HADA: Leveraging Multi-Source Data to Train Large Language Models for Hardware Security Assertion Generation

Weimin Fu
Kansas State University
Manhattan, USA
weiminf@ksu.edu

Yiting Wang
University of Maryland
College Park, USA
ywang144@umd.edu

Zelin Lu
University of Maryland
College Park, USA
zelinlu@umd.edu

Xiaolong Guo
Kansas State University
Manhattan, USA
guoxiaolong@ksu.edu

Gang Qu
University of Maryland
College Park, USA
gangqu@umd.edu

***Abstract*—Hardware security verification is critical but labor-intensive, particularly in crafting security-focused assertions for assertion-based verification. While large language models offer automation potential, their performance is constrained by limited domain-specific training data and fragmented hardware security knowledge. We present HADA (Hardware Assertion through Data Augmentation), a novel framework that systematically constructs a fine-tuning dataset for LLMs using three complementary sources: (1) formal verification outputs, (2) hardware vulnerability databases, and (3) version control histories of real-world designs. This multi-source data augmentation enables the generation of high-quality SystemVerilog assertions and their contextual explanations. We fine-tune several open-source LLMs and evaluate them on HSAEval, a new benchmark for hardware security assertion tasks. Results show that HADA significantly improves assertion correctness and coverage, outperforming prior syntactic and functional verification approaches. All models, datasets, and raw evaluations will be released to support open research in LLM-driven hardware security verification.**

***Index Terms*—Large Language Models, Hardware Security, Assertion Generation, Formal Methods, Data Augmentation, Verification Automation**

I. Introduction

The rapid evolution of hardware has fueled intelligent systems, yet this progress has also exposed critical security concerns. Software-based defenses often implicitly trust hardware, allowing vulnerabilities to bypass protection mechanisms—escalating risks as smart devices proliferate [1]–[5]. Post-silicon discovery of such flaws is often irreversible, and even when patchable, incurs significant performance overhead [6]. Early-stage detection is therefore essential. Verification ensures hardware behaves as specified, but **security verification** poses distinct challenges. Unlike functional assertions, which check intended behavior, **security assertions** aim to detect vulnerabilities, requiring deep expertise and reasoning about threat models [6], [7]. Constructing such assertions is labor-intensive and error-prone, and reliance on existing assertions is fragile due to limited coverage [8]. As a result, the lack of scalable security assertion generation has become a major bottleneck in hardware security verification.

Large language models (LLMs) have transformed automation across domains. In hardware verification, LLM-based approaches for assertion generation are emerging. For instance, LAAG-RV [9] improves assertion quality by incorporating hardware context into ChatGPT outputs. However, security assertions pose greater challenges due to reliance on specialized, scarce expertise [10]. One solution [10] uses detailed, handcrafted prompts to provide sufficient context to ChatGPT-3.5. While effective in controlled settings, it requires heavy prompt engineering and accesses only a limited portion of available security knowledge. Separately, LLM4SecHW [11] fine-tunes models for direct debugging tasks, but lacks integration into standard EDA workflows and does not generate formal assertions. A central barrier to scalable LLM-based security assertion generation is the scarcity of high-quality training data. While synthetic datasets have been proposed to address this gap, they often introduce token-level biases that degrade semantic precision. Thus, a key research question is how to construct synthetic datasets that meaningfully improve LLM performance in generating accurate and verifiable security assertions.

This paper introduces **HADA** (Hardware Assertion through Data Augmentation), an LLM-based framework for generating hardware security assertions. Given the fragmented, sparse nature of hardware security knowledge, we propose three data augmentation pipelines to transform disparate sources into a training dataset. These include: (1) vulnerability descriptions from the Common Weakness Enumeration (CWE), (2) version control data from open-source hardware projects, and (3) assertions derived from applying formal verification tools to designs. The resulting dataset captures the structure of hardware designs, assertions, and explanatory links between them, providing supervision for LLM training.

Our contributions are as follows:

1) We present HADA, an automated, domain-specific LLM-based framework that generates hardware security assertions.
2) Through novel data modeling and augmentation, HADA transforms hardware security knowledge bases, formal verification outputs, and bug-fix records into rich assertion–explanation pairs. This enables the use of widely available but assertion-sparse data, expanding dataset coverage and enhancing training quality.
3) We train and evaluate HADA on multiple open-source LLMs, demonstrating improvements in syntactic correctness and functional accuracy across real-world verification benchmarks.
4) We release HSAEval, an open-source benchmark for security assertion evaluation, along with all fine-tuned models, multi-source training datasets, and raw evaluation results to support reproducible research and community adoption.

II. Background

A. SystemVerilog Assertions

SystemVerilog Assertions (SVAs), analogous to assertions in C languages, serve as formal specifications for verifying hardware behavior. In functional verification, SVAs ensure that hardware operates as intended under various conditions. However, writing SVAs requires significant expertise in hardware design and verification [7], [12].

Security assertions extend this concept by checking for unintended behaviors that may compromise system security [6], [10]. These properties are often unrelated to functionality and fall outside the responsibilities of developers, making them difficult to construct.

