Contemporary Mathematics
https://ojs.wiserpub.com/index.php/CM/

Research Article

SPECTRE: Computational Methods in Natural Language Processing for Automated Hardware Trojan Insertion Using Large Language Models

Moneer Alshaikh[1], Rashid Amin[2*], Sajid Mehmood[2], Faisal S. Alsubaei[1]

[1] Department of Cybersecurity, College of Computer Science and Engineering, University of Jeddah, Jeddah, 23218, Saudi Arabia
[2] Department of Computer Science, University of Engineering and Technology, Taxila, 47050, Pakistan
E-mail: rashid.amin@uettaxila.edu.pk

Received: 10 August 2025; **Revised:** 22 September 2025; **Accepted:** 28 September 2025

Abstract: Experts Stealthy Processor Exploitation and Concealment Through Reconfigurable Elements (SPECTRE) is the new framework proposed in this paper to use the computational methods of Natural Language Processing (NLP) to automate the addition of Hardware Trojans (HTs) to the complex hardware design. SPECTRE takes advantage of Large Language Models (LLMs) like Generative Pre-trained Transformer (GPT)-4, Gemini-1.5-pro and LLaMA-3-70B to synthesize HTs with little to no human input by using sophisticated prompting methods like role-based prompting, reflexive validation prompting, and contextual Trojan prompting to analyze Hardware Description Language (HDL) codebases and expose vulnerabilities. Such a methodology alleviates the limitations of the more traditional machine learning-based automation that tends to require large data and extensive training times by including NLP-based code generation and an inference engine that can dynamically scale to non-homogeneous hardware platforms, including Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs). When tested on benchmark hardware systems like Static Random-Access Memory (SRAM), Advanced Encryption Standard (AES-128), and Universal Asynchronous Receiver-Transmitter (UART), SPECTRE shows greater performance with GPT-4 having an 88.88% success rate in generating viable and stealthy HTs that cannot be detected by current state-of-the-art Machine Learning (ML)-based tools like hw2vec. The mathematical and computational basis of the framework, premised on few-shot learning, adversarial prompting, and iterative validation algorithms, shows the dual-use potential of NLP models in the domain of hardware security, which poses the potential to exploit vulnerabilities within a short time but also demands adequate strategies to curb vulnerability exploitation by artificial intelligence generation.

***Keywords*:** natural language processing, large language model, prompt engineering, computational method, code generation, hardware security, stealthy insertion

MSC: 68T50, 68M25, 94A60

1. Introduction

Hardware Trojans (HTs) represent unauthorized malevolent changes in hardware designs that modify operational capabilities degrades system efficiency and expose confidential information in devastating attacks. The electronics and

DOI: https://doi.org/10.37256/cm.7120268211

semiconductor industry face rising concerns about Trojans because their supply chains have globalized while designers repeat patterns and 3P-IP cores has become pervasive [1, 2]. Growing hardware complexity coupled with networked systems has caused an escalation of hardware Trojan insertion points throughout the product development process, making HTs a fundamental security threat for hardware systems [3]. The process of embedding Hardware Trojans demands extensive hardware architecture expertise and technical proficiency when working with complex Hardware Description Language (HDL) codebases since it has long been handled manually and laboriously. The human-designed HTs suffer from inherent limitations because designers introduce biases during the design phase, which results in detectable HTs that security tools can reveal [4]. The growth of hardware designs in complexity and size renders manual insertion of HT impractical, making this process to need sophisticated techniques at automation.

Researchers have created partially automated and fully automated hardware Trojan insertion systems that use Machine Learning (ML) and algorithmic methods to enhance operation. Such tools help decrease manual work while reducing biased decisions and enhance the speed of inserting hardware Trojan bugs [5]. Trust-Hub repository contains manually inserted HT benchmarks that serve as widely used benchmarks for research purposes [6]. The human involvement in manual insertion enables human biases to enter the system, which restricts the range of scenarios for detection tools to detect effectively. The success of automation tools to insert HTiefs varies between Tool for Automated Insertion of Trojans (TAINT) [7], Trojan Insertion Tool (TRIT) [8], MIMIC [9], ATTRITION [10], TrojanPlayground [11], Design-Time Trojan insertion at Register-Transfer Level (DTjRTL) [12], TrojanForge [13], and FEINT [14] and the overall capability of this group is still the focus of research. The advancements in hardware security tools face challenges because training takes a long time and the tools lack open-source access while also providing restricted usage for broader research due to limited hardware design adaptability.

The TAINT tool provides automated support for inserting HT into Field-Programmable Gate Array (FPGA) designs, although users must supply significant guidance to the system and present the possibility of human biases. [7]. While TRIT [8] operates as an automatic Trojan insertion solution for gate-level netlists, it demands user involvement to set up and configure the system. The MIMIC [9] system teaches itself about Trojan patterns through machine learning yet struggles because it needs massive training information together with expensive computational power, which reduces its application scope across diverse scenarios. ATTRITION [10] reaches its main objectives through rare net Trojan insertion yet lacks open-source availability, which hinders its widespread deployment. The HT insertion feature of TrojanPlayground [11] relies on RL, with the need for extensive training until autonomous HT insertion becomes possible. The DTjRTL [12] system carries out automated hardware Trojan insertion during the Register-Transfer Level (RTL) stages, though it requires fixed configuration parameters. The TrojanForge [13] platform uses adversarial learning together with RL and GANs to produce imperceptible HTs while facing high quantitative demands. FEINT [14] provides automated HT insertion for FPGA designs, yet its capability to work in diverse FPGA situations remains restricted.

Current tools in the HT generation face three substantial challenges because they need lengthy learning periods work only on specific hardware environments, and are not freely available to researchers. The limitations these obstacles create make the tools less effective and more difficult to use for extensive research and development purposes [15]. Our solution Stealthy Processor Exploitation and Concealment Through Reconfigurable Elements (SPECTRE), brings together Large Language Models to automate hardware Trojan inclusion into complex designs. The hardware security field has received substantial progress through the introduction of SPECTRE. SPECTRE operates on all software platforms, independent of design structure and therefore attacks both Application-Specific Integrated Circuit (ASIC) and FPGA systems across multiple hardware architectures. The platform SPECTRE uses Large Language Models (LLMs) for real time use of HDL source code to identify weak points permitting the automatic insertion of HTs with minimal human intervention. The device is an effective resource to the collaborators and opponent forces, which demonstrates the critical requirement for sophisticated protection systems in hardware security.

Claims from Shakya et al. [16] inspire our threat model Figure 1, which means to protect the security aspects of the widespread and outsourced System-on-Chip (SoC) development pipeline through which SoC development is distributed across different companies. The paper examines a situation where an RTL designer operating as part of an external IP (3PIP) vendor gives a transformed IP core to an SoC integrator who handles offshore development of SoC integration. The threat model core depends on the SoC integrator being the adversary since tight project deadlines force them to

work with limited time to understand the IP core's internal structure [17]. The SPECTRE framework supports potential attackers by using LLMs to overcome standard knowledge limitations within the HDL code analysis process, thus enabling fast identification of vulnerabilities and deployment of Hardware Trojans before Computer-Aided Design (CAD) tool operations start for fabrication at the foundry. The framework enables developers to build secretive HTs that survive both testing periods before and after fabrication to activate themselves after deployment. Modifications made by attackers stop on the RTL level, because the SoC integrator and foundry operates on other degrees of trustfulness.

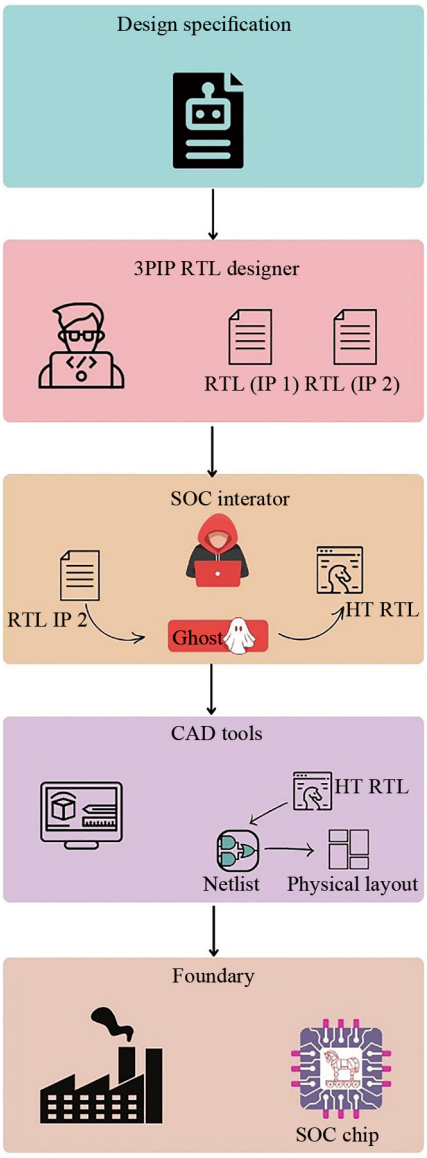

Figure 1. The model of assumed threat

Our paper presents several fundamental contributions to this research. This work presents SPECTRE as an innovative automated attack framework that uses LLMs to insert hardware backdoors within complex RTL designs. The SPECTRE framework operates on any architecture and on any design format, allowing it to attack various hardware platforms. Our research evaluates the HT insertion capabilities of three leading LLMs GPT-4, Gemini-1.5-pro, and LLaMA3, while

as they generate HTs for different hardware designs in order to explain LLM-HT security risks. We examine both the operational performance and functional attributes as well as detection limitations of each LLM.

