This paper was presented at IEEE VLSI Test Symposium (VTS) 2025

BugWhisperer: Fine-Tuning LLMs for SoC Hardware Vulnerability Detection

Shams Tarek, Dipayan Saha, Sujan Kumar Saha, Farimah Farahmandi
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA
{shams.tarek, dsaha, sujansaha}@ufl.edu, {farimah}@ece.ufl.edu

arXiv:2505.22878v1 [cs.CR] 28 May 2025

***Abstract*—The current landscape of system-on-chips (SoCs) security verification faces challenges due to manual, labor-intensive, and inflexible methodologies. These issues limit the scalability and effectiveness of security protocols, making bug detection at the Register-Transfer Level (RTL) difficult. This paper proposes a new framework named BugWhisperer that utilizes a specialized, fine-tuned Large Language Model (LLM) to address these challenges. By enhancing the LLM's hardware security knowledge and leveraging its capabilities for text inference and knowledge transfer, this approach automates and improves the adaptability and reusability of the verification process. We introduce an open-source, fine-tuned LLM specifically designed for detecting security vulnerabilities in SoC designs. Our findings demonstrate that this tailored LLM effectively enhances the efficiency and flexibility of the security verification process. Additionally, we introduce a comprehensive hardware vulnerability database that supports this work and will further assist the research community in enhancing the security verification process.**

***Index Terms*—Large Language Model, Fine-tuning, Hardware Security, Security Verification, Hardware Vulnerability Database**

I. Introduction

Hardware vulnerabilities at the SoC level present critical threats by potentially exposing sensitive user data, cryptographic keys, and essential system configurations. Increasingly sophisticated attacks targeting SoCs, such as information leakage [1], side-channel leakage [2], and violations of access control [3], emphasize the urgent need for rigorous SoC security verification. Addressing these vulnerabilities proactively is essential to prevent severe financial repercussions, product recalls, and reputational damage within the semiconductor industry.

Traditional SoC security verification approaches primarily concentrate on functional correctness, frequently overlooking critical vulnerabilities during pre-silicon verification. While methods such as information flow tracking [4], assertion-based security verification [3], fuzz testing [5], runtime verification monitoring, and static code analysis [6] have been developed, these techniques often encounter significant scalability and adaptability limitations. Furthermore, they usually require extensive manual intervention, thereby elevating both the complexity and cost of security verification. Consequently, there is an increasing demand for automated and versatile methods capable of efficiently handling diverse hardware architectures and dynamic security scenarios. Detecting vulnerabilities comprehensively at the RTL can notably reduce the time, effort, and expenses involved in SoC security verification.

Recently, LLMs have gained traction within hardware security and design communities due to their superior abilities in pattern recognition, knowledge generalization, and learning from extensive datasets [7], [8]. Consequently, researchers are leveraging LLMs to address complex security challenges at the SoC level [8]–[11]. For instance, prompting-based techniques utilizing both pre-trained and proprietary LLMs have been applied to identify security vulnerabilities within RTL designs [12]. Nevertheless, the limited hardware-specific knowledge inherent in pre-trained LLMs constrains their applicability in targeted security verification tasks. A comparative analysis of existing LLMs provides valuable insights into their effectiveness for vulnerability detection SoCs and hardware designs. Proprietary models, including well-known solutions such as Gemini and GPT [13], typically demonstrate superior detection accuracy compared to smaller open-source alternatives like LLama [14], Mistral, and CodeLlama [15]. However, despite their notable performance advantages, proprietary solutions present several significant challenges. These include high operational costs, limited accessibility, scalability constraints, and reduced operational flexibility. Such factors may substantially limit their broader adoption and applicability across various industries, potentially making open-source alternatives more attractive due to their flexibility, transparency, and accessibility.

Thus, the development of smaller, open-source LLMs explicitly fine-tuned to detect SoC-level security vulnerabilities in RTL designs becomes critical. To our knowledge, no such model currently exists. This paper addresses this significant gap by investigating the feasibility and effectiveness of fine-tuning an open-source LLM for hardware security vulnerability detection at the SoC level. By integrating comprehensive insights from the CWE hardware vulnerability database, our proposed model substantially enhances its capability to detect and mitigate critical security threats.

In addition, the model is trained using security vulnerability reports, which allows it to accurately identify the presence

We thank to U.S. National Science Foundation (NSF) for their support through CAREER Award under Grant 2339971.

