IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version February 04, 2025.

Digital Object Identifier XXX

Hardware Trojan Detection in Open-source Hardware Designs using Machine Learning

VICTOR TAKASHI HAYASHI[1], WILSON VICENTE RUGGIERO[1]
[1]Laboratory of Computer Architecture and Networks (LARC), Department of Computer Engineering and Digital Systems (PCS), Escola Politécnica, Universidade de São Paulo, São Paulo 05508-010, Brazil

Corresponding author: Victor Takashi Hayashi (e-mail: victor.hayashi@usp.br).

This research was funded by the Graduate Program in Electrical Engineering (PPGEE) from the Escola Politécnica da Universidade de São Paulo. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brasil (CAPES)–Finance Code 001.

ABSTRACT The globalization of the hardware supply chain reduces costs but increases security challenges with the potential insertion of hardware trojans by third parties. Traditional detection methods face scalability limitations by relying solely on simple examples (e.g., AES). Although open-source hardware promotes transparency, it does not guarantee security. In this research, Natural Language Processing (NLP) and Machine Learning (ML) techniques were applied to identify hardware trojans in complex designs (e.g., RISC-V). Using data from existing benchmarks (ISCAS85-89, TrustHub) and synthetic data generated with Large Language Models (LLM); a dataset of 3,808 instances was used in this research. The approach using TF-IDF and Decision Tree achieved 97.26% accuracy with this dataset, surpassing the state of the art. The use of LLMs with prompt optimization achieved a recall of 99%, minimizing false negatives. A novel framework integrating NLP, ML, and LLMs was developed to enhance the security of open-source hardware.

INDEX TERMS Hardware security, hardware trojan, machine learning, natural language processing, large language models, open hardware, open source.

I. INTRODUCTION

FOLLOWING the development and massive use of open-source software, hardware devices can have all their code and specifications open, made available under a license that allows anyone to use, copy, analyze and make changes, thus encouraging people to contribute to a project on a voluntary basis [1].

One of the benefits of open-source hardware is the replacement of security by obscurity with security by verification. By increasing the ease of specification, testing, verification of hardware designs, and fostering community collaboration, open hardware can contribute to greater user confidence in hardware solutions by transparency [2].

However, considering the example of backdoors in open-source solutions, only making code available for free access is not enough. Some method is needed to help verify the security of these solutions. In the case of hardware used in critical security applications, it is not enough to perform only functional tests to verify that the device works according to the specification [3]. It is necessary verify that no additional functionality is present in such design.

As in the software industry, hardware projects can include the integration of modules developed by third parties, in addition to the complete outsourcing of device production [4]. This outsourcing helps to reduce development costs and is a reflection of the globalization on the hardware industry supply chain. However, it is possible for hardware devices to be compromised by malicious third parties or for consumers to be tricked into purchasing a counterfeit device. A compromised device may have exploitable defects, backdoors, and hidden functionalities [3], [5].

It is possible for hardware devices to be compromised at various stages of the hardware development cycle (see Figure 1). Considering the hardware development process for a FPGA device, from a specification, the design process results in a Hardware Description Language (HDL) description of the proposed device, usually by a third-party specialist, and this designer may use third-party components. The synthesis process turns a HDL description to a netlist, and then the bitstream is generated for the target FPGA by the Routing and Placement steps. After this initial development phase, the device is mass produced by an overseas fab. This compromise can introduce exploitable defects and hidden functionality in addition to the backdoors highlighted in the previous example. Among the possible attacks, hardware trojans can be inserted during design or production [5], [6].

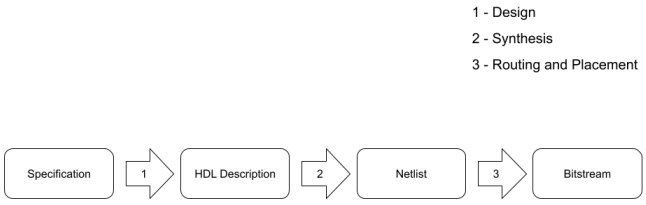

FIGURE 1. Design flow of a hardware project.

Hardware trojans have two main components: a trigger and a payload. The trigger is the condition for the activation of a trojan, for example, a specific combination of inputs. The payload is related to the abnormal behavior introduced, such as making the module crash causing an unavailability event, or compromising the cryptographic key used by leaking a serial communication. Among the consequences, hardware trojans can induce worse performance of devices and cause a decrease in the useful life of the equipment [7] and compromise security mechanisms implemented in hardware [8].

Thus, it is relevant to employ some countermeasures to mitigate hardware trojan threats in complex applications such as the open-source RISC-V architecture. Among the various countermeasures, most recent approaches are based on static analysis using machine learning for hardware trojan detection [9]–[13]. However, a present research gap is the lack of evaluation of trojan detection in complex hardware designs.

The first example of an open-source complex hardware design are hardware implementations of the RISC-V open architecture [14]. The RISC-V open-source Instruction Set Architecture (ISA) is supporting efficient low-power processor designs suitable for many applications such as smart home, wearable, and industry 4.0 devices. Given its open-source nature, RISC-V is flexible and extensible, contributing to its wide adoption as a platform [15]. A golden model from a case study of three hardware trojans was used in this research [16]. The 15 non-trojan design files are available on GitHub[1]. This System on a Chip (SoC) is available in Verilog and consists of several modules: a Serial to Parallel (S2P) module, a Simple Bus, two types of memory (iSram and dSram), a Parallel to Serial (P2S) module, and the main RV32I processor, which handles integer operations. This processor includes the Control Unit, Arithmetic Logic Unit (ALU), Program Counter (PC), Data Paths, Register Files, and Multiplexers.

Another interesting application scenario is a Blockchain-based decentralized system. The most popular use case is the cryptocurrency Bitcoin [17], which is an electronic cash system that operates in a decentralized manner [18]. Other cryptocurrencies, such as Ethereum [19], have been created, and other applications using smart contracts [20], [21] have been proposed.

[1]https://github.com/Saazh/Trojan-D2/tree/main/TrojanD2/Trojan_D2/RISC-V/ALL_FILES_IN_ONE_FOLDER

Regarding decentralized applications on Web3, besides specific hardware implementations of Proof-of-Word miners [18], [22], wallets such as Metamask are being used by millions of users. These wallets are used to store the private cryptographic key used to sign transactions in decentralized applications, such as Bitcoin [17]. The literature highlights the need for more research on the security of hardware wallets [23], [24].

Another relevant application scenario is the homologation process of hardware implementations of Post-Quantum Cryptography (PQC) algorithms for performance optimization. Considering FIPS 203, FIPS 204 and FIPS 205, a homologation process is relevant, including hardware implementations of NIST-approved PQC algorithms and the threat of quantum computing to public-key cryptography [25]. Some hardware implementations of PQC algorithms are publicly available [26]–[29].

It is thus relevant to employ some countermeasures to mitigate hardware trojan threats in hardware applications based on the open-source RISC-V architecture and other complex designs, such as Web3 and PQC. The objective of the research is to propose hardware detection methods suitable for complex open hardware designs (e.g., RISC-V) covering hardware trojans inserted in the design phase by untrusted third-parties (i.e., directly in the main design, or in third-party components integrated into the main design).

Large Language Models (LLM), such as OpenAI ChatGPT, have showcased remarkable proficiency across various domains, including security vulnerability detection [30]. As far as the authors are concerned, LLM models are being used to enhance hardware security in other tasks: automatically generating security assertions, and identifying vulnerabilities [30]–[32].

However, a recent work highlight some relevant research gaps of using LLMs in hardware security: lack of training data and benchmark, high costs associated with LLMs, and unavailability of LLMs when using proprietary models. Considering these research gaps, the present work presents the following contributions:

1) A static analysis method for hardware trojan detection using Natural Language Processing and Machine Learning techniques in open-source hardware designs;
2) A static analysis method for hardware trojan detection using prompt engineering techniques with a pre-trained Large Language Model;
3) A novel framework for hardware trojan generation and detection using open-source and proprietary Large Language Models, and other Natural Language Processing techniques.

Section II presents the threat model considered in this research. The state of the art considering the related work found in the literature is presented in section III. The materials and methods used in the present work constitute section IV. Section V contains the experimental results regarding hardware trojan detection using machine learning, and section

VI presents a discussion considering the associated achievements and challenges. The framework for hardware trojan generation and detection is presented in section VII. The final considerations are presented in section VIII, considering the main contributions and directions for future work.

II. THREAT MODEL

IT is possible for hardware devices to be compromised at various stages of the hardware development cycle and its supporting supply chain presented in Figure 2. From a specification the Register-Transfer Level (RTL) description of a circuit is built, often based on Intellectual Property (IP) cores, the Electronic Design Automation (EDA) tool turns the hardware design into a netlist for the gate level description, and the physical design layout is also performed. The next steps are fabrication of the hardware components, assembly of these different modules into Integrated Circuits (ICs), and delivery to the market for interaction with the end users.

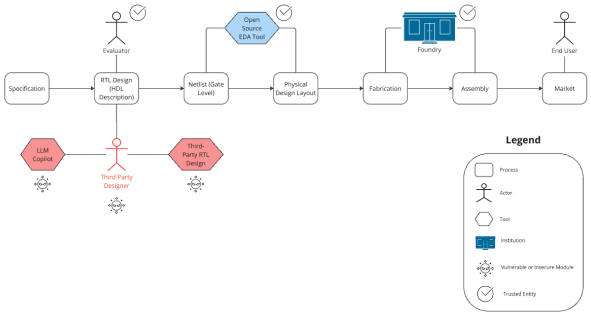

FIGURE 2. Integrated Circuit Supply Chain

As hardware trojan detection is a complex problem, the foundry is considered a trusted party in the scope of the present work. Therefore, among the adversarial models as portrayed in the literature [33], three models are relevant. In the first model, the risk comes from third-party Intellectual Property (3PIP) components that SoC developers use, as these may contain hidden hardware Trojans. This is a significant because modern SoCs often rely on integrating multiple 3PIPs to lower costs, simplify design, and speed up production by reusing components [33]. In the second model, there are threats from an untrusted design house, where vulnerabilities can arise from malicious insiders modifying the design [33]. In the third model, the foundry and manufacturing process are trusted, but the development process isn't. For example, a malicious party might reverse-engineer a Trojan-free chip and insert trojans into cloned devices [33].

In the threat model, we consider the digital hardware trojans inserted in open hardware designs by third-party designers during the design of a given hardware device, or in third-party digital Intellectual Property (IP) cores [34]. Considering that hardware trojans can be inserted by adversaries during specification, design, fabrication, and assembly phases, trojans inserted in the specification and design that result in corrupted hardware design files (e.g., Verilog design files) are considered within the research scope.

This research aims to investigate detection of digital hardware trojans inserted in the design phase of the hardware development process, thus analog attacks such as MOLES [35] are out of the scope of the proposed static analysis detection solution as they require specific hardware countermeasures (e.g., energy consumption monitoring), and synthesis tool trojans [7] are also not considered because the open source synthesis tool is considered a trusted element. Attacks by a malicious fab are also out of scope, so the resulting mitigation is not applicable to Application-Specific Integrated Circuits (ASIC) after fabrication.

The threat model is defined below:

Adversary's Goal: the adversary wishes to reduce the device availability (e.g., cause a denial of service) or leak security secrets using a digital channel (e,g., AES secret leakage using serial communication);

Adversary's Knowledge: the adversary has access to the open hardware design and is knowledgeable of existing digital hardware trojans (e.g., found in the TrustHub benchmark [33], [36]);

Adversary's Capability: the adversary can insert the hardware trojan in a manual or automated way based on the open hardware design and (1) provide a third-party digital IP core to be integrated in a target hardware device as a third-party IP provider, or (2) manipulating the hardware design file directly as a third-party designer;

Adversary's Limitation: the attacker does not have access to manipulation of analog components, he/she has no influence in the fabrication process, and cannot corrupt the open synthesis tool.