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

979-8-3315-3762-3/25/$31.00 © 2025 IEEE

TABLE I
DATA SOURCES AND LLM-AUGMENTED COMPONENTS IN FINE-TUNED HARDWARE SECURITY SYSTEMVERILOG ASSERTION DATASET

Source	Bug/CWE/Attack Inserted Hardware Design			Bug/CWE/Attack Design		SystemVerilog Assertion			
	Source Code	High-level Function abstraction	Detailed Function Implementation	Explanation	Location	Assertion	Explanation	Connection	
								to bug	to design
Selected CWE	LLM Gen ❷	LLM Gen ❼	LLM Gen ❼	Existing	LLM Gen ❷	LLM Gen ❶	LLM Gen ❼	LLM Gen ❼	LLM Gen ❼
Version Control	Existing ❸	LLM Gen ❼	LLM Gen ❼	LLM Gen ❼	Existing	LLM Gen ❹	LLM Gen ❼	LLM Gen ❼	LLM Gen ❼
Formal Verification	Existing ❻	LLM Gen ❼	LLM Gen ❼	Existing	Existing	Existing ❺	LLM Gen ❼	Existing	LLM Gen ❼

We propose using LLMs to automate SVA generation for security verification, reducing manual effort and integrating the process into existing EDA workflows.

B. Hardware Security Knowledge Database

Hardware security databases, such as CWE, provide structured descriptions of known vulnerabilities, offering valuable context for verification. These entries are often used as prompts in LLM-based assertion generation [10], but mapping them to concrete designs remains non-trivial. Human experts must interpret abstract descriptions into RTL-level semantics, while general-purpose LLMs are prone to misinterpretation or oversimplification. To bridge this gap, we generate RTL designs that explicitly reflect specific CWE cases and use LLMs to synthesize corresponding SVAs, grounding abstract knowledge in practical verification contexts.

C. Version Control System and Bug Fix Tracking

Version control systems offer rich, structured insights into the debugging lifecycle, including issues, pull requests, reviews, and merges, often marked by metadata such as the **bug** label. These generate useful data—issue descriptions, buggy/fixed designs, and developer comments—valuable for analyzing real-world vulnerabilities. Prior work [11], [13] has used this for LLM-driven debugging, focusing on code comparisons. However, such methods are not aligned with assertion-based verification or EDA integration. In our framework, we extract confirmed bug fixes, identify affected RTL modules, and use LLMs to generate SVAs targeting the security-relevant behavior—bridging version control and formal verification.

D. Formal Property Verification

Formal Property Verification (FPV) enables exhaustive hardware analysis by checking designs against formally defined safety and liveness properties. Tools such as AutoSVA2 [14] can automatically generate SVAs from symbolic signal representations and interface annotations, making formal properties applicable to RTL modules. However, the core strength of FPV lies in the rules it defines, not necessarily in their application via model checking. Applying these rules at scale is computationally expensive and often impractical due to state space explosion [9], especially for complex SoC designs.

Our framework takes a different approach: rather than executing FPV on large designs, we extract verified assertions and signal patterns from small-scale modules. These results are then used as training data for LLMs, allowing the model to generalize assertion logic without incurring analysis cost. This decouples insights from their computational burden, transforming symbolic constraints into semantically rich, learnable supervision for data-driven assertion generation. In essence, we translate formal rules into a form LLMs can internalize and reuse beyond the scope of model checkers.

E. Utilization of LLMs in Hardware Security Verification

In HADA, LLMs serve two distinct yet complementary roles. First, high-capacity commercial models (e.g., GPT-4o) are used as *intelligent agents* to assist with data augmentation: transforming incomplete or implicit information into structured assertion–explanation pairs. Second, open-source models are fine-tuned to serve as *domain experts*, capable of directly generating hardware security assertions in downstream tasks. This separation allows us to leverage the strengths of both: agent models for high-quality dataset construction, and target models for scalable, low-cost deployment.

1) LLMs as Intelligent Agents: Models like GPT-4o demonstrate exceptional reading comprehension and code, outperforming top human baselines in multiple domains [15]. In hardware tasks, GPT-4 has achieved 100% syntactic correctness in RTL generation [16], though its standalone bug detection accuracy remains limited (33%). When reformulated as a comparison task, accuracy improves to 99.7% [13], highlighting the importance of structured prompting. We adopt GPT-4o as our agent to generate annotations, clarify code semantics, and synthesize missing assertions from raw sources.

2) Fine-Tuned LLMs as Domain Experts: To support scalable, task-specific deployment, we fine-tune open-source models using our augmented dataset. Specifically, we use LLaMA 3 70B [17], LLaMA 3.1 70B [18], LLaMA 3.2 3B [19], and GPT-4o-mini [20]. These models are trained to generate SVAs and related security reasoning without external assistance. [1]

III. HARDWARE ASSERTION THROUGH DATA AUGMENTATION

This section introduces the data synthesis workflow in HADA, which transforms heterogeneous hardware security information into a dataset for LLM fine-tuning. We first outline the pipeline, then describe how each source is processed into design–assertion pairs, and finally explain how these are structured for supervised training.

A. Data Synthesis Workflow in HADA

Fig. 1 illustrates the pipeline of HADA, which transforms raw hardware security data into a fine-tuning dataset. Inputs are drawn from three sources: hardware security knowledge (e.g., CWE entries), version control records, and formal verification outputs. These sources are often incomplete, lacking executable RTL code, vulnerability annotations, or assertions, highlighted in yellow in Table I.

