(Special Session)

OpenAssert: Towards Secure Assertion Generation using Large Language Models

Anand Menon*, Samit Shahnawaz Miftah*, Amisha Srivastava*, Shamik Kundu†, Shovik Kundu†, Arnab Raha†, Suvadeep Banerjee†, Deepak Mathaikutty†, and Kanad Basu*
*University of Texas at Dallas, TX, USA
†Intel Labs, USA

2025 IEEE 43rd VLSI Test Symposium (VTS) | 979-8-3315-2144-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/VTS65138.2025.11022798

***Abstract*—Assertions are critical components used in hardware verification, ensuring robust functionality, fortifying design security, and providing essential verification features. Traditional hardware assertion methods are not automated, complicate security audits, and require effort, causing prolonged development cycles. Recent studies have highlighted the potential of commercial Large Language Models (LLMs) to generate security-focused assertions by leveraging textual data from design specifications. However, reliance on proprietary models like GPT-4 severely jeopardizes IP privacy and data confidentiality, undermining transparency and accountability in data handling practices. In this paper, we address secure hardware assertion generation by proposing a practical approach to significantly enhance the feasibility of open-source LLMs. Our proposed method, OpenAssert, involves fine-tuning existing models to be utilized locally at the user's end without compromising confidentiality. Additionally, we employ Retrieval Augmentation Generation to refine these models, mitigating hallucinations and security-related errors. OpenAssert demonstrates improvements, achieving up to a 44% increase in rouge-1 score, a 49% improvement in cosine similarity, and a 43.4% reduction in word error rate for security-critical designs compared to open-source models.**

***Keywords*—LLMs, Assertion, Verification**

I. Introduction

The landscape of computing systems is increasingly dominated by System-on-Chips (SoCs), which integrate various Intellectual Property (IP) cores for enhanced functionality. Although this integration accelerates market delivery and reduces costs, it introduces challenges in identifying security vulnerabilities within complex designs, potentially increasing development time by more than 70% [1]. Post-fabrication bug fixes can increase costs, making rigorous hardware verification crucial to ensure design integrity, performance, and security.

Hardware assertions using *SystemVerilog* Assertions (SVA) are a vital component in identifying security flaws and early detection of bugsbugs within the design process. Traditional methods for generating SVAs are manual and rely heavily on designer expertise. This process is labor-intensive and error-prone. As a result, it is increasingly susceptible to security vulnerabilities, especially as designs become more complex.

Large Language Models (LLMs) have emerged as a solution, utilizing textual data (*e.g.*, technical documentation, code) to efficiently produce SVAs. However, existing LLM solutions often rely on proprietary models like GPT-4 [2], [3], [4], restricting user control and heightening security and privacy concerns, particularly when deployed on untrusted servers.

This paper introduces OpenAssert, a framework that uses open-source LLMs to create tailored verification assertions for RTL designs. We developed a specialized dataset with 250 million tokens for SVA, fine-tuned the open-source CodeLlama model with this dataset, and incorporated advanced retrieval techniques to minimize errors and hallucinations.

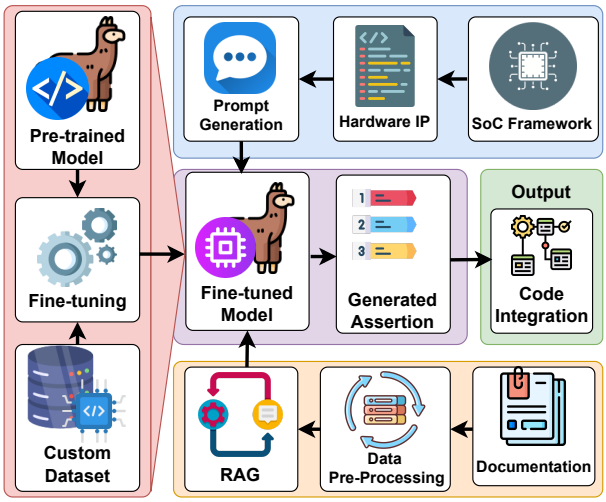

Fig. 1. Overview of proposed OpenAssert framework.