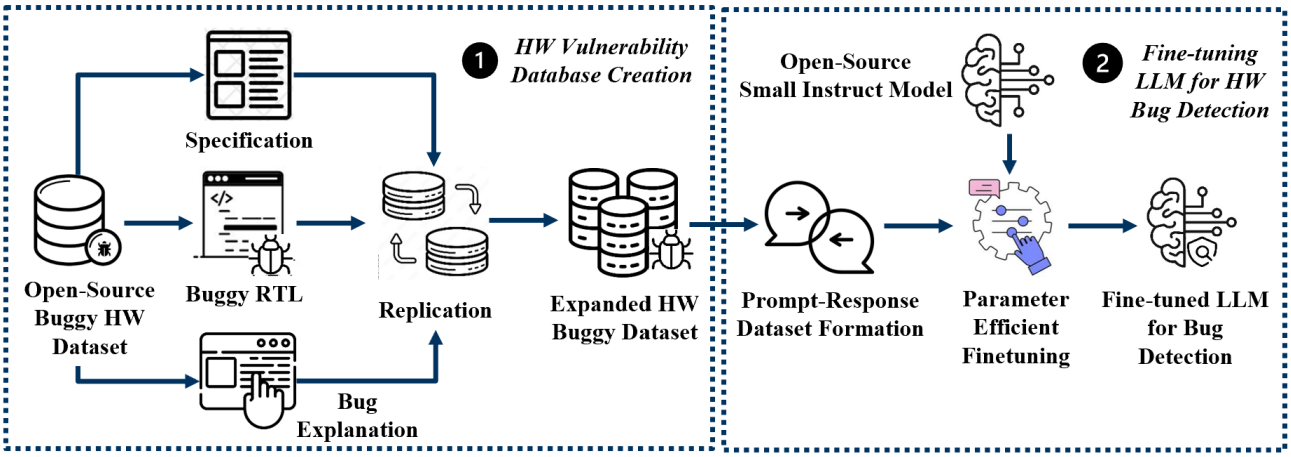

Fig. 1. Overview of the proposed BugWhisperer framework.

of security vulnerabilities. Using these task-specific datasets, our fine-tuned model improves the precision and efficiency in detecting 13 security vulnerabilities in SoC designs. The key contributions are:

- We present a comprehensive SoC hardware vulnerability database, open to the research community
- We introduce an open-source, fine-tuned LLM specifically designed for detecting hardware design bugs in the SoC
- We demonstrate that open-source LLMs can compete with proprietary LLMs if properly curated

In the remainder of this paper, Section II narrates the methodology used in this work. Next, Section III describes the training scheme and details the experimental result with analysis. Finally, Section IV concludes the paper.

II. Proposed Methodology

The proposed framework "BugWhisperer" comprises two key stages: (1) Generation of a hardware vulnerability database, and (2) Fine-tuning of a custom LLM for hardware vulnerability detection, as shown in Figure 1.

A. Database Generation for Vulnerable SoC Designs

To establish a high-quality database for vulnerable hardware designs, this work leverages existing Golden Vulnerable SoC design benchmarks available through the Cad4Security platform [16]. These benchmarks comprise 13 distinct SoC vulnerabilities introduced into the CVA6 Ariane core. Initially, each vulnerable design module is systematically separated and clearly labeled according to its specific vulnerability type. Subsequently, a comprehensive specification file is generated for each vulnerable module. Each specification file meticulously documents the baseline functionality of the respective Intellectual Property (IP), detailed descriptions of internal and external registers, Input/Output (I/O) ports, and explicit vulnerability characteristics. Such specification documentation is crucial, as it explicitly outlines both the intended baseline functionality and the inherent vulnerabilities within each module.

Following specification creation, the vulnerable designs are replicated using a Replicator LLM. This replication is not a simple duplication; rather, it involves a nuanced process engineered to retain original module functionality while systematically diversifying code expression. To achieve this diversity, various Verilog/SystemVerilog coding styles are adopted, including parameterization, Finite State Machine (FSM) architectures (single-process FSM and dual-process FSM), and varying signal nomenclatures. The replication process employs carefully curated Replicator prompts, manually tailored according to these different coding styles, to ensure each generated design instance remains distinct.

During replication, the specification file corresponding to each vulnerable IP is provided as context to the Replicator LLM, thereby ensuring fidelity to the original functionality and the embedded vulnerability. Furthermore, two pivotal LLM parameters—"Temperature" and "Top_p"—are utilized to enhance the uniqueness and diversity of the generated design instances. The "Temperature" parameter, adjustable between 0 and 2, influences response creativity and diversity, with typical optimal ranges identified as between 0.6 and 1.5. Higher temperature values increase diversity but may compromise functionality consistency due to enhanced creativity. The "Top_p" parameter controls randomness and coherence in the output tokens, aiding in the prevention of code similarity and token overlap among generated replicas.