To fill these gaps, we use prompt methods [10], [21] to synthesize missing components such as RTL modules ❷❸❻ and SVAs ❶❹❺, followed by validation using Synopsys VCS and Verilator ❽. The data is then standardized and passed through an augmentation pipeline ❼❾, producing 19,448 *Question–Answer* pairs. Each *Question* includes the hardware design, functional summary, and line-by-line explanation; each *Answer* contains vulnerability details, insertion locations, and the corresponding assertion with its explanation. The

[1]Larger proprietary models like GPT-4 or LLaMA3.1 405B are excluded due to practical computational constraints.

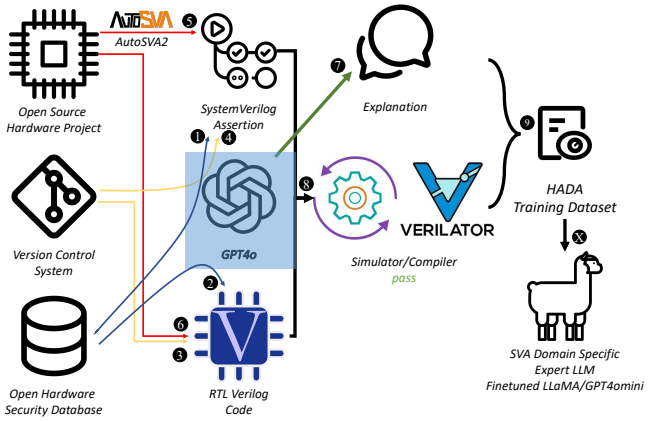


Fig. 1. **Dataset Construction Workflow: From Raw Data to Final Fine-Tuning Dataset.** The workflow illustrates the process of constructing our fine-tuning dataset starting from raw data collected from three primary sources: hardware security knowledge databases, version control records of open-source hardware projects, and outputs from formal property verification tools applied to open-source hardware designs.

dataset totals 73 million input tokens and 63 million output tokens (measured with the LLaMA 3 tokenizer).

Table I summarizes all components used. Each row corresponds to a data source, and each column captures a logical unit of information. Black-circled numbers map each field to its generation step in the workflow. Components marked as "Existing" (green) are directly sourced, while "LLM Gen" (yellow) are synthesized. The key columns include (1) RTL designs with inserted bugs/CWEs and their functional/contextual descriptions, (2) metadata on bug type and location, and (3) SVAs with natural language justifications.

B. Data Augmentation from Common Weakness Enumeration

We leveraged all 77 RTL-relevant entries from the CWE database. As shown in Table I, these entries provide only textual vulnerability descriptions, lacking essential elements such as hardware implementations and corresponding assertions. To bridge this gap, we employed an LLM to synthesize the missing components.

Using CWE classification metadata and descriptions, we first prompted the LLM to generate RTL designs exhibiting the target vulnerabilities ❷. For each design, the model also produced high-level functional summaries, I/O specifications, and line-by-line explanations ❼, resulting in complete, self-contained hardware modules.

Next, we paired the generated designs with their CWE descriptions and instructed the LLM to identify the vulnerability location within the code ❷. Based on this, the model generated tailored SystemVerilog assertions ❶ along with detailed natural language justifications ❼ linking the assertions to specific design behaviors and weaknesses.

To structure this pipeline, we developed three specialized prompts: (1) design generation for CWE-specific flaws, (2) assertion synthesis targeting the identified vulnerability, and (3) explanation of how each assertion mitigates the issue. All outputs were manually reviewed, aided by the generated explanations, and syntactically validated using RTL simulation. All designs compiled and executed successfully.

This process yielded 1,771 CWE-based hardware modules and 3,542 verified assertions.

C. Data Augmentation from Version Control Information

To ensure high-quality supervision, we selected nine actively maintained open-source hardware projects with substantial user engagement and clearly labeled issue reports. From these, we identified 3,172 confirmed bug fixes, each explicitly tagged as a bug. The version control data provided the buggy RTL code ❸ and the corresponding patch files, which revealed precise bug locations.

Building on this base, we employed LLMs to generate the remaining components. Using prompts adapted from Section III-B, we synthesized high-level functional descriptions, line-by-line RTL explanations ❼, and detailed bug summaries ❼. With the buggy design and associated bug metadata as input, the LLM then generated SVAs that directly target the identified issues ❹.

This workflow required two prompt stages: one for functional and bug-specific contextualization, and another for assertion generation. All LLM outputs were reviewed, and assertions were validated through RTL simulation to ensure syntactic and semantic correctness.

In total, the version control pathway contributed 3,172 designs and 15,281 verified assertions to the dataset.

D. Data Augmentation from Formal Verification Automation Tools

We selected AutoSVA2 [14] as the assertion generation tool. As it supports only module-level inputs and RISC-V cores, we collected 165 modules from five open-source RISC-V SoCs and cores ❻. Minor modifications and annotations were applied to enhance flexibility and enable assertion synthesis.

