 three times) and compared the results, as hallucinations are typically random and unlikely to converge in the same direction across different models. Due to variations in tokenizer specifications and content length restrictions among different models, models may not utilize the entire dataset during fine-tuning.

D. Fine tuning Target Model Choice

As discussed in Section II-C, for a model to show improved performance and generalization capabilities on specific problems after fine-tuning, the fine-tuning dataset must have a similar distribution to the pre-training dataset. However, while open-source models are available, the datasets used for training these models are often proprietary and not publicly disclosed. Consequently, we must navigate two variables within the *"black box"* of foundation models: 1) the differences in performance attributable to the model design, including architecture and parameter size, and 2) the variability in the quality of different pre-training datasets. The models we utilized are listed in Table III, which includes details such as model name, the total number of parameters, tokenizer used, context window size, and average scores across six benchmarks indicating the model's intelligence: AI2 Reasoning Challenge [74] $(25-shot)$, HellaSwag [75] $(10-shot)$, MMLU [76] $(5-shot)$, TruthfulQA [77] $(0-shot)$, Winogrande [78] $(5-shot)$, and GSM8k [79] $(5-shot)$. The scores for GPT-3.5 were provided by Vellum.ai [80], while the remaining scores were sourced from the Huggingface LLM leaderboard [81].

Our selection encompasses a diverse range of mainstream LLMs, including open-source options such as LLama2, CodeLLama, Phi, Mistral, Falcon, and Dolly, as well as private model ChatGPT-3.5 Turbo, which offers a paid fine-tuning service. These foundation models exhibit a wide range of capabilities, with average scores on six benchmark tests varying from 22.83 to 68.42, which we consider indicative of the models' inherent intelligence prior to fine-tuning.

TABLE III
LLM FOUNDATION MODELS INFORMATION

Model	Parameter	Tokenizer	Content Length	Average Score
Dolly 3B [82]	3B	Pythia	2048	22.83
Dolly 7B [82]	7B	Pythia	2048	39.24
Codellama 7B [23]	6.74B	CodeLlama	16384	39.81
Codellama 13B [23]	13.02B	CodeLlama	16384	43.35
Codellama 70B [23]	68.98B	CodeLlama	16384	58.93
Llama2 7B [83]	6.74B	Llama	4096	50.97
LLama2 13B [83]	13.02B	Llama	4096	55.69
LLama2 13B chat [83]	13.02B	Llama	4096	54.91
Llama2 70B [83]	68.98B	Llama	4096	67.87
Phi1.5 [84]	1.5B	codegen-mono	2048	47.69
Phi2 [85]	2.78B	codegen-mono	2048	61.33
Mixtral 8×7B [86]	46.7B	Llama	32000	68.42
GPT3.5 Turbo	N/A	cl100k_base	4096	65.46
Stablecode 3B [87]	2.8B	GPTNeoX	16384	41.53
Falcon 7B [88]	6.92B	Falcon	2048	44.17

We employed two fine-tuning approaches for models with a size of 7 billion parameters or fewer: LoRA and Llama Adapter. These methods were chosen to enhance model adaptability without extensively increasing parameter count. For models exceeding 7 billion parameters, computational resource constraints led us to use Quantized LoRA (QLoRA) [89], which quantizes the base model's weights to $4-$bits, thus reducing memory usage.

E. LLMs Performance Metric

A common metric LLMs use to evaluate domain-specific tasks (code generation evaluation, for example) is the $pass@k$ metric [6]. The metric roughly indicated the probability of obtaining one correct response when generating k prompt completions. For this work, we use the expression in Equation 6 described as a numerically stable and statistically unbiased method for computing $pass@k$.

$$pass@k = \begin{cases} 1, & \text{if } n-c<k \\ 1-\prod_{i=n-c+1}^{n}(1-\frac{k}{i}), & \text{otherwise} \end{cases} \tag{6}$$

where c is the number of correct responses, and n is the total number of responses generated by the model. We have selected $pass@1$ as our evaluation metric, where $n=10$ represents each LLM having ten attempts, and c denotes the number of correct responses.

VI. EXPERIMENT

Our experimental section consists of two parts. The first part details the technical aspects and results of fine-tuning LLMs, focusing on SFT configuration and training performance. It also provides a comprehensive overview of the fine-tuning process. The second part of the experiment evaluates the performance of these fine-tuned models on a validation set of tasks related to detecting and fixing hardware vulnerabilities. It also includes assessing the models' ability to recognize $(pass@1)$ vulnerabilities under test.

A. Supervised Fine-tuning

We employed three training platforms and fine-tuning strategies to balance the selection of model parameter scale and training costs, as detailed in Table IV. The *Training Token* illustrates each model's capacity to process training datasets. Different LLMs employ tokenizer implementations bound to their respective models to accomplish the initial step of mapping textual information to embedding spaces. Different tokenizer choices and context window sizes result in varying

