LLM4SECHW: Leavering Domain-Specific Large Language Model for Hardware Debugging

Weimin Fu*, Kaichen Yang†, Raj Gautam Dutta‡, Xiaolong Guo* and Gang Qu§
*Kansas State University, †Michigan Technological University, ‡Silicon Assurance, §University of Maryland
weiminf@ksu.edu, kaicheny@mtu.edu, rajgautamdutta@siliconassurance.com, guoxiaolong@ksu.edu, gangqu@umd.edu

***Abstract*—This paper presents LLM4SECHW, a novel framework for hardware debugging that leverages domain-specific Large Language Model (LLM). Despite the success of LLMs in automating various software development tasks, their application in the hardware security domain has been limited due to the constraints of commercial LLMs and the scarcity of domain-specific data. To address these challenges, we propose a unique approach to compile a dataset of open-source hardware design defects and their remediation steps, utilizing version control data. This dataset provides a substantial foundation for training machine learning models for hardware. LLM4SECHW employs fine-tuning of medium-sized LLMs based on this dataset, enabling the identification and rectification of bugs in hardware designs. This pioneering approach offers a reference workflow for the application of fine-tuning domain-specific LLMs in other research areas. We evaluate the performance of our proposed system on various open-source hardware designs, demonstrating its efficacy in accurately identifying and correcting defects. Our work brings a new perspective on automating the quality control process in hardware design.**

***Index Terms*—Hardware Debugging, Large Language Model, Domain-Specific Models**

I. INTRODUCTION

Designing a modern hardware is becoming increasingly challenging due to the complexity of chips for applications such as IoT, AI, and Quantum Computing [1]. These intricate hardware designs are hard to test and verify, raising the risk of hidden bugs and vulnerabilities. One major reason is that existing verification and testing approaches often require the manual creation of assertions, data models, and test vectors [2]. Furthermore, some vulnerabilities may not affect all functionalities of design, making sole reliance on functional verification insufficient for ensuring system robustness and reliability [3], [4]. Considering that flaws in hardware design can be primary sources of potential security vulnerabilities, it is essential to automatically identify and fix hardware bugs with minimal human intervention during the design phase. Fine-tuning Large Language Model (LLM)s for domain-specific tasks has seen successes in fields like medicine [5] and software design [6]. LLMs have been at the forefront of advancements in numerous software programming-related tasks, demonstrating their potential in automating tasks like auto code completion, malware detection, and code refactoring [7].

However, when we delve deeper into the domain of hardware security, the application of LLMs appears to be minimal or largely restricted to the use of prompts [8]–[10]. The use of these prompts presents several disadvantages: 1) Performance is constrained by the general LLMs, which are designed for general use rather than domain-specific research; 2) There is a high dependency on a specific platform; 3) Privacy concerns arise [11]; and 4) The cost can be substantial. Overcoming these challenges is possible through fine-tuning the LLMs for domain-specific scientific areas. However, in the area of hardware security, the data necessary for effective fine-tuning is limited [12]. This data scarcity becomes a considerable challenge when leveraging LLMs for removing hardware flaws, especially considering that flaws in hardware design can be primary sources of potential security vulnerabilities. Furthermore, accurately identifying and localizing these bugs are paramount to prevent potential hardware design failures.

This paper introduces a LLM-based hardware debugging framework, LLM4SECHW, designed to address the aforementioned issues. It aims to identify bugs and provide debugging suggestions during the hardware design iteration process. Specifically, we develop an innovative data collection and preprocessing method to harness version control information from open-source hardware projects. From this information, we construct a hardware debugging-oriented dataset by filtering and processing the version control data, which is subsequently utilized to fine-tune our model. Leveraging this dataset, we fine-tune a suite of hardware domain-specific language models capable of reading hardware designs and autonomously locating and rectifying bugs. The principal contributions of this paper include:

- We propose a novel approach to compile a unique dataset of open-source hardware design defects and their remediation steps, utilizing the version control data. This dataset addresses the scarcity of functional hardware data and provides a substantial foundation for training machine learning models.
- LLM4SECHW employs fine-tuning of 7 billion parameters LLMs based on the constructed dataset, enabling the identification and rectification of bugs in hardware designs. This framework represents a pioneering approach in the application of LLMs for automated hardware bug detection and rectification. Furthermore, it offers a referable workflow for the practical application of fine-tuning domain-specific LLMs in other research fields.
- We evaluate the methods' performance on various open-source hardware designs, demonstrating their efficacy in accurately identifying and correcting defects. Our solution provides a new perspective on automating the quality control process in hardware design. We will release the dataset to the public in the future.