The contributions of this paper are as follows:

1) We present OpenAssert, which adapts open-source LLMs for SVA generation to improve security.
2) We create and utilize a specialized SVA dataset of 250 million tokens to fine-tune the CodeLlama model [5].
3) We implement Retrieval-Augmented Generation (RAG) to improve accuracy and reduce hallucinations, optimizing overall security and performance.
4) Evaluation of OpenAssert on six OpenTitan IPs demonstrates improvements of up to 49% compared to the base Open-Source LLM and 23.7% compared to GPT-4o, highlighting its ability to address security-critical flaws.

II. Proposed Assertion Generation Framework

In this section, we introduce OpenAssert, a framework designed to generate security-oriented assertions for RTL designs. OpenAssert takes RTL designs and documentation, leveraging LLMs to produce SVA that detect vulnerabilities. The primary task of OpenAssert involves comprehending the functionalities and potential vulnerabilities outlined in documentation, localizing the corresponding modules within

TABLE I
EXAMPLES FROM OUR DATABASE.

Source	Explanation	Assertions	Type

Obtained from Online Source	Generate an assertion that checks asynchronously that the enable_signal is not active when reset is asserted.	`always_comb begin`
		`  assert (enable_signal || !reset) else $fatal("Enable signal should not be active during reset");`
		`end`
			Immediate asynchronous assertion

Manually Created	Generate an assertion that checks asynchronously at every positive edge of the clock that data_width is 8. If not, it triggers an error.	`always @(posedge clk) begin`
		`  assert (data_width == 8) else $fatal("Data width is not 8 bits");`
		`end`
			Immediate synchronous assertion

Obtained from an Open-Source repository	Generate an assertion that checks when state is IDLE at the positive edge of the clock, data_valid should be low.	`property p_sync_assertion;`
		`  @(posedge clk) disable iff (reset)`
		`  (state==IDLE)|=>(data_valid==1'b0);`
		`endproperty`
			Concurrent Synchronous assertion

Synthetically Generated	Generate an assertion that checks that whenever reset is asserted, the enable_signal should be immediately low	`property p_async_assertion;`
		`  disable iff (reset)`
		`  (!reset|->##0 enable_signal==1'b0);`
		`endproperty`
			Concurrent asynchronous assertion

You are an expert in verilog code assertions, answer the question in the following format:

property <name>;
(<list of conditions>) |-> (<list of assignment checks>);
endproperty

Fig. 2. Example of the System Prompt for fine-tuning the model.

the RTL code responsible for implementing these functionalities, and identifying the pertinent registers and conditions. OpenAssert generates assertions to validate functionality and detect vulnerabilities in the design. Our approach, OpenAssert, is equipped with three essential components, as shown in Fig. 1: (1) A dataset containing 250 million tokens, specifically tailored towards SVA. (2) An LLM model, trained on the dataset to grasp HDL designs and also identify threats. (2) Context injection via RAG for understanding documentation and RTL code.

A. Fine-tuned Model

Off-the-shelf pre-trained LLMs are not optimized for generating SVA. They often exhibit the following issues: (1) incorrect sequencing of conditions, (2) omission of necessary security conditions, and (3) hallucinations. To address these issues, we create a specialized dataset tailored to train our LLM for SVA generation. We then train the model on this dataset to produce correct synthesizable assertions.

1) Dataset Creation: In order to develop the training dataset, *we collected examples from various sources, including open-source repositories, academic literature, and online resources, supplemented by manually crafted examples [6], [7].* Table I presents examples from our dataset, detailing the source of each assertion in Column 1, the task description in Column 2, the assertion in Column 3, and the type of assertion in Column 4. While Table I highlights four common types of assertions—concurrent synchronous, immediate synchronous, concurrent asynchronous, and immediate asynchronous—these are illustrative examples, not exhaustive. The framework supports flexibility in creating other types of assertions beyond those shown in Table I, allowing users to tailor assertions to the specific requirements of their designs. Each assertion is linked to the relevant source code and is accompanied by a brief explanation of its function, providing context and clarifying its role within the design.