TABLE IV
TRAINING CONFIGURATION PARAMETERS FOR LARGE LANGUAGE MODELS

model	Training Token	Accerator	Duration (Hour)	Epoch	Optimizer	Learning Rate	Learning Rate Scheduler		PEFT Adapter	Training Loss
							Decay	Warmup Fraction		
Phi1.5	4502221	1 A100 40G	0.94	3	AdamW	0.0003	Cosine	0.01	LoRA (R:8, alpha:16, dropout:0.05)	1.0103
Phi2	4502221	3 A100 40G	1.55							1.3057
Dolly3B	4410489		0.69							1.1636
Dolly7B	4410489		1.39							1.1190
Falcon7B	4348393		2.09							1.2217
Llama2 7B	5556856		7.16							0.5728
StableCode 3B	18246844		7.12							0.8427
CodeLLama7B	20550919		17.16							0.4267
Llama2 13B	5400089	1 A10 24G	15.45	4	PagedAdam	0.0002		0.03	QLoRA (R:8, alpha: 16, dropout: 0.01, quantization: 4bits, type:bfloat16)	4.5653
Llama2 13B-chat	5400089		15.55							4.2543
CodeLLama13B	5400089		15.52							1.2156
LLama2 70B	5400089	1 A100 80G	37.35							1.5546
CodeLLama 70B	5400089		37.13							1.2444
Mistral8 × 7B	5305148		16.34							1.7434
Phi1.5	4502221	1 A100 40G	3.44	5	AdamW	0.0001		0.01	Llama-Adapter	1.5169
Phi2	4502221	3 A100 40G	4.19							0.8903
Dolly3B	4410489		1.5							0.5503
Dolly7B	4410489		5.63							0.6819
Falcon7B	4348393		5.66							0.9781
Llama2 7B	5556856		5.04							0.4136
StableCode 3B	18246844		7.88							0.9743
CodeLLama7B	20550919		16.11							0.2929
ChatGPT3.5 Turbo 0125	47236815	N/A	20.6	3	N/A	N/A	N/A	N/A	N/A	0.9846

amounts of usable data from the same dataset for SFT in this experiment. For instances where the length of the data exceeds the context window size, we opted to **discard** the data rather than truncate it. This preference stems from the typical structure of our data, which consists of a sequence in Equation 7.

$$\{\textit{hardware design raw code}, \quad \textit{question}, \quad \textit{response}\} \tag{7}$$

An excess length often indicates that the hardware design component occupies much space, potentially leading to truncation of the response. This could sometimes result in entries not containing the response. Such truncation undermines the model's ability to learn the intended tripartite relationship between design, question, and answer during fine-tuning. Therefore, we chose to discard these overly lengthy entries to prevent the model from encountering additional confusion during training.

The *Accelerator* column lists the accelerators used for each training project. Four configurations were provided: a single Nvidia A100 40G, three Nvidia A100 40Gs, a single Nvidia A10 24G, and a single Nvidia A100 80G. The column *Duration* indicates the training duration in hours, followed by the number of training epochs. This process, known as fine-tuning, primarily involves minor adjustments to the model's original parameters. Our training was conducted over 3, 4, and 5 epochs, which mitigates the risk of overfitting: a shorter training cycle is less likely to significantly alter the original parameters to fit the fine-tuning dataset, reducing the likelihood of overfitting.

The *learning rate* and *learning rate scheduler* columns indicated the learning rate and its adjustment strategy. We implemented a cos Decay strategy to gradually reduce the learning rate, finely controlling the pace of parameter updates during training. Additionally, a Warmup Fraction was used during the initial phase of training, where the learning rate gradually increases from a lower baseline to the set initial value over a certain proportion of the total training time, enhancing the model's stability in the initial stages. The *PEFT Adapter* column provides details of the PEFT Adapter, including rank, quantization, and others.

Training Loss column shows the model's efficacy on the training dataset, illustrating how effectively the model encodes the knowledge contained within that dataset. This metric serves as an indicator of the model's foundational performance, exclusive of its generalization capabilities. This column reports the model's loss on the training set upon completion of training. Although the training duration using the Llama Adapter is longer than that with LoRA, it demonstrates superior performance on training metrics. Consequently, in subsequent analyses, only the versions of smaller models fine-tuned using the Llama Adapter will be tested. The

In the experiments, QLoRA's memory usage is 66.38% of that of LoRA, while the LLama adapter consumes approximately 119.76%. Given these figures, adapter training exhibits superior performance compared to LoRA. Simultaneously, QLoRA conserves memory, facilitating the training of larger models. The benefits of increased model scale outweigh the precision losses and the negative impacts of the LoRA method. Consequently, QLoRA offers advantages in fine-tuning large models under resource constraints.

B. Validation Dataset Test Bench

We have selected a subset of 15 pairs (1.3%) from 1092 hardware iterations to serve as our validation dataset. As outlined in Table V, this dataset includes one negative test case, with the remaining fourteen encompassing a variety of updates: adding features, enhancing validation, correcting bugs, and patching potential vulnerabilities. We assess the model's performance on hardware debugging tasks before and after fine-tuning. We define bug detection as the process in which the LLM identifies issues within the hardware, and bug fixing as the process of providing effective solutions to those identified issues. The outcomes on the validation dataset are classified into four levels, utilized in Table VI and Table VII:

- × Bug could not be identified.
- △ Bug was detected but described inaccurately.
- ◯ Bug was correctly identified, but no accurate correction suggestion was provided. (Bug Detection task)
- ✓ Provide Appropriate correction advice. (Bug fix task)

We also include $pass@1$ for hardware bug detection as a reference.