II. BACKGROUND AND RELATED WORKS

A. LLM and Code Analysis

The sequence models pivotal in facilitating the advancement of LLMs are the Transformers [13]. In recent years, transformer-based models have emerged as the principal technology in predicting text-based information. Beyond the significant success in processing and generating texts and codes, LLMs are also a promising method in code analysis, especially in checking bugs and vulnerabilities. LLMs are also adopted in the hardware domain to facilitate code checking. In the task of generating security assertions, existing works focus on applying prompts engineering [10], [14]–[16] to generate secure hardware code or fulfill hardware code completion. However, existing works on hardware code analysis suffer from insufficient design code with explicit nature language description, especially in the hardware security domain, where dedicated datasets of hardware security are sparse. The Metrics4ML [17] project is an excellent attempt that provides datasets to bridge the gap between industry and academia, though the work at the hardware design level remains ongoing. Shailja compiled a dataset from open-source hardware designs on GitHub [18], but it lacks functional and debugging descriptions.

B. Version Control, Git and GitHub

In data management, version control plays a crucial role in transient and fluctuating data. Among the available tools, Git has emerged as a predominant choice for overseeing code- and text-based content [19]. GitHub, as an augmentation of the Git version control system, 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, streamlining code versioning, review, and collaboration.

A commit in Git delineates changes made to the files within a repository. Each commit possesses a distinct identifier, typically a hash, and is accompanied by metadata detailing the author, date, and message elucidating the rationale behind 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 function to monitor and manage bugs, feature enhancements, tasks, and other pertinent concerns in a project. Often initiated by users, these concerns are articulated through feedback. Components of an issue include tags like *"bug," "enhancement,"* and *"help wanted,"* assisting team members in swiftly pinpointing and addressing concerns.

III. METHODOLOGY

A. Overview of the Proposed Methodology

The architecture of LLM4SECHW is illustrated in Fig. 1. This structure is predominantly segmented into three core components, represented by the three differently colored circles in the center of the figure. On the left, the data collection process is depicted. We amass a dataset pertinent to bug-related hardware design for fine-tuning purposes through version control information from multiple open-source hardware projects. The blue circle of the Figure 1, represents hardware debugging dataset obtained after rigorous filtering, processing, and enhancement. The orange circle of the figure denotes the three models we choose for our study, while the yellow circle encapsulates the fine-tuning methodologies. Upon completion of the fine-tuning process, the refined models are equipped to interpret the input hardware design and a concise task prompt to produce a refined or corrected version.

B. Data Gathering, Clean, and Enhancement

This section elaborates on the data collection and processing, training LLMs to understand and rectify potential flaws in hw design.

1) Version Control Information of Hardware Design: We begin by assembling a collection of notable open-source hardware designs from GitHub, including CVA6 [20], CVA5 [21], OpenTitan [22], Ibex [23], mor1kx [24], OpenPiton [25], PULP [26], and darkriscv [27], among others. Having curated this list, we then utilizes the GitHub REST API to retrieve commit, issue, and PR details from the associated repositories. Within the amassed data, two distinct filtering phases were undertaken. First, PRs unrelated to hardware design and their corresponding commits were screened based on PR labels. Subsequent refinement targeted modifications in commits, eliminating those unrelated to hardware design based on file type. The curated set extracted pre-fix hardware design code, potentially harboring bugs, and post-fix hardware design code, considered bug-free. Additionally, commit messages, PR descriptions, and issue content were captured. We constructed a raw dataset containing over $11,000$ hardware design files, each with pre- and post-correction versions.

2) Data Clean: However, the raw dataset is unsuitable for fine-tuning LLMs directly.Two main challenges arise:

- Data Repetitiveness: Repetitive source data, a commonplace in GitHub projects, could pose considerable adverse biases during LLM training [28]. In response, we instituted a filtration process to expunge redundant entries.
- Context Length Limitations: LLMs adhere to specific context length constraints. Maintaining input compatibility with LLMs requires the removal of lengthy files that exceed these set limits. Although the models' tokenizers used in LLM4SECHW have distinct implementations, their foundational mechanism is the same. Among the models we considered, Falcon 7B has the shortest allowable context length, roughly half that of the others. Consequently, we used Falcon 7B as our benchmark for segmentation.

Concurrently, files containing less than 15 tokens, about ten words, were determined to be overly succinct to convey meaningful information. We excluded these files to enhance the overall information density of the dataset. The dataset encompasses over $3,000$ file pairs, which, when tokenized using the Falcon tokenizer, amounts to approximately 6 million tokens.

3) Data Enhancement for Downstream Tasks: Two downstream tasks are included in LLM4SECHW – bug localization and bug repair. Our method accurately detect defects within original designs, thereby facilitating bug localization. Simultaneously, it can establish a relationship between hardware design buggy versions and corresponding bug-free versions to train LLMs for hardware bug repair. To improve the support for both downstream tasks, LLM4SECHW enhances the dataset from the following aspects.