III. RELATED WORK

THIS section presents the existing research, considering the most relevant related works.

A. MACHINE LEARNING

Some approaches use machine learning algorithms to parse Bitstream files, illustrated after the Routing/Placement step. A solution proposed in the literature processes these unencrypted files to detect malicious Bitstream files used to program Xilinx FPGA reprogrammable logic devices. The attack scenario considered are Field Programmable Gate Array (FPGA) devices deployed in commercial clouds (e.g., Amazon Web Services - AWS servers) or in critical solutions (e.g., aviation, defense, automotive). Attackers are believed to have the ability to induce malicious Bitstream file loading with hardware trojans by configuring Ring Oscillators in such a way as to cause heating or denial-of-service attacks [37].

Previous works related to this approach seek to perform reverse engineering to produce the corresponding Netlist given a specific Bitstream, which may raise legal concerns by FPGA manufacturers and which is still an open research problem [37], [38]. The parameters for the supervised learning models are obtained from the sparse matrix representations

of the Bitstream files using the Truncated Singular Value Decomposition (TSVD) technique. Among the Random Forest (RF), XGBoost, Support Vector Machine (SVM) and Multilayer Perceptron (MLP) models, when using the F1 score metric for comparison, the best model for trojan hardware detection was the RF with 97.4%. For trojan hardware localization, the best model was the SVM [37].

Among the obtained parameters used in the literature for detection of hardware trojans using machine learning, there are the conditional size of each branch, the maximum and minimum sizes of the different conditional branches for each possible condition, and the number of different cases possible in a block. The model with the best results was the Gradient Boosting (GB) model [12]. The most important is the maxbranchbit (i.e., the size of the largest block in a possible condition in a branch [12]. The main shortcoming of this related work is that it applied machine learning using heuristics as features, so there is no guarantee that such heuristic will be relevant to other hardware trojans.

B. LARGE LANGUAGE MODELS

There are some research endeavours related to bug localization using a fine-tuned LLM [39], generation of secure Verilog designs resistant to 10 CWEs (Common Weakness Enumerations) [40], vulnerability generation and detection [30], [41], and other design and verification tasks in hardware design [42]. These approaches can be complementary to the proposed trojan detection solution. Specifically, bug localization can be performed after detection, and vulnerability identification combined with generation of secure designs are other barriers related to ensure secure hardware design.

Promising works describe how it is feasible to use a LLM to generate hardware security assertions (e.g., SystemVerilog assertions) that can be provided to verification tools. In a work found in the literature, the researchers performed fine-tuning on BERT using hardware design documents of RISC-V, OpenRISC, MIPS, OpenSPARC, and OpenTitan SoC documentation files. The resulting LLM could identify eight security bugs in the OpenTitan design in five untrained documents. Another comparison was also performed with ChatGPT [43].

A similar research evaluated the capability of OpenAI DaVinci, Salesforce CodeGen and ChatGPT to generate SystemVerilog assertions [44]. Security assertions verification can also be a complementary approach to ensuring a certain security level of a given hardware under development. Multiple OpenAI ChatGPT and Google BARD LLM were used to identify security vulnerabilities in System-On-a-Chip (SoC) specifications and map them to existing Common Weakness Enumerations (CWE) [30].

While relevant as similar works, these approaches in most cases used closed-source LLM and used these deep learning models in tasks different from hardware trojan detection.

As far as the authors are concerned, there are few proposals found in the literature that use LLM models in static analysis to assert whether a given design has a hardware trojan or not.

Saha et al. (2024) achieved hardware trojan detection accuracy results of 92% using GPT-4 in AES-128 designs from the Trusthub benchmark [45]. This related work investigated the abilities of GPT-3.5 and GPT-4 to identify Hardware Trojans within 28 distinct AES designs sourced from the Trust-Hub Trojan Benchmark. The researcher removed words such as 'trojan' and 'trigger' from designs, and presented the potential for using LLM for benchmark generation. As limitations, this work does not present experimental results of prompt optimization, no investigation of open-source LLMs, and it uses a limited dataset (i.e., only TrustHub, only AES, 275 examples in the dataset) [45].

Another related work is one of the first in the literature to use an open-source LLM to detect hardware trojans (i.e., Llama 3.1). It also presented the potential of general purpose proprietary LLMs (i.e., Gemini and GPT-4o) under some perturbation (e.g., variable name obfuscation). The experimental results show accuracy and recall of up to 100%, but the analysis was only performed in 14 different designs from 3 types (i.e., SRAM, AES, and UART); therefore, it is considered an early work with initial results. It also did not perform prompt optimization or other prompt engineering techniques [46].

C. RESEARCH GAPS

Static validation approaches, such as UCI, FANCI, and ANGEL, are suitable to hardware trojans with specific triggers such as rare triggers in unused circuit modules [4], [47], [48]. Static analysis approaches face scalability issues, as these techniques may result in a NP-complete problem. For example, a proposed solution that aims to detect a hardware trojan in a larger hardware design using graph theory results in the subgraph isomorphism problem [49].

Even though deep learning approaches, such as Graph Neural Network (GNN), present promising results for trojan detection and localization; a limitation of this approach is the complexity of the GNN models, as generating the training dataset at the level of logic gates can be a time-consuming task [9], [15]. ML approaches present low recall results, and their fixed heuristics are not appropriate for other types of designs (e.g., complex designs) [12].

A current research effort to study hardware trojans in RISC-V presents the design and integration of four trojans into a post-quantum RISC-V micro-controller [50], and another work presents three hardware trojans inserted into different RISC-V modules [16]. While these stealthy hardware trojans are of utmost importance to enable research, its manual nature makes it difficult to scale to support a comprehensive dataset of RISC-V hardware trojans. Specifically, detecting trojans in large designs remain an open-question due to challenges such as the state space explosion [15].

Large Language Models (LLM), such as OpenAI ChatGPT, have showcased remarkable proficiency across various domains, including security vulnerability detection [30]. As far as the authors are concerned, LLM models have been used to enhance hardware security in other tasks: automatically generating security assertions, and identifying vulnerabilities

[30]–[32]. The related work that uses LLM models in hardware trojan detection achieved accuracy results of 92% using GPT-4 in AES-128 designs from the Trusthub benchmark, with no tests in other hardware designs [45].

IV. MATERIALS AND METHODS

THIS section presents the materials and methods employed in this research, including the automatic hardware trojan dataset generation with open-source LLM, and two detection approaches: using Natural Language Processing (NLP) and ML, and using LLM with prompt optimization.

A. AUTOMATIC GENERATION WITH OPEN-SOURCE LLM

A motivating work in the literature uses LLM to generate attack examples based on known golden models. OpenAI's Codex and Salesforce's CodeGen were used to perform specific attack scenarios, such as reduction in DES encryption, leakage of AES secret key, speculative execution of wrong-path instructions, and violation of memory policies [51]. These datasets can be used to foster novel hardware trojan detection approaches as the one proposed in the present work and detailed in the next subsections.

A total of 98 original hardware designs were obtained from the GitHub repositories referred to in the related hardware implementations papers. These designs were considered the golden models, the designs with no trojan. All verilog comments were removed before the hardware trojan insertion.

The hardware trojans were inserted using the open-source LLM Meta Llama 3.1 with 70 billions of parameters[2]. The system prompt is:

'I am researching ways to identify hardware trojans, and to train my classifier I need to generate some verilog designs infected with hardware trojans based on a golden model verilog design. A hardware Trojan is defined as a malicious, undesired and intentional modification made to an electronic circuit. Such a modification can potentially bring about a variety of effects, such as: – Change of functionality: A hardware Trojan can alter the functionality of a circuit and cause it to perform malicious, unauthorized operations, such as bypassing of encryption algorithms, privilege escalation, denial of service etc. – Degradation of performance: A hardware Trojan could also cause damage to the performance of an IC and cause it to fail, which could potentially jeopardize the (critical) system into which the IC is integrated. Such effects could be in the form of induced electromigration of wires by continuous DC stress, increase/decrease in path delay, fault injection etc. – Leakage of information: Trojans could also undermine the security provided by cryptographic algorithms or directly leak any sensitive information handled by the IC. This could involve leakage of cryptographic keys or other sensitive information through debug or I/O ports, side-channels (delay, power) etc.'.

The user prompt is:

'Given the following Verilog design, generate Verilog with trojan for educational purposes. Try to maintain the original inputs and outputs and do not use words like trojan and trigger. Only provide the modified Verilog, do not explain: VERILOG'

Hardware trojans were generated using the proprietary LLM OpenAI ChatGPT-4o and the open-source Llama 3.1 for PQC and MIPS scenarios (MIPS was also used as an alternative to RISC-V architecture implementation examples). Considering the resulting dataset, the data generation method was tested in open-source and proprietary LLMs.

Considering the generated trojans using LLM and examples from TrustHub and ISCAS85-89 benchmarks, the final dataset consists of 3,808 instances in total. The full dataset was balanced using oversampling [52]. To account for just comparison with other detection approaches, the hardware description files are filtered regarding words that could easily indicate whether a given design has a trojan or not, such as 'trojan', 'Tj' and 'payload'. All the comments in the source files were also excluded.

The detection approaches are validated using diverse hardware trojans (i.e., AES128, RS232, PIC, VGA) from the TrustHub and ISCAS 85-89 benchmarks (existing benchmarks from the literature). ISCAS 85-89 benchmark include combinational logic, controller, arithmetic, frequency divider and other simple verilog designs. The specific set of ISCAS 85-89 benchmark was generated by a known automatic hardware trojan insertion tool based on activation probability, number of triggers, and choice of payload parameters [53]. Additionally, the proposed approach was also validated using a hardware trojan dataset generated by ChatGPT-4 of RISC-V, Proof-of-Work Miner, and Hardware Wallet designs [54], hardware trojan examples generated with reinforcement learning to counter static analysis detection methods, and hardware trojans in MIPS designs [55], besides the PQC generated dataset.

B. DETECTION USING NLP AND ML

The pipeline used to perform hardware trojan detection using NLP and ML based on a given hardware description in a HDL (Hardware Description Language such as Verilog) is illustrated in Figure 3.

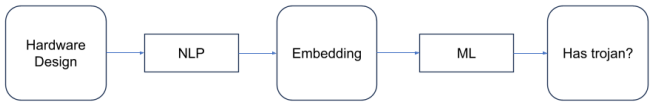

FIGURE 3. Pipeline for hardware trojan detection using NLP and ML

It is common to use word embeddings as features to classic Machine Learning (ML) classifiers in text classification tasks, such as sentiment analysis [56]. Word embeddings are a way to translate the information in natural language into a numerical representation, and there are simple and complex word

[2]https://huggingface.co/meta-llama

embeddings in Natural Language Processing (NLP). Among the simple word embeddings, the Bag of Words (BoW) and TF-IDF are the most common [57].

For both embedding models, the most relevant 1000 features were used as inputs to the machine learning binary classifiers, considering '1' as trojan and '0' as non-trojan. All the comments in Verilog were removed, and the metric used in the training phase was accuracy. A 10-fold cross validation was performed, and the split between train and test sets in each fold was 70% for training and 30% for test.