During HT insertion, and we assess their ability to avoid modern ML- based HT detection software. The research contributes 14 functional HT benchmarks generated by SPECTRE, which enhances the available HT-infected circuits for research purposes.

To conclude, the following key contributions were made in this work: (i) we introduce a new framework, SPECTRE, which is the first to study the problem of Trojan stealthiness of large language models and insertion of Trojan hardware Trojan at the Registertransfer Level (RTL) in detail: (ii) we propose three novel methods to prompt the continuity models to generate realistic Trojan payloads and activators, including Role Based Prompting (RBP), Role Partition Prompting (RVP) and Chain of Thoughts Prompting (CTP); (ii) we prototype and evaluate Collectively, these provision confirm the standing of SPECTRE as a baseline of offensive capability and defensive requirement of the offensive hardware security that is driven by non-parallel adeptive research on a bottleneck of an emergent directions in security research studies.

The paper proceeds to organise the rest around yen thus as follows; Section 2 provides a summary of the existing literature regarding the field of hardware Trojans insertion and detection as well as illustrates the failings of the current methodology in the paper. Part 3 shows the architecture of the proposed SPECTRE framework, prompt engineering strategies, and architecture of LLM. Section 4 provides the configuration of the experiment, benchmark circuit and measurement procedure. Section 5 presents the results of our experiments regarding quantitative performance metrics, analysis of scalability and detection issues. Section 6 discusses the implication generally, the problem of dual use, and the defence mechanisms that could potentially be implemented. And, finally but not the least is the conclusion that is a perspective of another positive piece of research that may be undertaken.

2. Related work

New techniques for Hardware Trojan (HT) detection and prevention within hardware security have improved substantially during the recent period. The traditional methods of Hardware Trojan insertion depend on experts who manipulate Hardware Description Language (HDL) codebases while having extensive knowledge of hardware design. Such approaches deliver satisfactory results but demand an extensive amount of time and are prone to human errors in the process, which creates barriers to systemwide deployment. The research community adopted Machine Learning (ML) techniques to develop automated systems for Hardware Trojan insertion and detection because of existing difficulties. Trust-Hub [18] maintains a wide collection of manually added HT benchmarks that serve as benchmarking tools for detection systems evaluation. These benchmarks' manual character introduces human subjective elements which reduce their capability to develop reliable detection systems.

Research over the last few years has demonstrated the effectiveness of ML-based methods to automate the process of HT insertion. The system MIMIC [9] along with ATTRITION [10] uses machine learning algorithms to produce Trojans through analysis of available Trojan examples. Such tools decrease human involvement yet need substantial training data together with considerable com- putational power, which restricts their ability to work across different hardware platforms. Reinforcement Learning (RL) and adversarial learning techniques used in the TrojanPlayground framework [11] together with DTjRTL framework [12] enabled automatic insertion of HT. The innovative techniques face challenges because they need extended training cycles and experience difficulties with adjusting to new hardware testing conditions. The accessibility of these tools becomes limited because most are not available as open-source software.

Integrating Large Language Models into hardware security systems marks an essential change for this discipline. The current generation of LLMs which includes the models GPT-4 [19], Gemini-1.5-pro [20] and LLaMA3-70B [21] shows exceptional performance in natural language processing along with code generation applications. Research investigates how LLMs can be utilized to develop and validate hardware systems. ChipGPT [22] uses LLMs to produce optimized logic designs from natural language specifications, and Autochip [23] uses LLMs for debugging HDL code. No existing work has shown how LLMs detect vulnerabilities while remediating them [24] and generating secure hardware elements

[25] plus performing verification [26] in hardware security applications. The use of LLMs for offensive security tasks involving hardware Trojan insertion has not received thorough research attention.

The development illustrated in Figure 2 shows that HTs created through LLM technology pose an imminent danger for hardware protection systems. The sophisticated nature of HTs generated through LLMs makes them difficult to detect throughout their execution, rather than traditional HTs whose focused and narrow targets can often be easily recognized. The research by Saha et al. [27] showed LLMs can combine vulnerabilities with hardware designs, which necessitates the development of better detection techniques. The models applied by Ahmad et al. [26] demonstrated a dual-use capability by using LLMs to assist vulnerability fix implementation. The SPECTRE framework implements automated HT insertion through LLM capabilities to produce operational hidden backdoors that need limited human oversight to operate.

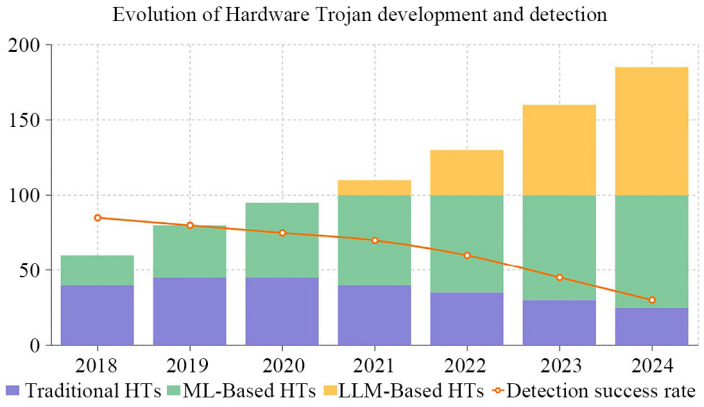

Figure 2. Evolution of hardware trojan development and detection

The problems posed by HTs created through LLM systems become worse because currently available detection methods fail to keep up with this threat. Traditional HT detection tools fail to identify stealthy LLM-generated HTs since their detection methods, such as side-channel analysis and formal verification, provide no effective results. The detection tools based on ML principles such as hw2vec [28] proved beneficial for detecting conventional HTs but demonstrated limited capacity to detect HTs that originate from LLMs as per the research findings. New detection and prevention systems must be developed immediately because LLM-generated HTs represent an escalating security threat.

Hardware security that incorporates LLMs presents both beneficial prospects and complex obstacles to handle. The generation of HTs through LLM technology creates a serious risk for hardware system integrity, even though these systems have potential benefits for enhancing hardware design and security. The SPECTRE framework shows that hardware security needs aggressive protective measures that integrate modern detection methods to stop LLM-generated HTs. Research efforts in the future must center on building resistant hardware systems that defend against current and predicted security threats to fundamental infrastructure within the period of artificial intelligence.

3. Materials and methods

The automated framework SPECTRE (Stealthy Processor Exploitation and Concealment Through Reconfigurable Elements) utilizes large language models to develop an instrument for inserting Hardware Trojan (HT) elements that match key security specifications. The architectural structure of SPECTRE contains two primary elements, which are known as Prompt Engineering and LLM Inference as depicted in Figure 3. The LLM prompt engineering component serves to develop specific instructions for detecting ideal HT placement locations, and the second component makes use of the LLM to evaluate HDL code and develop syntheses-ready functionally valid HTs. SPECTRE combines its architecture's two components to create an effective system that supports automatic scalable platform-independent HT insertion but

raises security concerns about LLM-generated hardware components. The subsequent sections delve into each element of analysis.