AutoSVA2 generated SVAs through symbolic analysis ❺, with annotations indicating vulnerability locations and logic for assertion placement. These outputs enabled extraction of vulnerability descriptions and linkage between RTL behavior and formal properties. However, while symbolically valid, the annotations lacked functional or contextual explanations.

To address this, we used LLMs to generate RTL explanations, line-by-line comments, assertion justifications, and mappings between SVAs and hardware structure ❼. Custom prompts ensured semantic alignment with the formal logic.

All outputs were manually verified for clarity. As AutoSVA2 ensures logical correctness, simulation-based validation was unnecessary. This process yielded 1,433 assertions across 165 hardware modules.

E. Unified Dataset Structure and Example

By combining the three data sources, HADA constructs a diverse fine-tuning dataset comprising `instruction, input, output` triplets. The following is a *compressed example* of a fine-tuning instance in HADA. It is simplified to fit the paper's format constraints and does not reflect the whole data structure used in training.

Instruction: *(Simplified)* Explain the design and the impact of the fault.

Input:
```
module memory_access(********)
*****************************
  if (we) mem[addr] <= data_in;
  else data_out <= mem[addr];
*****************************
endmodule
```

Output: *(Condensed)* This module handles memory read/write. The fault at line 12 causes out-of-bounds access (CWE-1259). The assertion `assert property (addr < 256);` ensures memory safety by bounding address range.
// Note: This is a shortened version shown for illustrative purposes only.

TABLE II
TRAINING AND INFERENCE RESOURCE USAGE

Activity	Usage
Training (A100 GPUs)	895.156 GPU hours, 268.55 kWh
Dataset Analysis	117.88 GPU hours, 35.67 kWh
Inference Time (per query)	
Llama3	1.08s
Llama3.1	1.36s
Llama3.2	0.56s
GPT-4o-mini	1.60s

IV. EXPERIMENTAL DETAILS

A. Experimental Setup

Dataset: We used the full dataset described in Section III, combining all three sources for general evaluation. To analyze individual contributions, we also trained models on source-specific subsets and a variant containing only raw (non-LLM-generated) data.

Models: We evaluated four LLMs: LLaMA 3 70B, LLaMA 3.1 70B, LLaMA 3.2 3B, and GPT-4o-mini.

Training: GPT-4o-mini was fine-tuned with standard supervised learning using a learning rate of $5e{-}5$. For other models, we applied LoRA [22] fine-tuning with a learning rate of $1e{-}4$. To mitigate quantization sensitivity, we used PISSA [23] in place of standard LoRA adapters (SVD iteration = 4, $r = 16$, $\alpha = 32$).

To improve training efficiency and reduce memory footprint, we used Unsloth [24], FlashAttention2 [25], and Liger Kernel [26], with 4-bit quantization via bitsandbytes [27] for the 70B models.

All models were trained for three epochs on 8×NVIDIA A100 80GB GPUs with at least 200GB memory allocated per run.

B. Energy Consumption and Training Time

As summarized in Table II, Llama3.2 achieves the fastest inference time, while training and analysis required over 1,000 GPU hours.

For a real-world SoC, we used OpenTitan as an example. Each module was processed independently by the LLM, allowing for efficient parallelization. Using the fine-tuned version of GPT-4o-mini, generating **10 assertions per design** for all OpenTitan IP cores took approximately **10 minutes**, with an API cost of around **$30**.

C. HSAEval: An Open-Source Benchmark for Security Assertions

We develop **HSAEval** (Hardware Security Assertion Evaluation), an open-source (soon-to-be-released) benchmark designed to evaluate the effectiveness of assertions in detecting security vulnerabilities. HSAEval is derived from real-world hardware projects to ensure diverse coverage of security issues. Specifically, it comprises **46** security verification tasks, including **18** from TrustHub, **20** from CWE, and **8** from OpenPiton.

Each task includes a SystemVerilog testbench and a specific security issue. These testbenches validate the syntactic correctness of the generated assertions and their functional effectiveness using JasperGold [28]. Performance evaluation in HSAEval is conducted using *zero-shot prompting* and *pass@1,5,10* as in Equation 1.

$$\text{pass@k} = \mathbb{E}_{\text{Problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{1}$$

where $n \geq k$ is the total number of generated samples per task, and $c \leq n$ is the number of samples that pass testing. The metric pass@k evaluates the probability that at least one correct assertion appears within the top k responses from the LLM.

TABLE III
PERFORMANCE COMPARISON OF LLMS PRE- AND POST-FINE-TUNING ON FVEVAL AND HSAEVAL (FUNC $pass@1,5$)