B. Fine-tuning LLMs for Vulnerability Detection

In the subsequent stage, open-source LLMs are fine-tuned for vulnerability detection tasks using the developed hardware vulnerability database. Initially, potential open-source models are carefully analyzed for suitability. Many open-source LLMs lack comprehensive hardware-specific domain knowledge essential for the effective detection of vulnerabilities in hardware designs. After rigorous assessment, we select *Llama-3.2-1B-instruct*, *Llama-3.2-3B-instruct*, *Llama-3.1-8B-instruct*, *Mistral-7B-instruct*, and *Codellama-7B-it* models. These models are chosen due to their instructional adaptability,

coding proficiency, and compatibility with hardware design tasks.

Despite their robust coding and instructional following capabilities, these models initially lack specialized knowledge regarding hardware security vulnerabilities, especially in RTL design contexts. Therefore, fine-tuning is necessary to embed this critical domain-specific knowledge. The fine-tuning process enhances the LLMs' capability to accurately recognize and classify hardware security vulnerabilities, thereby significantly improving their efficiency and reliability during security bug analysis. The incorporation of domain-specific expertise enables these models to effectively support hardware security assessments, contributing substantially to the robustness and security of future hardware designs. The detailed description of the vulnerabilities will be found here [16]. The fine- tuning and evaluation efforts discussed in this work focus on detecting the following 13 vulnerabilities:

- CWE-1198: Improper handling of privilege issues
- CWE-269: Improper privilege level during interrupt handling
- CWE-1245: Less secured FSM encoding
- CWE-1260: Overlapping between memory ranges
- CWE-506: Hardware trojan inside the decoder module
- CWE-310: Trojan in AES for information leakage
- CWE-310: Trojan in AES for denial of service
- CWE-310: Trojan in CSR module unauthorized access
- CWE-321: Use of hardcoded cryptographic key
- CWE-250: Improper trap privilege assignment
- CWE-1244: Unlocking JTAG during reset
- CWE-284: Improper direct memory access
- CWE-1271: Unauthorized access to important registers

The detailed description of the vulnerabilities will be found here [16].

During this phase, instruction fine-tuning is utilized to specialize the model for identifying security vulnerabilities. To accomplish this, the original data is reformatted via a Python script into a structured prompt-response format. Each prompt includes either a secure or a vulnerable SoC design accompanied by a query prompting a security evaluation targeting a particular security flaw. In the corresponding responses, it is clearly stated whether the specified vulnerability is present or not, along with explanations to support the decision. Such detailed rationales are essential during fine-tuning, as they assist the model in comprehending the logic underlying the security assessments, thus significantly improving its capability to identify and clearly articulate security issues within SoC designs.

A major challenge in dataset creation is the lack of sufficiently detailed annotations. To mitigate this limitation, GPT-4o is employed to generate thorough explanatory annotations across different hardware modules. These generated explanations are subsequently integrated into the prompt-response pairs, thereby enhancing the dataset with the necessary context for effective model training. For a rigorous evaluation of the model's effectiveness, the dataset is partitioned into distinct training, validation, and testing subsets based on individual designs. Such an approach ensures that the RTL designs allocated to the test set remain entirely unseen during the training phase, thereby enabling an accurate assessment of the model's ability to generalize beyond the provided training examples.

III. Experimentation and Evaluation

A. Dataset Replication

For replicating the dataset, available open-source SoC vulnerability benchmarks were used. The coding and replicating capabilities of GPT-4 and GPT-4o were leveraged as the replicator LLMs. All datasets were replicated using GPT API calls. To maintain consistent functionality throughout the process, the "Temperature" value was kept between 0.6 and 1.5. Using this process, a set of 4000 vulnerable SoC hardware designs were generated. The dataset is available here: https://github.com/shamstarekargho/Hardware-Vulnerability-Dataset

B. Training Setup

As described in Section III-A, 4,000 Verilog codes are used during training the open-source models. For training, We implemented a parameter-efficient fine-tuning approach using low-rank adaptation (LoRA) [17], enhancing training efficiency by adding minimal additional parameters to the open-source models. The LoRA configuration was set with a rank size of 128, an alpha value of 256, and a dropout rate of 0.1, optimizing the balance between parameter efficiency and model accuracy. Fine-tuning was conducted on two NVIDIA A100 GPUs with 4-bit quantization (NF4) and a float16 compute data type to manage memory demands, leveraging the GPUs' computational power. To ensure stable weight updates, we employed a low learning rate of 2×10^{-6}, a batch size of 4, and gradient accumulation steps of 1 across three training epochs, mitigating overfitting risks while maintaining effective gradient updates. The training utilized the paged adamw 32bit optimizer with a weight decay of 0.001 for regularization, a maximum gradient norm of 0.3 for clipping to prevent exploding gradients, and a constant learning rate scheduler with a warmup ratio of 0.03 to gradually increase the learning rate. Gradient checkpointing was enabled to reduce memory usage, and the maximum sequence length was capped at 512 tokens to balance computational efficiency and context retention.