TABLE V
15 TEST BENCH INSIDE THE VALIDATION DATASET

#	Module	Function	Code Changes
1	piso_shift_register	Parallel-In Serial-Out (PISO) shift register	data_reg <= 0; → data_reg <= {WIDTH{1'b0}}
2	pinmux_wkup	For a pin multiplexer wakeup control	cnt_q == wkup_cnt_th_i → cnt_q >= wkup_cnt_th_i
3	riscv_hwloop _controller	Manage hardware loops in a RISC-V processor	current_pc_i == hwlp_end_addr_i[i] → current_pc_i + 4 == hwlp_end_addr_i[i]
4	cv32e40x_wb _stage_sva	SVAs for write-back (WB) stage	added new assertion (a_nonlsu_error)
5	aes_mix_single _column	mix a single column of the state matrix in AES	assign z_muxed[0] = (op_i == CIPH_FWD) ? 8'b0 : z[0]; → assign z_muxed[1] = (op_i == CIPH_FWD) ? 8'b0 : z[1];
6	uart_transceiver	UART transceiver module for the Milkymist VJ SoC project	if (enable16 && tx_busy) begin → if (1 && tx_busy) begin
7	qf_rhw	A module to store, update, and output data	Negative case
8	prim_subreg	A programmable sub-register within a larger digital system	wr_en_buf and q_buf are unnecessary → remove
9	ROM	Read-Only Memory	Addressing starts from 8'h01 instead of 8'h00, incorrect branching instructions and data values
10	jtagdpi	A DPI component for a JTAG interface	ctx = 0; → ctx = null;
11	class chip_sw _lc_ctrl_ transition_vseq	UVM	get_otp_token uses c_dpi_cshake128, potentially bypasses specific decoding/manipulation of LC CSRs
12	NICNAC16	Control seven-segment display and generate a CPU clock	parameter MAX_COUNT = 5_000_000 -1; → parameter MAX_COUNT = 2_000_000 -1;
13	a2_bootrom	A verilog module for a boot ROM	Q <= value[A]; without checking CEN
14	cube_controller	LED cube	Add a test_panel_select_n signal for test panel mode
15	cv32e40x_pma	PMA check mechanism for the CV32E40X processor module	Added check for misaligned accesses handled by PMA configuration

C. Performance on Validation dataset

To illustrate our approach to bug detection and fixing, we used a specific example from [90]. For the Bug Detection task, the conversation with the LLM would proceed as follows:

- **User:** There may be a bug in the following hardware design code. Identify the bug and Locate the statements and variables related to it.*Hardware Design [90]*[2]
- **FT-LLama2 Response:** The bug pertains to the handling of the `op_i` signal during the conditional assignment to `z_muxed`. In the original code, the `z_muxed` values were assigned to zero unconditionally for the forward cipher operation (`CIPH_FWD`) but did not explicitly handle the inverse cipher operation (`CIPH_INV`), potentially causing incorrect behavior during decryption. The bug is related to the following blocks and variables:
  - **Variables**: `z_muxed[0]` and `z_muxed[1]`
  - **Statements**:
    * `assign z_muxed[0] = (op_i == CIPH_FWD)? 8'b0 : z[0];`
    * `assign z_muxed[1] = (op_i == CIPH_FWD)? 8'b0 : z[1];`

For the Bug Fix task, the conversation would proceed as follows:

- **User:** There may be a bug in the following hardware design code. Explain how to fix the bug. *Hardware Design [90]*
- **FT-LLama2 Response:** The bug would be fixed by adding a condition to check if the operation performed is `CIPH_INV` (cipher inverting). When this condition is met, the `z[0]` is assigned to `z_muxed[0]` and `z[1]` is assigned to `z_muxed[1]`. This ensures the code always outputs the correct data based on the current operation mode.

Table VI shows the performance of untrained models on the validation dataset. Most original models, especially those with fewer parameters, exhibited zero performance on both tasks under consideration. We specifically show Phi1.5 and CodeLLama7B as representative models with this minimal performance, while these models may achieve comparable scores to larger models on specific LLM metrics.

Among the open-source models, LLama2 70B had the best performance, completing $3(20\%)$ of the Bug Detection tasks (achieve ◯) and $1(6.7\%)$ of the Bug Fix tasks (achieve ✓), surpassing the performance of the proprietary, paid LLM, ChatGPT3.5 Turbo. The best-performing model overall was ChatGPT4, which accomplished $5(33.3\%)$ Bug Detection tasks and $2(13.3\%)$ Bug Fix tasks.

[2]In the actual conversation, here is the hardware design code from [90]

TABLE VI
ORIGINAL MODEL PERFORMANCE OF BUG DETECTION AND BUG FIX TASKS ON VALIDATION DATASET

Module #	ChatGPT3.5 Turbo	ChatGPT4 Turbo	LLama2 70B	Phi1.5	Codellama 7B
1	×	×	×	×	×
2	△	◯	×	×	×
3	△	△	×	×	×
4	×	×	×	×	×
5	△	◯	◯	×	×
6	×	△	×	×	○
7	×	◯	×	×	×
8	×	✓	△	×	×
9	×	△	×	×	×
10	△	△	×	×	×
11	△	△	◯	×	×
12	△	△	×	×	×
13	✓	✓	✓	×	×
14	×	×	×	×	×
15	×	×	△	×	×
Bug Detection	1(6.7%)	5(33.3%)	3(20%)	0	1(6.7%)
Bug Fix	1(6.7%)	2(13.3%)	1(6.7%)	0	0

As demonstrated in Table VII, there was a noticeable improvement in the performance on two specific tasks after fine-tuning. Specifically, we observed that while models with a capacity under 7B could exhibit some improvement through fine-tuning (from 0% to 6%), these changes are not particularly significant. Although these smaller models are more accessible to train and less costly, their baseline performance is relatively poor, and the capacity to assimilate domain-specific knowledge through fine-tuning is limited.

Models with 7B parameters reach the performance levels of the untrained GPT-4. However, despite having similar parameter counts, differences in hardware-specific knowledge in their pre-training datasets lead to varying performance outcomes post-fine-tuning. Among the four 7B models, Dolly 7B performed the weakest, showing almost no correlation with hardware content. Falcon 7B ranked slightly higher but failed to complete bug-fix tasks, indicating minimal hardware knowledge. In contrast, Codellama7B and Llama2 7B, both from the Meta Llama project, performed better, suggesting that their shared general-purpose code datasets positively influenced their hardware task performance. Despite similar performance in bug detection (53.3% *vs.* 53.3%) and bug fix tasks (13.3% *vs.* 6.7%), CodeLLama is superior due to its longer context window length, which is more advantageous in coding environments.

As model size exceeded 7B parameters, quantization became necessary due to resource constraints, yet the benefits of larger models remained significant despite potential performance degradation. In bug fix tasks, the performance of Codellama 13B (20%) exceeded that of Codellama 7B(13.3%), reflecting the advantages of increased parameter count. However, in bug detection tasks, Codellama 13B saw a performance decline (33.3%) compared to the 7B version (53.3%). A similar pattern was observed with Llama2.