Next, we adapted the dataset to the Llama-2 instruction format, comprising a *system instruction*, *input*, and *output*, ensuring an easier learning process for the model from the structured data. Furthermore, we included a structural template describing the construct of an SVA to guide the LLM in generating assertions with valid syntactical forms [8]. Figure 2 provides an example of the system prompt along with the aforementioned structural template [8].

2) Fine-tuning: In our proposed approach, we utilize open-source CodeLlama models derived from Llama 2. Specifically, the 34 billion parameter variant of CodeLlama was selected due to resource constraints. *However, researchers with greater resources can utilize higher parameter models for improved performance*. These models are widely embraced across various frameworks and remain the best-performing open-source models in code generation [5]. First, we fine-tune the model on our dataset. Due to constraints in training hardware, we opted to utilize Low-Rank Adapters (LoRA) because it can fine-tune pre-trained models while conserving resources [9]. We carefully chose the model parameters to optimize the fine-tuning process for hardware verification. With computing limitations, we decided on a batch size of 16 samples and a context length of 8192. We trained the model over one epoch with a learning rate of 3×10^{-4}.

We opted for a LoRA rank of 256, where LoRA rank refers to the dimensionality of the low-rank approximation matrix used to modify the model's parameters. The alpha, acting as the scaling factor, was set to double the rank (512). We chose these values because higher LoRA ranks can lead to over-fitting. Targeting all transformer layers results in better adaptation than utilizing only the Q and K layers, as further corroborated by QLoRA [10].

B. Context Injection through RAG

To generate assertions for a specific design, the model must comprehend design-specific register names and logic. **This is accomplished by feeding the design documentation into the model through RAG, enabling it to discern crucial details on the design's high-level structure, including major components and interactions.** Injecting context into the

model involves pre-processing the documentation and then providing the processed data to the model using RAG.

1) Preprocessing Data: RAG struggles with large blocks of text, necessitating segmentation into smaller chunks. For this purpose, the system loads markdown files that contain detailed explanations and annotations on the source code's functionality, architecture, and usage. Since these files can be of considerable size, exceeding the model's context capacity, we divide them into manageable segments. Through context-aware splitting, we obtain semantically relevant chunks with pertinent metadata—such as register names and event flow—extracted from the markdown files.

2) RAG Integration: After preprocessing the data, we integrate RAG into OpenAssert to incorporate relevant hardware documentation during assertion generation. This enhancement utilizes the *all-roberta-large-v1* model, known for strong text-processing capabilities in capturing semantic meaning from extensive documentation [11]. The model produces 1024-dimensional vectors to represent text essence, enabling advanced semantic searches within a vector database. Queries encoded in the same vector space facilitate precise retrieval of documentation snippets based on cosine similarity. This integration enriches the LLM's contextual grasp by injecting specific register details and design functionality. Consequently, the fine-tuned LLM achieves notably improved accuracy, reducing mistakes in assertion generation for hardware design and effectively leveraging design-specific information directly from documentation.

III. Results

A. Experimental Setup

We evaluated OpenAssert on the OpenTitan SoC, chosen for its popularity, security emphasis, and robust FSM implementation. Key IPs used include *UART, I2C, LC CTRL, OTBN, KMAC*, and *SYSRST CTRL*. Our platform includes an Nvidia DGX server with four A-100 GPUs.

