

Towards Trustworthy LLM-Based Assertion Generation: A Data Augmentation Framework with Formal Check Approach

Qingchen Zhai[†], Hao Yu^{‡*}, Chen Bai[‡], Charles Young[§], Frank Qu[¶], Dezhi Ran[§], Yuan Xie[‡], Tao Xie^{§*}

[†]Institute of Automation, Chinese Academy of Sciences, Beijing, China

[§]School of Computer Science, Peking University, Beijing, China

[‡]Hong Kong University of Science and Technology, Hong Kong, China

[¶]Independent Research

Abstract—Formal verification is a major bottleneck in integrated circuit (IC) design due to the inefficiency and inaccuracy of manual assertion writing and the limitations of existing automation approaches. While large language models (LLMs) offer a promising alternative for assertion generation, their effectiveness has been constrained by the scarcity of high-quality, formally verified training data. To address these challenges, we propose AutoAssert, an framework of automated assertion generation leveraging formal equivalence checking into the assertion generation pipeline, and introduce TrustAssert, a public dataset containing 110K formally verified assertions. By fine-tuning LLMs on TrustAssert, we achieve substantial improvements across four representative hardware modules. Our approach significantly outperforms GPT-4 in terms of the ratio of non-trivial assertions generated, syntactic correctness, and functional verification accuracy.

I. INTRODUCTION

Rising integrated circuit (IC) complexity has made functional verification a critical bottleneck. Assertion-based verification addresses this bottleneck by enabling formal specification of design properties. These assertions act as embedded monitors, checking signals, timing, and protocols during simulation and formal verification [1]–[3]. However, manual assertion writing remains a severe challenge [4]. While SystemVerilog assertions provide a rich language, creating accurate assertions requires deep design expertise and is time-consuming, error-prone, and limited by expert availability [5]. Automating assertion generation is therefore a crucial open problem in EDA.

Previous research into automation can be broadly categorized into two strands. The first employs static analysis techniques, which generate assertions through analysis of RTL code or specifications without requiring simulation [6], [7]. However, these approaches often fail to capture complex temporal relationships and tend to produce a significant number of redundant assertions, adversely impacting verification efficiency. The second strand utilizes dynamic analysis, mining assertions from simulation waveforms [8]. Despite being capable of uncovering sophisticated temporal behaviors,

these approaches are inherently limited by the coverage of the simulation stimuli. Scenarios not exercised by the tests remain unverified. To overcome the limitations of both static and dynamic approaches, researchers have recently attempted to abstract the problem as a sequence-to-sequence (S2S) translation task, employing neural networks to capture hidden patterns in specifications or code [9]–[11]. Nevertheless, constrained by data availability and the structural heterogeneity of specifications and RTL, traditional language-processing-based techniques exhibit limited generalization capability, hindering their practical adoption.

The advent of large language models (LLMs) presents a transformative opportunity, offering an unprecedented ability to parse and interpret natural language. Their success has spurred exploration into various hardware design tasks, including RTL generation, testbench creation, and synthetic data generation [12]–[14]. However, in the domain of assertion generation, current research remains predominantly focused on prompt engineering strategies [15], [16]. The application of LLMs is severely constrained by the scarcity of high-quality, large-scale assertion datasets for training. While small, hand-crafted datasets exist, the open-source community lacks a substantial corpus of verified assertions with description [17]. This data scarcity prevents the effective fine-tuning of general-purpose LLMs and stifles progress in the field.

To break this critical bottleneck, we propose a novel framework of data generation grounded in formal verification principles. Our work goes beyond yet another assertion generation tool. It introduces a rigorous methodology for constructing a scalable dataset of provably correct assertions, specifically designed to enable and accelerate AI research in this domain. By leveraging equivalence checking between a design and its formally transformed variants, we automatically produce a large and diverse set of correct assertions. This paper makes the following main contributions:

- We present a comprehensive analysis of assertion patterns and distributions from a large corpus of open-source RTL projects (2020 ~ 2025), highlighting the current landscape and the need for curated data.
- We propose **AutoAssert**, a formal-verification-based

*Corresponding authors.

framework that automatically generates high-quality assertion data. We release its output as the **TrustAssert** dataset, a corpus of 110K formally verified assertions. The dataset is publicly available at <https://github.com/sjowoj/coderust/tree/master/paper/AutoAssert>.

- We fine-tune Llama2-7B [18] on TrustAssert and achieve performance that surpasses GPT-4 [19] in key scenarios. This result demonstrates the effectiveness of targeted fine-tuning with formally verified data.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes the dataset collection and analysis. Section IV presents the AutoAssert framework. Finally, Section VI concludes this paper.

