# LLM-guided Formal Verification Coupled with Mutation Testing

Muhammad Hassan[1,2], Sallar Ahmadi-Pour[1], Khushboo Qayyum[2], Chandan Kumar Jha[1], Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{hassan, sallar, khushboo, chajha, drechsler}@uni-bremen.de

***Abstract*—The increasing complexity of modern hardware designs poses significant challenges for design verification, particularly defining and verifying properties and invariants manually. Recently, *Large Language Models* (LLMs) such has GPT-4 have been explored to generate these properties. However, assessing the quality of these LLM generated properties is still lacking. In this paper, we introduce a LLM-guided formal verification methodology combined with mutation testing for creating and assessing invariants for *Design Under Verification* (DUV). Utilizing OpenAI's GPT-4, we automate the generation of invariants and formal models from design specifications and Verilog behavioral models, respectively. We further enhance this approach with mutation testing to validate the quality of the invariants. We use a 27-channel interrupt controller (C432) from ISCAS-85 benchmarks as a complex case-study to showcase the methodology.**

## I. Introduction

The complexity of the modern hardware designs has increased significantly. Consequently, the task of formal verification has become increasingly daunting [1], [2]. As hardware complexity continues to grow, traditional verification methods struggle to keep pace with it, leading to increased development time and potential for errors. This necessitates a rigorous verification process, where properties and invariants must be accurately defined to ensure system correctness. However, manually writing these properties for complex designs is not only labor-intensive but also prone to errors, making the verification process cumbersome and less reliable. The challenge lies not only in the complexity but also in ensuring that these designs are robust against a wide range of potential design or implementation errors.

Recent advances in *Artificial Intelligence* (AI), particularly in the domain of *Large Language Models* (LLMs), offer new avenues for addressing these challenges [3]. LLMs, such as OpenAI's GPT-4, have shown remarkable capabilities in understanding and processing natural language, making them well-suited for interpreting and translating hardware specifications into *SystemVerilog Assertions* (SVA) [4]–[6], *System-on-Chip* (SOC) security properties [7]–[9], and stimuli generation [10]. However, the dynamic and often unpredictable nature of LLM-generated outputs (e.g., properties) poses a unique challenge in consistently evaluating their effectiveness across different hardware specifications.

In this regard, mutation testing emerges as a viable solution to this problem. By introducing controlled modifications to the hardware design, mutation testing allows for a comprehensive assessment of the quality and thoroughness of the verification process [11]. This method provides quantitative metrics for evaluating the effectiveness of LLMs in generating accurate and robust verification models. Consequently, providing higher confidence in using LLM-generated properties.

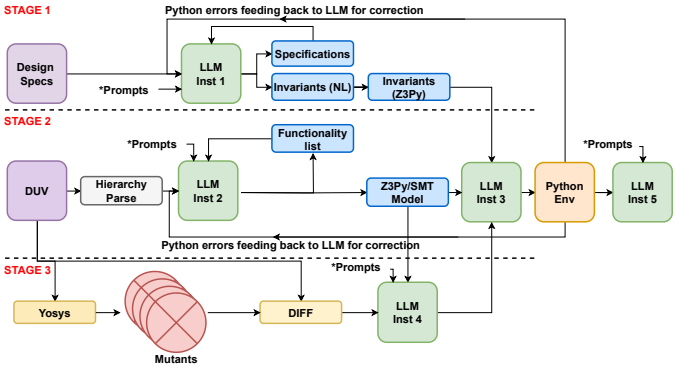


Fig. 1. Overview of LLM-guided Methodology

In this paper, we propose a LLM-guided formal verification methodology to generate invariants for *Design Under Verification* (DUV). An invariant is a condition or property that is consistently maintained as true throughout the execution of a system. In addition to invariants, we couple the LLM with mutation testing to perform qualitative analysis of the LLM-generated invariants. The methodology leverages OpenAI's GPT-4 as a key component, to automatically and systematically generate formal properties from design specifications and formal model from the Verilog behavioral model. We go one step further and integrate mutation testing to ensure that the generated properties are of high quality. The experiments were carried out on ISCAS-85 C432 27-channel interrupt controller [12], [13] as it shows the complexity in terms of hierarchy and functionality.

## II. LLM-guided Formal Verification

In this section, we provide an overview of the proposed LLM-guided verification methodology (see Fig. 1) coupled with mutation testing. The methodology is divided into three stages, 1) generation of invariants, 2) generation of the formal model, and 3) qualitative analysis of invariants. In stage 1, the design specification is given as input to the LLM-inst1, where three unique prompts are given as input, one at a time. First, the LLM is requested to create a list of specifications from the design specifications. The purpose of this prompt is twofold; 1) clarity in what LLM has extracted, and 2)

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project ECXL under contract no. 01IW22002, the project PaSVer under contract no. 16ME0855, and the project SASPIT under contract no. 16KIS1852K.

provide structure. This prompt provides guidance to the LLM. We have observed that LLMs work significantly better with structured data, than unstructured data. Afterwards, the LLM is requested to cross-check the specification it has listed against the provided specifications and add the missing specifications. This step provides an additional layer to ensure nothing is missed out. Finally, the LLM is requested to create a list of invariants in natural language and Z3Py representation for formal verification.