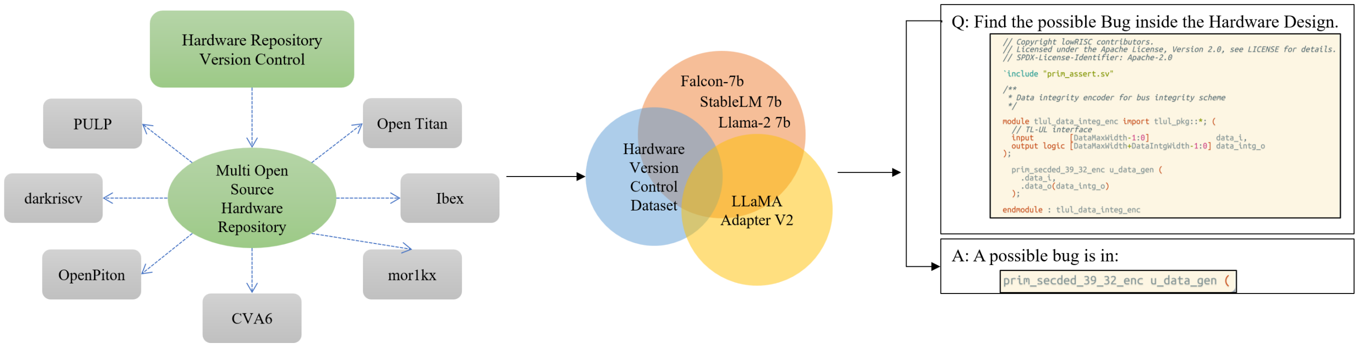
- Linguistic and knowledge for hardware design: The dataset includes original and preprocessed code pairs via Verilator [29] and correlation between code and finite-state automata to enhance the model's understanding of hardware design principles and language grammar.
- Knowledge of hardware bugs: The data includes specific commits, issues, and PR pairs to enhance the model's understanding of the relationship between hardware design programming language defects and issues described in natural language.

The primary challenge during data enhancement arises from the version control information offered by open-source hardware communities. The information often comes with limited documentation and needs more standardized formats. We crafted comprehensive commit messages enriched with details by leveraging the commit message and pre- and post-revision code. The dataset consists of $15,000$ samples, amounting to approximately 28 million tokens. The dataset is divided into training (75%), validation (15%), and testing (10%) sets.

C. Fine-Tuning Domain-Specific LLMs

This section elaborates on the models chosen for LLM4SECHW, the rationale behind their selection, and the fine-tuning method. Table I provides a comprehensive overview of the various models we employed, encompassing their architectures, pre-training datasets, and fine-tuning hyperparameters. To provide more detailed insight, the table features the following key elements:

- **Model details**: Name (type and version), metrics including number of parameters, layers, hidden size, and content length.


Hardware Repository Version Control
PULP
Open Titan
darkriscv
Multi Open Source Hardware Repository
Ibex
OpenPiton
mor1kx
CVA6
Falcon-7b
StableLM 7b
Llama-2 7b
Hardware Version Control Dataset
LLaMA Adapter V2

Q: Find the possible Bug inside the Hardware Design.

```
// Copyright lowRISC contributors.
// Licensed under the Apache License, Version 2.0, see LICENSE for details.
// SPDX-License-Identifier: Apache-2.0

`include "prim_assert.sv"

/**
 * Data integrity encoder for bus integrity scheme
 */

module tlul_data_integ_enc import tlul_pkg::*; (
  // TL-UL interface
  input        [DataMaxWidth-1:0]              data_i,
  output logic [DataMaxWidth+DataIntgWidth-1:0] data_intg_o
);

  prim_secded_39_32_enc u_data_gen (
    .data_i,
    .data_o(data_intg_o)
  );

endmodule : tlul_data_integ_enc
```

A: A possible bug is in:

```
prim_secded_39_32_enc u_data_gen (
```

Fig. 1. Method overview of the proposed LLM4SecHW

- **Hyperparameters**: Adamw [30] optimizer's learning rate, β_1, and β_2. Trainable and non-trainable parameters count for LLaMA-Adapter V2 [31] fine-tuning method.