II. RELATED WORK

In this section, we introduce related work on assertion generation approaches (Section II-A) and relevant datasets (Section II-B).

A. Assertion Generation Approaches

Assertion-based verification has emerged as the most widely adopted solution for pre-silicon design functional validation [20]. While static approaches such as HARM [21] generate assertions at the pre-RTL stage using user-defined hints without requiring simulation, they often struggle to capture complex temporal dependencies. In contrast, dynamic analysis approaches extract assertions from simulation traces, enabling the automatic mining of intricate temporal properties and producing high-quality assertions for long sequential behaviors [8]. However, dynamic analysis approaches are inherently limited by simulation coverage and cannot generate assertions for scenarios not present in the waveform traces. With the recent advances in LLMs, researchers have begun exploring their potential to infer critical functional properties directly from natural language specifications. AssertLLM [15] employs prompt engineering and multiple customized LLMs to process multimodal specification inputs for assertion generation. Building upon AssertLLM, SPEC2Assert [16] further enhances the generation pipeline with more sophisticated LLM-based workflow engineering, achieving higher SystemVerilog assertion (SVA) accuracy on the same benchmark compared to AssertLLM.

B. Datasets for Hardware

Multiple datasets have been developed to support LLM-driven hardware design and verification tasks. The MG-Verilog dataset [22] provides over 11K Verilog code samples paired with natural language descriptions at varying levels of detail, facilitating tasks such as code generation and summarization. The SA-DS dataset [23] offers a collection of spatial array designs conforming to the Gemmini accelerator template [24], supporting research in neural network accelerator architectures. For RTL design evaluation, the RTL-Repo benchmark [25] includes over 4K Verilog samples from public GitHub repositories, each annotated with full repository context to enhance both training and inference. The OpenLLM-RTL framework [26] provides 50 hand-crafted

designs, each accompanied by natural language descriptions, test cases, and correct RTL implementations. It also includes an extended dataset of 80K instruction–code pairs, of which 7K are high-quality verified samples, to support model training and evaluation. VerilogEval [27], derived from HDLBits [28], contains 156 problems covering topics from combinational logic to finite-state machines, and supports automated functional testing by comparing generated designs against golden references. Additionally, VeriGen [29] aggregates Verilog code from GitHub and Verilog textbooks, and has been used to fine-tune general-purpose LLMs for hardware code generation. VERT [17] is a dataset with 18K assertion samples for LLM fine-tuning or pretraining. While a number of existing efforts have focused on hardware-related datasets, there remains a noticeable lack of large-scale, well-annotated assertion datasets.

III. AUTOASSERT AUGMENTATION FRAMEWORK

This section presents the AutoAssert framework for automated assertion generation, as shown in Fig. 1. Our approach begins with collecting all open-source hardware designs from GitHub, followed by rigorous quality filtering through compilation verification (Section III-B). To extract initial assertion snippets, we employ regular-expression-based matching for SVA extraction (Section III-C). Addressing the scarcity of available assertion data, we introduce a novel methodology that first identifies sequential logic modules from RTL code through pattern matching (Section III-C), and then transforms these modules into novel assertion constructs using our innovative RTL-to-assertion conversion framework (Section III-D). With the augmented assertion dataset, we generate natural language descriptions through customized LLM processing (Section III-E). We subsequently produce mutated assertion variants to enhance diversity (Section III-F). Crucially, we implement a formal verification framework to ensure semantic consistency between generated descriptions and mutated assertions, thereby guaranteeing dataset quality (Section III-G).

A. Mining Open-Source Hardware Repositories

Our initial dataset comprises 2.2 terabytes of GitHub repositories containing Verilog and SystemVerilog code, collected from January 2020 to February 2025. We employ the Icarus Verilog 10.3 compiler [30] for syntactic validation and functional verification through testbench simulation. Repository selection is rigorously conditioned on successful compilation and testbench execution to ensure code quality for subsequent analysis stages. As detailed in Fig. 2, the curated dataset contains 2,521,029 Verilog or SystemVerilog files. Among these files, only 6,415 files (0.25%) incorporate syntactically correct assertions, demonstrating exceptionally sparse adoption across the codebase. The yearly distribution reveals that files containing assertions remain consistently scarce throughout the six-year period, with a maximum of only 517 files containing any assertions in the peak year of 2024. Statistical analysis confirms critically low assertion prevalence, with over 99.7% of files completely lacking assertion constructs. This extreme scarcity is evident in the fact that fewer than 1 in 400 files contain any verification statements. The resulting file-