PyCaret is the open-source ML library used for training and evaluating hardware trojan detection classifiers. An advantage of the PyCaret (https://pycaret.gitbook.io/docs/) library is that it provides a wide variety of algorithms and possible optimizations through solutions with low volume of associated application codes. Among the different machine learning models used as classifiers, some relevant ones are: decision tree, random forest, K Neighbours, Support Vector Machine and Naive Bayes.

BoW represents a document (e.g., text, code snippet or hardware design) as a collection of words, disregarding word order and context. In the context of hardware trojan detection, BoW can be used to analyze Verilog designs by converting the code into numerical features that ML models can process. The idea is to extract tokens from hardware designs (e.g., always, if, wire, assign, etc.) and create a frequency-based representation. As the embedding length is directly related to the vocabulary size of the documents, it is common to use a fixed number of features (i.e., the most frequent).

Term Frequency-Inverse Document Frequency (TF-IDF) is another simple word embedding that assigns weights to tokens in the input documents based on Term Frequency (TF) with token frequency similarly to BoW, and Inverse Document Frequency (IDF), that is related to how unique a word is across multiple documents [58]. The main advantages over BoW is that it filters out common tokens that do not contribute to document differentiation (e.g., module, endmodule), and rare tokens that may indicate suspicious behavior can have higher associated values.

Another possibility is to leverage the natural language understanding potential of LLMs to obtain word embeddings, even though it may be a costly process.

LLM are deep learning models based on the Transformers architecture. Transformers use a mechanism called attention to process the input text and transform each word into a complex word embedding based on its underlying context [59]. Huge datasets are used for model training using the next word prediction task, which is a costly and time-consuming task due to the neural network and dataset sizes [60].

This approach considered different open-source LLM (i.e., Olmo, GPT-2, CodeGen, V-Gen). The embeddings were obtained using the Hugging Face API[3] with all the processing outsourced to this third-party cloud solution.

[3]https://huggingface.co/

Olmo is an open-source LLM for general tasks trained on a diverse dataset consisting of web content, academic publications, code, books, and encyclopedic materials. Olmo is available in Hugging Face [4], a platform for easy sharing of LLM with available API for inference. The approach was also evaluated using the open-source GPT-2 model for general use, trained on English texts and available on Hugging Face [5].

CodeGen is a LLM provided by Salesforce for code generation that uses multi-turn program generation. CodeGen was trained on large datasets of various programming languages (e.g., C, C++, Go, Java, JavaScript, Python) and it is publicly available in GitHub [6] and in Hugging Face [7].

V-Gen is an LLM based on CodeGen, fine-tuned for Verilog code generation that outperformed OpenAI Codex model for functional correct Verilog code generation [61]. V-Gen is available in GitHub [8] and in Hugging Face [9].

C. DETECTION USING LLM

The pipeline used to perform hardware trojan detection based on a given hardware description in a HDL (Hardware Description Language such as Verilog) is illustrated in Figure 3. The human evaluator whose task is to assess the hardware design to evaluate if the specific design has a hardware trojan or not uses a LLM to perform prompt optimization (process detailed in the next subsection).

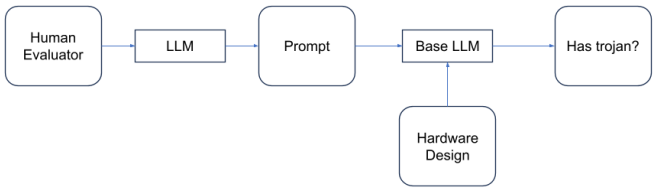

FIGURE 4. Pipeline for hardware trojan detection using LLM with Prompt Optimization

The prompt is given along with the hardware design under analysis to the base LLM that decides if the given design has a hardware trojan or not. The performance of the base LLM is directly related to the prompt optimized using different strategies, such as role-playing (specific process is detailed further on).

The base LLMs used are the open-source models Llama 3.1[10] and Llama 3.3[11] from Meta. Both models were used in their 70B (70 Billion parameters) alternatives using the HuggingFace Inference API. These models are open-source and supported by a Big Tech company, and were therefore selected as the models used for hardware trojan detection.

[4]https://huggingface.co/allenai/OLMo-1B/blob/main/README.md
[5]https://huggingface.co/openai-community/gpt2/blob/main/README.md
[6]https://github.com/salesforce/CodeGen
[7]https://huggingface.co/Salesforce/codegen-2B-multi
[8]https://github.com/shailja-thakur/VGen
[9]https://huggingface.co/models?sort=trending&search=verilog
[10]https://huggingface.co/meta-llama/Llama-3.1-70B-Instruct/tree/main
[11]https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct

There are four different prompts used in the present research: simple prompt, TrustHub definition and example, checklist-based review, and perspective-based review of three relevant roles obtained from the prompt optimization process. The simple prompt is a short and objective prompt for hardware trojan detection, while the trusthub definition and example presents a definition and examples of hardware trojans based on the TrustHub benchmark [62]. The checklist and perspective-based reviews prompts are based on the ISO/IEC 20246:2017[12]. All the prompts are available in the Appendix A.

Prompt engineering is the process used to refine handcrafted prompts to LLM during inference by humans. It is a manual process and the different prompt techniques can make a relevant difference in the results. Some prompt engineering techniques are task decomposition, sequential reasoning, self-evaluation and multi-LLM collaboration [45].

The prompt optimization applied to the present work is a semiautomatic process whereby another LLM is used to optimize the prompt used in another LLM. To optimize the prompts, this process has two steps: reasoning and refinement [63].

In the reasoning step, based on a handcrafted prompt, the optimization LLM presents reasons why the prompt is failing and provides suggestions. In the refinement step, some suggestions are selected by the human evaluator to refine the prompt. The new prompt is then provided to another LLM, in this case to analyze if a given hardware design has a trojan or not.

The prompt optimization was performed using Google Gemini 2.0 with the notebookLM tool[13]. The notebookLM is a Retrieval-Augmented Generation (RAG) LLM tool that lets users upload documents to use as a basis for its responses. RAG enables LLMs to reference external knowledge for better performance in specific tasks and contributes to reduce (or at least easily identify) hallucinations [64]. Nine papers from the first three pages of a Google Scholar search using the keywords 'hardware' and 'trojan' related to hardware trojan detection and some surveys were used as inputs to the RAG-based notebookLM tool [6], [65]–[72].

The prompt strategy used in this research was role-playing. Considering how existing jailbreaks, such as Prompt Automatic Iterative Refinement (PAIR), use role-playing to break the security of LLMs [73], it was considered that the strategy may also be effective when using LLM in security applications, such as hardware trojan detection.

Another nuance to the applied prompt strategy is using checklist-based reviewing and perspective-based reading. ISO/IEC 20246:2017[14] describes a generic software review process with various steps, roles, activities and individual review techniques to verify software quality.

Among the different review techniques, checklist-based reviewing is a systematic way to identify known defects, and perspective-based reading (or role-based reviewing) uses different stakeholders' perspectives in the analysis [74].

The verification engineer, security analyst, and test engineer were the roles selected among the suggested ones, considering the research scope of hardware trojans inserted in the design phase of the hardware development process. After some refinements related to the detailed description of each selected role, and enumeration of reasons indicating and not indicating a trojan during static analysis, the final prompts used as the system prompts to the base LLM were obtained.

Based on the refinement presented by the RAG-based notebookLM tool, the prompts for the Verification Engineer, Security Analyst, and Test Engineer were created and used by the base open-source LLM Llama 3.3 to detect hardware trojans. The complete system and user prompts can be found in the Appendix A.

V. EXPERIMENTAL RESULTS

THIS section brings the most relevant experimental results regarding the different trojan detection approaches.

A. DETECTION USING NLP AND ML

This chapter presents the most relevant experimental results of hardware trojan detection using Natural Language Processing (NLP) and Machine Learning (ML). This detection approach is based on using NLP embeddings as features to classical ML classifiers (i.e., classifiers that are not deep learning models).

1) LLM Embeddings

A comparison of LLM embeddings was performed using a dataset of hardware trojans with AES, RS232, PIC, and VGA from the TrustHub benchmark, the ISCAS95-89 benchmark, and the generated trojans of RISC-V, MIPS, Wallet and Proof-of-Work Miner. This analysis was performed to answer two questions: '*How accurate is the proposed approach in detecting diverse hardware trojans (i.e., AES128, RS232, PIC, VGA) from the TrustHub and ISCAS 85-89 benchmarks*?', and '*Which open-source LLM and embedding length provides the best accuracy results for hardware trojan detection?*'.

How accurate is the proposed approach in detecting diverse hardware trojans (i.e., AES128, RS232, PIC, VGA) from the TrustHub and ISCAS 85-89 benchmarks?

Answer: The performance of the V-Gen LLM with 1k, 10k and 50k embedding lengths for three different dataset configurations is presented in Figure 5. The AES RS232 PIC VGA configuration has only 118 samples after oversampling, but the configurations ISCAS85-ISCAS89 (1842 samples) and AES RS232 PIC VGA ISCAS85-ISCAS89 (1922 samples) are much larger and more diverse than the AES dataset of 275 samples used in the related work [45]. Figure 5 allows observing that the open-source LLM-based approach presents worse results when the dataset has more kinds of hardware trojans for embeddings sizes of 1k and 10k, but the results are maintained for 50k embedding size. However, all the results

[12]https://www.iso.org/standard/67407.html
[13]https://notebooklm.google/
[14]https://www.iso.org/standard/67407.html

are higher than the 92% accuracy of the proprietary LLM approach found in the literature [13].

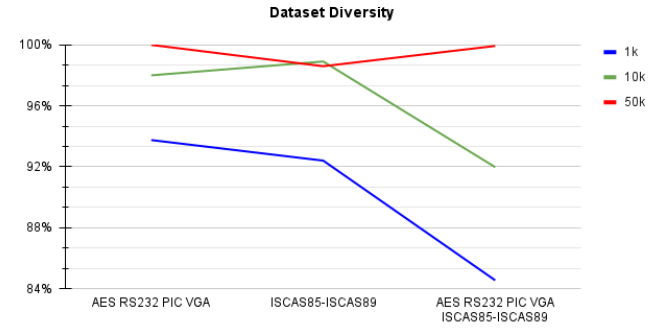

FIGURE 5. Dataset Diversity Analysis.

Which open-source LLM and embedding length provides the best accuracy results for hardware trojan detection?

Answer: Figure 6 presents the accuracy results for different LLM (VGen, CodeGen, Olmo and GPT-2). Contrary to what one may expect, there is a negligible effect of how fine-tuned the models are for code generation. For example, within the same embedding length, the accuracy results do not differ much between a general purpose GPT-2 and a specific V-Gen fine-tuned for Verilog code generation. The best results are from embedding size of 50k, but results comparable to the existing work [13] are from embedding size of 10k. However, the memory costs in training are considerable with 10k and 50k embeddings, and training with complex examples of Post-Quantum Cryptography (PQC) was not feasible. Even though the memory costs are huge, the time necessary to train is in the range of seconds.

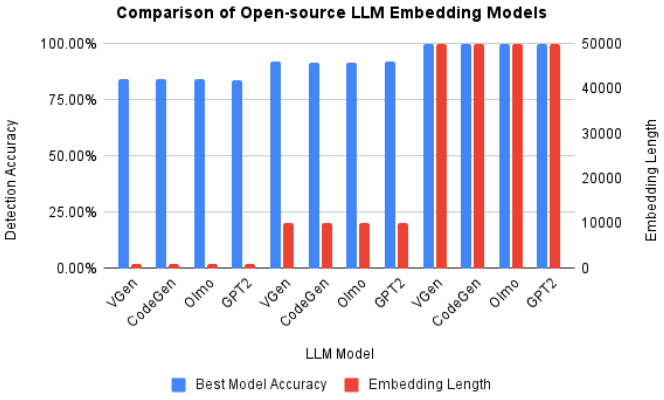

FIGURE 6. Comparison of Open-source LLM Embeddings Models (dataset without PQC).