In stage 2, first the DUV is parsed to extract the hierarchical modules. With complex designs, the LLM responds with excuses and tries to guide the user with model creation instead of creating it. However, our goal is to prompt LLM to generate the formal model. Hence, we start creating the formal model in a bottom-up manner, i.e., starting from the lowest hierarchy in DUV to the top module. For each sub-module, the LLM-inst2 generates a functionality list and then a Z3Py model. At this point, LLM-inst3 is requested to combine the invariants from stage 1 into the model from stage 2. Please note, the invariants are written in two formats, 1) consistency check, 2) bug finding (negation of original invariant). The complete DUV Z3Py model with invariants is given to the Z3 SMT solver in a Python environment. If the Z3 solver returns *SAT*, everything is fine. In case it returns *UNSAT* with a counter example, it is given as input to the LLM-inst5, requesting it to create a testcase for Verilog testbench.

In stage 3, we perform a qualitative analysis of the invariants generated in stage 1. In particular, we use mutation testing to check if the invariants are of high quality or not. The DUV is given as input to Yosys which generates several mutants. Since, each mutant has only one mutation inserted, we take difference (*Diff*) between the golden DUV and the mutant to extract that particular changed line. Once extracted, we ask the LLM-inst4 to convert that particular line to Z3Py representation. Afterwards, it is merged in the original model and given to Z3 solver. All mutants are checked in this manner. Finally, based on the number of mutants detected, a mutation score is calculated to assess the quality of the generated invariants.

In the next section, we discuss a case-study to show the efficacy of our LLM-guided methodology.

## III. Case-study: 27-channel interrupt controller

In this section we present a case study to demonstrate the LLM-guided formal properties generation and assessment methodology. We consider the ISCAS-85 C432 27-channel interrupt controller [12], [13] which has been modeled as a Verilog behavioral model. It comprises of five sub-modules, *PriorityA, PriorityB, PriorityC, EncodeChan,* and *DecodeChan*. As shown in Fig. 1, we give the C432 circuit to Yosys and generate 20 mutants. Each mutant has a distinct mutation, i.e., no two mutants are the same. We generated 2 invariants from the given specifications to verify the correct priority order among buses as well as correct priority order within bits of a bus. The original DUV Z3Py model executed successfully with the invariants, i.e., the invariants were satisfied for both consistency and bug finding. However, the invariants failed when mutants were executed. Out of 20 mutants, all were detected successfully resulting in a mutation score of 1. As an additional step to verify our own results, we also performed equivalence checking of DUV Z3Py model and mutants. The Z3 solver reported *UNSAT* for all mutants. For all the cases where the invariants failed, the reported counter example was added as a test-case to the testbench.

## IV. Insights into LLM Behavior

While developing our methodology, we gained several insights (in addition to what is already mentioned in the paper):

- We observed that if the counter example is given as input to LLM to patch the complex circuit, it was unable to do so. On the contrary, the LLM was able to patch a simple circuit, e.g., Full-adder.
- Furthermore, we also observed that positive feedback to LLM improves the LLM output, i.e., acknowledging the result by saying *thank you* and requesting a task with *please*.
- Inside the prompt, if a monetary incentive, e.g., $100 tip is promised, the quality of results improve. The higher the incentive the better the results.
- The LLMs work very well with structured data. Hence, we always requested the LLM instances to create lists.
- The LLMs find it very difficult to extract meaning from the Verilog netlist and structural gate-level models. However, behavioral models work very well.

## V. Conclusion

In this paper, we proposed a LLM-guided formal verification methodology to generate invariants as well as evaluate their quality. At the heart of the methodology lies the LLM which performs a set of tasks as requested by the prompts. Our findings demonstrated that LLMs, particularly OpenAI's GPT-4, can effectively automate the generation of invariants from hardware design specifications. Coupling it with mutation testing as a method for qualitative analysis further strengthens our methodology.

Our research also uncovered some challenges, LLM's are not equally good at patching the complex code only using counter-example. Secondly, the context window size for LLMs is limited. Hence, techniques like slicing of the Verilog code needs to be incorporated. The slicing criteria could be based on data-flow, hierarchy or some certain functionality. In addition to aforementioned challenges, in future we plan to also automate the methodology using frameworks for automated interaction with LLM instances.

## References


[1] R. Drechsler, *Advanced formal verification*. Springer, 2004.
[2] ——, *Formal verification of circuits*. Springer Science & Business Media, 2013.
[3] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," *arXiv preprint arXiv:2305.13243*, 2023.
[4] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," *arXiv preprint arXiv:2306.14027*, 2023.
[5] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "From rtl to sva: Llm-assisted generation of formal verification testbenches," *arXiv preprint arXiv:2309.09437*, 2023.
[6] C. Sun, C. Hahn, and C. Trippel, "Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions," in *First International Workshop on Deep Learning-aided Verification*, 2023.
[7] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "Llm for soc security: A paradigm shift," *arXiv preprint arXiv:2310.06046*, 2023.
[8] X. Meng, A. Srivastava, A. Arunachalam, A. Ray, P. H. Silva, R. Psiakis, Y. Makris, and K. Basu, "Unlocking hardware security assurance: The potential of llms," *arXiv preprint arXiv:2308.11042*, 2023.
[9] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," *arXiv preprint arXiv:2302.01215*, 2023.
[10] 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.
[11] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.
[12] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a targeted translator in fortran," in *Proc. Intl. Symp. Circuits and Systems, 1985*, 06 1985.
[13] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the iscas-85 benchmarks: a case study in reverse engineering," *IEEE Design Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.