Model		FVEval: Nvidia FV Real World Benchmark				HSAEval: Benchmark from open source SoC	
		FSM		Pipeline			
		Functionality					
		pass@1	***pass@5***	***pass@1***	***pass@5***	***pass@1***	***pass@5***
GPT4o-mini	base	10.11%	41.37%	8.81%	37.01%	11.96%	23.91%
	HADA	9.42%	39.08%	34.52%	88.03%	15.22%	32.60%
LlaMA3 70B	base	17.08%	60.89%	12.03%	47.39%	11.96%	17.39%
	HADA	30.58%	83.95%	23.19%	73.35%	17.39%	34.78%
LlaMA3.1 70B	base	24.26%	75.16%	18.93%	65.06%	7.17%	15.22%
	HADA	30.58%	83.95%	23.19%	73.35%	12.60%	30.43%
LlaMA3.2 3B	base	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	HADA	2.41%	11.49%	4.10%	18.91%	1.30%	2.17%
SOTA General Proprietary LLM							
GPT4o		10.40%	42.70%	37.30%	90.00%	26.09%	28.26%
Prompt Methods - base on GPT4o mini							
RTLFixer [30]		0.00%	0.00%	0.00%	0.00%	0.22%	1.19%
DIVAS [31]		0.00%	0.00%	0.00%	0.00%	0.22%	1.19%
LAAG [9]		0.00%	0.00%	0.00%	0.00%	0.22%	1.19%
Fine Tuned LLM							
LLM4SecHW [11]		0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Additionally, we incorporated **FVEval** [29], a formal verification benchmark for digital hardware, as a supplementary evaluation. FVEval consists of two categories: **FSM** and **Pipeline**, each containing **96** designs with varying difficulty distributions. Like HSAEval, we assessed FVEval's performance using JasperGold, zero-shot prompting, and *pass@k* metrics to evaluate syntactic and functional correctness.

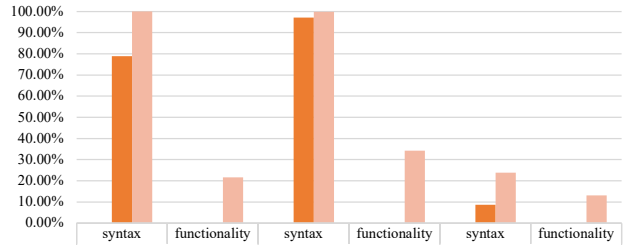
D. HADA Framework Evaluations

We conducted four experiments using **HSAEval** and **FVEval**. For each experiment, we evaluated *syntactic correctness* and *functional correctness* using the pass@k metric under zero-shot prompting.

1) Base Model Performance and Training Effectiveness: Table III compares the performance of different LLMs before and after fine-tuning in generating valid assertions for hardware tasks using **FVEval** and **HSAEval**. The results show that, except for a slight decline (5%) by GPT-4o-mini on the FSM task in FVEval, all other models exhibited improvements after fine-tuning. Notably, **LLaMA 3.2 3B**, a typical small model, was initially unable to handle hardware verification tasks but gained the ability to generate valid assertions after fine-tuning. Larger models, including **GPT-4o-mini** and the **70B** versions of the LLaMA series, also demonstrated positive performance gains.

2) Ablation: Augmented vs. Raw: In Fig. 2, we fine-tuned the **LLaMA 3.2 3B** model on raw data and compared it with the same model fine-tuned on the augmented dataset, with all evaluations measured using $pass@10$. The base model exhibited no hardware-related capabilities, scoring **zero** on both **FVEval** and **HSAEval**. After fine-tuning on raw data, the model showed significant improvements in syntax comprehension, achieving $pass@10$ scores of **78.84%**, **97.11%**, and **8.70%** across evaluation tasks. However, its functionality remained at zero, indicating that while the model learned SystemVerilog syntax and the basic construction of SVAs, it still lacked security-related reasoning. Fine-tuning on the augmented dataset led to further improvements in syntax capabilities, with $pass@10$ scores increasing by **26%**, **2%**, and **175%**, reaching **0.9999**, **0.9988**, and **0.2391**, respectively. More importantly, the augmented model demonstrated the ability to generate correct assertions, highlighting the effectiveness of HADA.

3) Ablation Study: Independent Source Data Scaling: We fine-tuned the **LLaMA 3.2 3B** model on varying-sized subsets (500, 1,000, and 1,500 samples) from each of the three data sources—**CWE**, **VC**, and **FPV**—and evaluated performance on



	FSM syntax	FSM functionality	Pipeline syntax	Pipeline functionality	HSAEval syntax	HSAEval functionality
base	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Raw data	78.84%	0.00%	97.11%	0.00%	8.70%	0.00%
HADA	99.99%	21.71%	99.88%	34.32%	23.91%	13.04%

Fig. 2. Comparisons of training results using augmented and raw data, $pass@10$.

FVEval (Fig. 3). Results show that **VC**-trained models consistently outperform others across all dataset sizes, highlighting the value of diverse real-world bug patterns in version control data. In contrast, **FPV** data, though derived from actual hardware designs, suffers from low variability due to repetitive assertion patterns. **CWE**-generated samples exhibit limited diversity, as LLM outputs for similar vulnerabilities often converge. These findings suggest that source diversity, not just data quantity, improves model performance.

E. Ablation Study: Impact of Data Source Combinations

We constructed datasets by selecting two of the three available sources (resulting in three combinations) or combining all three. Each dataset was adjusted to the same size, and the **LLaMA 3.2 3B** model was fine-tuned on these datasets. This ablation study evaluates the relative importance of different data sources. Fig. 4 presents the evaluation of four models trained on datasets combining two sources (**1,350** samples from each) and all three sources (**900** samples from each). The results indicate that models trained on datasets including **VC** consistently outperformed others. The model trained on all three sources (**All**) achieved superior performance in the hardware-focused **Pipeline** task compared to the **CWE+VC** combination, suggesting that **FPV** data enhances performance in hardware-specific scenarios. However, this improvement came at the cost of weaker performance in the **FSM** task, due to the repetitive and less diverse nature of FPV assertions. The lowest performance was observed with the **CWE+FPV** combination, emphasizing the critical role of **VC** data in achieving robust results across tasks.