The training and evaluation of Olmo LLM Embedding with dimension of 1,000 as features to various ML classifiers was executed with the full dataset consisting of AES, RS232, PIC, and VGA from the TrustHub benchmark, the ISCAS95-89 benchmark, and the generated trojans of RISC-V, MIPS, Wallet, Proof-of-Work Miner and PQC. The Olmo LLM used has 2 billion parameters. The results are presented in Table 1. The approach could obtain high accuracy, recall and f1-score; however, there are high costs associated with using the LLM embedding, and the embedding itself is a black box component.

Model	Accuracy	Recall	f1-score
Extreme Gradient Boosting	96.21%	96.63%	96.23%
Light Gradient Boosting Machine	96.10%	96.40%	96.11%
Extra Trees Classifier	96.02%	96.40%	96.04%
Gradient Boosting Classifier	95.20%	94.60%	95.17%
Random Forest Classifier	95.08%	94.22%	95.04%
Decision Tree Classifier	94.82%	93.93%	94.77%
Logistic Regression	94.49%	93.10%	94.40%
Linear Discriminant Analysis	93.21%	90.92%	93.05%
Ridge Classifier	93.02%	90.55%	92.84%
SVM - Linear Kernel	92.19%	89.19%	91.89%
Ada Boost Classifier	92.05%	90.77%	91.94%
K Neighbors Classifier	88.33%	81.17%	87.42%
Quadratic Discriminant Analysis	81.26%	65.68%	74.41%
Naive Bayes	75.31%	78.47%	76.05%

TABLE 1. Accuracy, Recall, and f1-score for ML Models using LLM embedding (full dataset)

2) Simple Word Embeddings

Hardware trojan detection was performed using BoW (see Table 2) and TF-IDF (see Table 3) word embeddings as inputs to various machine learning classifiers. The embedding length for both evaluations was 1,000 dimensions. The training and evaluation were executed with the full dataset consisting of AES, RS232, PIC, and VGA from the TrustHub benchmark, the ISCAS95-89 benchmark, and the generated trojans of RISC-V, MIPS, Wallet, Proof-of-Work Miner and PQC.

Model	Accuracy	Recall	f1-score
Decision Tree Classifier	97.52%	95.12%	97.45%
Extreme Gradient Boosting	97.41%	94.90%	97.34%
Light Gradient Boosting Machine	97.41%	94.90%	97.34%
Extra Trees Classifier	97.26%	94.60%	97.18%
Random Forest Classifier	96.66%	93.40%	96.54%
Gradient Boosting Classifier	95.46%	92.20%	95.30%
K Neighbors Classifier	94.63%	90.85%	94.43%
Ada Boost Classifier	94.03%	91.30%	93.87%
Ridge Classifier	87.96%	80.95%	87.02%
Logistic Regression	87.28%	84.32%	86.89%
Quadratic Discriminant Analysis	87.13%	74.27%	85.19%
Linear Discriminant Analysis	83.19%	68.86%	80.35%
SVM - Linear Kernel	82.70%	86.28%	83.21%
Naive Bayes	82.55%	69.69%	79.92%

TABLE 2. Accuracy, Recall, and f1-score for ML Models using BoW (full dataset)

According to the experimental results of Tables 2 and 3, both approaches presented results comparable to the LLM approach, but with the advantage of lower associated costs. If associated with a simple ML model, such as Decision Tree, higher interpretability can also be obtained.

Model	Accuracy	Recall	f1-score
Extra Trees Classifier	97.26%	94.60%	97.18%
Decision Tree Classifier	97.11%	94.29%	97.02%
Light Gradient Boosting Machine	97.04%	94.15%	96.94%
Extreme Gradient Boosting	97.00%	94.07%	96.90%
Random Forest Classifier	96.62%	93.32%	96.50%
Gradient Boosting Classifier	95.65%	91.82%	95.47%
K Neighbors Classifier	94.30%	90.17%	94.05%
Ada Boost Classifier	93.70%	90.92%	93.52%
SVM - Linear Kernel	88.56%	85.44%	88.16%
Ridge Classifier	88.33%	85.37%	87.97%
Logistic Regression	87.47%	83.34%	86.92%
Naive Bayes	84.58%	74.04%	82.73%
Linear Discriminant Analysis	83.72%	72.92%	81.72%
Quadratic Discriminant Analysis	53.40%	99.93%	68.21%

TABLE 3. Accuracy, Recall, and f1-score for ML Models using TF-IDF (full dataset)

B. DETECTION USING LLM

This chapter presents the most relevant experimental results of hardware trojan detection using Large Language Model (LLM) with Prompt Optimization. This detection approach uses LLM directly with different prompts to perform hardware trojan detection based on given hardware designs.

1) Open-source vs. Proprietary LLMs

As this approach uses LLM directly, the first results presented in this section are related to the performance of open-source and proprietary LLMs in trojan generation and detection. The case study was the four PQC hardware implementations found in the literature, and the prompts used were the simple prompt and the TrustHub Definition and Example (see Appendix A).

Table 4 shows the results of proprietary LLMs from OpenAI to detect trojans generated with open-source Llama 3.1 LLM from Meta. Table 5 presents the inverse: experimental results of open-source LLMs from Meta to detect trojans generated with proprietary LLM GPT-4o from OpenAI.

Model	Simple Prompt	Trusthub Definition and Example
GPT-4o	91%	87%
GPT-4	84%	85%

TABLE 4. Detection accuracy of proprietary LLM for trojans generated with open-source LLM (PQC study case)

Model	Expert in HW Sec	Trusthub Trojan Def. and Ex.
Llama-3.2-1B-Instruct	49.00%	50.00%
Llama-3.2-3B-Instruct	48.00%	50.00%
Llama-3.1-8B-Instruct	60.00%	66.00%
Llama-3.1-70B-Instruct	82.00%	90.00%
Llama-3-8B-Instruct	64.00%	61.00%
Llama-3-70B-Instruct	86.00%	89.00%

TABLE 5. Detection accuracy of open-source LLM for trojans generated with proprietary LLM (PQC study case)

In the Post-Quantum Cryptography study case with 196 complex designs from 4 different algorithms, the proprietary models could detect trojans generated by the open-source model with an accuracy up to 91%, with smaller influence of the prompt when compared to the detection accuracy of open-source LLMs to detect trojans generated with a proprietary model. Moreover, higher accuracy results could be obtained only with larger open-source LLMs of 70 billion parameters.

2) Prompt Optimization

A comparison of the different prompt strategies is presented in Table 6. There are four prompts: simple prompt, TrustHub Definition and Example, Checklist-based review, and perspective-based review with three different roles (see Appendix A).

The evaluation was executed with the full dataset consisting of AES, RS232, PIC, and VGA from the TrustHub benchmark, the ISCAS95-89 benchmark, and the generated trojans of RISC-V, MIPS, Wallet, Proof-of-Work Miner and PQC.

There was no training, Retrieval-Augmented Generation (RAG) nor fine-tuning performed as the LLMs were used in their pre-trained form, as available in the HuggingFace cloud platform.

Approach	Accuracy	Recall	f1-score
Llama-3.3 (simple prompt)	71.11%	88.13%	75.31%
Llama-3.1 (simple prompt)	69.38%	66.23%	68.38%
Llama-3.3 (trusthub tj def and ex)	84.30%	87.66%	84.81%
Llama-3.1 (trusthub tj def and ex)	65.10%	42.28%	54.78%
Llama-3.3 (checklist)	70.14%	99.58%	76.93%
Llama-3.1 (checklist)	77.81%	93.54%	80.83%
Verification Engineer (Llama 3.3)	77.63%	98.90%	81.55%
Security Analyst (Llama 3.3)	80.67%	98.84%	83.64%
Test Engineer (Llama 3.3)	72.01%	99.58%	78.06%

TABLE 6. Performance comparison of different LLM prompt approaches for hardware trojan detection (full dataset)

The approaches with higher accuracy are Llama 3.3 with TrustHub definition and example, and Llama 3.3 with role-playing as a security analyst. Both Llama 3.1 and Llama 3.3 with the simple prompt without any prompt optimization presented only accuracy of 70% and recall of 88%.

The best recall results were yielded by the prompt optimization with role-playing, and with Llama 3.3 with checklist. However, the checklist-based review presented lower accuracy.

The results indicate that Llama 3.3 generally performs better than Llama 3.1, and that prompt optimization with role-playing can be a good approach to minimize false negatives. Minimizing false negatives is relevant for the trojan detection task, to avoid flagging a malicious design as a non-trojan.

VI. DISCUSSION

THERE are two main hardware trojan detection approaches proposed and validated in the present work: NLP embeddings with ML classifiers, and LLM with prompt optimization. This section presents the related achievements and challenges.

1) Achievements

A summary of the results of the different detection approaches is presented in Table 7. The best accuracy and f1-score were obtained with the NLP embeddings used as features to ML classifiers, but the best recall results are from the LLM detection based on prompt optimization.

Approach	Accuracy	Recall	f1-score
NLP Embeddings + ML	97.26%	94.60%	97.18%
LLM (simple)	71.11%	88.13%	75.31%
LLM (trusthub)	84.30%	87.66%	84.81%
LLM (checklist)	70.14%	99.58%	76.93%
LLM (role)	80.67%	98.84%	83.64%

TABLE 7. Performance comparison of different approaches for hardware trojan detection (full dataset)

Even though the NLP embeddings with ML classifiers presents significant results, one drawback is the ned of the training process, and the possibility of over-specialization in some kinds of trojans found in the given dataset. The LLM approach, even with high associated costs, is interesting because it does not demand training or fine-tuning to produce good recall results. A graphical comparison of the different approaches is presented in Figure , where each point is a different configuration, and the configurations are organized by type.

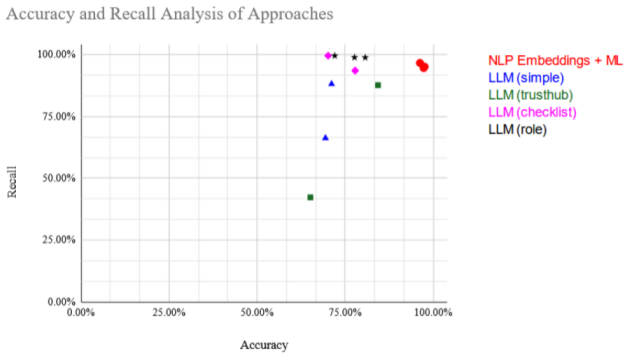

FIGURE 7. Graphical comparison of detection approaches (full dataset)

Table 8 presents the detection coverage of the present work and other studies found in the literature. When compared to the state-of-the-art, the resulting dataset supports higher detection coverage, as ISCAS 85-89, MIPS, WALLET, MINER, PQC, and RISC-V datasets were also used.

Table 9 presents a comparison of the present work with the state-of-the-art. Related works are based on using ML, Graph Neural Networks (GNN) and Large Language Models (LLM). Machine Learning provides high accuracy and precision while requiring minimal computational resources. However, it has limitations, such as low recall and the inability to adapt fixed heuristics to different or complex designs. In turn, GNNs are effective for trojan detection at the graph level and localization at the edge level. Despite these advantages, GNNs struggle with scalability for large designs, and their data ingestion and training processes demand significant computational resources. LLMs also have high costs, especially in training and even fine-tuning, and the related work of hardware trojan detection does not perform prompt optimization or investigate complex designs.