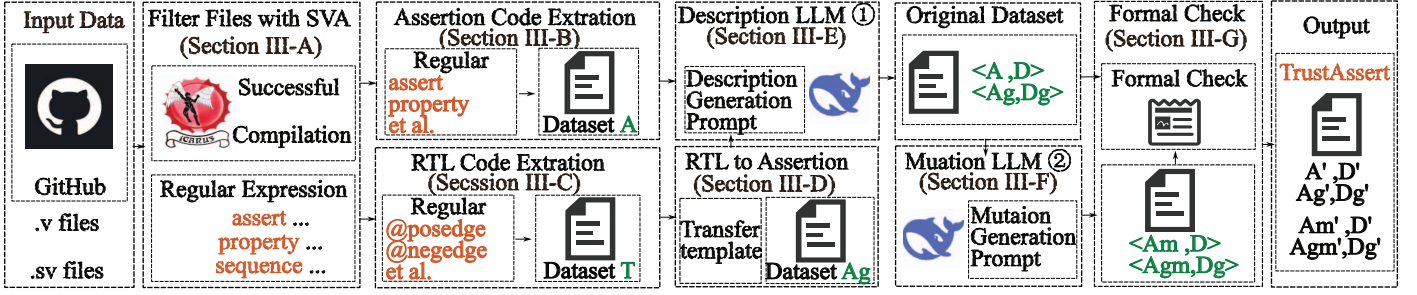


Fig. 1. The overview of AutoAssert framework.

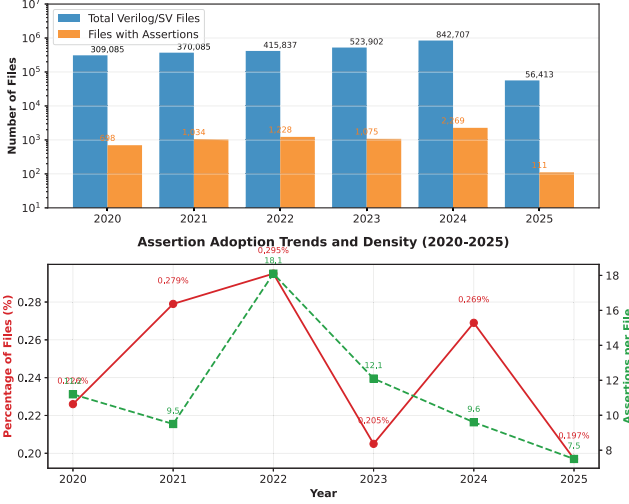


Fig. 2. Assertion trend and density Analysis (2020-2025).

level distribution falls orders of magnitude short of what is needed for effective LLM fine-tuning. Consequently, relying on naturally occurring files that contain assertions from open-source repositories is fundamentally inadequate for training data-intensive LLM models of assertion generation. This factor necessitates sophisticated data augmentation frameworks to overcome the critical scarcity gap.

B. Assertion Extraction

To systematically extract assertion constructs from the compiled dataset obtained in Section III-A, we employ a regular-expression-based methodology targeting SVA syntax patterns. Our approach specifically identifies three fundamental assertion types. The first type is immediate assertions, which correspond to `assert` statements. The second is concurrent assertions, implemented using `assert property` constructs. The third is temporal assertions, which are based on expressions of timing relationships. The extraction pipeline utilizes pattern matching rules that capture both assertion keywords and their corresponding syntactic blocks, including clocking expressions and temporal operators. Through this process, we successfully extract 74,682 unique assertion instances from 6,415 Verilog or SystemVerilog files. All extracted assertion snippets are aggregated into the dataset $\langle A \rangle$, which serves as the foundation for subsequent augmentation stages.

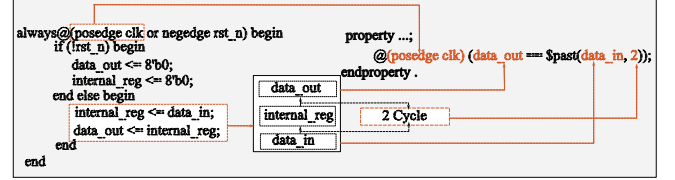


Fig. 3. Verilog code snippet is converted to the temporal assertion.

C. Temporal RTL Extraction

To address the scarcity of assertions obtained in Section III-B and further exploit temporal assertion patterns, we develop a targeted extraction methodology applied to the filtered dataset from Section III-A. Our approach employs regular expression matching to identify clock-triggered procedural blocks, specifically targeting `always@(posedge)` constructs. This process yields the dataset $\langle T \rangle$, comprising 53,317 sequential Verilog code snippets including conditional statements (e.g., `case`, `if-else`), reset-controlled flip-flops, and other temporal code logic structures, providing a foundation for temporal assertion generation.