C. Result Analysis

The detection accuracy results for different models, including proprietary, fine-tuned, and non-fine-tuned open-source models, are shown in Figure 2 The results indicate a clear distinction in performance between these categories, demonstrating the effectiveness of fine-tuning in adapting LLMs for hardware security vulnerability detection.

Among the fine-tuned models, *Mistral-7B-instruct* achieved the highest accuracy of 84.8%, significantly outperforming its non-fine-tuned counterpart, which only reached 42.5%. This improvement of over 40 percentage points highlights the effectiveness of fine-tuning in equipping open-source

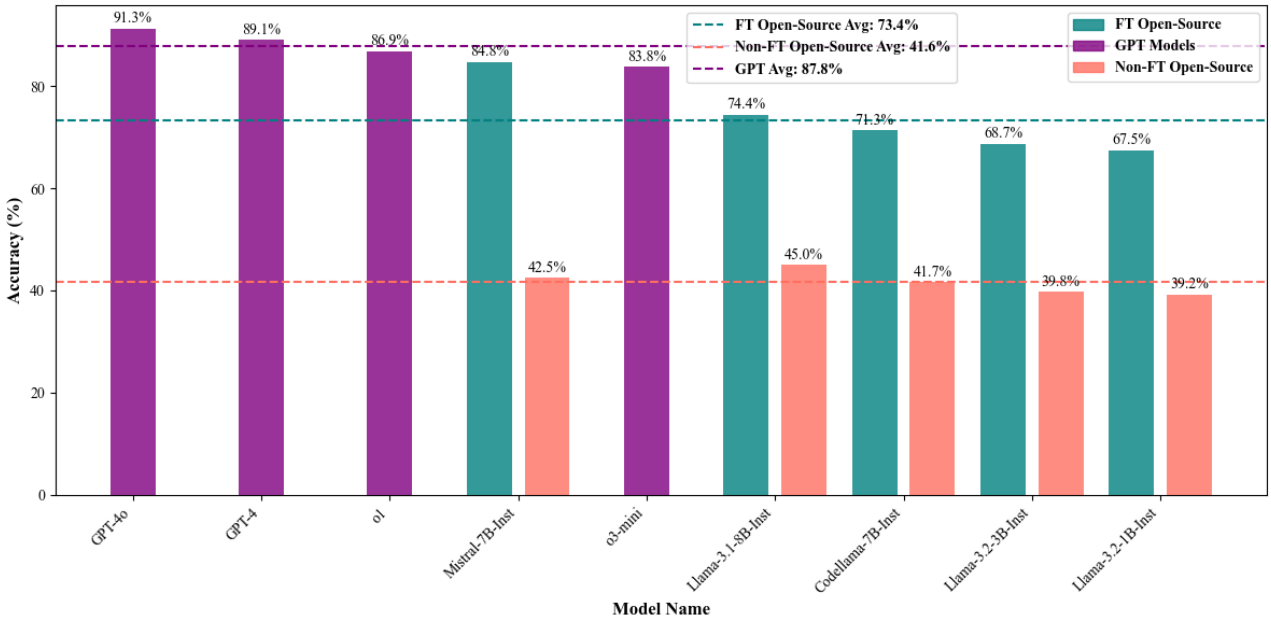

Fig. 2. Performance evaluation of the large proprietary models, fine-tuned and non-fine-tuned open-source models.

models with domain-specific knowledge. Although *Mistral-7B-instruct* does not surpass the performance of proprietary models, its high accuracy suggests that open-source models can be viable alternatives with sufficient domain adaptation. Similar trends are observed for *Llama-3.1-8B* and *Llama-3.2-3B*, where fine-tuning elevates detection accuracy to 74.4% and 68.7%, respectively, from their significantly lower non-fine-tuned baselines. The best-performing model (Mistral-7b-instruct-Bug-Whisperer) has been released to the research community for further use in the hardware security verification domain. The model can be found here: https://huggingface.co/shamstarek/Mistral-7B-instruct-Bug-Whisperer

The proprietary models, as expected, demonstrated superior performance, with *GPT-4o* achieving the highest accuracy of 91.3%, followed by *o-1* (86.9%) and *o3-mini* (83.8%). These results align with expectations, as proprietary models benefit from extensive pretraining on diverse datasets, including specialized knowledge in hardware security. However, their reliance on closed-source architectures and significant usage costs make them less accessible for widespread deployment.