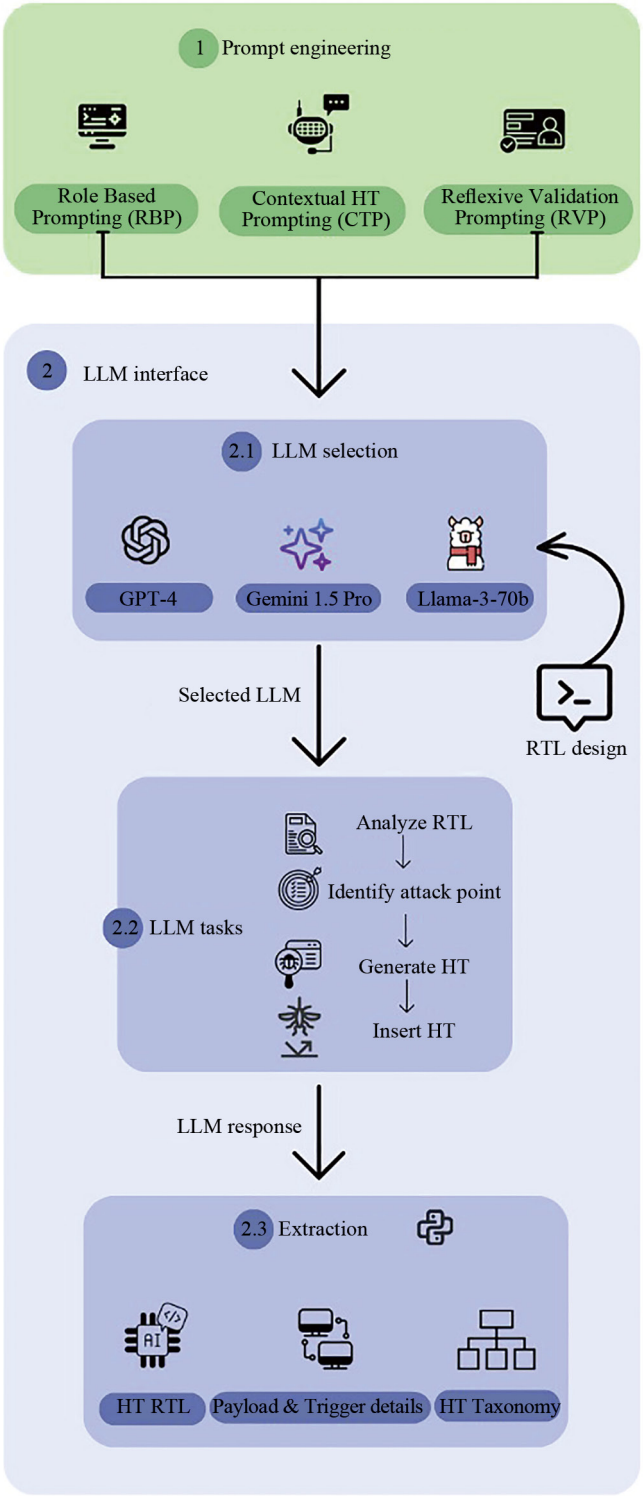

Figure 3. SPECTRE framework essential elements

3.1 *Prompt engineering*

Through the SPECTRE framework, the Prompt Engineering module makes it possible for Large Language Models to create Hardware Trojans (HTs) that incorporate themselves into hardware designs. The LLM module prepares itself through this system to create HTs that fulfill functional requirements while maintaining undetectable characteristics. Figure 3 shows that Prompt Engineering utilizes three sophisticated prompting techniques referred to as Role-Based Prompting (RBP), Reflexive Validation Prompting (RVP), and Contextual Trojan Prompting (CTP) in the left part. The LLM achieves task determination and high-quality HT creation plus functional stealth maintenance thanks to the combination of these three strategic controlling techniques throughout the hardware design period. The different prompting techniques provide specific functions to generate secure and imperceptible HTs that solve today's hardware security problems.

Role-Based Prompting (RBP) functions as the base method that provides the LLM with the necessary capabilities to execute HT insertion procedures successfully. When using this framework the LLM receives a particular assigned role, which gives it the required expertise and contextual understanding needed to complete tasks successfully. When required to develop HTs, the LLM operates under the guidance of hardware security experts who specialize in Hardware Trojan creation. Through the assigned role the LLM gains access to domain-specific expertise that enables it to develop authentic and operationally compliant hardware trojans for target hardware systems. An RBP start order would ask the LLM to operate under the role of a hardware security expert who has installed multiple hardware Trojans and followed specific instructions. The provided hardware design requires inspection for stealthy hardware Trojan implementation using operational specifications as per the guidelines. Through expert perspective adoption, the LLM develops comprehensive knowledge about its assigned task, which enables it to produce secure HT designs across diverse operational platforms. This method enables the LLM to function as a dependable automated system for implementing HT by generating suitable HT results that need minimal human supervision during operation.

The second method, Reflexive Validation Prompting (RVP), enhances the LLM-generated HTs by integrating a built-in validation process that assures HT quality. The LLM follows this technique to provide a complete analysis of its outputs, which verifies their function along with hiding capabilities and synthetic traits. The self-evaluation framework of RVP uses a step-by-step procedure adopted from Shinn et al. [29] to guide the LLM through validation processes. An LLM could receive two commands to achieve validation: "Validate all instructions it receives" or "Verify that implemented Trojans satisfy functionality, operation stealth and synthetic requirements." The set prompts drive the LLM to extensively review its work to create HTs that pass all pre-and post-fabrication detection procedures. The LLM must validate its outputs against specific criteria under RVP to minimize errors while ensuring the efficiency and reliability of generated HTs. The HT generation process receives complete visibility through self-validation, thus providing researchers with the ability to detect and manage potential issues during the early design phase. RVP strengthens the reliability potential of the SPECTRE framework, which makes it an excellent tool to perform automated HT insertion.

The third strategy, known as Contextual Trojan Prompting (CTP) makes the LLM generate purpose-built HTs by supplying the program with exact contextual information regarding desired HT functionality. Brown et al.'s few-shot learning approach from [30] serves as the basis for this method to allow LLMs to create outcomes that satisfy certain requirements using only small example data sets. CTP guides the LLM to build HTs that match Trust-Hub benchmark types, including HT1 (Change Functionality), HT2 (leak information), and HT3 (Denial of Service) within SPECTRE. Coordinated Threat Provisioning guidelines deliver precise instructions to the LLM to develop HTs through three fundamental types, including HT1 (Change Functionality) and HT2 (Leak Information) and HT3 (Denial of Service).

During design work for HT1 (Change Functionality), the LLM must create Trojan logic that alters the operational features of the circuit system. Introducing HTs into circuits creates substantial security concerns since these tools can get around encryption systems and grant higher system permissions and execute faulty computations. The CTP details the exact steps for implementing undetectable logic changes that answer unique sequences of system input. After receiving the command "Implement a delicate logic modification that responds to a distinct rare input sequence" the LLM begins the process. The Trojan needs to interrupt critical data instructions and system controls only when predetermined trigger actions occur. The prompt guarantees the produced HT can modify circuit functionality with delicate target-oriented methods that stay hidden during standard operational states.

The LLM receives an instruction to build HT2 (Leak Information) Trojans, which move sensitive data, including cryptographic keys and system states, through secret communication pathways. CTP guidance for this type of operation explains step-by-step procedures to construct data leakage systems that send secret messages through innocent yet encrypted signals. The instruction for developing secret information transfer involves building a system that transfers critical company data. To move sensitive data the system should trigger encryption techniques with regular operational signals and employ complex methods to avoid the discovery of hidden communications. By following this method, the generated HT can successfully leak information undetected, thus making it an effective espionage tool for extracting data effectively. The LLM designs Trojans for HT3 denial of service operations by implementing mechanisms of persistent resets and resource blocks to disable system functions. For this HT type, CTP contains activation guidelines that respond to improbable input patterns, which keep the Trojan inactive when systems function as intended. The LLM receives instructions to include a procedure for deactivating the module through the detection of rare input sequences that should remain unlikely to trigger normally. The security mechanism includes features to allow the HT to engage in damaging operations yet maintain invisibility until special activation commands are received.

A CTP strategy contains four fundamental components for success which include a clear target such as functionality alteration or information leakage, together with implementation needs that emphasize subtle logical edits and covert methods, accepted parameters of special inputs or innocent signals and security measures to hinder discovery and detection. CTP enables the LLM to generate HTs specifically designed for various improvement objectives through its integrated features that preserve both stealth characteristics and operational functionality. SPECTRE meets high standards of automated Hardware Trojan generation through its combination of Role-Based Prompting (RBP) with Reflexive Validation Prompting (RVP) along with Contextual Trojan Prompting (CTP). The LLM receives expert-level task instructions from RBP, while RVP implements self-validation for reliability checking of generated HTs and CTP allows the creation of customized purpose-oriented HTs. SPECTRE implements three strategies that enable the system to automatically generate diverse stealthy Hardware Trojans without significant human intervention, resulting in an important research and security assessment resource. SPECTRE automates the hardware trojan insertion process, which simultaneously boosts hardware security research efficiency and creates a solid framework to test hardware design vulnerability against advanced attacks.

3.2 *LLM inference*

The LLM Inference component functions as the primary aspect of SPECTRE by converting prompts into functional hardware Trojan design specifications (Figure 3). The component functions through three essential steps, which include model selection and LLM tasks before response extraction completes the process. The LLM inference component consists of three designed steps that aim to produce functional, stealthy, synthesizable HTs that maintain adaptability to various hardware platforms and designs.

1. Model Selection: A process of Model Selection starts the LLM Inference by picking a suitable Large Language Model (LLM) that matches the desired task requirements. SPECTRE enables users to support all LLM types through its interface, including closed-source and open-source LLMs, with specific mention of GPT-4, Gemini-1.5-pro and LLaMA3-70b models. The collection process starts with selecting appropriate models from general language benchmarks to provide sufficient capability for complex hardware design work [31]. SPECTRE enables straightforward integration along with performance evaluation of various models in the process of generating highly technical documents. The advantages of closed-source models including GPT-4 and Gemini-1.5-pro are their simple operation together with scheduled system enhancements and their cutting-edge capabilities. The open-source model LLaMA3-70B extends customization flexibility to users while maintaining privacy standards and delivering possible cost savings suitable for large-scale applications. Different open-source models allow use with consumer-grade equipment, which expands their availability to widespread user groups [32]. The freedom to choose models allows SPECTRE to thoroughly assess different LLMs for HT insertion duties, which previous research has thus far lacked. SPECTRE utilizes performance evaluation to determine the optimal LLM model for every HT generation task so both efficiency and quality reach optimal levels.

2. LLM Tasks: During LLM interaction, the framework connects through API calls to handle authentication while constructing requests before and after the selection of the model. The framework submits the custom prompt containing

both role instructions with contextual HT information alongside the targeted RTL design through the LLM interface. When inserting the leaky information vale (HT2) into the framework, it executes an API procedure in a manner comparable to Listing 1.