D. Assertion Generation From RTL Code

Building upon the sequential code snippets in $\langle T \rangle$ obtained from Section III-C, we next focus on generating assertions to augment both the volume and diversity of training data. A fundamental challenge in assertion generation lies in producing complex, multi-cycle temporal assertions rather than simple single-cycle checks. Generating excessive simplistic assertions would misdirect LLM training, resulting in redundant and unusable outputs that increase verification overhead. To address this issue, we propose a novel approach for identifying patterns of multi-stage pipeline data propagation.

As illustrated in Fig. 3, we leverage the structured syntax of Verilog to extract key signals from `always@(posedge/negedge)` blocks. The clocks involve clock and reset signals as well as assignment logic. In contrast to approaches such as VERT [17], which are limited to capturing single-level temporal relations (e.g., from `case` or `if-else` statements), our approach detects pipeline structures spanning multiple cycles. Signals and their assignment dependencies are organized into a unidirectional linked list. A list length exceeding two indicates a multi-cycle pipeline path. The depth of the pipeline is then translated into explicit cycle delays, enabling automatic generation of complex temporal

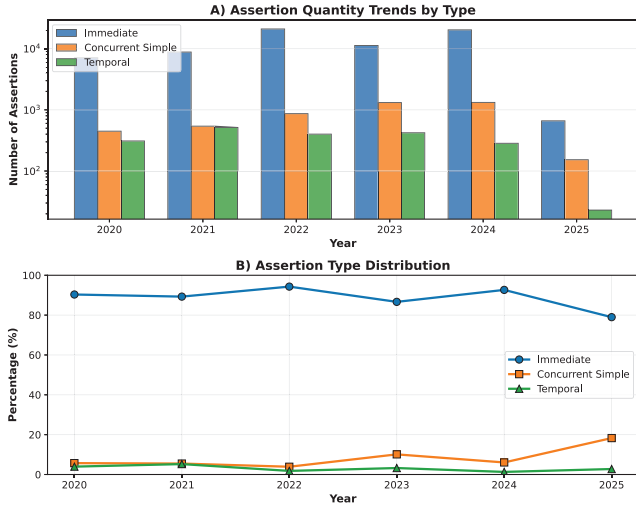


Fig. 4. Overview of the assertion type trends (2020-2025). (a) Counts of Immediate, Concurrent Simple, and Temporal assertions. (b) Proportional trends showing relative usage evolution.

assertions that accurately capture multi-cycle signal behavior.

E. Natural Language Description Generation

To address the critical absence of natural language descriptions for both naturally occurring and generated assertions, we develop a specialized LLM-based description generation framework. Our approach employs carefully crafted system instructions to create a dedicated assertion description generator that produces high-quality textual annotations suitable for supervised fine-tuning and reliable model training. The generator takes two primary inputs: the complete contextual code and the specific assertion code segment itself. This contextual understanding enables the model to generate semantically accurate descriptions that capture both the functional intent and verification purpose of each assertion.

We apply this approach to both dataset $\langle A \rangle$ from Section III-B (including naturally occurring assertions) and the dataset $\langle A_g \rangle$ from Section III-D (containing generated assertions). The resulting enriched datasets, denoted as $\langle A, D \rangle$ and $\langle A_g, D_g \rangle$, now include comprehensive natural language descriptions that provide semantic context and functional explanations for each assertion statement.

F. Mutation Assertion Generation

To maximize the utility of the collected assertion dataset and enhance its diversity, we introduce a specialized mutation generation approach that employs a systematically instructed LLM. This approach is designed to produce functionally equivalent assertion variants while altering their syntactic representations, thereby expanding the assertion pattern coverage without compromising semantic integrity. The mutation generator takes as input the natural language descriptions from Section III-E coupled with the complete RTL code, generating novel assertion formulations that preserve identical functional behavior but employ divergent syntactic structures. This process ensures that each mutated assertion maintains logical equivalence with its source assertion while exhibiting

variations in expression style, operator usage, and structural organization. We apply this mutation approach to both enriched datasets: $\langle A, D \rangle$ and $\langle A_g, D_g \rangle$ (Section III-E). The mutation process yields two augmented datasets correspondingly being $\langle A_m, D \rangle$ and $\langle A_{gm}, D_g \rangle$. Crucially, the descriptions remain unchanged throughout the mutation process, ensuring consistent semantic grounding while enabling syntactic diversification.

G. Maintain Assertion-Description-Assertion Consistency

To ensure the reliability of LLM-generated descriptions and mutated assertions obtained in Section III-E and Section III-F, we propose a novel formal-verification-based approach that establishes consistency between original assertions, their natural language descriptions, and mutated variants. The core approach employs formal equivalence checking to validate semantic preservation across the assertion-description-assertion transformation chain.