Performance metrics: To evaluate OpenAssert's ability to generate security SVA effectively, it is essential to ensure that the produced assertions demonstrate comparable syntax and logical attributes to the reference code. To accomplish this, we used the following three performance metrics:

- **Rouge-1** or Recall-Oriented Understudy for Gisting Evaluation measures the overlap of unigram tokens between the generated output and the reference summaries. A higher Rouge-1 score would imply that the generated assertions capture more of the essential features and logic present in the reference assertions, indicating a better similarity between the two.
- **Word Error Rate (WER)** evaluates automatic speech recognition and machine translation. It calculates the percentage of differing words (substitutions, insertions, deletions) in the generated output compared to the reference. A lower WER would suggest that there are fewer discrepancies or differences between the generated assertions and the reference assertions.
- **Cosine Similarity** measures the similarity between vectors by the cosine of the angle based on various aspects of the code, including syntax and structure. Cosine similarity is particularly useful for comparing the semantic similarity between texts, as it considers the orientation (direction) of the vectors rather than their magnitude [12].

B. Experimental Results

In this section, we present our results comparing the performance of the different strategies used to enhance the baseline LLM's performance, which are showcased in Table II. The table is organized into five major columns that evaluate the performance of different language models, specifically the baseline CodeLlama-34B LLM, the baseline model supplemented with RAG, the fine-tuned 34B LLM, and OpenAssert, which is fine-tuned and assisted by RAG. Each column is further divided into three subcolumns corresponding to the performance metrics used. The generation process continues iteratively until all high-priority requirements specified in the design documentation (provided by RAG) have been translated into SVAs. The flow stops generating assertions when no additional specification items remain unaddressed.

Coverage Evaluation: We evaluate the generated assertions using line coverage, a fundamental metric in assertion coverage [13]. **This ensures that the lines in the design source code are functionally correct.** Our evaluation using mutation testing demonstrates up to 100% line coverage, showing that every individual condition contributing to the assertion logic has been executed [14], [15]. This approach involved introducing small, intentional code modifications (which we call *mutants*) that deviated from the expected assertion logic. By detecting these mutants, we demonstrate the effectiveness of the generated assertions in identifying logical inconsistencies and validating their utility.

Base 34B LLM with RAG: A comparative analysis between columns 2-4 and 5-7 of Table II showcases the outcomes of employing RAG solely to enrich the model with additional context. The performance metrics for the designated hardware blocks within the OpenTitan benchmark are derived utilizing manually crafted assertions as golden references. This practice is essential since *existing automated systems fail to address the intricate details of complex hardware designs adequately*.

The consistent use of RAG demonstrably enhances performance compared to the base model, as evidenced by the comparison of rouge-1 and cosine similarity scores. The rouge-1 scores range from 0.56 to 0.81, surpassing the baseline model's scores, which range from 0.47 to 0.73, yielding an improvement of up to 25%. In terms of cosine similarity, the RAG-enhanced model achieves scores between 0.62 and 0.84. This is a notable increase from the baseline model's scores, which range from 0.42 to 0.74. Overall, these improvements correspond to an enhancement of up to 34%, indicating a significant increase in the semantic coherence of the RAG-enhanced model compared to the baseline.

However, WER scores reveal limitations when relying solely on RAG. The baseline model exhibits WER ranging from 68% to 86%, whereas the use of RAG results in rates between 67% and 85%. It is observed that despite RAG providing additional context, the model lacks the requisite knowledge for precise SVA generation. Moreover, specific instances, such as with the hardware blocks *LC CTRL* and *KMAC*, illustrate scenarios where RAG fails to supply accurate context, resulting in erroneous assertions and a higher WER.