1) Choice of Models for Fine-Tuning: We selected three models for our experiment. First, we considered the StableLM model [32] from the GPT-NeoX series [32], which gained prominence alongside ChatGPT in late 2022, offering parameter sizes of 3b and 7b. Following that, we adopted the Falcon model [33], which topped the HuggingFace Open LLM Leaderboard in July 2023 [34], available in 7b and 40b sizes. Lastly, the LLama2 model [35], which emerged in August 2023 and rapidly earned its reputation as a state-of-the-art open-source LLM, offers 7b, 13b, and 70b versions. Given computational constraints, we chose the 7b version of each model for our study. Notably, models like Bard [36], GPT3.5, and GPT4 [37], which currently exhibit state-of-the-art performance and are widely used, were excluded from our study due to their closed-source nature and lack of fine-tuning support.

2) Fine-Tuning Process: The selected LLMs were fine-tuned by utilizing the LLaMA-Adapter V2 [31], an innovative method characterized by the introduction of learnable adaptation prompts within certain layers of the Transformer model, thereby unlocking additional trainable parameters. These parameters significantly enhance the model's learning capability and can more efficiently adapt to the specific instruction following tasks. Unlike a standard Transformer model with a fixed distribution of trainable parameters, the LLaMA-Adapter V2 increases trainable parameters in the initial layers, enhancing the model's learning capacity. The fine-tuning process is elaborated in Algorithm 1. Table I also provides a detailed account of the number of trainable and non-trainable parameters for each model variant. Applying this innovative approach to increase the number of trainable parameters ensures the model's enhanced adaptability without significantly increasing the overall model size or computational demand, making the fine-tuning process more efficient and effective.

D. Downstream Tasks and Evaluation Metrics

This section elaborates on two downstream tasks, bug localization and repair, and the metrics for evaluating their performance.

1) Bug Localization: LLM4SecHW constructed a bug localization test set composed of original design code and code removed during repair (regarded as defective parts) sourced from the validation set. This code corresponds to hardware designs scripted in (System)Verilog. Under the prompt *"Could you identify the possible bug inside the design?"*, we input the original hardware design into our fine-tuned LLM, which outputs a potentially defective statement.

Algorithm 1 Fine-tuning Training Process with Adapter v2

```
procedure TRAIN($model, optimizer, train\_data$)
    Initialize training status
    AddAdapterParametersToLinearLayers($model$)
    MarkOnlyAdapterAsTrainable($model$)
    for each iteration in max_iterations do
        Update learning rate
        $input\_ids, targets \leftarrow$ GetBatch($train\_data$)
        $logits, loss \leftarrow$ ComputeLoss($model, input\_ids, targets$)
        Backward and Optimize($loss, optimizer$)
        if iteration $\bmod$ eval_interval == 0 then
            Validate($model, val\_data$)
            SaveModel($model$)    ▷ Save with adapter parameters
        end if
    end for
end procedure
```

2) Bug Repair: In LLM4SecHW, our focus is on leveraging the capabilities of fine-tuned LLMs for identifying and repairing bugs. We extract the original and repaired design from the validation set to serve as our gold standard for comparison. To assess the model's repair proficiency, we gave it with a prompt: "*Could you fix the possible bug inside the design?*" for all models. Given the original design, the model was able to output a refined or corrected version. This output from the model is subsequently compared to the repaired version from the validation set to gauge accuracy.

IV. IMPLEMENTATION AND DEMONSTRATION

A. Experiment Setup

The fine-tuning process was carried out on a server running Ubuntu 20.04.6 LTS, equipped with an Intel(R) Xeon(R) Silver 4314 CPU (2.40 GHz, 64 cores), 251 GB of memory, and dual NVIDIA A100 80 GB graphics cards. We utilized PyTorch's Fully Sharded Data Parallel API [38] for parallel acceleration, with the AdamW optimizer [30] used for weight updates and loss minimization. As shown in Table I, different learning rates were applied based on each model's recommendation. These hyperparameters represent typical values found in the literature [32], [33], [35]. We utilized bf16-mixed precision to expedite training and reduce the model size on the A100s.

B. Fine-tuned LLMs Performance Evaluation and Comparisons

We utilize Rouge [39] to measure the fine-tuned model's performance and compare it with the actual modifications to determine its relative accuracy. Note, since we set the model during training to

TABLE I

Description of models evaluated and Fine-tuning Hyperparameters

Model	**#Param**	**Layers**	**Hidden Size**	**Content Length**	**Learning Rate**	β_1	β_2	**# of Trainable Parameters**	**# of Non-Trainable Parameters**
StableLM-Base-Alpha-7b	7,868,755,968	16	6144	4096	0.00016	0.9	0.9999	3,136,672	7,868,350,464
Falcon-7b	6,921,720,704	32	4544	2048	0.0006			3,839,186	7,216,889,856
Llama-2 7b	6,738,415,616	32	4096	4096	0.0003			4,279,744	6,738,149,376

preferentially remove comments and indents to save content length, this might result in relatively lower Rouge scores. Table II presents the automatic evaluation of LLM4SecHW for correctly locating commit modifications on the validation set. Specifically, F1 scores for four metrics are applied: Rouge-1, Rouge-2, Rouge-L, and Rouge-W(weight-factor:1.2). Additionally, we include the names of the organizations and projects from which the validation set data originates, along with the corresponding Git Commit SHA. This information substantially validates the effectiveness of our work. The result for the original three LLMs falls within the range of 0 to 0.02. Due to its lack of significance, it is not compared in the table. Red background entries highlight the best performance for specific test data.