Our approach performs systematic equivalence check between pairs of $\langle A, D \rangle$ and $\langle A_m, D \rangle$, along with pairs of $\langle A_g, D_g \rangle$ and $\langle A_{gm}, D_g \rangle$. This verification ensures that the natural language description D accurately captures the semantic meaning of assertion A , and that the mutated assertion A_m maintains functional equivalence with the original assertion. The formal analysis reveals two critical failure scenarios when equivalence checking fails. First, description D is correct but the mutation generation produces an erroneous assertion A_m . Second, both description D and mutated assertion A_m are incorrect. To ensure dataset quality, our methodology adopts a conservative approach. Namely, if all mutated assertions generated from a description fail formal equivalence check, such data instance is discarded to prevent contamination of the training dataset with unreliable samples. To ensure dataset reliability, we implement a strict filtering protocol. If all mutation assertions generated from a single description fail consistency check, the entire data instance is discarded. This conservative approach eliminates potentially erroneous samples that could compromise training quality. The final dataset comprises four rigorously verified components. They are verified natural assertions A' , verified generated assertions A'_g , verified mutation assertions A'_m , and verified generated mutation assertions A'_{gm} , respectively. This curation process guarantees that only formally consistent assertions with corresponding valid descriptions are included in the final dataset.

IV. EXPERIMENTS

We present our experiments in this section.

A. Assertion Empirical Analysis

To establish a rigorous empirical foundation for assertion generation research, we present a large-scale study of assertion usage across 124,537 public GitHub repositories with Verilog/SystemVerilog files (2020-2025). Our curated dataset reveals that only 14.65% of 2.84 million source files contain assertions, highlighting a critical need for automated assertion generation in contemporary design practices. As shown in Fig. 4, the data reveals a critical imbalance in assertion

TABLE I

COMPARISON OF ASSERTION GENERATION PERFORMANCE ACROSS BENCHMARKS WITH BASELINE AND FINE-TUNED MODELS. *Non-trivial/Total* COUNTS ASSERTIONS THAT CAPTURE NON-TRIVIAL DESIGN PROPERTIES; *Syntax Correct* VERIFIES SYNTACTIC VALIDITY; *FPV Correct* MEASURES ASSERTIONS PROVEN VALID BY FORMAL PROPERTY VERIFICATION.

Module	Metric	Goldmine [6]	SPEC2Assert [16]	AssertLLM [15]	Llama2-7B [18]	Llama2-7B-T	Qwen-3B [31]	Qwen-3B-T	GPT4 [19]
UART [32]	Non-trivial/Total	0/0	54/54	48/59	0/12	25/47	0/10	6/23	13/43
	Syntax Correct	0	47	42	3	16	0	7	8
	FPV Correct	0	17	9	0	6	0	1	5
HATX [33]	Non-trivial/Total	41/83	153/153	73/90	12/25	35/67	5/18	15/57	23/76
	Syntax Correct	41	152	17	5	30	3	10	21
	FPV Correct	2	15	2	0	12	0	2	13
I2C [15]	Non-trivial/Total	18/18	111/111	104/130	15/28	46/62	3/22	37/62	32/71
	Syntax Correct	18	99	88	12	42	3	23	23
	FPV Correct	11	33	26	3	18	0	12	6
SOCKIT [32]	Non-trivial/Total	37/906	161/161	95/109	2/25	47/62	0/32	32/52	43/83
	Syntax Correct	37	148	71	18	33	12	25	32
	FPV Correct	7	53	27	4	23	3	12	29

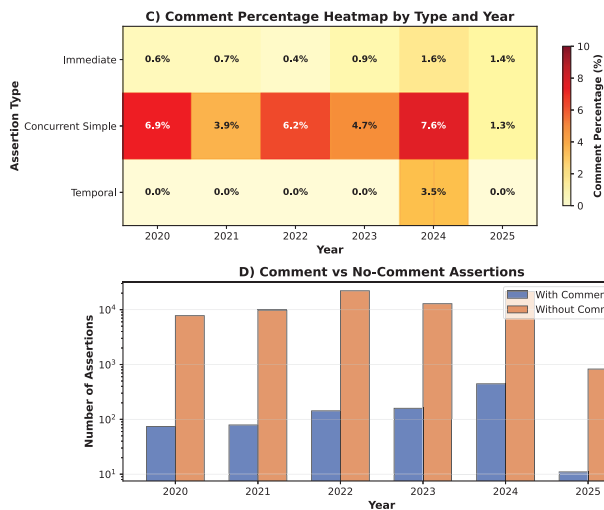


Fig. 5. Assertion comment analysis results (2020-2025). (c) Heatmap showing annual distribution of commented assertions across three categories: Immediate assertion, Concurrent Simple assertion (excluding temporal assertion), and Temporal assertions. (d) Annual counts of assertions with and without comments (vertical axis in logarithmic scale, base 10).