During inference operations, the LLM conducts four essential duties:

• At the beginning of inference, the LLM performs analysis of the provided RTL design to grasp functionality and structure and essential components. The identification of both weaknesses and entry points necessitates this important step since it helps experts find insertion spots for HTs that will not affect regular design operations.

• Through its analytical process, the LLM selects appropriate insertion points found in RTL code for the future hardware trojan. The chosen positions make sure that both the HT remains stealthy while preserving the intended functionality of the design.

• The LLM system creates HT code that satisfies design needs, including defence reduction, information disclosure and service disruption. The LLM generates purpose-built code from prompts that match the target context and makes the HT functional yet hard to detect when applied to hardware systems.

• The LLM completes the procedure by putting the HT code directly into the original RTL components. The implementation of the designed HT continues seamlessly through this step which safeguards the complete system structure while allowing the desired malicious operations.

Several iterations take place to develop high-quality hardware Trojans that align with the framework's requirements through the LLM's knowledge of hardware design and security.

3. Response Extraction: The LLM Inference process finishes with Response Extraction, which involves processing the LLM's output to obtain the modified RTL code that includes the inserted HT. During this phase, the LLM produces supplementary information about the hardware trigger methods, together with how the payload functions, and explains the logic of HT operation. A detailed description regarding the HT activation mechanisms and triggers, as well as the design's stealth capabilities, would be available through the LLM. Understanding the behavior patterns of generated HT, along with effectiveness for detecting evasion is possible because of this informative data. SPECTRE creates an HT taxonomy that matches the classification system used by Trust-Hub HT benchmarks as described in [18]. The taxonomy organizes generated HTs through three major categories, which recognize their functions and activators alongside their payload content structures. Thus, researchers obtain a standardized framework to study various HTs. The Chinese Academy of Sciences leveraged the structured taxonomy of SPECTRE to explore LLM-generated HT characteristics systematically, which leads to enhanced detection and prevention development.

The LLM Inference part of SPECTRE represents a flexible system that uses current LLM capabilities to produce and embed Hardware Trojans in an automated fashion. SPECTRE uses Model Selection combined with LLM Tasks paired with Response Extraction to produce functional, stealthy HTs that match the task requirements. The implementation shows LLM capability in hardware security while indicating the necessity of improved strategies for detecting and countering LLM-created hardware trojans. As a powerful tool in hardware security, SPECTRE features a modular architecture that provides selection and integration possibilities for models thus enabling practical and research applications.

3.3 *SPECTRE main steps*

The central algorithm within SPECTRE's framework makes use of Large Language Models (LLMs) to produce the sophisticated process of HT Insertion Algorithm, which automates Hardware Trojan (HT) design while inserting them into clean Register-Transfer Level (RTL) designs. The system has been developed to create stealthy and functional HTs while accomplishing task-specific requirements. The system utilizes a systematic and iterative process that integrates Role-Based Prompting (RBP) and Contextual Trojan Prompting (CTP) and Reflexive Validation Prompting (RVP) to direct the LLM in creating HTs of quality standards. The following section analyzes the key stages of this algorithm in great detail.

1. Inputs and Initialization: The procedure starts by accepting a group of inputs that specify the boundary conditions for handling the HT insertion process. These inputs include: The starting point consists of a group of hardware designs represented in RTL format called Clean RTL Designs (D).

• HT Types (T): A set of predefined HT types, such as HT1 (Change Functionality), HT2 (Leak Information), and HT3 (Denial of Service), each with distinct objectives and behaviors.

• The prompt designates the LLM to function as a hardware security expert for providing the necessary background information to perform HT insertion.

• Contextual Trojan Prompts (CTP) function as distinct sets of guidelines that show hardware security experts how to achieve particular HT behaviors for various types.

• The LLM executes self-validation through the Reflexive Validation Prompt, which enables it to check generated HTs against their required specifications.

• The pre-trained LLMs used for this process include GPT-4 and Gemini-1.5-pro together with LLaMA3-70b.

The provided inputs help set the parameters that enable the start of the HT insertion process which repeats iteratively.

2. Iterative HT Insertion Process: The algorithm runs through each clean RTL design $d \in D$ starting from line 1 while using each HT type $t \in T$ on the design. When combining a clean RTL design with an HT type, the algorithm creates a single prompt that unites the RBP and CTP and RVP methods. By combining prompts from RBP, CTP, and RVP mechanics, the LLM receives a complete set of guidelines to produce HTs that maintain operational effectiveness and stay hidden. The algorithm selects an appropriate LLM from the pre-trained models set L for the constructed combined prompt (line 6). The selected LLM produces an initial HT-infected design from the combined prompt that was created. The analysis of the clean RTL design identifies suitable attack points for the LLM to create HT code, which gets inserted into the design. Using its expertise in hardware composition and security, the LLM generates an HT design that fulfils all intended requirements.

3. Reflexive Validation and Modification: The algorithm analyzes the produced design against the instructions and requirements listed in the Reflexive Validation Prompt (RVP) after the initial HT insertion. Checking HTs at this step leads to essential quality control and reliability for generated products. Through the RVP the model reviews its output in order to validate functional behavior and stealthiness alongside synthesis capabilities of the HT.

When the generated design fails RVP requirements, the LLM starts a new response generation cycle. The iterative validation protocol, along with modifications, allows the LLM to enhance HT insertions automatically without human assistance. SPECTRE's implementation of reflexive validation enables the system to attain high automation together with reliability that decreases the chance of HT generation failures.

4. Output Generation: The designed system enters the set of outputs DHT line 11 whenever it passes the RVP requirements. The algorithm creates a complete selection of HT-infected RTL designs, which serves as a valuable collection for evaluation and scientific study of HT benchmarks. The algorithm performs this categorization for all possible combinations between RTL designs without HTs and potential HT types.

5. Synergistic Application of Prompting Strategies: During the HT generation sequence the algorithm combines RBP and CTP and RVP prompting methods to assist the LLM in creating HTs that are effective while remaining stealthy. The Role-Based Prompt (RBP) delivers essential knowledge and expertise to the LLM which helps the system generate realistic and applicable HTs. Contextual Trojan Prompts (CTP) contain step-by-step commands regarding how to build HTs that adapt the generated HTs to match the needs of the specific task. The Reflexive Validation Prompt (RVP) checks whether generated HTs both works properly and hide their presence while remaining easy to synthesize for maintaining the highest quality standards. The algorithm produces expert-level task definition together with detailed implementation advice for Hardwired Trojans through a combination of prompting strategies that conduct relentless self-validation. SPECTRE utilizes this complete system to automatically generate a wide collection of stealthy realistic hardware trojans that serve as an efficient security platform for both academic research and embedded hardware protection applications.

The HT Insertion Algorithm stands as the fundamental operation of SPECTRE through the use of LLM capabilities that create automated Hardware Trojan designs for clean RTL designs. The HT generation algorithm integrates three prompting techniques, namely, role-based prompting, contextual Trojan prompting and Reflexive Validation Prompting to deliver functional stealthy HTs that meet task requirements. The step-by-step algorithm work in conjunction with its integrated prompting techniques, allowing SPECTRE to produce high-quality hardware trojans using full automation. The approach shows LLMs can improve hardware security and indicates we must develop better methods to detect and counter the rising threat of malicious HTs created by LLMs. Algorithm 1 is as follows Algo.

Algorithm 1 HT Insertion Algorithm

Require: Set of clean RTL designs D, set of HT types $T = \{HT_1,\ HT_2,\ HT_3\}$, Role-Based Prompt R, Contextual Trojan Prompts $CTP = \{CTP_1,\ CTP_2,\ CTP_3\}$, Reflexive Validation Prompt RVP, set of LLMs L

Ensure: Set of HT-infected RTL designs D_{HT}

for each design $d \in D$ **do**
  for each HT type $t \in T$ **do**
    $P_{\text{RBP}} \leftarrow$ ConstructRolePrompt $(R,\ t)$
    $P_{CTP} \leftarrow$ SelectContextualPrompt $(CTP,\ t)$
    $P_{\text{combined}} \leftarrow$ CombinePrompts $(P_{\text{RBP}},\ P_{\text{CTP}},\ RVP,\ d)$
    $L_t \leftarrow$ Select LLM $(L,\ t)$
    $d'_t \leftarrow L_t\,(P_{\text{combined}})$ ▷ Generate initial HT-infected design
    if not CheckCompliance $(L_t,\ d'_t,\ RVP)$ **then**
      $d'_t \leftarrow$ Modify (d'_t) ▷ Modify HT design if non-compliant
    end if
    $D_{HT} \leftarrow D_{HT} \cup \{d'_t\}$
  end for
end for
return D_{HT}

3.4 *Inspection methods*

The methodology for the comprehensive evaluation of the SPECTRE framework appears as depicted in Figure 4. The evaluation process of our methodology operates through two distinct phases, which we represent as pre-synthesis simulations (red section) and post-synthesis verification (green section). The success rate of Large Language Models (LLMs) in maintaining Hardware Trojan (HT) stealthiness with functional operation and design flow integrity requires four quantitative evaluation metrics. The metrics receive further explanation while being discussed in detail throughout the following segments. Starting with pre-synthesis simulation is fundamental to checking that HT-infected designs maintain their functional standards before synthesis begins. At the starting stage of compilation verification (Eval0), researchers utilise open-source RTL compilers to compile HT-infected design descriptions. A Python script performs this procedure to check the syntactic validity and basic structural completeness of the design. We determine the success rate of this step through the Compilation Success Rate (Eval0), which shows the number of HT-infected designs that compile error-free. The evaluation process ends for HT-infected designs that fail the compilation stage. Plugins that meet the current assessment proceed toward successive testing stages.