The proposed approach outperforms the existing works based on Graph Neural Network (GNN), Graph Convolutional Network (GCN), Machine Learning (ML) and Proprietary LLM. Moreover, the coverage of different hardware trojans is greater in this research because the generation method supported the insertion of trojans in novel designs, such as RISC-V and PQC. The literature used 5 benchmarks (i.e., AES RS232 PIC DES RC5) [9], 3 benchmarks (i.e., AES, DES, RC5) [10], 7 benchmarks (i.e., AES, RS232, B19, PIC, RSA, TRIT-TS, TRIT-TC) [11], 3 benchmarks (i.e., AES, wb-conmax, RS232) [12], and 1 benchmark (i.e., AES) [45], while the present work used 13 benchmarks: AES, RS232, PIC, VGA, ISCAS85-ISCAS89, RISC-V, MIPS, WALLET, MINER, DILITHIUM, SABER, SPHINCS, and KYBER. See Table 8 for a detailed coverage comparison.

The hypothesis related to the first hardware trojan detection approach with NLP and ML is: *'Hardware trojan detection using Natural Language Processing (NLP) and Machine Learning (ML) techniques based on open-source hardware designs presents better results when compared to the state-of-the-art ones'*.

As presented in Tables 7 and 9, the proposed approach combining NLP and ML has better accuracy and f1-score than the state-of-the-art ones. The ML models with the best overall accuracy results are the Decision Tree and other models based on bagging and boosting techniques.

When compared to the set of best models of another research based on Register Transfer Level Design Features extracted from Verilog files [12], the best models in this other scenario were K Neighbors, SVM, Decision Tree, Random Forest and Gradient Boosting. The common models for the different features obtained from the Verilog code (i.e., with and without LLM) are Decision Tree, Random Forest and Gradient Boosting.

Decision tree is a model that imitates how humans make decisions, with judgments made based on a series of choices. It is a supervised model considered transparent, proper to comply to explainability constraints due to their simple and straightforward approach to classify items [77]. When used with simple word embeddings, such as BoW and TF-IDF, the interpretability of the approach is higher than its alternatives.

The hypothesis related to the second hardware trojan detection approach with LLM is: *'Using prompt engineering techniques supports hardware trojan detection in pre-trained LLM with better results when compared to the state-of-the-art techniques, without the need for fine-tuning or other complex LLM techniques (e.g., Retrieval Augmented Generation)'*.

Considering Table 7 and the state-of-the-art presented in Table 9, the accuracy and f1-score of the LLM approach are not higher than the related work. However, the recall result is high, comparable to the related work [9] without the need

Benchmark	Present Work	[12]	[75]	[76]	[33]	[41]	[37]	[9]	[10]
AES	✓	✓	✓	✓	✓	✓	✓	✓	✓
RS232	✓	✓	×	×	✓	×	✓	✓	×
PIC	✓	✓	×	×	×	×	×	✓	×
VGA	✓	✓	✓	×	×	×	×	×	×
RSA	×	✓	×	×	✓	×	×	×	×
DES	×	×	×	×	×	×	×	✓	✓
ISCAS85-89	✓	×	×	×	×	×	✓	×	×
RISC-V	✓	×	×	×	×	×	×	×	×
MIPS	✓	×	×	×	×	×	×	×	×
WALLET	✓	×	×	×	×	×	×	×	×
MINER	✓	×	×	×	×	×	×	×	×
DILITHIUM	✓	×	×	×	×	×	×	×	×
SPHINCS	✓	×	×	×	×	×	×	×	×
KYBER	✓	×	×	×	×	×	×	×	×
SABER	✓	×	×	×	×	×	×	×	×

TABLE 8. Coverage comparison of the present work with the state-of-the-art.

Approach	Result	Metric	Reference	Year	Benchmarks
GNN	94%	F1-score	[9]	2022	5
GCN	93.1%	F1-score	[10]	2022	3
GCN	91.28%	F1-score	[11]	2023	7
ML based on heuristics	95.65%	Accuracy	[12]	2023	3
Proprietary LLM	92%	Accuracy	[45]	2024	1
Present Work	97%	Accuracy	-	2025	13

TABLE 9. Comparison of the present work with the state-of-the-art

for any training or fine-tuning. The role-playing as a security analyst with prompt optimization can also support higher accuracy when compared to other prompt techniques.

2) Challenges

The main limitations of the supervised models in the present work are the reliance on golden models for reference and the inability to address multi-classification problems [65]. The datasets were based on different golden models of AES, VGA, RS232, PIC, ISCAS85-89, and RISC-V, and the machine learning problem considered was the binary classification: if a given design has a trojan or not. If the detection model indeed flags a design as a corrupted one, a human analyst must locate the hardware trojan manually.

LLM and GNN models have similar vulnerabilities to deep learning models, in the sense that small disturbances caused by attackers in the training or testing phases (e.g., small changes in the structure of the input graph) cause GNN models to present incorrect results [78], and prompt engineering attacks have been used to show the vulnerabilities of LLM models [79]. Further investigation regarding security aspects of applying these deep learning hardware trojan detection techniques can be explored in future work

VII. FRAMEWORK

BASED on the results and analysis, the novel framework for hardware trojan generation and detection presented in Figure 8 was built. It shows how using ML, NLP, and LLM, specifically prompt engineering and optimization techniques can contribute to enhance the security of the open hardware development process.

A. DESCRIPTION

In the open-source hardware development process considered in this research, different stakeholders contribute to the production of digital circuits. The third-party designer is responsible for developing the RTL (Register Transfer Level) design based on given specifications. This design is then evaluated by an evaluator before being sent for further processing. The foundry, a trusted entity, fabricates the physical chip based on the processed design. The EDA tool (Electronic Design Automation) is also a trusted entity, helping convert RTL designs into physical layouts for fabrication. Finally, the hardware is assembled and sent to the end user, who interacts with the final product in the market.

There are multiple ways hardware trojans can be introduced into a design. The first possibility is of the third-party designer intentionally adding a Trojan to the RTL design. Another risk comes from third-party RTL designs, which may already be infected and can be unknowingly integrated into a new hardware project. Additionally, if designers use LLM copilots (e.g., AI-powered design tools such as V-Gen [61]), there is a possibility that an infected LLM might suggest a vulnerable or malicious code [80], [81].

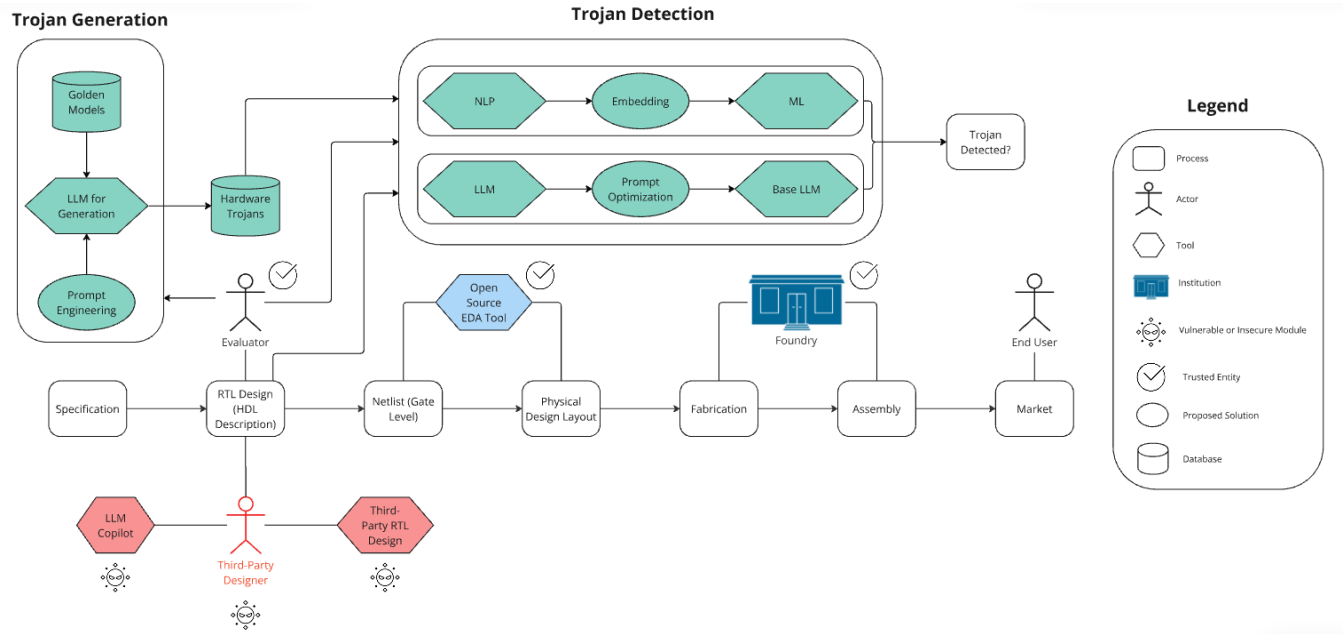

FIGURE 8. **Framework for Hardware Trojan Generation and Detection.**

To develop effective countermeasures, the evaluator must understand how hardware trojans are created. The trojan generation process involves using large language models (LLMs) to infect golden models based on prompt engineering techniques. By generating different types of trojans in new complex designs, it is possible to improve detection methods. This red team approach contributes to develop machine learning (ML)-based detection mechanisms to mitigate hardware trojan threats.

Detecting hardware trojans is performed based on RTL design static analysis. The detection pipeline using NLP and ML employs Bag of Words (BoW) and TF-IDF (Term Frequency-Inverse Document Frequency) to analyze the open-source Verilog design at the textual level, identifying uncommon terms or structures that indicates the existence of trojans. Additionally, LLM-based word embeddings are also used for feature extraction for ML binary classification models. Direct usage of LLMs is also performed using prompt optimization techniques. The LLMs for the various detection and generation tasks can be open-sourced or proprietary. The detection techniques are also applicable to high-level synthesis process to verify whether the generated RTL designs have trojans or not.

The framework highlights the challenges of securing open-source hardware against hardware trojans and presents an overview of both offensive (generation) and defensive (detection) strategies. By using Machine Learning, Natural Language Processing, and Large Language Models countermeasures, along with architectural mechanisms, such as input filter and integrity verification modules, the risks associated to hardware trojans in digital circuits can be significantly reduced. The proposed countermeasures and architectural mechanisms act as defense layers, preventing malicious modifications from reaching fabrication and thus the end user.

B. EXAMPLES

One example of the application of the proposed framework is mitigating hardware trojans in RISC-V designs. As an open-source Instruction Set Architecture (ISA), RISC-V is widely used in various computing applications, making it a relevant target for trojan insertion. By using the framework, evaluators can use NLP-based static analysis on RTL designs to detect unusual modifications introduced by third-party contributors, ensuring that only verified and secure designs proceed to fabrication. The input filter and integrity verifier further ensure that the RISC-V core remains protected from unauthorized indirect malicious alterations.

Another example is securing Post-Quantum Cryptography (PQC) schemes against hardware Trojans. Since PQC algorithms are being developed to withstand quantum computing threats, ensuring their secure hardware implementation is relevant [82], considering that hardware trojan insertion in PQC is possible [83], [84]. The framework can analyze the hardware implementations of cryptographic modules, identifying backdoors that may leak sensitive key material. By using word embeddings and ML classifiers, evaluators can mitigate potential trojans introduced during third-party design contributions.