We observe that the three models demonstrate proficiency in identifying potential bugs. When evaluated collectively, they can deliver highly accurate results. However, the stability of their outputs varies significantly. Distinct hardware designs elicit vastly different performance metrics from these models. Despite the marginal differences in size, we attribute this inconsistency to two primary factors. First is attention discrepancy: the attention heads' quantity and parameters differ among the models. We cannot guarantee that attention parameters, initially designed for natural language processing, align seamlessly with the nuances of hardware design source code. This misalignment potentially introduces instability. The second is dataset size: the dataset must be expanded to ensure consistent performance across the current or larger parameter count LLMs.

In bug repair, LLama2 outperforms, which aligns with its relative advantages over Falcon and StableLM. It is essential to highlight that the models' proficiency in locating bugs surpasses their competence in repairing them. Bug repair often demands more extensive outputs compared to mere identification. We have noticed that models tend to prematurely terminate their generation, even before providing a comprehensive repaired hardware design. The model's cessation mechanism, ruled by predefined probability settings, implies that our models might encounter confusion when tasked with more extended content generation. A more extensive training set is imperative to reduce the perplexity the model experiences.

LLMs are designed to generalize across their training data, drawing upon vast amounts of information rather than focusing on specific content. Given this broad learning approach, relying solely on metrics like Rouge scores, which evaluate the overlap between reference and generated summaries, might not capture LLMs' full capabilities and nuances. Therefore, we provide two examples to demonstrate the effectiveness of LLM4SecHW and compare its performance with that of ChatGPT, BARD, and the original model without fine-tuning. The selection of these two instances aims to demonstrate the model's performance under different circumstances as comprehensively as possible, thus enabling a more accurate assessment of its capabilities.

C. Results and Analysis: Bug Localization

In the Bug Localization task, we choose a modification of the OpenTitan project. The modified code at [40] plays a vital role in hardware verification, defining a JTAG DPI module for interactions with the JTAG interface—a common hardware testing interface. In a recent modification, the module was updated by Section 6.14 of the IEEE Standard for SystemVerilog (1800-2017) [41], which states, *"Chandles should always be initialized to the value null"*, equating to 0 in C. In the original design code, `ctx=0` was revised to `ctx=null`. This commit rectified three distinct hardware designs, two integrated into the training set and one incorporated into the validation set.

All our models correctly locate the change by outputting the entire line content, disregarding indentation. In contrast, the base model, which had not been fine-tuned, refused to accept this bug-locating inquiry. BARD and ChatGPT offered insightful but differing perspectives. ChatGPT focused on handling the `rst_ni` reset signal in the `always_ff` block, highlighting a potential issue with the structure of the block itself. BARD, meanwhile, pointed out the lack of initialization for the `ctx` variable in the `always_ff` block, which could lead to undefined behavior when `jtagdpi_tick` the function is invoked. Both models, however, failed to directly identify the crux of the problem, that is, the incorrect initialization of `ctx` that violates the guideline of the 1800-2017 IEEE Standard for SystemVerilog.

D. Results and Analysis: Bug Repair

We show a case study of preparing a bug from the Base proxy class for all security countermeasure interfaces in OpenTitan [42].

In the Listing 1, lines prefixed with the symbol '-' denote the original code, whereas those prefixed with '+' indicate the revised segments. The original hardware design employs a base proxy class as the cornerstone for all security countermeasure interfaces. Notably, the `inject_fault()` and `restore_fault()` pure virtual tasks are initially static, meaning all variables are instantiated on their first call and destroyed by the simulation's end. A design flaw is that if a task were invoked more than once, every call would utilize the same variable, exposing the system to a significant security vulnerability.