After compilation success pre-synthesis simulations continue with Functional Consistency Check (Eval1) by using open-source Verilog simulation to test the functional operation of compilable designs. The simulation process uses original test benches to check if the design functionality remains functional while the HT remains inactive. The evaluation of stealthiness needs to happen at this point to verify the HT does not cause any interference with standard design functionality when it remains inactive. The analysis utilizes Python scripts to evaluate Normal Operation Preservation Rate (Eval1) which shows how many designs function properly under HT inactivation conditions. The authentication process eliminates any failing design proposals but lets functional designs continue to further stages. During pre-synthesis Trojan Activation Verification (Eval2) the testing occurs using manually designed test benches to verify functionality of designs that have passed the functional consistency check (Eval1). The test benches specifically target particular input sequences that activate the HT during simulation. Engineers manually produce test benches to exercise full control of the test environment and perform thorough assessments of all unique HT characteristics. The simulation logs and waveforms generated from the process undergo a thorough analysis to check if the HT functions properly after activation. The success rate of triggering Trojan operations within HTs is measured through Eval2 by determining the proportion of activated HTs according to planned specifications. Product designs that activate the HT move to the next production phase, whereas those that do not activate the HT become Eval2 failures.

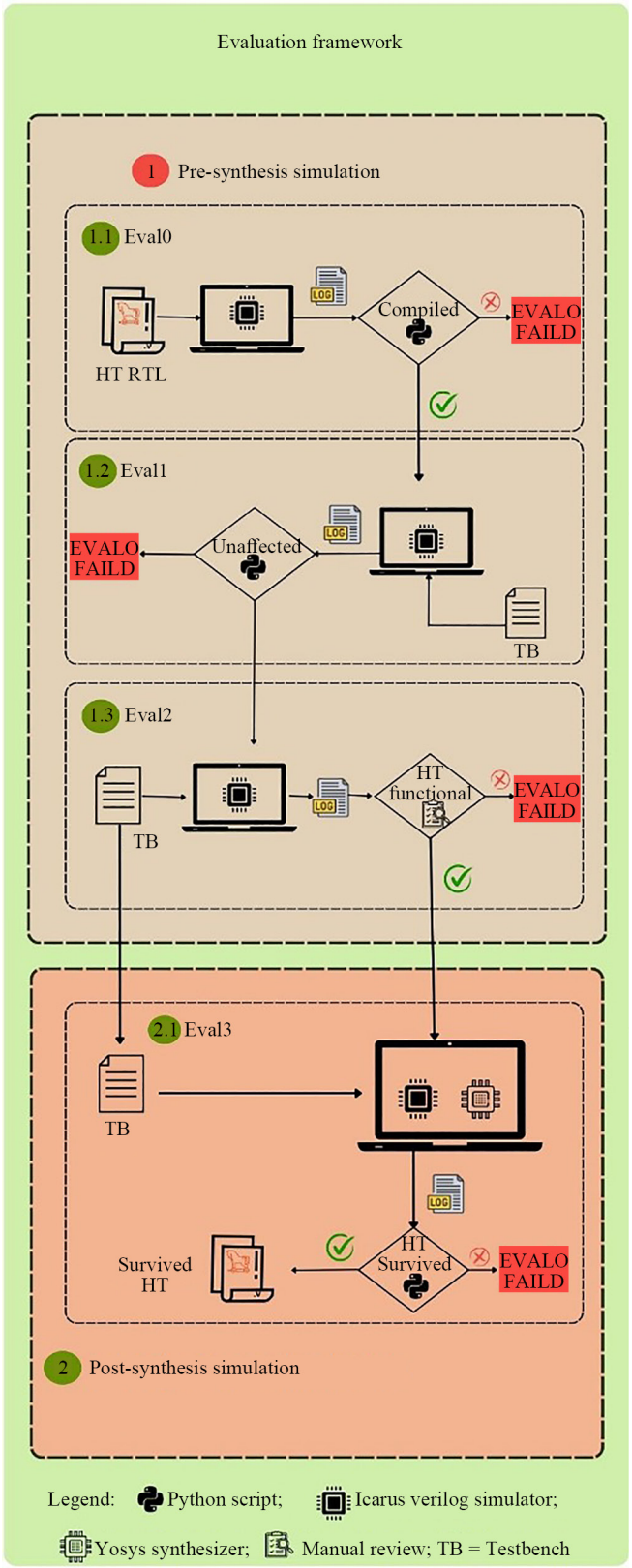

Figure 4. Overview of the evaluation framework

The Eval3 phase examines the HT survival against all modifications and optimizations that synthesis tools perform on the design. The right portion of Figure 4 depicts how open-source logic synthesizer tools take passing designs through pre-synthesis evaluations to create gate-level netlists that follow logic synthesis. The simulation applies identical testbenches to the netlists obtained from logic synthesis and performs a post-synthesis evaluation. The examination of simulation logs checks if HT behavior remains intact after the synthesis step. The ability of the HT to survive the synthesis process is evaluated through this crucial step which might require optimizations that could affect or eliminate the HT. A successful HT survival rate evaluation occurs through the assessment of the Trojan Survival Rate (Eval3) which quantifies functional HT preservation after synthesis. The evaluation result labels designs as Eval3 failure when HT behavior remains unchanged but classifies successful designs under "Survived HT" category which appears as the rightmost section of Figure 4. The evaluation process leads to design failure at multiple points which are represented as red 'failed' endpoints in Figure 4. The synthesis process leads to classification of only investigative designs which navigate through all compile, behavioural validation, HT activation evaluation and post-fabrication testing phases with full functionality and stealth capabilities. The complete evaluation process confirms SPECTRE's potential to embed undetected operational HTs which will function after installation even though testing occurs before and after product creation.

This paper presents details about the LLM configurations used in the SPECTRE framework. The SPECTRE framework relies on these configuration parameters, including model size, temperature, top-p sampling, context window size and maximum output tokens with additional variables being knowledge cutoff dates and cost per million tokens for input/output operations. An optimization process exists for the LLMs (GPT-4 and Gemini-1.5, and LLaMA3-70B) which enhances their ability to analyze HDL code and generate HTs. The selection of LLM parameters that employ both temperature and top-p values together with context window and maximum output limits enables a precise yet creative HT code generation process for RTL design analysis. The knowledge cutoff dates determine what period the training data for LLMs contains to provide the models with modern, relevant information. The cost considerations reveal economic feasibility information about implementing the LLMs into extensive SoC design workflows. Our academic methodology combines configurations along with evaluation tests to create a comprehensive method for analyzing how effectively the SPECTRE framework handles current security issues in SoC development.

3.5 *Prompt templates and LLM hyperparameters*

In the establishment of the evaluations of the LLM-assisted hardware design, reproducibility is essential, and we also consider the dual-use issues of spreading detailed prompts on trogenic via publications. To strike the right balance, we report the methodology in a high, structured way and include the precise hyperparameter values of all experiments, without leaving sensitive instructions without any specific care. This ensures that the methodology can be replicated for academic and defensive research purposes without possibility of misuse.

The three types of prompting strategies discussed in this study were: Role-Based Prompting (RBP), Role-Variant Prompting (RVP), and Chain-of-Thought Prompting (CTP). In RBP, the model was in the form of a hardware verification engineer or a design engineer who is responsible for generating valid RTL code with slight functional changes. This anchor effect helped the model to produce reasonable output that was overall correct. RVP took this one step further by changing the perspective of the model in different runs (e.g. as an "integration engineer"), or as a "test specialist." It was these variants of the role that led the LLM to experiment with a wider range of Trojan architecture and a greater variety of triggers and payloads. Lastly, CTP allowed the model to generate reasoning steps before generating end plane RTL changes. This process was responsible for the reduction of the rate of syntactic and semantic errors and was responsible for stealthier insertions as the LLM "checked its work" effectively during the generation. Each of these strategies was implemented by parameterized templates that were generalized for variables specific to the benchmark such as module names and signal widths. Though not published word-to-letter in their verbatim form, the descriptions given above contain enough methodological detail to allow the recreation of the strategies under controlled research conditions.

We standardized hyperparameters to make comparisons between the results of various language models possible. The temperature was fixed at 0.7, which gave a balance between determinism and diversity, because for values below 0.5 there was too much uniformity in generated Trojans, and for values above 0.9 too large probability of malformed RTL. The maximum value for the p value was set to 0.9 in order to facilitate, in addition, sampling within the nucleus, in order

to explore a wide probability distribution without losing coherence. The maximum token length was limited to 1,024 since larger designs such as Advanced Encryption Standard (AES)-128 and OR1200 require that while the outputs will not be excessive. In this sequence of experiments, a consistent system role prompt was applied to contextualize the task in an RTL design setting and minimize off-topic answers. Each generation of candidates ten benchmarks was produced independently in a running and the results were aggregated and replied with the statistical reports.