F. Conclusion from Experimental Results

Based on the four experiments, we conclude that data augmentation in HADA significantly enhances the performance of domain-specific LLMs in generating hardware security assertions. Increasing the diversity of training data during augmentation substantially improves the quality of hardware datasets.

Among the different data synthesis approaches, **FPV** subset provides less benefit than **CWE**, and both are less effective than incorporating **VC** subset from diverse open-source projects. The repetitive patterns inherent in formal method-based and LLM-assisted generation negatively impact data quality, whereas leveraging version control data ensures greater diversity and robustness.

V. Discussion

A. Is HADA a Distillation of GPT-4o?

In short, no. Traditional LLM distillation involves using a teacher model to generate answers for a predefined question set, implicitly transferring its knowledge through the resulting dataset. This process often risks accumulating hallucinations and errors—a phenomenon we refer to as an "LLM runaway effect."

In contrast, **GPT-4o in our workflow is used solely for explanation and format transformation**, not as a source of core knowledge. Our approach derives its supervision from three independent sources: (1) version control data curated by open-source project maintainers (**VC**), (2) formal security features defined by tool developers and open hardware contributors (**FPV**), and (3) structured domain knowledge from the **CWE** database.

B. Is Manual Verification Still Complex?

No, it is significantly simplified. Including detailed textual explanations, covering hardware functionality, defect causes, and assertion justifications, substantially reduces the reasoning burden. As a result, manual verification primarily depends on reading speed rather than specialized domain knowledge.

In contrast, verifying **raw data** requires a deep understanding of hardware design and formal assertions, demanding considerable expertise and making the process more complex.

C. Are There Additional Statistical Supports?

No. The generative behavior of LLMs is inherently stochastic. Metrics such as **pass@k** represent mathematical expectations, capturing the probability that at least one correct output appears within k generations, rather than deterministic guarantees.

Furthermore, LLM training involves randomness at multiple stages, and repeated training runs do not yield identical results. As such, variations in performance (e.g., across different randomly selected subsets of 500 training samples, as discussed in Section IV-D3) are expected and do not provide statistically meaningful insights beyond what the stochastic optimization process already entails.

D. Open-Source Plan

We will release all key components of our work to promote transparency, reproducibility, and future research. Specifically, we will open-source:

- All fine-tuned models based on open LLMs;
- The full training datasets collected from three distinct sources;
- Raw outputs and evaluation results on the test sets.

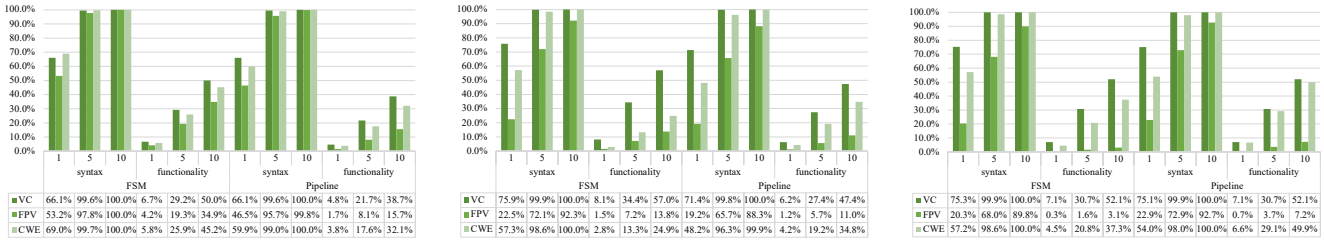
This release will facilitate further development in hardware security verification and encourage the community to build upon our approach.

VI. Related Work

Table IV and Table III compare our approach with prior methods. While some achieve **100%** syntactic correctness by injecting external knowledge, this often leads to semantic errors that degrade overall performance. These methods fail on real-world verification benchmarks, with an effective success rate of **0**—even missing correct answers previously achievable without such enhancements.

Unlike LLM4SecHW [11] and [13], which rely on version control data, our approach integrates directly with EDA workflows by generating SVAs without requiring additional input. While all approaches depend on maintainers to provide structured data, we employ data augmentation to make raw inputs more suitable for LLMs, improving usability and performance beyond patch-based methods.

Compared to AutoSVA2 [14], HADA scales better and reduces computational cost by leveraging longer LLM contexts. While formal methods rely heavily on expert-defined security rules—a major bottleneck in practice—our approach sidesteps this by using LLM-based



(a) 500 samples from each data subset (b) 1,000 samples from each data subset (c) 1,500 samples from each data subset

Fig. 3. Comparisons of fine-tuning results of independent source quantity scaling, where $k \in \{1, 5, 10\}$ in $pass@k$.

TABLE IV
FEATURE COMPARISON BETWEEN THE PROPOSED METHOD AND RELATED WORKS.