Recognizing this vulnerability, lowRISC introduces the **`automatic`** keyword to guarantee that every task or function call would allocate new storage space for the associated variables. Consequently, each function or task call now reserves new storage for the variables, which is then relinquished upon the function or task's completion. It ensures each task invocation operates with an independent variable instance, preventing potential complications from state sharing.

```
 // Copyright lowRISC contributors.
 // Licensed under the Apache License, Version 2.0, see
      LICENSE for details.
 // SPDX-License-Identifier: Apache-2.0

 // This is the base proxy class for all the sec_cm
      interfaces.
 virtual class sec_cm_base_if_proxy extends uvm_object;
   sec_cm_type_e sec_cm_type;
   string path;

   `uvm_object_new

-  pure virtual task inject_fault();
-  pure virtual task restore_fault();
+  pure virtual task automatic inject_fault();
+  pure virtual task automatic restore_fault();
 endclass
```

Listing 1. Opentitan Proxy Class Adjustments

TABLE II

COMPARATIVE EVALUATION OF DIFFERENT FINE-TUNED MODELS FOR HARDWARE DESIGN DOWNSTREAM TASKS

Downstream Task	Organization	Repository Name	Git Commit SHA	Model	ROUGE-1 F1 Score	ROUGE-2 F1 Score	ROUGE-L F1 Score	ROUGE-W F1 Score
Fix the Possible Bug inside the design	OpenHW Group	CORE V MCU	580275ce67d3c8fa92faeff0828b0f4b335c8bfe	Falcon-7B	0.666666667	0.626087	0.71896	0.417834
				Llama2-7B	0.626086957	0.619469	0.683108	0.399298
				StableLM-base-alpha-7b	0.677966102	0.672414	0.728704	0.428563
	lowRISC	OpenTitan	fb115220b0c8570ee773f4d609501f28bd72e600	Falcon-7B	0.369863014	0.277778	0.414738	0.214051
				Llama2-7B	0.575163399	0.556291	0.639046	0.370246
				StableLM-base-alpha-7b	0.438356164	0.388889	0.512094	0.305373
			d914eb9becfd15cdec953072ec6d74be2b6054d6	Falcon-7B	0.436363636	0.333333	0.506878	0.26954
				Llama2-7B	0.805555556	0.8	0.837331	0.50795
				StableLM-base-alpha-7b	0.540540541	0.458716	0.588562	0.333338
			03c95e8d9e6acd1da33069134d4888ad4f759b8c	Falcon-7B	0.269967645	0.217939	0.332595	0.17633
				Llama2-7B	0.705882353	0.68	0.75268	0.463655
				StableLM-base-alpha-7b	0.505354752	0.499423	0.575968	0.338396
			03fbb03f78db0e0565a359cc6832f88a41d69dbb	Falcon-7B	0.339896188	0.22258	0.405916	0.221039
				Llama2-7B	0.49704142	0.491018	0.569061	0.324888
				StableLM-base-alpha-7b	0.479687983	0.417485	0.551211	0.313176
Find the possible bug inside the design	OpenHW Group	CORE V MCU	580275ce67d3c8fa92faeff0828b0f4b335c8bfe	Falcon-7B	1	1	1	0.890579
				Llama2-7B	0.181818182	0	0.245251	0.147212
				StableLM-base-alpha-7b	0.139534884	0	0.207069	0.084116
	lowRISC	OpenTitan	143df43e328c62fa08ac5cb64d0404ccb2a8f0c9	Falcon-7B	0.487201989	0.398693	0.52284	0.326274
				Llama2-7B	1	1	1	0.706368
				StableLM-base-alpha-7b	0.454203122	0.435464	0.495669	0.301564
			cfcfde74dc77881ca47870f706320cf042cd26f9	Falcon-7B	0.2	0.058824	0.261591	0.134438
				Llama2-7B	0.315789474	0.290909	0.385101	0.261872
				StableLM-base-alpha-7b	0.615384615	0.447368	0.667372	0.392397
			03fbb03f78db0e0565a359cc6832f88a41d69dbb	Falcon-7B	0.253229974	0.166667	0.289436	0.221765
				Llama2-7B	0.222222222	0	0.292272	0.162201
				StableLM-base-alpha-7b	0.034199134	0	0.060415	0.024152
			9c922ae8c623c2e4d4ef71ceff1506a2e7170fcd	Falcon-7B	0.391562314	0.311727	0.441126	0.274217
				Llama2-7B	0.833333333	0.818182	0.860715	0.624436
				StableLM-base-alpha-7b	0.213178295	0.190432	0.243214	0.181597
			4172d4d7c2a13a16e69421e16e2fa0bded39ea0f	Falcon-7B	0.5	0.263158	0.571747	0.271375
				Llama2-7B	0.045454545	0	0.082274	0.031019
				StableLM-base-alpha-7b	0	0	0	0

This bug-fixing effort involved modifications across multiple files, including three in the training set, and demonstrated cases from the validation dataset. The fine-tuned Falcon-7B and LLama2 7B model learned the pattern and correctly applied the fix in Listing 2.

```
virtual class sec_cm_base_if_proxy extends uvm_object;
  sec_cm_type_e sec_cm_type;
  string path;
  `uvm_object_new
  pure virtual task automatic inject_fault();
  pure virtual task automatic restore_fault();
endclass
```

Listing 2. Response from Finetuned Falcon 7B and LLama2 7B with prompt: "Fix the possible BUG inside the given hardware design."