TABLE II
PERFORMANCE COMPARISON OF RAG ASSISTED MODEL, FINE-TUNED MODEL AND OPENASSERT WITH BASE CODELLAMA MODEL

Evaluation Module	Base 34B LLM (CodeLlama)			Base 34B LLM with RAG			Fine-Tuned 34B LLM			OpenAssert		
	Rouge	Cosine	Word ER	Rouge	Cosine	Word ER	Rouge	Cosine	Word ER	Rouge	Cosine	WordER
UART	0.56	0.50	72.73%	0.81	0.84	77.36%	0.76	0.81	45.46%	0.87	0.84	36.36%
I2C FSM	0.48	0.58	68.3%	0.62	0.78	67.3%	0.71	0.76	61.5%	0.70	0.84	44.05%
LC CTRL	0.72	0.74	61.54%	0.83	0.77	77.58%	0.77	0.87	46.15%	0.91	0.88	31.58%
OTBN	0.63	0.63	86.64%	0.67	0.71	85.6%	0.72	0.79	54.5%	0.84	0.81	33.33%
KMAC	0.73	0.74	68.42%	0.72	0.76	81.42%	0.74	0.81	63.6%	0.81	0.78	40%
SYSRST CTRL	0.47	0.42	72.73%	0.56	0.62	68.3%	0.83	0.71	46.6%	0.91	0.91	29.41%

TABLE III
COMPARISON OF OPENASSERT WITH GPT-4

Evaluation Module	OpenAssert			GPT-4		
	Rouge	Cosine	Word ER	Rouge	Cosine	Word ER
UART	**0.87**	0.84	**36.36%**	0.68	**0.89**	38.46%
I2C FSM	**0.70**	**0.84**	**44.05%**	0.66	0.83	58.33%
LC CTRL	**0.91**	**0.88**	**31.58%**	0.86	0.79	45.5%
OTBN	0.84	0.81	33.33%	**0.86**	**0.82**	33.33%
KMAC	**0.81**	0.78	**44%**	0.69	**0.82**	67.74%
SYSRST CTRL	**0.91**	**0.91**	29.41%	0.78	0.88	**27.30%**

As a result, we contend that the exclusive use of RAG will not suffice for the model to generate accurate SVA.

Fine-tuned 34B LLM: Columns 2-4 and 8-10 in Table II reveal the performance boost achieved with the model fine-tuned on our custom dataset over the base model. Specifically, the fine-tuned model consistently outperforms the base model across several metrics. For example, rouge-1 scores range from 0.70 to 0.83 for the fine-tuned model, compared to 0.47 to 0.73 for the base model, resulting in an improvement of up to 36%. This indicates a more substantial overlap in the correct assertions produced by the fine-tuned model. In addition, the cosine similarity scores show a notable enhancement, with values from 0.81 to 0.79 for the fine-tuned model, significantly higher than the 0.42 to 0.74 range seen in the base model. This signifies an improvement of up to 31%, reflecting superior semantic alignment. The WER values further support these findings, as the fine-tuned model achieves WER values ranging from 45% to 61%, noticeably lower than the 68% to 86% range of the base model. This showcases a reduction of up to 27.3% in WER, highlighting the improved performance of the fine-tuned model.

Overall, the fine-tuned model demonstrates enhanced performance over the RAG-only model, particularly in its handling of SVA. This is evidenced by higher rouge and cosine similarity scores across various IP cores, such as in the *SYSRST CTRL* test where the fine-tuned model achieves a rouge-1 score of 0.83 and a cosine similarity score of 0.71, compared to the RAG-only model's scores of 0.56 and 0.62, respectively. These results indicate a superior grasp of the syntactic and semantic aspects of SVA by the fine-tuned model. However, the model exhibits limitations in its handling of variable names, as it lacks the context provided by RAG.