Solution	Method	Information Source	Assertion	Fully Open	User Requirements	Automation Level	Security & Functionality
LLM4SecHW [11]	SFT	VC	✗	✓	None	4	Both
[32]		Design Document	✗	✓	None	4	Functionality Only
DIVAS [31]	Prompt	Design converted to CWE	✓	✗	Every test target	1	Both
[10]		Not Apply	✓	✗	Sample assertion of the property	2	Security Only
LLM4DV [33]			✗	✗	Task details	2	Functionality Only
[34]			✗	✗	Error injection details	2	Functionality Only
RTLFixer [30]			✗	✗	Description and modifications	3	Functionality Only
MEIC [35]			✓	✗	Feedback integration with LLM	3	Functionality Only
Verilog Reader [36]			✗	✗	All format information provided	1	Both
[21]			✓	✗	All details	1	Both
AutoSVA2 [14]	FPV		✓	✗	None	4	Both
HADA	SFT	VC, FPV, CWE	✓	✓	None	4	Security Only

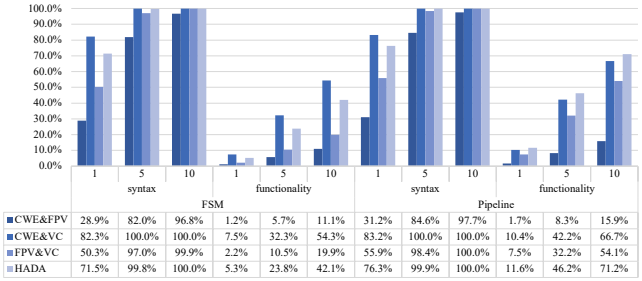


Fig. 4. Fine-tuning results comparison of balanced combined data sources, where $k \in \{1, 5, 10\}$ in $pass@k$.

static analysis to approximate formal reasoning, without requiring precise rule definitions, and incorporates outputs from multiple tools for broader coverage than single-tool formal approaches.

[32] employs BERT (2018) trained on hardware documentation, which introduces additional overhead for project maintainers. In contrast, our approach leverages standard development workflows, utilizes a more advanced model, and draws from a broader, more representative dataset, resulting in more robust and up-to-date outcomes. By integrating SVA generation into existing EDA pipelines, we enhance usability and facilitate adoption while avoiding the scalability limitations of manually curated documentation.

Prompt-based approaches in this field generally fall into two categories. The first, exemplified by DIVAS [31], LLM4DV [33], Verilog Reader [36], and others [10], [21], [34], requires users to provide detailed, tutorial-like prompts. These methods rely heavily on the user's domain knowledge, resembling guided code completion rather than autonomous verification.

The second category involves LLM agents, such as RTLFixer [30] and MEIC [35], which incorporate simulation feedback into LLM interactions. While offering more automation, they remain limited by predefined scenarios and simulator-specific constraints.

In contrast, our approach minimizes user input and eliminates the need for security or hardware expertise. It can also act as an LLM agent when needed, positioning it as a more general and adaptable engine that complements existing agent-based systems.

VII. Conclusion

This work presents HADA, a domain-specific framework that leverages multi-source data to fine-tune LLMs for hardware security assertion generation. By integrating formal verification outputs, structured vulnerability knowledge, and real-world version control data, we construct a comprehensive dataset that enables LLMs to generate high-quality SystemVerilog assertions with contextual understanding. Experimental results on the HSAEval and FVEval benchmarks demonstrate substantial syntactic and functional correctness improvements over baseline models. Unlike prior prompt-based or tool-constrained methods, HADA supports scalable, low-intervention assertion generation and aligns with modern EDA workflows. All datasets, models, and evaluation tools will be released to facilitate reproducible research and future development in this emerging area.

References

[1] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014. [Online]. Available: https://doi.org/10.1109/JPROC.2014.2335155

[2] K. Xiao, D. Forte *et al.*, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Design Autom. Electr. Syst.*, vol. 22, no. 1, pp. 6:1–6:23, 2016. [Online]. Available: https://doi.org/10.1145/2906147

[3] A. Chakraborty, N. G. Jayasankaran *et al.*, "Keynote: A disquisition on logic locking," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 1952–1972, 2020. [Online]. Available: https://doi.org/10.1109/TCAD.2019.2944586

[4] K. Basu, S. M. Saeed *et al.*, "Cad-base: An attack vector into the electronics supply chain," *ACM Trans. Design Autom. Electr. Syst.*, vol. 24, no. 4, pp. 38:1–38:30, 2019. [Online]. Available: https://doi.org/10.1145/3315574

[5] M. Ender, A. Moradi, and C. Paar, "The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series fpgas," in *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, S. Capkun and F. Roesner, Eds. Boston, MA, USA: USENIX Association, 2020, pp. 1803–1819. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/ender

[6] G. Dessouky, D. Gens *et al.*, "Hardfails: Insights into software-exploitable hardware bugs," in *28th USENIX Security Symposium, USENIX Security 2019, August 14-16, 2019*, N. Heninger and P. Traynor, Eds. Santa Clara, CA, USA: USENIX Association, 2019, pp. 213–230. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky

[7] H. Witharana, A. Jayasena *et al.*, "Automated generation of security assertions for RTL models," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 1, pp. 8:1–8:27, 2023. [Online]. Available: https://doi.org/10.1145/3565801