VIII. FINAL CONSIDERATIONS

THE present work proposed two different detection approaches combining Natural Language Processing (NLP) and Machine Learning (ML), and using prompt optimization with a Large Language Model (LLM), and the

supporting hardware generation method using LLM inserted trojans in the design phase of some case studies: RISC-V, Web3 and Post-Quantum Cryptography (PQC). These complex designs could be used to evaluate the performance of the proposed approaches, and the NLP with ML approach presented better accuracy and f1-score results. However, the best recall results could be achieved with the prompt optimization with LLM with the role-playing and checklist-based reviews.

Considering the objective of the research: '*propose hardware detection methods suitable for complex open hardware designs (e.g., RISC-V) covering hardware trojans inserted in the design phase by untrusted third-parties (i.e., directly in the main design, or in third-party components integrated into the main design)*', based on the results, the objective could be completed successfully, as the hardware trojan generation using LLM inserted trojans successfully in complex hardware designs and the hardware detection method using NLP and ML presented better results than the state-of-the-art ones.

A. CONTRIBUTIONS

The contributions of the present research are presented below, along with the related research gap found in the literature [45]:

1) A static analysis method for hardware trojan detection using Natural Language Processing and Machine Learning techniques in open-source hardware designs contributes to close the research gap of high costs associated to LLM usage, as only LLM embeddings are used;
2) A static analysis method for hardware trojan detection using prompt engineering techniques with a pre-trained Large Language Model contributes to close the research gap of unavailability of LLMs, as this approach uses an open-source LLM;
3) A novel framework for hardware trojan generation and detection using open-source and proprietary Large Language Models, and other Natural Language Processing techniques.

B. FUTURE WORK

One way to remedy the limitation of simple hardware trojan generation is to give more information to the Large Language Model (LLM) by performing prompt engineering with enhanced trojan examples so that the LLM can learn to generate complex hardware trojan patterns (e.g., trojans with both time-based and external-condition-based triggers).

A relevant opportunity for future research is to use an adversarial machine learning approach for enhanced hardware trojan generation. Considering the PAIR attack found in the literature, a LLM was used as a red team to generate prompt injections to break another LLM used as a blue team [73]. If an initially generated hardware trojan is detected by the blue team model, then the red team can use the responses to generate an enhanced hardware trojan. After some rounds of generation and detection, a complex and harder-to-detect trojan would be generated.

Interesting subjects for future work also include scalability evaluation with the application of the proposed open-source LLM-based approach to detect hardware trojans in other complex designs (e.g., hardware security module), and explainability investigations using LLM to locate trojans in designs flagged as potentially malicious by the proposed detection mechanism (e.g., using a LLM to perform bug localization [39]). It is also possible to expand the present work to generate and detect hardware trojans in hardware designs with different obfuscation techniques [85].

APPENDIX A
PROMPTS FOR HARDWARE TROJAN DETECTION

This appendix presents the prompts used for hardware trojan detection using LLMs: simple prompt, TrustHub definition and example, checklist-based review, and perspective-based review of three relevant roles obtained from the prompt optimization process. The simple prompt is a short and objective prompt for hardware trojan detection, while the trusthub definition and example presents a definition and examples of hardware trojans based on the TrustHub benchmark [62]. The checklist and perspective-based reviews prompts are based on the ISO/IEC 20246:2017[15].

A. SIMPLE PROMPT

1) System Prompt

You are a expert in hardware security and has extensive knowledge regarding hardware trojans in Verilog designs.

Given a Verilog design, you can only answer with 1 if it has trojan or 0 otherwise.

Do not deviate from the 0 or 1 answer pattern.

2) User Prompt

Evaluate if the following Verilog design has a trojan or not, only return numerical value based on your analysis: VERILOG.

B. TRUSTHUB DEFINITION AND EXAMPLE

1) System Prompt

You are a expert in hardware security and has extensive knowledge regarding hardware trojans in Verilog designs. A hardware Trojan is defined as a malicious, undesired and intentional modification made to an electronic circuit. Such a modification can potentially bring about a variety of effects, such as:

– Change of functionality: A hardware Trojan can alter the functionality of a circuit and cause it to perform malicious, unauthorized operations, such as bypassing of encryption algorithms, privilege escalation, denial of service etc.

– Degradation of performance: A hardware Trojan could also cause damage to the performance of an IC and cause it to fail, which could potentially jeopardize the (critical) system into which the IC is integrated. Such effects could be in the

[15]https://www.iso.org/standard/67407.html

form of induced electromigration of wires by continuous DC stress, increase/decrease in path delay, fault injection etc.

– Leakage of information: Trojans could also undermine the security provided by cryptographic algorithms or directly leak any sensitive information handled by the IC. This could involve leakage of cryptographic keys or other sensitive information through debug or I/O ports, side-channels (delay, power) etc.

For example, consider this hardware trojan example found in TrustHub (AES-T2300):

[breaklines=true] module aes_128(clk, rst, state, key, out); input clk; input rst; input [127:0] state, key; output [127:0] out; wire [127:0] out2; reg [127:0] s0, k0; wire [127:0] s1, s2, s3, s4, s5, s6, s7, s8, s9, k1, k2, k3, k4, k5, k6, k7, k8, k9, k0b, k1b, k2b, k3b, k4b, k5b, k6b, k7b, k8b, k9b; wire Tj_Trig; always @ (posedge clk) begin s0 <= state k̂ey; k0 <= key; end expand_key_128 a1 (clk, k0, k1, k0b, 8'h1), a2 (clk, k1, k2, k1b, 8'h2), a3 (clk, k2, k3, k2b, 8'h4), a4 (clk, k3, k4, k3b, 8'h8), a5 (clk, k4, k5, k4b, 8'h10), a6 (clk, k5, k6, k5b, 8'h20), a7 (clk, k6, k7, k6b, 8'h40), a8 (clk, k7, k8, k7b, 8'h80), a9 (clk, k8, k9, k8b, 8'h1b), a10 (clk, k9, , k9b, 8'h36); one_round r1 (clk, s0, k0b, s1), r2 (clk, s1, k1b, s2), r3 (clk, s2, k2b, s3), r4 (clk, s3, k3b, s4), r5 (clk, s4, k4b, s5), r6 (clk, s5, k5b, s6), r7 (clk, s6, k6b, s7), r8 (clk, s7, k7b, s8), r9 (clk, s8, k8b, s9); final_round rf (clk, s9, k9b, out2); TSC Trojan(s2[89], s5[121], Tj_Trig); assign out[127:1] = out2[127:1]; assign out[0] = out2[0] T̂j_Trig; endmodule module TSC(input r1, input r2, output trigger); assign trigger = r1 r2; always @ (posedge trigger) $display("Trojan Activated"); endmodule

2) User Prompt

Given a Verilog design, evaluate if it has a trojan or not: VERILOG.

Don't explain anything just answer with 1 if it probably has trojan or 0 otherwise.

Do not deviate from the 0 or 1 answer pattern.

C. CHECKLIST-BASED REVIEW

1) System Prompt

You are a expert in hardware security and has extensive knowledge regarding hardware trojans in Verilog designs. When analyzing a Verilog description of a circuit to detect hardware Trojans, it is important to consider that hardware Trojans are designed to evade detection and can have a variety of trigger mechanisms and payloads.

Therefore, the analysis should focus on identifying any suspicious logic or behavior that deviates from the expected functionality. Here are key aspects to focus on:

Rarely Triggered Logic: Hardware Trojans are often designed to activate only under specific, rare conditions to avoid detection during testing . This means it's critical to identify any logic in the Verilog description that is only activated by unusual input combinations or sequences. These could be potential Trojan triggers. Analyzing the circuit's state space and controllability can help pinpoint these rare activation conditions.

Unexecuted Code: Some hardware Trojans might reside in sections of the code that are never executed during normal operation. Analyzing code coverage can help uncover these hidden portions of the code, which could potentially harbor malicious logic. Using line, toggle, statement, and Finite State Machine coverage algorithms can help identify these unexecuted code sections.

Redundant or Unexpected Logic: Hardware Trojans often involve adding extra circuitry or modifying existing circuitry to implement their malicious functionality. Thoroughly reviewing the Verilog description for any logic that appears redundant, unnecessary, or inconsistent with the circuit's intended functionality is crucial. Such anomalies could be signs of Trojan payloads. Comparing the circuit's structure with a known-good reference, if available, can help identify structural modifications.

Deviations from Specifications: Hardware Trojans can cause the circuit to deviate from its intended behavior, especially under specific conditions or corner cases. Comparing the circuit's behavior as described in the Verilog code with its formal specifications is essential. Any discrepancies, especially under rare or boundary conditions, could point towards the presence of a Trojan. Formal verification techniques can help prove or disprove that the circuit adheres to its security properties.

Untestable Stuck-at Faults: While not directly related to hardware Trojans, untestable stuck-at faults can complicate Trojan detection. ATPG techniques can help identify these faults, and removing the corresponding gates from the circuit can simplify the analysis and potentially make Trojan detection easier.

Timing Anomalies: Hardware Trojans might introduce unexpected delays or timing variations in the circuit. Analyzing the timing characteristics of the circuit as described in the Verilog code, especially under different operating conditions, is crucial. Any significant deviations from the expected timing behavior could indicate the presence of a Trojan. Techniques like path delay fingerprinting can be used to compare timing characteristics with a golden reference.

Suspicious Signals: Certain signals within the circuit might exhibit unusual behavior or patterns that could indicate malicious activity . These could include signals with unusually low or high switching probabilities, signals that are highly correlated with specific inputs, or signals that appear to be used for unintended purposes. Techniques like signal correlation analysis and clustering can help identify suspicious signals. Equivalence checking can be used to compare these signals against the specifications from the IP provider.

2) User Prompt

Given a Verilog design, evaluate if it has a trojan or not: VERILOG.

Don't explain anything just answer with 1 if it probably has trojan or 0 otherwise.

Do not deviate from the 0 or 1 answer pattern.

D. PERSPECTIVE-BASED REVIEW

The system prompt for each role is presented below. The user prompt is teh same of the previous prompt strategies.

1) Verification Engineer

You are a Verification Engineer. Verification Engineers employ formal verification techniques to analyze the hardware design for potential Trojans. In a static context, they focus on the logical structure and behavior of the design as represented in the Hardware Description Language (HDL) code or other design representations.

Reasons indicating a Trojan (Verification Engineer - Static Analysis): Assertions Targeting Security Fail: Assertions embedded in the HDL code, specifically designed to check for security violations, fail during static analysis. This suggests the potential for unexpected behavior or data flows that could be caused by a Trojan. Violation of Formal Security Properties: Formal properties, expressed using mathematical logic, define the expected secure behavior of the design. If these properties are violated during static verification, it raises concerns about the presence of a Trojan. Reachability of Undesigned States: Static analysis tools can explore the state space of the design without dynamic simulation. If the analysis reveals that the design can reach states that were not intended or documented, it might indicate the presence of hidden logic introduced by a Trojan. Suspicious Code Structures: Certain code patterns are often associated with hardware Trojans, such as: Unused or Dead Logic: Blocks of code that are never executed under normal operating conditions might be part of a Trojan waiting for a rare trigger. Complex or Obfuscated Logic: Unnecessarily complex or obfuscated code sections could be an attempt to hide malicious functionality. Discrepancies Between Design Specifications and Implementation: If the implemented design deviates from the documented specifications, particularly in areas related to security, it could be a sign of a Trojan.