When further scaling up the models, quantization allowed for using even larger models under limited computational resources, which helped mitigate the adverse effects of reduced precision. Larger models like Llama2 70B and Codellama 70B exhibited exceptional performance, with Llama2 70B completing 60% of bug detection tasks and 40% of bug fix tasks. The achievements are notable, given that these tasks were framed in real-world scenarios. The Mistral $8 \times 7B$ model, with a smaller scale (56B parameters) and lower memory requirements, still demonstrated commendable performance.

In contrast, the performance of ChatGPT3.5 Turbo, fine-tuned with

TABLE VII
FINE-TUNED MODEL PERFORMANCE OF BUG DETECTION AND BUG FIX TASKS ON VALIDATION DATASET

Module #	Phi1.5	Phi2	Dolly 3b	Dolly 7b	StableCode 3b	Falcon 7b	Codellama 7b	Codellama 13b	Codellama 70b	Llama2 7b	Llama2 13b	Llama2 13b chat	Llama2 70b	mixtral 8×7b	Finetuned GPT3.5 Turbo	Ablation LLama2 70B
1	△	△	△	△	○	△	×	×	×	✓	×	×	×	×	✓	×
2	×	×	×	×	×	×	△	×	×	○	△	△	×	×	○	×
3	×	×	×	×	×	×	○	✓	△	○	○	✓	✓	✓	○	×
4	×	×	×	×	×	×	○	✓	✓	×	✓	✓	✓	×	×	○
5	×	×	×	×	×	×	✓	○	○	○	✓	○	✓	✓	○	×
6	×	×	×	×	×	×	○	○	△	○	△	△	✓	△	△	×
7	×	×	○	×	×	×	○	△	○	△	△	△	✓	✓	○	✓
8	×	×	△	×	△	✓	×	○	△	○	×	△	✓	△	×	×
9	△	×	×	○	×	×	✓	×	✓	×	✓	○	×	✓	△	×
10	○	△	×	△	△	×	○	✓	○	○	△	✓	○	✓	✓	×
11	×	×	×	×	×	○	△	△	△	△	△	△	△	△	△	×
12	×	△	×	△	○	○	△	△	✓	○	△	△	△	○	△	×
13	△	×	×	×	×	×	○	○	✓	△	△	×	○	○	✓	○
14	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
15	×	×	×	×	×	×	×	○	✓	×	○	✓	○	△	×	×
Bug Detection	1(6.7%)	0	1(6.7%)	1(6.7%)	2(13.3%)	3(20%)	8(53.3%)	5(33.3%)	8(53.3%)	8(53.3%)	5(33.3%)	6(40%)	9(60%)	7(46.7%)	7(46.7%)	3(20%)
Bug Fix	0	0	0	0	0	1(6.7%)	2(13.3%)	3(20%)	5(33.3%)	1(6.7%)	3(20%)	4(26.7%)	6(40%)	6(40%)	3(20%)	1(6.7%)

OpenAI's paid service, did not show a clear advantage over other models. Considering the fine-tuning cost of nearly $400, compared to about $50 for tuning other large models on the cloud, ChatGPT3.5 does not represent a cost-effective investment.

Concurrently, Table VIII illustrates that when considering the $pass@1$ result for the bug detection task, GPT-4 stands out as the only LLM with the potential to address the challenge effectively. GPT-4 can reliably complete the task for 6% of the designs, generates at least one correct answer in 12% of the cases after five attempts, and achieves a correct response once every ten tries in 20% of the tasks. The performance of others is negligible by comparison; for instance, GPT-3.5, on average, requires 30 attempts to produce a correct answer, while LLama2 13B need 142 attempts on average.

Significant performance improvements are observed post-fine-tuning in medium—to large models. The fine-tuned LLama2 70B averages just 2.6 attempts to generate a correct answer. Despite its inability to generate correct analyses in 40% cases, expanding the dataset size, which is our method's strength, could mitigate this issue.

In our study, we also conducted an ablation experiment on our best-performing model, Llama2 70B. We retrained the model using only the collected version control dataset, omitting the 5W1H modeling while keeping the fine-tuning parameters. The results revealed that, while the ablation model solved some new problems compared to the original version, it lost the ability to solve a subset of previously handled issues. As a result, the overall performance of the ablation model was comparable to that of the original model. However, both models significantly underperformed compared to the version fine-tuned with the semi-synthetic dataset incorporating 5W1H. These findings demonstrate that the semi-synthetic dataset and the 5W1H modeling approach can significantly enhance model performance.

In conclusion, the performance tests on hardware debugging tasks conducted with our models after fine-tuning demonstrate that our dataset enhances the performance of general-purpose LLMs in the hardware domain. All the results above effectively validate the efficacy of our semi-synthetic dataset construction methodology.

VII. RELATED WORK

Typical prompt-based approaches [91], [92] implicitly include efforts to reduce the dependency on general-purpose LLMs for hardware expertise. These methods often involve manually crafted complex tutorials [93] or a random combination of basic designs [16]. However, our approach, which utilizes version control information and the 5W1H pattern, offers three advantages:

1) Fully automated process, using mature LLMs for a generation. The cost of data generation is linearly related to the scale of the data and does not depend on labor with specific expertise.
2) Fully directed generation. Although no additional tutorials or prompts are provided to constrain the behavior of the LLM, the continuity between two versions of the same design ensures that the generated content is inherently aligned with the direction of the development iterations, much like how two points on a plane dictate the direction of a line.
3) Comprehensive problem descriptions and solutions. Although we do not always require a full 5W1H description, this framework thoroughly summarizes the issues and solutions in a hardware design, allowing for further process automation.