distribution. Immediate assertions dominate (85+%) while temporal assertions comprise less than 3%. The abundance of simple immediate assertions provides limited verification value, whereas the paucity of temporal assertions directly constrains formal verification effectiveness. This imbalance underscores the urgent need for targeted temporal assertion generation to advance practical formal verification adoption. As shown in Fig. 5, concurrent assertions show reasonable commenting (5.2%) while temporal assertions suffer from near-zero documentation despite their critical importance. With over 99% of assertions lacking descriptions, the reliability crisis is particularly severe for complex temporal properties where semantic precision is paramount. This dual deficiency demands simultaneous solutions: both increasing temporal assertion quantity and ensuring their semantic documentation to enable effective verification reuse and validation.

B. TrustAssert Dataset

The TrustAssert dataset represents a significant advancement in assertion verification data, constructed through our rigorous

TABLE II
DATASET COMPOSITION W. TRUSTASSERT.

Source	Immediate	Concurrent_simple	Temporal	Total
A'	20,000	4,216	1,827	26,043
A'_g	0	7,327	18,732	26,059
A'_m	22,530	4,216	1,827	28,573
A'_{gm}	0	10,483	20,863	31,346
Total	42,530	26,242	43,249	112,021

data augmentation framework. While the original open-source data contains 69,082 immediate assertions, we strategically select only 20,000 for the dataset A to maintain type balance and avoid over-representation of simpler assertion types. A comprises 26,043 assertions in total, including 4,216 concurrent simple and 1,827 temporal assertions. To address the critical scarcity of complex assertions, we generate A_g containing 26,059 assertions from RTL code segments, consisting of 7,327 concurrent simple and 18,732 temporal assertions. Our mutation approach further expands the dataset through formal-equivalence-preserving transformations, producing A_m (28,573 assertions) and A_{gm} (31,346 assertions). The final TrustAssert dataset integrates all verified components, totaling 112,021 high-quality assertions as shown in TABLE II. This dataset represents a carefully balanced composition with 42,530 immediate (38.0%), 26,242 concurrent simple (23.4%), and 43,249 temporal assertions (38.6%). Our framework achieves a 62% increase in overall assertion quantity while significantly improving type diversity, with temporal assertions experiencing a 2,267% increase compared to the original GitHub extraction.

C. Baseline

Our experimental evaluation comprises two comprehensive components designed to rigorously assess the effectiveness of the TrustAssert dataset. We select Qwen-Coder-3B [31] and Llama-2-7B [18] for fine-tuning based on their open-source availability, moderate parameter scale, and demonstrated capabilities in code-related tasks, making them representative models for evaluating assertion generation improvement through our dataset. These fine-tuned models are evaluated across four diverse hardware verification modules to quantify per-

formance gains against established baselines including Gold-Mine [6], a seminal simulation-based assertion mining tool; AssertLLM [15], a recent LLM-based approach generating assertions from specifications; SPEC2Assert [16], a contemporary SOTA approach; and GPT-4 [19] as a reference for high-capacity general-purpose model performance.

D. Model SFT Settings

We employ full parameter fine-tuning to comprehensively adapt the base model to assertion generation tasks. The training configuration utilizes automatic half-precision computation to optimize memory efficiency while maintaining numerical stability. Our experimental setup processes 112,021 training examples over three complete epochs, ensuring thorough exposure to the diverse assertion patterns in the TrustAssert dataset. The training procedure employs a batch size of two per device with gradient accumulation over eight steps, resulting in an effective batch size of 64. We fine-tune our models on 16 Nvidia Tesla V100S GPUs.

E. Benchmark

Our benchmark, consistent with SPEC2Assert [16], comprises four representative hardware modules: SOCKIT [32], an open-source socket controller implemented on an FPGA-based SoC; UART [32], a UART-to-bus interface from OpenCores; I2C [15], an I2C master/slave logic controller from AssertLLM; and HATX [33], a custom high-throughput arbitration and crossbar module. These designs span 3-9 logic levels and contain 400-1200 lines of functionally correct RTL code, providing diverse complexity for comprehensive evaluation of assertion generation techniques.

F. Methods and Metrics

Our assertion generation methodology follows a similar approach to AssertLLM [15] and SPEC2Assert [16], where natural language specifications are processed by LLMs to produce corresponding assertions. To ensure fair comparison across all models, we utilize GPT-4o [34] to convert all specification materials into pure textual descriptions, eliminating potential biases from multimodal capabilities that our fine-tuned models lack. Evaluation is conducted using three key metrics: the ratio of non-trivial assertions to total assertions generated, measuring semantic relevance; syntax correctness rate, assessing adherence to assertion language grammatical rules; and functional correctness verified through formal property verification (FPV).