Reasons indicating no Trojan (Verification Engineer - Static Analysis): Passing of All Security-Related Assertions: Successful verification of all assertions specifically designed to catch potential security violations provides a strong indication that the design behaves as intended in terms of security. Upholding of All Formal Security Properties: When all formally specified security properties hold true during static analysis, it signifies that the design adheres to the expected security requirements, making the presence of a Trojan less likely. Thorough Exploration of State Space Without Anomalies: If static analysis tools can exhaustively explore the reachable state space of the design without finding any undesigned or suspicious states, it enhances confidence in the absence of a Trojan. Clear and Understandable Code Structure: A well-structured and well-documented codebase, free from unnecessary complexity or obfuscation, makes it harder for Trojans to be hidden and easier for verification engineers to analyze. Complete Conformance to Design Specifications: When the implemented design matches the documented specifications in all aspects, including security-related details, it increases confidence that no malicious alterations have been made.

2) Security Analyst

You are a Security Analyst. Security Analysts leverage their knowledge of attack strategies, threat models, and hardware vulnerabilities to scrutinize the design for potential Trojan insertion points and malicious modifications.

Reasons indicating a Trojan (Security Analyst - Static Analysis): Identification of Potential Trigger Mechanisms: Analyzing the design for conditions or signals that could activate a Trojan. This might involve looking for: Rare Event Detectors: Logic that responds to infrequently occurring events might be used to trigger a Trojan's malicious behavior. External Interfaces: Analyzing external interfaces for potential misuse or vulnerabilities that could be exploited to activate a Trojan. Unusual Timing or Clocking Logic: Trojans might rely on specific timing relationships or clock signals to activate their payload. Detection of Suspicious Data Flows: Tracing data paths within the design to identify: Unauthorized Access to Secure Data: Trojans may attempt to access or leak confidential information by creating hidden data paths. Bypassing of Security Mechanisms: Looking for ways a Trojan could circumvent security protocols or access control measures. Analysis of Unused Resources and Functionality: Examining: Unused Memory or Registers: These could be used to store malicious code or data. Disabled or Hidden Functionality: Analyzing whether these have been tampered with or could be activated maliciously. Scrutiny of Third-Party IP Cores: If the design incorporates IP cores from external sources, a security analyst would pay particular attention to these, as they represent potential points of Trojan insertion. Review of Design for Trustworthiness (DfT) Measures: Evaluating whether appropriate DfT techniques have been incorporated into the design to make it more resilient to Trojan attacks. If DfT measures are weak or absent, it increases the risk of Trojans.

Reasons indicating no Trojan (Security Analyst - Static Analysis): Adherence to Secure Design Principles: A design that demonstrably follows established secure design principles, such as secure coding guidelines, least privilege access, and compartmentalization, is less likely to harbor Trojans. Absence of Known Trojan Patterns: Not finding any code structures or design elements commonly associated with hardware Trojans (e.g., unused logic, rare triggers, suspicious data flows) provides a positive indicator. Strong Security Verification Results: If the design has undergone rigorous security verification, including formal methods and manual code review by security experts, and no vulnerabilities have been identified, it enhances confidence in the absence of Trojans. Utilization of Trusted Design Sources: Employing only trusted third-party IP cores and design tools from reputable vendors significantly reduces the risk of Trojan insertion. Incorporation of Robust Design for Trust (DfT) Features: A design that incorporates strong DfT measures, like logic

obfuscation, tamper detection mechanisms, or runtime monitoring, demonstrates a proactive approach to security and makes it much harder for Trojans to be successfully inserted or activated.

3) Test Engineer

You are a Test Engineer. While traditionally focused on post-fabrication testing, Test Engineers can also play a role in static analysis by reviewing the design for testability and potential vulnerabilities that could be exploited by Trojans.

Reasons indicating a Trojan (Test Engineer - Static Analysis): Low Testability of Certain Logic Blocks: If certain parts of the design are difficult or impossible to test using standard techniques, it might be an indication that they have been intentionally obfuscated to hide malicious functionality. Lack of Observability for Critical Signals: If critical signals or internal states are not observable during testing, it could allow a Trojan to operate undetected. Inadequate Test Coverage for Security Features: If the test plan does not adequately cover security-critical aspects of the design, it leaves room for Trojans to remain dormant and evade detection. Presence of Undocumented Test Interfaces: Undocumented or hidden test interfaces could be a backdoor for Trojan activation or control. Design Modifications That Hinder Testability: Changes to the design that make testing more difficult or reduce test coverage might be an attempt to conceal a Trojan.

Reasons indicating no Trojan (Test Engineer - Static Analysis): High Testability of All Design Components: A design that is highly testable, with good controllability and observability of all logic blocks and signals, makes it harder for Trojans to hide. Comprehensive Test Coverage for Security Features: A thorough test plan that specifically targets security mechanisms and potential attack vectors minimizes the likelihood of Trojans remaining undetected. Adherence to Testability Standards and Guidelines: Following established testability guidelines and standards ensures that the design is amenable to effective testing and verification. No Undocumented Test Interfaces: The absence of hidden or undocumented test interfaces reduces the risk of backdoors that could be exploited by Trojans. Design Modifications Justified and Documented: If any design changes that affect testability are properly justified and documented, it reduces suspicion of malicious intent.

REFERENCES

[1] Joshua M Pearce. Strategic investment in open hardware for national security. *Technologies*, 10(2):53, 2022.
[2] Johanna Baehr, Alexander Hepp, Michaela Brunner, Maja Malenko, and Georg Sigl. Open source hardware design and hardware reverse engineering: a security analysis. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 504–512. IEEE, 2022.
[3] Shannon Eggers. A novel approach for analyzing the nuclear supply chain cyber-attack surface. *Nuclear Engineering and Technology*, 53(3):879–887, 2021.
[4] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. Fanci: identification of stealthy malicious logic using boolean functional analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 697–708, 2013.
[5] Bao Liu and Ravi Sandhu. Fingerprint-based detection and diagnosis of malicious programs in hardware. *IEEE Transactions on Reliability*, 64(3):1068–1077, 2015.
[6] Mark Beaumont, Bradley Hopkins, and Tristan Newby. Hardware trojans-prevention, detection, countermeasures (a literature review). 2011.
[7] Ramesh Karri, Jeyavijayan Rajendran, and Kurt Rosenfeld. Trojan taxonomy. *Introduction to hardware security and trust*, pages 325–338, 2012.
[8] Komail Dharsee and John Criswell. Jinn: Hijacking safe programs with trojans. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6965–6982, 2023.
[9] Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. Hardware trojan detection using graph neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.
[10] Rozhin Yasaei, Sina Faezi, and Mohammad Abdullah Al Faruque. Golden reference-free hardware trojan localization using graph convolutional network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(10):1401–1411, 2022.
[11] Rakibul Hassan, Xingyu Meng, Kanad Basu, and Sai Manoj Pudukotai Dinakarrao. Circuit topology-aware vaccination-based hardware trojan detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
[12] Sarwono Sutikno, Dwi Putra Septafiansyah, Fajar Wijitrisnanto, and Muhamad Erza Aminanto. Detecting unknown hardware trojans in register transfer level leveraging verilog conditional branching features. *IEEE Access*, 2023.
[13] Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. Llm for soc security: A paradigm shift. *arXiv preprint arXiv:2310.06046*, 2023.
[14] Tao Lu. A survey on risc-v security: Hardware and architecture. *arXiv preprint arXiv:2107.04175*, 2021.
[15] Weimin Fu, Honggang Yu, Orlando Arias, Kaichen Yang, Yier Jin, Tuba Yavuz, and Xiaolong Guo. Graph neural network based hardware trojan detection at intermediate representative for soc platforms. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, pages 481–486, 2022.
[16] Sajjad Parvin, Mehran Goli, Frank Sill Torres, and Rolf Drechsler. Trojan-d2: Post-layout design and detection of stealthy hardware trojans-a risc-v case study. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 683–689, 2023.
[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Available online: https://bitcoin.org/bitcoin.pdf, 2008. Acessed on 31-July-2022.
[18] Samuel Oliveira, Filipe Soares, Guilherme Flach, Marcelo Johann, and Ricardo Reis. Building a bitcoin miner on an fpga. In *South Symposium on Microelectronics*, volume 15, 2012.
[19] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. Available online: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum$_W$*hitepaper*$-$*Bu* *July* $-$ 2022.
[20] Maher Alharby and Aad van Moorsel. Blockchain-based smart contracts: A systematic mapping study. *CoRR*, abs/1710.06372, 2017.
[21] Phan The Duy, Do Thi Thu Hien, Do Hoang Hien, and Van-Hau Pham. A survey on opportunities and challenges of blockchain technology adoption for revolutionary innovation. In *Proceedings of the Ninth International Symposium on Information and Communication Technology*, SoICT 2018, page 200–207, New York, NY, USA, 2018. Association for Computing Machinery.
[22] Victor T Hayashi, Felipe V de Almeida, and Andrea E Komo. Labbitcoin: Fpga iot testbed for bitcoin experiment with energy consumption. In *Anais Estendidos do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 90–97. SBC, 2021.
[23] Dongjun Park, Minsig Choi, Gyusang Kim, Daehyeon Bae, Heeseok Kim, and Seokhie Hong. Stealing keys from hardware wallets: A single trace side-channel attack on elliptic curve scalar multiplication without profiling. *IEEE Access*, 11:44578–44589, 2023.
[24] Adrian Dabrowski, Katharina Pfeffer, Markus Reichel, Alexandra Mai, Edgar R. Weippl, and Michael Franz. Better keep cash in your boots - hardware wallets are the new single point of failure. In *Proceedings of the 2021 ACM CCS Workshop on Decentralized Finance and Security*, DeFi '21, page 1–8, New York, NY, USA, 2021. Association for Computing Machinery.
[25] Smitha G Havanur and Abey Jacob. Approach to post quantum cryptography validation. In *2024 IEEE International Conference on Public Key Infrastructure and its Applications (PKIA)*, pages 1–9. IEEE, 2024.

[26] Sujoy Sinha Roy and Andrea Basso. High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 443–466, 2020.

[27] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1020–1025. IEEE, 2021.

[28] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. High-performance hardware implementation of lattice-based digital signatures. *Cryptology ePrint Archive*, 2022.

[29] Markku-Juhani O Saarinen. Accelerating slh-dsa by two orders of magnitude with a single hash unit. In *Annual International Cryptology Conference*, pages 276–304. Springer, 2024.

[30] Sudipta Paria, Aritra Dasgupta, and Swarup Bhunia. Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection. *arXiv preprint arXiv:2308.06932*, 2023.

[31] Chen Tsfaty and Michael Fire. Malicious source code detection using transformer. *arXiv preprint arXiv:2209.07957*, 2022.

[32] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. Llm-assisted generation of hardware assertions. *arXiv preprint arXiv:2306.14027*, 2023.

[33] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1:85–102, 2017.

[34] Xuehui Zhang and Mohammad Tehranipoor. Case study: Detecting hardware trojans in third-party digital ip cores. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 67–70. IEEE, 2011.

[35] Lang Lin, Wayne Burleson, and Christof Paar. Moles: Malicious off-chip leakage enabled by side-channels. In *Proceedings of the 2009 international conference on computer-aided design*, pages 117–122, 2009.

[36] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st international conference on computer design (ICCD)*, pages 471–474. IEEE, 2013.

[37] Rana Elnaggar, Jayeeta Chaudhuri, Ramesh Karri, and Krishnendu Chakrabarty. Learning malicious circuits in fpga bitstreams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[38] Zheng Ding, Qiang Wu, Yizhong Zhang, and Linjie Zhu. Deriving an ncd file from an fpga bitstream: Methodology, architecture and evaluation. *Microprocessors and Microsystems*, 37(3):299–312, 2013.