Compared to the approach [15], [94], our work does not rely on Retrieval-Augmented Generation, which shows performance degradation in non-demo scenarios, and it is not dependent on data specifically constructed for particular LLMs. We have observed that the data quality generated by GPT-3.5, GPT-4, and Mistral 8×7 is remarkably similar, indicating that we do not expect general-purpose LLMs to possess the capabilities of hardware domain experts. This assumption aligns with the current performance standards of modern general-purpose LLMs. Additionally, our approach does not depend on proprietary datasets from corporations, and we believe our method maintains excellent performance across any dataset, demonstrating considerable versatility.

VIII. DISCUSSION

A. Supervised Fine-Tuning Enhances LLM Performance in Hardware Domain Tasks

SFT with our dataset in our testing led to significant performance improvements for LLMs. For models with non-zero success rates in both pre-and post-tuning phases, the performance on the bug detection task improved by 287.5%. For the bug fix task, there was a 400% increase. A comparative analysis across different models revealed that the fine-tuned ChatGPT3.5 Turbo exceeded the performance of GPT4, while Codellama 7B and Llama2 7B outperformed the original 70B version. Despite these models not being designed explicitly for hardware-related tasks, the semi-synthetic dataset significantly enhanced their capabilities in this domain. Contrasting with previous efforts, our work transitions from a proof-of-concept to a scalable approach for dataset generation and validates its effectiveness in real-world scenarios.

B. Pretrain still effect finetuned model

We observed that the intrinsic traits of the original models, shaped by their pre-training datasets, persisted even after fine-tuning with our dataset. These traits are especially noticeable in the models' response styles, which differ markedly across various architectures and sizes.

For instance, the Llama2 13B model tends to generate descriptions in natural language, evident in its detailed explanations of hardware

TABLE VIII
$pass@1$ OF BUG DETECTION TASK ON VALIDATION DATASET

	Original Model								Fine-tuned model															Ablation
DUT	GPT 4	GPT 3.5	Mixtral 8×7B	Codellama 70B	Llama2 70B	Code-llama 13B	Llama2 13B chat	Llama2 13B	GPT 3.5	Mixtral 8×7B	Codellama 70B	Llama2 70B	Code-llama 13B	Llama2 13B chat	Llama2 13B chat	Codellama 7B	Llama2 7B	Dolly 7B	Dolly 3B	Falcon 7B	StableCode 3B	Phi 1.5	Phi 2	LLama2 70B
1	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0.1	0	0	0	0.1	0	0	0
2	0.1	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0
3	0	0	0	0.1	0	0	0	0	0.3	0.2	0	1	0.1	0.2	0.4	0.2	0.1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0.7	0.1	0.1	0.2	0.2	0.1	0	0	0	0	0	0	0	0.5
5	0.1	0	0.2	0	0.2	0	0	0	0.3	0.5	0.2	1	0.1	0.2	0.2	0.2	0.2	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0.7	0.4	0	0	0.2	0.1	0	0	0	0	0	0	0
7	0.2	0.2	0	0	0	0	0	0	0.2	1	0.6	0.3	0	0	0	0.1	0	0	0.1	0	0	0	0	0.3
8	0.1	0.1	0.3	0.1	0	0	0	0	0	0	0	1	0.1	0	0	0	0.1	0	0	0.1	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0.9	0.2	0	0	0.1	0.5	0.2	0	0.1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.6	1	1	0.1	1	0.4	0.1	0.2	0.3	0	0	0	0	0.1	0	0
11	0	0	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0
12	0	0	0	0.1	0	0.1	0	0	0	0.2	0.2	0	0	0	0	0	0.2	0	0	0.1	0.1	0	0	0
13	1	0.2	0.4	0.5	0.2	0.1	0.1	0.1	0.2	1	0.5	0.9	0.3	0	0	0.1	0	0	0	0	0	0	0	0.4
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0.7	0.7	0.6	0.7	0.2	0	0	0	0	0	0	0	0	0
Overall	0.1	0.033	0.06	0.053	0.04	0.013	0.007	0.007	0.133	0.32	0.273	0.387	0.18	0.12	0.107	0.087	0.08	0.007	0.007	0.02	0.013	0.007	0	0.08

design codes and interfaces:

> ... for a Serial Peripheral Interface (SPI) host IP. This module interfaces with a TileLink (TL-UL) bus, handling transactions

In contrast, Codellama 13B retains more of the coding language, including variable names and more technical descriptions:

> ... between the host and the SPI device, including sending data (tx_data_o) and signaling when data is valid (tx_valid_o),...

The Llama2 70B model combines both approaches, offering comprehensive explanations that incorporate narrative and details:

> ... managing the data flow between the SPI host and the TX FIFO. It outputs data (tx_data_o), byte enable signals (tx_be_o), ...

The Mixtral 8×7B model delivers the most intriguing responses, generating structured, highly detailed, and functional content:

> ...**Key Functionalities:**
> - **Window Management:** The module manages the TX FIFO window based on the input win_i and outputs win_o...

These examples highlight that when a base model is initially trained on actual documentation, it tends to replicate the format and style of those documents. As a result, even after fine-tuning, it retains these distinctive features, emphasizing the significant impact of pre-training characteristics on model behavior in specific applications.

IX. CONCLUSION

We developed an automated method for generating semi-synthetic datasets, validating its effectiveness in hardware debug tasks using general-purpose LLMs. Despite limitations due to scarce raw data, our approach enhances data utilization efficiency and allows scalable, fully automated data generation without expert involvement. This success integrates LLMs into the hardware domain. Our method avoids redundancy and low quality by using real-world data compared to existing hardware synthetic methods. We plan to expand its application through future partnerships.

ACKNOWLEDGMENT

Portions of this work were supported by the National Science Foundation (2340949, 2419880).