A key observation from the results is the poor performance of non-fine-tuned open-source models, which average around 40% accuracy. This highlights a major limitation of general-purpose LLMs when applied to domain-specific tasks without adaptation. The lack of pretraining on hardware security datasets renders them ineffective for vulnerability detection, reinforcing the necessity of fine-tuning to bridge this knowledge gap.

Model size is another influential factor affecting detection accuracy. The results indicate that larger open-source models tend to achieve higher accuracy post-fine-tuning. For instance, *Llama-3.2-3B* (3B parameters) reaches 68.7% accuracy, whereas *Llama-3.1-8B* (8B parameters) achieves 74.4%, suggesting that increased model capacity contributes to better representation of security-related patterns. However, fine-tuning smaller models effectively can still yield competitive performance while reducing computational overhead.

These findings suggest broader implications for AI-driven hardware security. The substantial improvement in fine-tuned models highlights their potential as cost-effective alternatives to proprietary solutions. Unlike closed-source models, fine-tuned open-source LLMs provide transparency, cost efficiency, and flexibility for integration into security workflows. However, Despite these promising improvements, limitations remain. While fine-tuning enhances detection accuracy, some vulnerabilities may still go undetected due to complex attack patterns. Further refinements in dataset expansion, fine-tuning methodologies, and architectural modifications could push open-source models closer to proprietary-level performance.

IV. Conclusion

This paper presents a fine-tuned LLM-based approach, “BugWhisperer”, to SoC security verification, addressing the limitations of manual and inflexible methodologies. The proposed model significantly improves vulnerability detection accuracy, outperforming non-fine-tuned counterparts by over 40% and demonstrating the potential of open-source LLMs as cost-effective alternatives to proprietary solutions. Additionally, the introduction of a comprehensive hardware vulnerability database enhances research in automated security verification. These findings highlight the scalability and adaptability of LLMs for hardware security, with future work focusing on further optimization and dataset expansion.

REFERENCES

[1] G. K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, and M. Tehranipoor, "Security vulnerability analysis of design-for-test exploits for asset protection in socs," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 617–622.

[2] P. Mishra, M. Tehranipoor, and S. Bhunia, "Security and trust vulnerabilities in third-party ips," *Hardware IP Security and Trust*, pp. 3–14, 2017.

[3] N. Farzana, F. Rahman, M. Tehranipoor, and F. Farahmandi, "Soc security verification using property checking," in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–10.

[4] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1691–1696.

[5] K. Z. Azar, M. M. Hossain, A. Vafaei, H. Al Shaikh, N. N. Mondol, F. Rahman, M. Tehranipoor, and F. Farahmandi, "Fuzz, penetration, and ai testing for soc security verification: Challenges and solutions," *Cryptology ePrint Archive*, 2022.

[6] R. Kibria, F. Farahmandi, and M. Tehranipoor, "Arc-fsm-g: Automatic security rule checking for finite state machine at the netlist abstraction," *Cryptology ePrint Archive*, 2023.

[7] S. Tarek, D. Saha, S. K. Saha, M. Tehranipoor, and F. Farahmandi, "Socurellm: An llm-driven approach for large-scale system-on-chip security verification and policy generation," *Cryptology ePrint Archive*, 2024.

[8] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "Llm for soc security: A paradigm shift," *IEEE Access*, vol. 12, pp. 155 498–155 521, 2024.

[9] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "ChipGPT: How far are we from natural language hardware design," May 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. Akyash and H. M. Kamali, "Self-hwdebug: Automation of llm self-instructing for hardware security verification," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 391–396.

[12] D. Saha, K. Yahyaei, S. K. Saha, M. Tehranipoor, and F. Farahmandi, "Empowering hardware security with llm: The development of a vulnerable hardware database," in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 233–243.

[13] "Gpt-4 technical report." [Online]. Available: https://arxiv.org/pdf/2303.08774.pdf

[14] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[15] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[16] S. Tarek, H. A. Shaikh, S. R. Rajendran, and F. Farahmandi, "Benchmarking of soc-level hardware vulnerabilities: A complete walkthrough," in *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2023, pp. 1–6.

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.