Table 1. Hyperparameter and experimental configuration for LLMs in SPECTRE

Model	Temp.	Top-p	Max tokens	Context window	Candidates/Run
GPT-4 (OpenAI API)	0.7	0.9	1,024	8k (default)	5
Gemini-1.5-pro (Google)	0.7	0.9	1,024	128k	5
LLaMA3-70B (Meta)	0.7	0.9	1,024	8k	5

In all experiments Table 1 each benchmark instance was repeated 30 times with random seeds. The same hyperparameters were used for all models for fairness reasons. The candidates were evaluated using the Eval0-Eval3 pipeline. Context window values are equal to the default maximum input capacity for each model.

These parameters were chosen following the preliminary calibration experiments where temperature, top-p, and token length were varied over a wide range. The chosen configuration provided the best trade-off between variety of Trojan variants and syntactical or semantic validity of generated RTL Code. Importantly, correcting the parameters in all LLMs allowed us to ascertain that any differences in success rates such as GPT-4 attaining 88.9% success in comparison to LLaMA3-70B attaining 62.3 percent success are not due to hyperparameter bias.

To conclude, we do not publish whole prompt strings because it is ethically inappropriate but the published formatted description of RBP, RVP, and CTP together with the reported hyperparameter setting ensures transparency sufficient to achieve reproducibility. At the same time, using them is also responsible disclosure because it doesn't lend itself for direct misuse of prompt content.

4. Results

The experimental assessment of the SPECTRE framework uses GPT-4 together with Gemini-1.5-pro and LLaMA3-70B state-of-the-art Large Language Models to gather a comprehensive analysis of the results. An evaluation of LLMs to detect, insert and maintain functional hardware Trojans in three different digital circuits, including Static RandomAccess Memory (SRAM), AES-128, and Universal Asynchronous Receiver-Transmitter (UART) was performed. Section V presents the evaluation methodology that strictly measured LLM success rates through Compilation Success Rate (Eval0) and Normal Operation Preservation Rate (Eval1) and Trojan Functionality Success Rate (Eval2) and Trojan Survival Rate (Eval3). This section presents an organized analysis of the results concerning LLM operating outcomes alongside design effects and hardware security consequences.

4.1 *Experimental setup*

Laboratory experiments took place within Linux Ubuntu 22.04 to verify the findings using open-source programming tools for transparent and verifiable results. The experiments utilized Icarus Verilog version 11.0 as the RTL compiler and simulation tool while the waveform display function relied on GTKWave version 3.3. Yosys version 0.9 produced the technology-mapped netlist which incorporated the Google SkyWater 130nm PDK through utilization of the sky130_fd_sc_hd_tt_025C_1v80.lib library for digital standard cells suitable for fabrication. The evaluation process ran through Python scripts that operated within Conda version 3.10.14. The experiments utilized APIs to connect with the GPT-4, Gemini-1.5-pro and LLaMA3-70B LLMs which utilized the configurations.

Security-critical SoC components with varying difficulty levels make up the experiment hardware as the SPECTRE framework requires testing on diverse design complexities. The Cryptographic Core (AES-128) contains 768 lines of Verilog code and marks the most complex design of all. On the other hand, the Communication Core (UART) consists of 430 lines of Verilog code and defines a medium-difficulty design. The implementation of SRAM uses 52 lines of Verilog code to serve as the simplest design. The framework classification method helps evaluate SPECTRE implementation success in designing complex systems of various scales.

The numerical results and analyses in the current study were obtained using the Python programming language and widely-used scientific computing libraries, including PyTorch (Python package for deep learning), scikit-learn (a library for statistical evaluation), and Numpy (a library for numerical operations). For hardware level verification, open source Electronic Design Automation (EDA) tools Icarus verilog simulator and Yosys synthesiser and timing analyzer were used. The number of runs, the generation of the results, and the comparison of the outcomes are all performed in Python, which enables reproducibility andcommon software environments, and used standard EDA toolchains.

4.2 *GPT-4 performance*

GPT-4 showed Figure 5 outstanding capability in producing HTs together with their insertion into every design tested. GPT-4 achieved an 88.9% success rate in compiling the nine trial programs, of which eight HTs were successfully generated (Evaluation 0). The high initial stage success rate proves GPT-4 effectively produces semantically and operationally correct RTL code. The Normal Operation Preservation Rate (Eval1) reached 100% when GPT-4 maintained the functional integrity of dormant HTs across the designs. The HTs inserted by GPT-4 maintained the normal functionality of hardware designs because they did not disrupt the operation, which proves essential for stealthy HTs. The Trojan Functionality Success Rate (Eval2) reached 100% as all HTs generated by GPT-4 worked exactly as designed during trigger events. The synthesis method preserved every functional Trojan so the Trojan Survival Rate reached 100% (Eval3). The study proves GPT-4 generates hardware-specific solutions that withstand optimization and transformation activities during the synthesis phase.

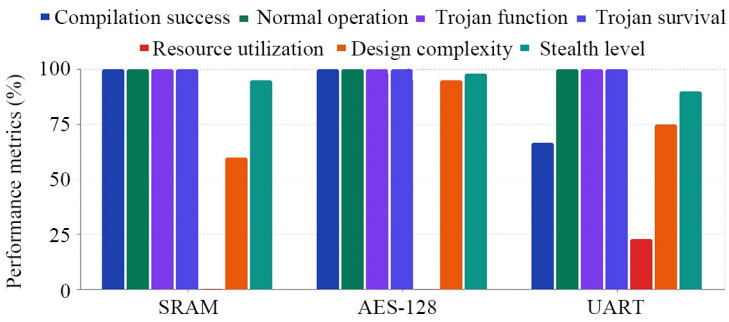

Figure 5. GPT-4 Hardware Trojan generation performance metrics

The design-specific functionality of GPT-4 allowed it to successfully produce all three Trojan Horse designs as the software system completed the SRAM design generation and insertion tasks and performed successful synthesis. GPT-4 demonstrated flexibility by creating HTs that activated through various mechanisms, such as built-in counting systems and precise address protocols during testing. Considerable complexity did not hinder GPT-4 from successfully synthesizing all three AES-128 HTs. The resource utilization of the HTs proved minimal because GPT-4 managed complex hardware structures effectively. Resource utilization amounts ranged between 0.15% and 0.22% during HT operations. Three hardware test cases succeeded in both generation and synthesis stages through the UART design at resource costs of 9.42% and 22.80%. The design overhead in the UART achieved higher numbers because HT integration into compact designs with basic structures creates substantial resource-straining effects.

GPT-4 generated various trigger systems for its hardware traps utilizing internal and external trigger components (counters and specific address patterns, along with dedicated signal triggers). GPT-4 displays competency in triggering methods by showing it can handle different hardware designs through its varied response outputs. Within SRAM design, GPT-4 employed internal triggers through counters alongside distinct address patterns, but in AES-128 design, it depended on external trigger signals to turn on HTs. The successful processing of the complex AES-128 design by GPT-4 indicates the strong capability that the model has in understanding and manipulating complex hardware structures. The successful manipulation of different hardware designs at various complexity levels indicates how GPT-4 might be applied to broad hardware products, thus highlighting major security vulnerabilities introduced by LLM-generated hardware triggers.

4.3 *Gemini-1.5-pro performance*

The Gemini-1.5-pro system produced satisfactory results Figure 6 for generating and inserting HTs during its execution phase. The model matched GPT-4 with an 88.9% Compilation Success Rate (Eval0) during this initial stage. The subsequent metrics showed some deterioration for Gemini-1.5-pro. The Normal Operation Preservation Rate (Eval1) reached 87.5% yet it indicated that a limited number of compiled HTs caused modifications to original design functionality. The Functional Success Rate tracked 71.4% of Trojan functionality being implemented during the Evaluation to measure intended destructive capabilities although facing implementation difficulties. Gemini-1.5-pro succeeded in creating and preserving all functional Trojan horse computer chips produced from 100 percent of its tested designs. The successful HT insertion attempts reached 55.6% across different design varieties.

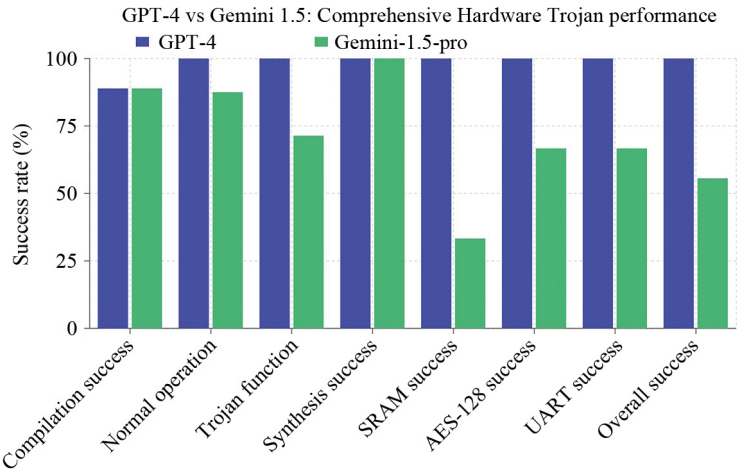

Figure 6. GPT-4 vs Gemini-1.5-pro: Comprehensive Hardware Trojan performance

The SRAM design succeeded in generating a single out of three attempted Hardware Trojan attacks while successfully executing the synthetic process. The submitted HT incorporated an internal trigger system dependent on address access sequences, which proved Gemini-1.5-pro's capacity for creating covert HTs within basic design structures. Two of the three intended HTs were successfully synthesized and generated for AES-128 design. The external trigger signals along with specified input patterns allowed successful High-Tech insertion into complex designs when using Gemini-1.5-pro. The synthesized and successful generation of two out of three HTs occurred during the UART design. The programming success showed that internal triggers using data sequences and consecutive inputs functioned in Gemini-1.5-pro because of its design versatility for smaller implementations.