REFERENCES

[1] K. Laeufer, B. Fajardo et al., "Rtl-repair: Fast symbolic repair of hardware design code," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2024, pp. 867–881.
[2] D. Yang, J. He et al., "Strider: Signal value transition-guided defect repair for hdl programming assignments," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023.
[3] T. Brown, B. Mann et al., "Language models are few-shot learners," Advances in neural information processing systems, vol. 33, pp. 1877–1901, 2020.
[4] H. Xiong, S. Wang et al., "Doctorglm: Fine-tuning your chinese doctor is not a herculean task," arXiv preprint arXiv:2304.01097, 2023.
[5] S. Yue, W. Chen et al., "Disc-lawllm: Fine-tuning large language models for intelligent legal services," arXiv preprint arXiv:2309.11325, 2023.
[6] M. Chen, J. Tworek et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
[7] L. Zhong, Z. Wang, and J. Shang, "Debug like a human: A large language model debugger via verifying runtime execution step by step," in Findings of the Association for Computational Linguistics ACL 2024, 2024, pp. 851–870.
[8] Anthropic, "Claude 3.5 sonnet," https://www.anthropic.com/news/claude-3-5-sonnet, 2024.
[9] M. Liu, N. Pinckney et al., "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.
[10] F. Cui, C. Yin et al., "Origen: Enhancing rtl code generation with code-to-code augmentation and self-reflection," arXiv preprint arXiv:2407.16237, 2024.
[11] Anthropic, "Introducing the next generation of claude," 2024. [Online]. Available: https://www.anthropic.com/news/claude-3-family
[12] E. Perez, S. Huang et al., "Red teaming language models with language models," in Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, 2022, pp. 3419–3448.
[13] W. Fu, S. Li et al., "Hardware phi-1.5 b: A large language model encodes hardware domain specific knowledge," arXiv preprint arXiv:2402.01728, 2024.
[14] J. Hoffmann, S. Borgeaud et al., "Training compute-optimal large language models," arXiv preprint arXiv:2203.15556, 2022.
[15] M. Liu, T.-D. Ene et al., "Chipnemo: Domain-adapted llms for chip design," arXiv preprint arXiv:2311.00176, 2023.
[16] S. Liu, W. Fang et al., "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," arXiv preprint arXiv:2312.08617, 2023.
[17] G. Hart, "The five w's: An old tool for the new task of task analysis," Technical communication, vol. 43, no. 2, pp. 139–145, 1996.
[18] W.-L. Chiang, L. Zheng et al., "Chatbot arena: An open platform for evaluating llms by human preference," 2024.
[19] J. Achiam, S. Adler et al., "Gpt-4 technical report," arXiv preprint arXiv:2303.08774, 2023.
[20] J. Bhandari, J. Knechtel et al., "Llm-aided testbench generation and bug detection for finite-state machines," arXiv preprint arXiv:2406.17132, 2024.
[21] C. E. Shannon, "A mathematical theory of communication," The Bell system technical journal, vol. 27, no. 3, pp. 379–423, 1948.
[22] E. Nijkamp, B. Pang et al., "Codegen: An open large language model for code with multi-turn program synthesis," arXiv preprint arXiv:2203.13474, 2022.
[23] B. Roziere, J. Gehring et al., "Code llama: Open foundation models for code," arXiv preprint arXiv:2308.12950, 2023.
[24] R. Li, L. B. Allal et al., "Starcoder: may the source be with you!" arXiv preprint arXiv:2305.06161, 2023.
[25] Z. Luo, C. Xu et al., "Wizardcoder: Empowering code large language models with evol-instruct," arXiv preprint arXiv:2306.08568, 2023.
[26] N. Muennighoff, Q. Liu et al., "Octopack: Instruction tuning code large language models," arXiv preprint arXiv:2308.07124, 2023.

[27] W. Fu, K. Yang et al., "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in 2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2023, pp. 1–6.

[28] J. Liu, Y. Zhu et al., "Rltf: Reinforcement learning from unit test feedback," arXiv preprint arXiv:2307.04349, 2023.

[29] B. Shen, J. Zhang et al., "Pangu-coder2: Boosting large language models for code with ranking feedback," arXiv preprint arXiv:2307.14936, 2023.

[30] B. Chen, F. Zhang et al., "Codet: Code generation with generated tests," arXiv preprint arXiv:2207.10397, 2022.

[31] A. Ni, S. Iyer et al., "Lever: Learning to verify language-to-code generation with execution," in International Conference on Machine Learning. PMLR, 2023, pp. 26 106–26 128.

[32] R. Kande, H. Pearce et al., "Llm-assisted generation of hardware assertions," arXiv preprint arXiv:2306.14027, 2023.

[33] B. Ahmad, S. Thakur et al., "Fixing hardware security bugs with large language models," arXiv preprint arXiv:2302.01215, 2023.

[34] W. Fang, M. Li et al., "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," arXiv preprint arXiv:2402.00386, 2024.

[35] S. Paria, A. Dasgupta, and S. Bhunia, "Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection," arXiv preprint arXiv:2308.06932, 2023.

[36] T. Y. Zhuo, M. C. Vu et al., "Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions," arXiv preprint arXiv:2406.15877, 2024.

[37] M. Chen, J. Tworek et al., "Evaluating large language models trained on code," 2021.

[38] C. E. Jimenez, J. Yang et al., "SWE-bench: Can language models resolve real-world github issues?" in The Twelfth International Conference on Learning Representations, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66

[39] Y. Huang, J. Zhang et al., "Compression represents intelligence linearly," arXiv preprint arXiv:2404.09937, 2024.

[40] C. Raffel, N. Shazeer et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," arXiv e-prints, 2019.

[41] T. Computer, "Redpajama: An open source recipe to reproduce llama training dataset," 2023. [Online]. Available: https://github.com/togethercomputer/RedPajama-Data

[42] P. Zhang, G. Zeng et al., "Tinyllama: An open-source small language model," 2024.