[8] W. Fu, O. Arias *et al.*, "Fuzzing hardware: Faith or reality? : Invited paper," in *IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2021, November 8-10, 2021*. AB, Canada: IEEE, 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1109/NANOARCH53687.2021.9642252

[9] K. Maddala, B. Mali, and C. Karfa, "LAAG-RV: LLM assisted assertion generation for RTL design verification," *CoRR*, vol. abs/2409.15281, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.15281

[10] R. Kande, H. Pearce *et al.*, "(security) assertions by large language models," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 4374–4389, 2024. [Online]. Available: https://doi.org/10.1109/TIFS.2024.3372809

[11] W. Fu, K. Yang *et al.*, "Llm4sechw: Leveraging domain-specific large language model for hardware debugging," in *Asian Hardware Oriented Security and Trust Symposium, AsianHOST 2023, December 13-15, 2023*. Tianjin, China: IEEE, 2023, pp. 1–6. [Online]. Available: https://doi.org/10.1109/AsianHOST59942.2023.10409307

[12] H. Witharana, Y. Lyu *et al.*, "A survey on assertion-based hardware verification," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 225:1–225:33, 2022. [Online]. Available: https://doi.org/10.1145/3510578

[13] W. Fu, S. Li *et al.*, "A generalize hardware debugging approach for large language models semi-synthetic, datasets," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 2, pp. 623–636, 2025.

[14] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using llms to facilitate formal verification of RTL," *CoRR*, vol. abs/2309.09437, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.09437

[15] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[16] Y. Lu, S. Liu *et al.*, "RTLLM: an open-source benchmark for design RTL generation with large language model," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference, ASPDAC 2024, January 22-25, 2024*. Incheon, Korea: IEEE, 2024, pp. 722–727. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473904

[17] A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," *CoRR*, vol. abs/2407.21783, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.21783

[18] M. AI, "Llama 3.1 - 70b instruct model," https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct, 2024.

[19] ——, "Llama 3.2 - 3b model," https://huggingface.co/meta-llama/Llama-3.2-3B, 2024.

[20] OpenAI, "Gpt-4o mini: Advancing cost-efficient intelligence," https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

[21] W. Fang, M. Li *et al.*, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *CoRR*, vol. abs/2402.00386, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.00386

[22] E. J. Hu, Y. Shen *et al.*, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*. Virtual: OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[23] F. Meng, Z. Wang, and M. Zhang, "PiSSA: Principal singular values and singular vectors adaptation of large language models," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA: Curran Associates, Inc., 2024. [Online]. Available: https://openreview.net/forum?id=6ZBHIEtdP4

[24] U. Contributors, "Unsloth: Efficient data loading and transformation for machine learning," https://github.com/unslothai/unsloth, 2023.

[25] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," in *The Twelfth International Conference on Learning Representations, ICLR 2024, May 7-11, 2024*. Vienna, Austria: OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=mZn2Xyh9Ec

[26] P. Hsu, Y. Dai *et al.*, "Liger kernel: Efficient triton kernels for LLM training," *CoRR*, vol. abs/2410.10989, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.10989

[27] H. Face, "Bitsandbytes documentation," https://huggingface.co/docs/bitsandbytes/main/en/index, 2024.

[28] Cadence Design Systems, *JasperGold Apps User's Guide*, Cadence Design Systems, San Jose, CA, 2015, version 2015.12.

[29] M. Kang, M. Liu *et al.*, "Fveval: Understanding language model capabilities in formal verification of digital hardware," 2024. [Online]. Available: https://arxiv.org/abs/2410.23299

[30] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing RTL syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC 2024, June 23-27, 2024*, V. De, Ed. San Francisco, CA, USA: ACM, 2024, pp. 53:1–53:6. [Online]. Available: https://doi.org/10.1145/3649329.3657353

[31] S. Paria, A. Dasgupta, and S. Bhunia, "DIVAS: an llm-based end-to-end framework for soc security analysis and policy-based protection," *CoRR*, vol. abs/2308.06932, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.06932

[32] X. Meng, A. Srivastava *et al.*, "Unlocking hardware security assurance: The potential of llms," *CoRR*, vol. abs/2308.11042, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.11042

[33] Z. Zhang, G. Chadwick *et al.*, "LLM4DV: using large language models for hardware test stimuli generation," *CoRR*, vol. abs/2310.04535, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.04535

[34] L. J. Wan, Y. Huang *et al.*, "Invited paper: Software/hardware co-design for LLM and its application for design verification," in *Proceedings of the 29th Asia and South Pacific Design Automation Conference, ASPDAC 2024, January 22-25, 2024*. Incheon, Korea: IEEE, 2024, pp. 435–441. [Online]. Available: https://doi.org/10.1109/ASP-DAC58780.2024.10473893

[35] K. Xu, J. Sun *et al.*, "MEIC: re-thinking RTL debug automation using llms," *CoRR*, vol. abs/2405.06840, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.06840

[36] R. Ma, Y. Yang *et al.*, "Verilogreader: LLM-Aided hardware test generation," in *Proceedings of the 2024 IEEE LLM Aided Design Workshop (LAD)*. San Jose, CA, USA: IEEE, 2024, pp. 1–5.