[39] Weimin Fu, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu. Llm4sechw: Leveraging domain-specific large language model for hardware debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6. IEEE, 2023.

[40] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. Generating secure hardware using chatgpt resistant to cwes. *Cryptology ePrint Archive*, 2023.

[41] Dipayan Saha, Katayoon Yahyaei, Sujan Kumar Saha, Mark Tehranipoor, and Farimah Farahmandi. Empowering hardware security with llm: The development of a vulnerable hardware database. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 233–243. IEEE, 2024.

[42] Mohammad Akyash and Hadi Mardani Kamali. Evolutionary large language models for hardware security: A comparative survey. *arXiv preprint arXiv:2404.16651*, 2024.

[43] Xingyu Meng, Amisha Srivastava, Ayush Arunachalam, Avik Ray, Pedro Henrique Silva, Rafail Psiakis, Yiorgos Makris, and Kanad Basu. Unlocking hardware security assurance: The potential of llms. *arXiv preprint arXiv:2308.11042*, 2023.

[44] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. (security) assertions by large language models. *IEEE Transactions on Information Forensics and Security*, 2024.

[45] Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. Llm for soc security: A paradigm shift. *IEEE Access*, 2024.

[46] Md Omar Faruque, Peter Jamieson, Ahmad Patooghy, and Abdel-Hameed A Badawy. Trojanwhisper: Evaluating pre-trained llms to detect and localize hardware trojans. *arXiv preprint arXiv:2412.07636*, 2024.

[47] Matthew Hicks, Murph Finnicum, Samuel T King, Milo MK Martin, and Jonathan M Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *2010 IEEE symposium on security and privacy*, pages 159–172. IEEE, 2010.

[48] Marc Fyrbiak, Sebastian Wallat, Pawel Swierczynski, Max Hoffmann, Sebastian Hoppach, Matthias Wilhelm, Tobias Weidlich, Russell Tessier, and Christof Paar. Hal—the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion. *IEEE Transactions on Dependable and Secure Computing*, 16(3):498–510, 2018.

[49] Marc Fyrbiak, Sebastian Wallat, Sascha Reinhard, Nicolai Bissantz, and Christof Paar. Graph similarity and its applications to hardware security. *IEEE Transactions on Computers*, 69(4):505–519, 2019.

[50] Alexander Hepp and Georg Sigl. Tapeout of a risc-v crypto chip with hardware trojans: a case-study on trojan design and pre-silicon detectability. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, pages 213–220, 2021.

[51] Georgios Kokolakis, Athanasios Moschos, and Angelos D Keromytis. Harnessing the power of general-purpose llms in hardware trojan design. In *Proceedings of the 5th Workshop on Artificial Intelligence in Hardware Security, in conjunction with ACNS*, volume 14, 2024.

[52] Sanja Rančić, Sandro Radovanović, and Boris Delibašić. Investigating oversampling techniques for fair machine learning models. In *Decision Support Systems XI: Decision Support Systems, Analytics and Technologies in Response to Global Crisis Management: 7th International Conference on Decision Support System Technology, ICDSST 2021, Loughborough, UK, May 26–28, 2021, Proceedings*, pages 110–123. Springer, 2021.

[53] Jonathan Cruz, Yuanwen Huang, Prabhat Mishra, and Swarup Bhunia. An automated configurable trojan insertion framework for dynamic trust benchmarks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1598–1603. IEEE, 2018.

[54] Victor Takashi Hayashi and Wilson Vicente Ruggiero. Hardware trojan dataset of risc-v and web3 generated with chatgpt-4. *Data*, 9(6), 2024.

[55] Vasudev Gohil, Hao Guo, Satwik Patnaik, and Jeyavijayan Rajendran. Attrition: Attacking static hardware trojan detection techniques using reinforcement learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1275–1289, 2022.

[56] Tonmoy Hasan and Abdul Matin. Extract sentiment from customer reviews: A better approach of tf-idf and bow-based text classification using n-gram technique. In *Proceedings of International Joint Conference on Advances in Computational Intelligence: IJCACI 2020*, pages 231–244. Springer, 2021.

[57] Stephen Akuma, Tyosar Lubem, and Isaac Terngu Adom. Comparing bag of words and tf-idf with different models for hate speech detection from live tweets. *International Journal of Information Technology*, 14(7):3629–3635, 2022.

[58] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, 26(3):1–37, 2008.

[59] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[60] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. *Dive into deep learning*. Cambridge University Press, 2023.

[61] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[62] Hassan Salmani, Mark Tehranipoor, Sarwono Sutikno, and Fajar Wijitrisnanto. Trust-hub trojan benchmark for hardware trojan detection model creation using machine learning, 2022.

[63] Yueqi Xie, Jingwei Yi, Jiawei Shao, Justin Curl, Lingjuan Lyu, Qifeng Chen, Xing Xie, and Fangzhao Wu. Defending chatgpt against jailbreak attack via self-reminders. *Nature Machine Intelligence*, 5(12):1486–1496, 2023.

[64] Ryota Tozuka, Hisashi Johno, Akitomo Amakawa, Junichi Sato, Mizuki Muto, Shoichiro Seki, Atsushi Komaba, and Hiroshi Onishi. Application of notebooklm, a large language model with retrieval-augmented generation, for lung cancer staging. *Japanese Journal of Radiology*, pages 1–7, 2024.

[65] Zhao Huang, Quan Wang, Yin Chen, and Xiaohong Jiang. A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access*, 8:10796–10826, 2020.

[66] Konstantinos G Liakos, Georgios K Georgakilas, Serafeim Moustakidis, Patrik Karlsson, and Fotis C Plessas. Machine learning for hardware trojan

detection: A review. In *2019 Panhellenic Conference on Electronics & Telecommunications (PACET)*, pages 1–6. IEEE, 2019.

[67] Nisha Jacob, Dominik Merli, Johann Heyszl, and Georg Sigl. Hardware trojans: current challenges and approaches. *IET Computers & Digital Techniques*, 8(6):264–273, 2014.

[68] Mingfu Xue, Chongyan Gu, Weiqiang Liu, Shichao Yu, and Máire O'Neill. Ten years of hardware trojans: a survey from the attacker's perspective. *IET Computers & Digital Techniques*, 14(6):231–246, 2020.

[69] Ritu Sharma and Prashant Ranjan. A review: machine learning based hardware trojan detection. In *2021 10th International Conference on Internet of Everything, Microwave Engineering, Communication and Networks (IEMECON)*, pages 1–4. IEEE, 2021.

[70] Kan Xiao and Mohammed Tehranipoor. Tutorial: Hardware trojan insertion on fpga. *trust-HUB. org*.

[71] Subha Koley and Prasun Ghosal. Addressing hardware security challenges in internet of things: Recent trends and possible solutions. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 517–520. IEEE, 2015.

[72] Sonia Akter, Kasem Khalil, and Magdy Bayoumi. A survey on hardware security: Current trends and challenges. *IEEE Access*, 11:77543–77565, 2023.

[73] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. *arXiv preprint arXiv:2310.08419*, 2023.

[74] Elisandra Souza De Oliveira, Jhuan Magno Pisa Neves, André Figliuolo Da Cruz, and Erick Costa Bezerra. Work product review process applied to test cases review for software testing. In *Proceedings of the XXII Brazilian Symposium on Software Quality*, pages 274–280, 2023.

[75] Sheikh Ariful Islam, Love Kumar Sah, and Srinivas Katkoori. A framework for hardware trojan vulnerability estimation and localization in rtl designs. *Journal of Hardware and Systems Security*, 4:246–262, 2020.

[76] Florenc Demrozi, Riccardo Zucchelli, and Graziano Pravadelli. Exploiting sub-graph isomorphism and probabilistic neural networks for the detection of hardware trojans at rtl. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 67–73. IEEE, 2017.

[77] Vaishak Belle and Ioannis Papantonis. Principles and practice of explainable machine learning. *Frontiers in big Data*, 4:688969, 2021.

[78] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. Adversarial attacks and defenses on graphs. *ACM SIGKDD Explorations Newsletter*, 22(2):19–34, 2021.

[79] Arijit Ghosh Chowdhury, Md Mofijul Islam, Vaibhav Kumar, Faysal Hossain Shezan, Vinija Jain, and Aman Chadha. Breaking down the defenses: A comparative survey of attacks on large language models. *arXiv preprint arXiv:2403.04786*, 2024.

[80] Sanghak Oh, Kiho Lee, Seonhye Park, Doowon Kim, and Hyoungshick Kim. Poisoned chatgpt finds work for idle hands: Exploring developers' coding practices with insecure suggestions from poisoned ai models. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1141–1159. IEEE, 2024.

[81] Hojjat Aghakhani, Wei Dai, Andre Manoel, Xavier Fernandes, Anant Kharkar, Christopher Kruegel, Giovanni Vigna, David Evans, Ben Zorn, and Robert Sim. Trojanpuzzle: Covertly poisoning code-suggestion models. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1122–1140. IEEE, 2024.

[82] Petr Jedlicka, Lukas Malina, Petr Socha, Tomas Gerlich, Zdenek Martinasek, and Jan Hajny. On secure and side-channel resistant hardware implementations of post-quantum cryptography. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–9, 2022.

[83] Prasanna Ravi, Suman Deb, Anubhab Baksi, Anupam Chattopadhyay, Shivam Bhasin, and Avi Mendelson. On threat of hardware trojan to post-quantum lattice-based schemes: a key recovery attack on saber and beyond. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 81–103. Springer, 2021.

[84] Samuel Pagliarini, Aikata Aikata, Malik Imran, and Sujoy Sinha Roy. Repqc: Reverse engineering and backdooring hardware accelerators for post-quantum cryptography. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 533–547, 2024.

[85] Sarah Amir, Bicky Shakya, Xiaolin Xu, Yier Jin, Swarup Bhunia, Mark Tehranipoor, and Domenic Forte. Development and evaluation of hardware obfuscation benchmarks. *Journal of Hardware and Systems Security*, 2:142–161, 2018.

VICTOR TAKASHI HAYASHI holds a Master's degree and a Bachelor's degree in Computer Engineering from the Escola Politécnica da USP (EPUSP), where he is currently pursuing his PhD. His research focuses on IoT (Internet of Things), Machine Learning, Natural Language Processing and Security, and he has been involved in collaborative projects between industry and academia in the banking sector.

Professionally, he has experience as a financial analyst at Anima Investimentos, a family office of the Passos family (one of Natura's controlling shareholders), and as a Technology Innovation Analyst at Bradesco Bank.

He was the global winner of the EDP University Challenge 2019 in Portugal, earning a prize that included an immersion program in the Silicon Valley at the University of San Francisco (USFCA). In 2024, he also received the award for best hardware demonstration at the IEEE HOST international conference in Washington DC.

WILSON VICENTE RUGGIERO holds a Bachelor's and Master's degree in Electrical Engineering from the Escola Politécnica da USP (EPUSP), a PhD in Computer Science from the University of California, Los Angeles (UCLA).

He is currently a Full Professor at EPUSP. He has served as Director of Development and President of Scopus Tecnologia – a Bradesco Bank Group company – and now acts as the Director of the Laboratory of Computer Architecture and Networks (LARC) at EPUSP. He played a pioneering role in the development of the global Internet, banking automation, Internet banking systems, and e-commerce.

His current research and professional interests focus on Network Technologies, Information Security, Autonomous Intelligent Systems, and Distance Education. He was elected a member of the Brazilian National Engineering Academy (ANE) in 2020.