OpenAssert: OpenAssert combines RAG's broad knowledge retrieval with fine-tuning's precise context-aware enhancements, leveraging both for improved language performance. Comparing columns 11-13 with columns 2-10, we observe that OpenAssert shows notable improvements across most benchmarks when measured against the base CodeLlama model, the RAG-assisted LLM, and the solely fine-tuned LLM. For instance, in the *UART* benchmark, OpenAssert achieves a higher rouge-1 score (0.91 vs. 0.47 base, 0.56 RAG, 0.83 fine-tune) and a competitive cosine similarity (0.91 vs. 0.42 base, 0.62 RAG, 0.71 fine-tune), along with a significantly lower WER of 29.41% compared to 72.73% in the base model, 68.3% in RAG, and 46.6% in the fine-tune model. This demonstrates its superior understanding, contextual alignment, and accuracy in language processing. Compared to baseline CodeLlama-34B LLM, OpenAssert shows an improvement of up to 44% in rouge-1 score, 49% in cosine similarity, and 43.3% reduction in WER. Overall, OpenAssert achieves the highest rouge-1 and cosine similarity scores while significantly lowering the WER compared to all other models.

Comparison with GPT-4o: Table III provides a detailed comparison between OpenAssert and GPT-4o, Open AI's latest state-of-the-art model. OpenAssert demonstrates superior performance in rouge-1, achieving an average score of 0.84 across the benchmark modules than those achieved by GPT-4o, which scores an average of 0.755 as seen in columns 2 and 5, indicating a relative increase of 11.3%. The cosine similarity scores reflect competitive performance, with OpenAssert outperforming GPT-4o in the SYSRST CTRL, I2C FSM, and LC CTRL modules overall exhibiting a relative improvement of 0.5%. Furthermore, columns 4 and 7 show that OpenAssert exhibits lower WER in the UART, LC CTRL, KMAC, and I2C FSM modules of 36.36%, 31.58%, 44%, and 44.05%, respectively, compared to 38.46%, 45.5%, 67.74%, and 58.33% by, respectively, GPT-4o, demonstrating a relative improvement of 23.7%.

IV. CONCLUSION

In this paper, we introduce OpenAssert, a framework that leverages open-source LLMs to securely generate hardware assertions for SoC security verification. Unlike methods relying on proprietary models such as GPT-4, which risk exposing sensitive design data, OpenAssert fortifies the Open-Source CodeLlama model with a specialized database, parameter-efficient fine-tuning, and context injection to reduce hallucinations and protect confidentiality. Our comparative analysis demonstrates that OpenAssert outperforms the baseline by up to 49% on OpenTitan. Moreover, OpenAssert surpasses GPT-4 by up to 23.7%. In the future, we plan to extend OpenAssert to additional large-scale commercial SoCs.

V. ACKNOWLEDGMENTS

This research is supported by NSF grant #222304 and Intel Corporation.

REFERENCES

[1] F. Farahmandi *et al.*, *System-on-chip security*. Springer, 2020.

[2] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms," *arXiv preprint arXiv:2402.00386*, 2024.

[3] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "(security) assertions by large language models," *IEEE Transactions on Information Forensics and Security*, 2024.

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

[5] B. Roziere *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[6] S. Ganesh, "Systemverilog assertions basics - systemverilog.io." [Online]. Available: https://www.systemverilog.io/verification/sva-basics/

[7] B. Cohen, S. Venkataramanan, and A. Kumari, *SystemVerilog Assertions Handbook:–for Formal and Dynamic Verification*. vhdlcohen publishing, 2005.

[8] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[9] 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," 2021.

[10] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 10 088–10 115. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf

[11] D. Gunawan *et al.*, "The implementation of cosine similarity to calculate text relevance between two documents," in *Journal of physics: conference series*, vol. 978. IOP Publishing, 2018, p. 012120.

[12] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *2016 4th International Conference on Cyber and IT Service Management*, 2016, pp. 1–6.

[13] J. G. Tong, M. Boulé, and Z. Zilic, "Defining and providing coverage for assertion-based dynamic verification," *Journal of Electronic Testing*, vol. 26, pp. 211–225, 2010.

[14] Y. Zhang and A. Mesbah, "Assertions are strongly correlated with test suite effectiveness," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 214–224.

[15] M. Hassan, S. Ahmadi-Pour, K. Qayyum, C. K. Jha, and R. Drechsler, "Llm-guided formal verification coupled with mutation testing," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–2.