G. Results and Discussion

1) Cross-Module Comparison with Baselines

As demonstrated in TABLE I, we conduct a comprehensive evaluation of assertion generation performance across four hardware modules. In the HATX module, Llama2-7b-T improves the non-trivial assertion ratio by 191% over its base model, while narrowing the performance gap with GPT-4 to within 15%. Notably, in functional correctness, Llama2-7b-T achieves a 400% improvement over its base version in the I2C module and reduces the performance gap with GPT-4 to just 33%. Similarly, Qwen-3B-T shows exceptional

TABLE III
ABLATION STUDY ON E203 MODULE USING QWEN-CODER-3B.

Metric	Unverified Dataset	RVAssert (Ours)
Non-trivial/Total	13/72	37/62
Syntax Correct	10	23
FPV Correct	6	12

progress, improving its non-trivial assertion ratio by 280% in the SOCKIT module and surpassing GPT-4 in functional correctness for the I2C module by 100%. Although GPT-4 maintains an advantage in certain contexts, particularly in syntax correctness for the UART module, our fine-tuned models deliver comparable functional correctness while utilizing only a fraction of the computational resources.

2) Ablation Study

We conduct a controlled ablation study comparing the TrustAssert dataset (with formal verification) against a 50K-sample unverified dataset. Both datasets are used to fine-tune the same base model, Qwen-Coder-3B, with comprehensive evaluation performed on the I2C module. As quantitatively demonstrated in TABLE III, the formally verified TrustAssert dataset delivers substantially superior performance across all evaluation metrics: the non-trivial assertion ratio improves by 230%, syntax correctness increases by 166%, and most significantly, functional correctness under FPV rises by 100%. These results unequivocally demonstrate that formal verification is essential for cultivating high-quality training data.

V. CONCLUSION

In this paper, we have introduced AutoAssert, a novel framework of automated assertion generation using formal equivalence checking, and released TrustAssert, a publicly available dataset of 110K formally verified assertions. Experimental results have shown that after fine-tuning LLMs on this high-quality data, multiple hardware modules have shown significant improvements in non-trivial assertion ratio, syntactic validity, and functional correctness, even surpassing GPT-4 in key metrics with minimal computational overhead.

VI. ACKNOWLEDGMENTS

Tao Xie is with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China; Fudan University Institute of Systems for Advanced Computing, Shanghai, China; Beijing Tongming Lake Information Technology Application Innovation Center, Beijing, China; Shanghai Institute of Systems for Open Computing, Shanghai, China. This work was partially supported by National Natural Science Foundation of China under Grant No. 623B2006, 92464301 and U25A6023. This research was also partially conducted by ACCESS AI Chip Center for Emerging Smart Systems, supported by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government. Additional support was provided by Research Grants Council of Hong Kong SAR (16213824 & 16212825).