[43] S. Gunasekar, Y. Zhang et al., "Textbooks are all you need," arXiv preprint arXiv:2306.11644, 2023.

[44] H. Liu, C. Li et al., "Improved baselines with visual instruction tuning," 2023.

[45] R. Taori, I. Gulrajani et al., "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[46] I. Shumailov, Z. Shumaylov et al., "Model dementia: Generated data makes models forget," arXiv e-prints, pp. arXiv–2305, 2023.

[47] D. Mendoza, "Openai has hired an army of contractors to make basic coding obsolete," https://www.semafor.com/article/01/27/2023/openai-has-hired-an-army-of-contractors-to-make-basic-coding-obsolete, Feb 2024.

[48] N. Rajani, L. Tunstall et al., "No robots," https://huggingface.co/datasets/HuggingFaceH4/no_robots, 2023.

[49] A. Köpf, Y. Kilcher et al., "Openassistant conversations-democratizing large language model alignment," Advances in Neural Information Processing Systems, vol. 36, 2024.

[50] Y. Wang, Y. Kordi et al., "Self-instruct: Aligning language model with self generated instructions," 2022.

[51] B. Peng, C. Li et al., "Instruction tuning with gpt-4," arXiv preprint arXiv:2304.03277, 2023.

[52] S. Thakur, B. Ahmad et al., "Verigen: A large language model for verilog code generation," ACM Transactions on Design Automation of Electronic Systems, 2023.

[53] P. Srikumar, "Fast and wrong: The case for formally specifying hardware with llms," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM. ACM Press, 2023.

[54] Y. Bai, A. Jones et al., "Training a helpful and harmless assistant with reinforcement learning from human feedback," arXiv preprint arXiv:2204.05862, 2022.

[55] R. Nakano, J. Hilton et al., "Webgpt: Browser-assisted question-answering with human feedback," in arXiv, 2021.

[56] S. Thakur, J. Blocklove et al., "Autochip: Automating hdl generation using llm feedback," arXiv preprint arXiv:2311.04887, 2023.

[57] S. Duan, N. Kanakaris et al., "Leveraging reinforcement learning and large language models for code optimization," arXiv preprint arXiv:2312.05657, 2023.

[58] S. Ugare, T. Suresh et al., "Improving llm code generation with grammar augmentation," arXiv preprint arXiv:2403.01632, 2024.

[59] E. J. Hu, Y. Shen et al., "Lora: Low-rank adaptation of large language models," arXiv preprint arXiv:2106.09685, 2021.

[60] R. Zhang, J. Han et al., "Llama-adapter: Efficient fine-tuning of language models with zero-init attention," arXiv preprint arXiv:2303.16199, 2023.

[61] P. K. Ponuthorai and J. Loeliger, Version Control with Git. " O'Reilly Media, Inc.", 2022.

[62] R. Kipling, Just so stories for little children. Oxford University Press, 2009.

[63] Z. Pan and G. M. Kosicki, "Framing analysis: An approach to news discourse," Political communication, vol. 10, no. 1, pp. 55–75, 1993.

[64] P. D. Schiavone, F. Conti et al., "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, 2017, pp. 1–8.

[65] S. Johnson, D. Rizzo et al., "Titan: enabling a transparent silicon root of trust for cloud," in Hot Chips: A Symposium on High Performance Chips, vol. 194, 2018, p. 10.

[66] P. D. Schiavone, D. Rossi et al., "Quentin: an ultra-low-power pulpissimo soc in 22nm fdx," in 2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2018, pp. 1–3.

[67] M. Gautschi, P. D. Schiavone et al., "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700–2713, 2017.

[68] P. Davide Schiavone, F. Conti et al., "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017, pp. 1–8.

[69] M. Cavalcante, F. Schuiki et al., "Ara: A 1-ghz+ scalable and energy-efficient risc-v vector processor with multiprecision floating-point support in 22-nm fd-soi," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 2, pp. 530–543, 2020.

[70] J. Balkind, M. McKeown et al., "Openpiton: An open source manycore research framework," in Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 217–232. [Online]. Available: https://doi.org/10.1145/2872362.2872414

[71] E. Matthews and L. Shannon, "Taiga: A new risc-v soft-processor framework enabling high performance cpu architectural features," in 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–4.

[72] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2629–2640, Nov 2019.

[73] V. Gangadhar, R. Balasubramanian et al., "MIAOW: an open source GPGPU," in 2015 IEEE Hot Chips 27 Symposium. IEEE, 2015, pp. 1–43. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/HOTCHIPS.2015.7477460

[74] P. Clark, I. Cowhey et al., "Think you have solved question answering? try arc, the ai2 reasoning challenge," arXiv preprint arXiv:1803.05457, 2018.

[75] R. Zellers, A. Holtzman et al., "Hellaswag: Can a machine really finish your sentence?" arXiv preprint arXiv:1905.07830, 2019.

[76] D. Hendrycks, C. Burns et al., "Measuring massive multitask language understanding," arXiv preprint arXiv:2009.03300, 2020.

[77] S. Lin, J. Hilton, and O. Evans, "Truthfulqa: Measuring how models mimic human falsehoods," arXiv preprint arXiv:2109.07958, 2021.

[78] K. Sakaguchi, R. L. Bras et al., "Winogrande: An adversarial winograd schema challenge at scale," 2019.

[79] K. Cobbe, V. Kosaraju et al., "Training verifiers to solve math word problems," 2021.

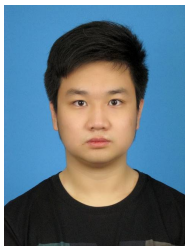
[80] "LLM Leaderboard," https://www.vellum.ai/llm-leaderboard, 2024.