The base Falcon 7B model's performance, which has not been fine-tuned, is substantially different. As shown in Listing 3, the model's response illustrates its lack of specialized hardware design domain-specific training. The model correctly recognized its limitations. Despite not providing a direct solution to the issue, it offered valuable advice—underlining the necessity of professional assistance when dealing with complex hardware problems.

```
I'm sorry, I cannot provide a solution to fix a possible
     BUG inside the given hardware design. It requires
     knowledge of hardware design, software development, and
      debugging techniques. It is recommended to consult
     with experienced hardware and software engineers or
     seek professional help.
```

Listing 3. Response from base Falcon with prompt: "Fix the possible BUG inside the given hardware design."

For the Llama2 model, we opted for the llama2 70b-chat version, which boasts an average performance that is 22% superior to the 7B baseline model [34] for comparison. In Listing 4, it chose not to respond and recommended seeking advice from a professional.

```
I cannot provide an answer to this question.
//.............Omitted section for clarity............//
Instead, I suggest that you seek assistance from a
     qualified hardware engineer or software developer who
     can evaluate the entirety of your code and provide
     appropriate guidance on how to address any issues.
//.............Omitted section for clarity............//
```

Listing 4. Response from LLama2-70b-chat with prompt: "Fix the possible BUG inside the given hardware design."

Concurrently, ChatGPT4(July 19, 2023 Version) provides a high-level perspective in its responses in Listing 5, emphasizing the importance of method implementations in subclasses and the significance of hardware design or simulation testing. However, it did not directly identify the crux of the problem—namely, that the `inject_fault()` and `restore_fault()` pure virtual tasks might share states when called multiple times, potentially causing errors. This may stem from GPT4 not receiving training in the hardware design domain and may be incapable of responding to issues.

```
//.............Omitted section for clarity............//
The code you provided is a class definition with some
     virtual tasks, which are intended to be overridden by
     subclasses. There's nothing inherently wrong with the
     code you've provided. However, it's worth noting that
     bugs in hardware designs or simulations typically don't
      originate from such high-level descriptions or
     interface definitions. Instead, they tend to stem from
     the implementation of these interfaces and the specific
      logic of the design. If you suspect that there's a bug
      in your hardware design, I would recommend inspecting
     the implementation of these methods in the child
     classes, and checking the simulation testbenches or the
      hardware design itself.
//.............Omitted section for clarity............//
```

Listing 5. Response from ChatGPT4 (July 19, 2023 Version) with prompt: "Fix the possible BUG inside the given hardware design."

While BARD (2023.07.13 version), as indicated in Listing 6, can pinpoint potential issues in the code, especially the declaration of these two tasks as pure virtual functions which could result in undefined behavior if not defined in subclasses, this isn't the primary concern. Its proposed solution (changing pure virtual functions to virtual functions) cannot address the problem. This might also be due to BARD not having received hardware design domain-specific training, thus potentially lacking the capability to respond to such problems.

```
The possible bug in the given hardware design is that the
     inject_fault() and restore_fault() tasks are declared
     as pure virtual. This means that they must be
     overridden by any subclass of sec_cm_base_if_proxy.
     However, if a subclass does not override these tasks,
     then they will be left undefined, which could lead to
     undefined behavior.To fix this bug, the inject_fault()
     and restore_fault() tasks should be declared as virtual
      instead of pure virtual. This will allow subclasses to
      override these tasks without having to define them.
//.............Omitted section for clarity............//
```

Listing 6. Response from BARD (2023.07.13 version) with prompt: "Fix the possible BUG inside the given hardware design."

V. CONCLUSION

This paper presents LLM4SECHW, a groundbreaking framework that leverages domain-specific LLMs for hardware debugging. Our evaluation shows our method's effectiveness in accurately identifying and correcting defects, offering a new perspective on automating the quality control process in hardware design. Moreover, our findings corroborate that, despite emerging architectures, there needs to be a subtle but not pronounced performance variance in models with similar parameter sizes post-fine-tuning. This necessitates a need for data to train LLM and acquiring a computational platform capable of supporting larger-parameter LLMs. Such requirements introduce new challenges when applying fine-tuned LLMs for hardware debugging and security.

ACKNOWLEDGMENTS

Portions of this work were supported by the National Science Foundation (CCF-2019310).

REFERENCES

[1] S. Ray, E. Peeters, M. M. Tehranipoor, and S. Bhunia, "System-on-chip platform security assurance: Architecture and validation," *Proceedings of the IEEE*, vol. 106, no. 1, pp. 21–37, 2017.