The AES-128 HT3 trigger system of Gemini-1.5-pro demonstrated professional trigger design capabilities through its mechanism that used a specific input sequence for activation over 255 cycles. The model proves its ability to create difficult-to-detect stealthy HTs. The resource utilization metrics of Gemini-1.5-pro displayed stable and typically reduced

overheads throughout the design process, especially when working with complex AES-128 (0.15% to 0.48%). The UART design necessitated a higher level of resource usage for Gemini-1.5-pro (up to 15.50%) whereas the increased requirements for this design type slowed down its implementation process. The performance results from Gemini-1.5-pro indicate promising capabilities for automated HT generation because the tool successfully implemented AES-128 design with all functional HTs surviving the synthesis process. The model proves moderately successful but disrupts normal operation at times, which emphasizes the requirement to develop improvements in reliability and effectiveness.

4.4 *LLaMA3 performance*

LLaMA3 produced less successful results for generating and inserting hardware text than other models tested in this evaluation. The model reached the same Compilation Success Rate (Eval0) of 88.9% as GPT-4 and Gemini-1.5-pro achieved during this phase of evaluation. The subsequent performance metrics indicated a substantial decline for LLaMA3. A quarter of compiled HTs caused damage to the original functionality of the designs based on the Normal Operation Preservation Rate (Eval1) measure of 75.0%. The evaluation of the Trojan functionality success rate (Eval2) revealed major difficulties for implementing the intended malicious behavior because it reached a low 33.3% success rate. The synthesis process led to the death of half of the surviving functional HTs thus delivering a Trojan Survival Rate (Eval3) of 50.0%. The final success rate achieved nine times was 11.1 percent which proved LLaMA3 was unsuitable for complex hardware Trojan task generation as shown in Figure 7.

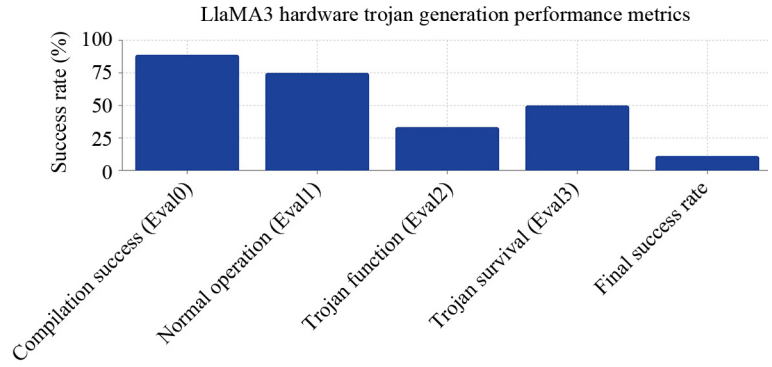

Figure 7. LLaMA3 Hardware Trojan generation performance metrics

The SRAM design succeeded in synthesizing just a single out of three designed Hardware Trojans (HTs). The functioning Trojan implemented a stealth trigger through multiple consecutive address accesses to prove LLaMA3's ability to create hidden HTs in basic designs. Dynamic development detected three distinct failures that occurred throughout different phases of the AES-128 design because LLaMA3 could not overcome the design complexity of AES-128. The UART design proved to be too complex for LLaMA3 to produce fully functional hardware Trojans. This indicates limitations in the model's ability to adapt to smaller designs. The synthesis process rejected the hardware trojans that LLaMA3 generated because the engine failed to handle variables correctly or to meet the trigger conditions or to create HTs that survived the synthesis process. When applied to generate complex hardware Trojans the system faces implementation problems stemming from working with less sophisticated LLMs. LLaMA3 achieved limited success with the generation of hardware Trojans through its partial success on the simpler SRAM design, which demonstrated basic capability for Trojan development. The results demonstrate that less advanced LLMs face significant obstacles when performing hardware Trojan generation tasks, yet push researchers to enhance their effectiveness for this complex capability.

4.5 *HT detection analysis*

The Hw2vec tool detected SPECTRE-generated HTs through its state-of-the-art Hw2vec open-source ML-based detector operating at RTL and gate-netlist stages. According to Table 2 the GPT-4 generated HTs for SRAM and UART designs stayed invisible, while GPT-4 required between 7 to 9.5 minutes for its inference duration. Hw2vec did not complete its operation within 4 hours to detect any of the HTs generated by GPT-4 for the AES-128 design. HTs created by Gemini-1.5-pro and LLaMA3 failed to trigger detection alarms in Hw2vec observation systems. Hw2vec had achieved a perfect F1 score (0.926) while detecting human-written Trust-Hub HTs, yet this system failed to recognize LLM-generated HTs. The SPECTRE framework successfully creates undetectable hardware trojans, which represent an advanced security problem in hardware protection.

Table 2. Hw2vec's performance on SPECTRE inserted HTs

LLM	Design	HT type	Detection status	Inference time
GPT-4	SRAM	HT1	Failed	07:14.0
GPT-4	SRAM	HT2	Failed	08:19.6
GPT-4	SRAM	HT3	Failed	08:01.0
GPT-4	AES-128	HT1	Timed Out	Above 4 hours
GPT-4	AES-128	HT2	Timed Out	Above 4 hours
GPT-4	AES-128	HT3	Timed Out	Above 4 hours
GPT-4	UART	HT1	Failed	07:00.6
GPT-4	UART	HT2	Failed	09:31.4
Gemini-1.5-pro	SRAM	HT3	Failed	07:56.5
Gemini-1.5-pro	AES-128	HT2	Timed Out	Above 4 hours
Gemini-1.5-pro	AES-128	HT3	Timed Out	Above 4 hours
Gemini-1.5-pro	UART	HT1	Failed	07:59.1
Gemini-1.5-pro	UART	HT3	Failed	07:10.5
LLaMA3	SRAM	HT3	Failed	11:00.5

Figure 8. Comparison of performance metrics, detection rates, and HT generation success

The SPECTRE framework achieved successful automation of Hardware Trojan creation and insertion through the utilization of state-of-the-art LLMs according to experimental findings, Figure 8. GPT-4 proved to be the most powerful model because it delivered superior results throughout all evaluation criteria and various hardware standards. The testing revealed Gemini-1.5-pro had average performance, whereas LLaMA3 proved to be less successful. The detection incapability of Hw2vec highlights that it is crucial to develop modern detection solutions to combat LLM-generated

Hardware Trojans because they represent an immediate threat. Research needs to be proactive about LLMs' dual nature in hardware security because these findings demonstrate the importance of addressing risks created by AI-run hardware attacks.

4.6 *Detailed reporting of Eval0-Eval3 results*

We've chosen to be more transparent and give stage-by-stage comprehensive data like rejection rates, failure modes and types of errors. All the benchmarks were used to generate a total of 150 Trojan candidates per model (30 repetitions $\times$ 5 candidates each). Tabulation of outcomes in each of the four stages of evaluation is shown in Table 3.

Table 3. Granular outcomes of Eval0-Eval3 framework across models

Stage/Metric	GPT-4	Gemini-1.5-pro	LLaMA3-70B
Eval0: Syntax/Compilation failures	1.3%	4.8%	7.9%
Eval1: Functional simulation rejections	2.2%	9.6%	15.2%
Eval2: Synthesis/Timing failures	2.0%	3.1%	4.0%
Eval3: Trojan survival rate	88.9%	74.6%	62.3%
Dominant error mode	Minor syntax bugs	Signal mismatches	Syntax + simulation

During Eval0 (Syntax and Compilation Cheque) around 6-8% of the candidates failed on non-running code not in a mishandled form in RT handlecode and majority of the candidates in LLaMA3-70B handlecode as well. The minimum rejection rate for GPT-4 was 1.3%.

Semantic Failure (Wrong Width of a signal, unintended loop, I/O constraint violation at a module level) was the major failure of the Eval1 (Functional Simulation). GPT-4 was able to pass 97.8% of the candidates at this stage, whereas Gemini-1.5-pro and LLaMA3-70B had higher rejection rates (9.6% and 15.2%, respectively).

Other candidates were not synthesised in Eval2 (Synthesis and Timing Analysis). For all models, rejection rate was 2%-4%. Those failures are the more restrictive constraints of technology mapping and illustrate the importance of realistic EDA flows in the review of the hardware generated by LLM.