[81] E. Beeching, C. Fourrier et al., "Open llm leaderboard," https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2023.

[82] M. Conover, M. Hayes et al. (2023) Free dolly: Introducing the world's first truly open instruction-tuned llm. [Online]. Available:

https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm
[83] H. Touvron, L. Martin et al., "Llama 2: Open foundation and fine-tuned chat models," arXiv preprint arXiv:2307.09288, 2023.
[84] Y. Li, S. Bubeck et al., "Textbooks are all you need ii: phi-1.5 technical report," arXiv preprint arXiv:2309.05463, 2023.
[85] Microsoft, "phi-2," https://huggingface.co/microsoft/phi-2, 2024.
[86] A. Q. Jiang, A. Sablayrolles et al., "Mixtral of experts," arXiv preprint arXiv:2401.04088, 2024.
[87] N. Pinnaparaju, R. Adithyan et al., "Stable code 3b." [Online]. Available: https://huggingface.co/stabilityai/stable-code-3b
[88] E. Almazrouei, H. Alobeidli et al., "Falcon-40B: an open large language model with state-of-the-art performance," 2023.
[89] T. Dettmers, A. Pagnoni et al., "Qlora: Efficient finetuning of quantized llms," Advances in Neural Information Processing Systems, vol. 36, 2024.
[90] lowRISC Contributors, "Opentitan aes mix single column module," 2024. [Online]. Available: https://github.com/lowRISC/opentitan/blob/6f1056276066efe3336abaf71e92993ada67a5c2/hw/ip/aes/rtl/aes_mix_single_column.sv
[91] B. Ahmad, S. Thakur et al., "On hardware security bug code fixes by prompting large language models," IEEE Transactions on Information Forensics and Security, pp. 1–1, 2024.
[92] H. Pearce, B. Tan et al., "Examining zero-shot vulnerability repair with large language models," in 2023 IEEE Symposium on Security and Privacy (SP), 2023, pp. 2339–2356.
[93] R. Kande, H. Pearce et al., "(security) assertions by large language models," IEEE Transactions on Information Forensics and Security, 2024.
[94] X. Yao, H. Li et al., "Hdldebugger: Streamlining hdl debugging with large language models," arXiv preprint arXiv:2403.11671, 2024.

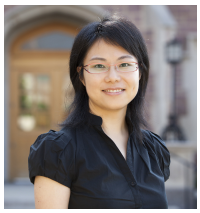
Weimin Fu (S'19) is a Ph.D. candidate in the Mike Wiegers Department of Electrical and Computer Engineering at Kansas State University. He holds an M.S. in Electrical Engineering from The George Washington University and a B.S. in Information Engineering from Zhejiang University. His current research focuses on domain-specific large language models and hardware security. He can be reached via email at weiminf@ksu.edu.



Shijie Li (S'23) is a Ph.D. student in the School of Cyber Science and Technology at the University of Science and Technology of China. He received a B.S. degree in Information Security from Shanghai Jiao Tong University, China, in 2022. His research interests include Hardware Security and Large Language Models for hardware. He can be reached via shijie_li@mail.ustc.edu.cn.

Yifang Zhao (S'23) is a Ph.D. student in the School of Cyber Science and Technology at the University of Science and Technology of China. She earned her B.S. degree in Information Security from Taiyuan University of Technology, China, in 2021. Her current research focuses on Hardware Security and Large Language Models. She can be contacted at zhaoyifang@mail.ustc.edu.cn.

Kaichen Yang (S'16-M'20) is an Assistant Professor in the Electrical and Computer Engineering Department at Michigan Technological University. He earned his Ph.D. in Electrical and Computer Engineering from the University of Florida in 2022. His research focus on the fields of hardware security, CPS security and their intersections with machine learning. He has published many papers in top-tier AI and security conferences including NDSS, AAAI, AsiaCCS, Host and DAC.



Xuan Zhang (S'08, M'15, SM'22) is an Associate Professor in the Electrical and Computer Engineering Department at Northeastern University. She works across the fields of integrated circuits/VLSI design, computer architecture, and electronic design automation. Dr. Zhang is an IEEE Women in Engineering (WiE) Distinguished Lecturer for 2023-2024 and IEEE Circuits and Systems Society (CAS) Distinguished Lecturer for 2022-2023. Her work has received numerous best paper awards and nominations including ISLPED Best Paper Award in 2022, AsianHOST Best Paper Award in 2020, DATE Best Paper Award in 2019, and Best Paper nomination at DAC 2022, ASP-DAC 2021, MLCAD 2020, DATE 2019, and DAC 2017.

Yier Jin (M'12-SM'19) is a Professor in University of Science and Technology of China (USTC). Before joining in USTC, he was an Associate Professor and IoT Term Professor in the Department of Electrical and Computer Engineering (ECE) in the University of Florida (UF). He received his PhD degree in Electrical Engineering in 2012 from Yale University. His research focuses on the areas of hardware security, embedded systems design and security, trusted hardware intellectual property (IP) cores and hardware-software co-design for modern computing systems. Dr. Jin is a recipient of the DoE Early CAREER Award in 2016 and ONR Young Investigator Award in 2019. He received Best Paper Award at DAC'15, ASP-DAC'16, HOST'17, ACM TODAES'18, GLSVLSI'18, DATE'19, AsianHOST'20, Oakland'22, and CCS'23.

Xiaolong Guo (S'13-M'19) is an Assistant Professor in the Department of Electrical and Computer Engineering at Kansas State University. His research focuses on large language models, formal verification, and hardware security. Dr. Guo has received several accolades, including Best Paper Awards at DATE 2019 and AsianHost 2020, and a Best Paper Nomination at ASP-DAC 2021. He was honored with the Outstanding Assistant Professor Award from the Carl R. Ice College of Engineering at K-State. Most recently, he is the recipient of the 2024 NSF CAREER Award.