REFERENCES

- [1] L.-C. Wang, M. Abadir, and N. Krishnamurthy, "Automatic generation of assertions for formal verification of PowerPC/Sup TM/Microprocessor Arrays using symbolic trajectory evaluation," in *ACM/IEEE Design Automation Conference*, 1998, pp. 534–537.
- [2] S. Hertz, D. Pal, S. Offenberger, and S. Vasudevan, "A figure of merit for assertions in verification," in *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2019, pp. 675–680.
- [3] Z. Zhang, W. Weng, Y. Li, L. Cai, H. Wang, D. Boland, Y. Bao, and K. Shi, "Hassert: Hardware assertion-based verification framework with FPGA acceleration," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2024, pp. 142–154.
- [4] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A survey on assertion-based hardware verification," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–33, 2022.
- [5] A. B. Mehta, *System verilog assertions and functional coverage: Guide to language, methodology and applications*, 3rd ed. Springer, 2020. [Online]. Available: <https://link.springer.com/10.1007/978-3-030-24737-9>
- [6] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "GoldMine: Automatic assertion generation using data mining and static analysis," in *IEEE/ACM Proceedings Design, Automation and Test in Europe*, 2010, pp. 626–629.
- [7] A. Danese, N. D. Riva, and G. Pravadelli, "A-TEAM: Automatic template-based assertion miner," in *2017 54th ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 1–6.
- [8] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, and J. Raik, "Engineering of an effective automatic dynamic assertion mining platform," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration*, 2019, pp. 111–116.
- [9] C. Hahn, "Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions," 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:259289057>
- [10] C. B. Harris and I. G. Harris, "GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions," in *2016 Conference on Design, Automation & Test in Europe*, San Jose, CA, USA, 2016, p. 966–971.
- [11] R. Krishnamurthy and M. S. Hsiao, "Controlled natural language framework for generating assertions from hardware specifications," in *2019 IEEE 13th International Conference on Semantic Computing*, 2019, pp. 367–370.
- [12] X. Wang, G.-W. Wan, S.-Z. Wong, L. Zhang, T. Liu, Q. Tian, and J. Ye, "ChatCPU: An agile CPU design & verification platform with LLM," in *2024 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.
- [13] Z. He, Y. Pu, H. Wu, T. Qiu, and B. Yu, "Large language models for EDA: Future or mirage?" *ACM Transactions on Design Automation of Electronic Systems*, vol. 30, no. 6, pp. 1–53, 2025.
- [14] Y. Lu, C. Bai, Y. Zhao, Z. Zheng, Y. Lyu, M. Liu, and B. Yu, "DeepVerifier: Learning to update test sequences for coverage-guided verification," *ACM Transactions on Design Automation of Electronic Systems*, Mar. 2025, just Accepted. [Online]. Available: <https://doi.org/10.1145/3721133>
- [15] Z. Yan, W. Fang, M. Li, M. Li, S. Liu, Z. Xie, and H. Zhang, "AssertLLM: Generating hardware verification assertions from design specifications via multi-LLMs," in *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2025, pp. 614–621.
- [16] F. Wu, E. Pan, R. Kande, M. Quinn, A. Tyagi, D. K. Houngrinou, J. Rajendran, and J. Hu, "Spec2Assertion: Automatic pre-RTL assertion generation using large language models with progressive regularization," 2025. [Online]. Available: <https://arxiv.org/abs/2505.07995>
- [17] A. Menon, S. S. Miftah, S. Kundu, S. Kundu, A. Srivastava, A. Raha, G. T. Sonnenschein, S. Banerjee, D. Mathaikutty, and K. Basu, "Enhancing large language models for hardware verification: A novel systemverilog assertion dataset," 2025. [Online]. Available: <https://arxiv.org/abs/2503.08923>
- [18] H. Touvron, L. Martin, K. Stone, and P. A. et al., "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: <https://arxiv.org/abs/2307.09288>
- [19] OpenAI, "GPT-4 technical report," 2024. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [20] A. Gupta, "Assertion-based verification turns the corner," *IEEE Design & Test*, vol. 19, no. 4, p. 131–132, Jul. 2002.
- [21] S. Germiniani and G. Pravadelli, "HARM: A hint-based assertion miner," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4277–4288, 2022.
- [22] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "MG-Verilog: multi-grained dataset towards enhanced LLM-assisted Verilog generation," in *IEEE LLM Aided Design Workshop*, 2024, pp. 1–5.
- [23] D. Vungarala, M. Nazzal, M. Morsali, C. Zhang, A. Ghosh, A. Khreishah, and S. Angizi, "SA-DS: A dataset for large language model-driven AI accelerator design generation," in *IEEE International Symposium on Circuits and Systems*, 2025, pp. 1–4.
- [24] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao et al., "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *ACM/IEEE Design Automation Conference*, 2021, pp. 769–774.
- [25] A. Allam and M. Shalan, "RTL-Repo: A benchmark for evaluating LLMs on large-scale RTL design projects," in *IEEE LLM Aided Design Workshop*, 2024, pp. 1–5.
- [26] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "OpenLLM-RTL: Open dataset and benchmark for LLM-aided design RTL generation," 2025. [Online]. Available: <https://arxiv.org/abs/2503.15112>
- [27] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating large language models for Verilog code generation," in *IEEE/ACM International Conference on Computer-Aided Design*, 2023, pp. 1–8.
- [28] H. Zhou. HDLbits. [Online]. Available: https://hdlbits.01xz.net/wiki/Main_Page
- [29] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated Verilog RTL code generation," in *IEEE/ACM Proceedings Design, Automation and Test in Europe*, 2023, pp. 1–6.
- [30] S. Williams and M. Baxter, "Icarus Verilog: Open-source Verilog more than a year later," *Linux Journal*, vol. 2002, no. 99, p. 3, 2002.
- [31] B. Hui, J. Yang, Z. Cui, J. Yang, and D. L. et al., "Qwen2.5-Coder technical report," 2024. [Online]. Available: <https://arxiv.org/abs/2409.12186>
- [32] OpenCores, *OpenCores: Open source hardware designs*, <https://opencores.org>.
- [33] HyperTransport Consortium, *HyperTransport Advanced X-bar (HTAX) specification*, 2008.
- [34] OpenAI, "Gpt-4o," Online, May 2024, multimodal large language model. [Online]. Available: <https://openai.com/index/hello-gpt-4o/>