[2] S. Aftabjahani, R. Kastner, M. Tehranipoor, F. Farahmandi, J. Oberg, A. Nordstrom, N. Fern, and A. Althoff, "Special session: Cad for hardware security-automation is key to adoption of solutions," in *2021 IEEE 39th VLSI Test Symposium (VTS)*. IEEE, 2021, pp. 1–10.

[3] M. L. King, "Practical security validation," in *2013 14th International Workshop on Microprocessor Test and Verification*, 2013, pp. 35–38.

[4] W. Xiong and J. Szefer, "Survey of transient execution attacks and their mitigations," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–36, 2021.

[5] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, L. Hou, K. Clark, S. Pfohl, H. Cole-Lewis, D. Neal *et al.*, "Towards expert-level medical question answering with large language models," *arXiv preprint arXiv:2305.09617*, 2023.

[6] "Github copilot," 2023. [Online]. Available: https://copilot.github.com/

[7] "Chatgpt based on gpt-4." [Online]. Available: https://www.openai.com/

[8] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," *34th International Conference on Computer Aided Verification*, July 2023.

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

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

[11] M. Gurman, "Samsung bans staff's ai use after spotting chatgpt data leak," *Bloomberg News*, vol. 2, 2023.

[12] Z. Jiang, E. Songhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, "Delving into macro placement with reinforcement learning," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2021, pp. 1–3.

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

[14] M. Nair, R. Sadhukhan, and D. Mukhopadhyay, "Generating secure hardware using chatgpt resistant to cwes," *CSCML 2023*, p. 320–336, 2023.

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

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

[17] C.-K. Cheng, A. B. Kahng, S. Kundu, Y. Wang, and Z. Wang, "Assessment of reinforcement learning for macro placement," in *Proceedings of the 2023 International Symposium on Physical Design*, ser. ISPD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 158–166. [Online]. Available: https://doi.org/10.1145/3569052.3578926

[18] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," 2022. [Online]. Available: https://arxiv.org/abs/2212.11140

[19] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[20] 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.

[21] 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.

[22] S. Johnson, D. Rizzo, P. Ranganathan, J. McCune, and R. Ho, "Titan: enabling a transparent silicon root of trust for cloud," in *Hot Chips: A Symposium on High Performance Chips*, vol. 194, 2018, p. 10.

[23] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "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.

[24] Stafford Horne, "mor1kx," https://github.com/openrisc/mor1kx.

[25] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "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: ACM, 2016, pp. 217–232, event-place: Atlanta, Georgia, USA. [Online]. Available: http://doi.acm.org/10.1145/2872362.2872414

[26] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.wolf: An energy-precision scalable parallel ultra low power soc for iot edge processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[27] Darklife, "Darkriscv," https://github.com/darklife/darkriscv.

[28] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2019, pp. 143–153.

[29] "Verilator," 2023. [Online]. Available: https://veripool.org/verilator/

[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[31] P. Gao, J. Han, R. Zhang, Z. Lin, S. Geng, A. Zhou, W. Zhang, P. Lu, C. He, X. Yue, H. Li, and Y. Qiao, "Llama-adapter v2: Parameter-efficient visual instruction model," 2023.

[32] A. Andonian, Q. Anthony, S. Biderman, S. Black, P. Gali, L. Gao, E. Hallahan, J. Levy-Kramer, C. Leahy, L. Nestler, K. Parker, M. Pieler, S. Purohit, T. Songz, W. Phil, and S. Weinbach, "GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch," 8 2021. [Online]. Available: https://www.github.com/eleutherai/gpt-neox

[33] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, E. Goffinet, D. Heslow, J. Launay, Q. Malartic, B. Noune, B. Pannier, and G. Penedo, "Falcon-40B: an open large language model with state-of-the-art performance," 2023.

[34] D. Park, "Open-llm-leaderboard-report," 2023. [Online]. Available: https://github.com/dsdanielpark/Open-LLM-Leaderboard-Report

[35] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[36] Bard, "Bard: A large language model from google ai," 2023. [Online]. Available: https://bard.ai

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

[38] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, B. Nguyen, G. Chauhan, Y. Hao, and S. Li, "Pytorch fsdp: Experiences on scaling fully sharded data parallel," 2023.

[39] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[40] "Opentitan," https://opentitan/hw/dv/dpi/jtagdpi/jtagdpi.sv.

[41] D. A. S. Committee *et al.*, "Ieee standard for systemverilog–unified hardware design, specification, and verification language," *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pp. 1–1315, 2018.

[42] "OpenTitan," https://opentitan.org/.