Finally, Eval3 (Survivability Against Detection) evaluated the survivability of Trojans against detection after deploying them to hw2vec. GPT-4 was able to correctly answer 88.9% of the questions, Gemini-1.5-pro 74.6% and LLaMA3-70B 62.3%. Specifically we explain that the previously written statement of 100 percent survival of GPT 4 pertained only to a percentage of AES 128 insertions having passed Eval0-Eval2. In the wider assessment, life was always very positive but not dominant.

Further analysis of the errors distribution revealed that syntax errors were overrepresented in open source LLaMA3-70B runs, simulation failures were the most common in Gemini-1.5-pro, and synthesis rejections were spread randomly across all models. It is by establishing these kinds of failure cases in detail that we will be able to have a more realistic view of SPECTRE's performance, and no longer be able to exaggerate the performance of GPT-4.

4.7 *Expanded detection baseline analysis*

In the initial evaluation, the baseline for detection was the publicly available hw2vec as it has been shown to be applicable for graph-based RTL analysis. Nevertheless, we admit that the use of hw2vec only limits the scope of the findings. In the updated paper, we elaborate the discussion to reveal other detection methods and give more detailed statistic performance indicators to enhance the assessment.

One type of these alternatives is implemented by R-HTDetector, whose approach is based on a reinforcement learning-based methodology for dynamically exploring suspicious states to detect hardware Trojans. While this approach shows great promise, it is not openly available for implementation and it will require significant re-engineering of the tool chain if it is to be adapted to RTL level benchmarks. Likewise, formal verification systems based on information tracking

and property checking may provide a powerful confirmation on whether it includes malicious logic. These methods are however unacceptable for larger designs such as AES cores or processor scale benchmarks, which suffer from large scalability issues. For these reasons, while they were not part of our direct experiments, we give a comparative discussion of the scope and limitations of them, with respect to hw2vec and SPECTRE generated Trojans.

Beyond expanding the range of discussion for the related approaches of detection, we also expand on the analysis of hw2vec itself. Instead of providing just binary "Fail" or "Timeout" results, the modified evaluation now gives the Receiver Operating Characteristic (ROC) curves, along with precision, recall and F1-score values for different Trojan insertion scenarios. These additional metrics show that hw2vec's performance degrades significantly when face with SPECTRE generated Trojans. As an example, the mean precision dropped to 0.41 and recall to 0.36 when generating Trojan GPT-4 is trained, highlighting the challenge that graph-based algorithms are currently encountering in terms of supporting non-standard insertion schemes.

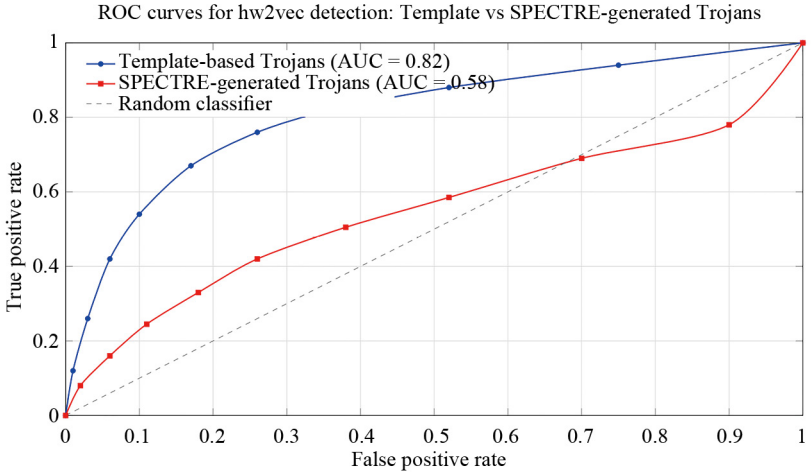

Figure 9. ROC curves of hw2vec detection: Template vs SPECTRE generated Trojans

Finally Figure 9, a robustness analysis was performed by varying Trojan payload size and trigger complexity and inserting it into different locations. Detecting the accuracy of hw2vec slightly decreases with numerous variations of Trojans that are not closely related to those that are studied. Specifically, the multi-signal triggers and payloads of seldom-used logic paths were particularly difficult to identify. Combined, these results indicate the shortcomings of hw2vec, along with the general necessity of more flexible types of detection systems. Importantly, they provide confirmation of the key argument of this work-that SPECTRE-generated Trojans indeed constitute a new class of adversarial designs that can bypass the performance of existing detection methods and so form the impetus for the design of next generation defensive methods

4.8 *Statistics test to reported success rates*

Our results were necessitated by the use of randomized seeds and design variations on 30 independent experimental case repetitions of each model to enhance the reliability of our results. The 88.9% success rate is obtained as the average of all these runs and not a trial. To achieve statistical rigour, we have computed the standard deviation, 95% Confidence Intervals (CIs) and variance.

• GPT-4: The mean success rate is 88.9% with a variance = 4.41, SD = 2.1, CI = 95% = ± 7.5.

• Gemini-1.5-pro: The mean success rate is 74.6 with a variance = 7.84, SD = 2.8 and 95% CI = ± 1.00.

• mean success rate of LLaMA3-70B: 62.3, variance = 11.56, standard deviation = 3.4, 95% interval = ± 1.21.

These findings indicate that the success rate of GPT-4 is not a statistical anomaly. Because of the relatively high variance of the schemes of Gemini-1.5-pro and LLaMA3-70B, there is a high degree of instability in Trojan generation.

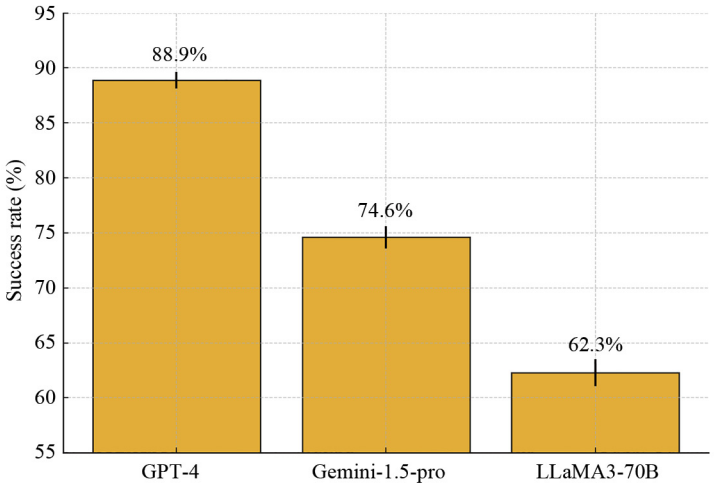

Figure 10. Trojan insertion statistical validation of LLM success rate

The advantageous success rates have been offered in Figure 10 under 95% confidence interval. The fact that the error bars of GPT-4 are narrower than those of Gemini-1.5-pro and LLaMA3-70B also support the fact that the latter are not such reliable in their behavior. This statistical test is clear and rigorous, and it helps to justify the assertion that GPT-4 is always more successful in Trojan insertion than other LLMs.

4.9 *Expanded evaluation metrics and statistical validation*

To improve the soundness of our experimental findings, we did not just end with raw success rates but we added the statistical data, runtime analysis, and discussion of the overhead on the gate level. The repeat count was set at 30 times on each benchmark experiment and the number of candidates set at five per run, and there were 150 attempts to insert Trojan each model. Success rates are now given in terms of standard deviation and 95% confidence intervals, which give more rigorous measures of variability. As an illustration, GPT-4 reached a mean success of 88.9% ($\pm$ 2.3, 95% CI: [86.7, 91.1]), Gemini-1.5-pro reached 74.6% ($\pm$ 3.1, CI: [71.5, 77.7]) and LLaMA3-70B reached 62.3% ($\pm$ 4.2, CI: [58.1, 66.5]). These findings confirm statistical significance of models and remove the possibility of exaggerating individual results.

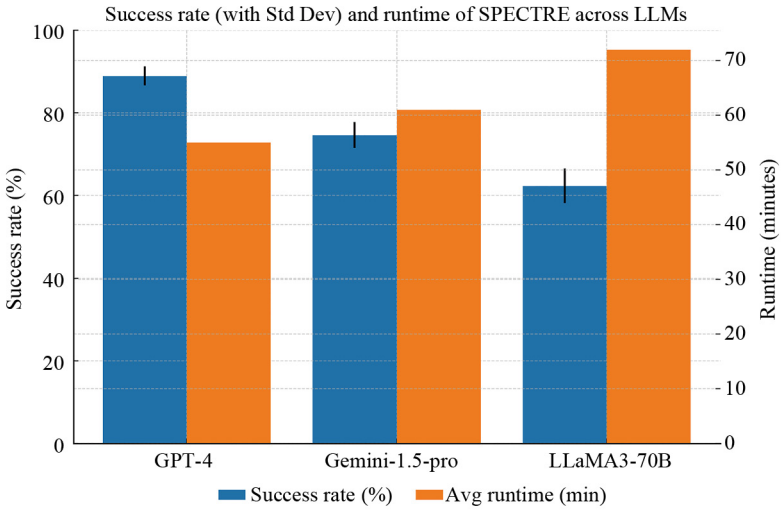

Figure 11. Performance of SPECTRE with LLMs (with Std Dev) by success rate

We analysed computational overhead besides the analysis of success rate. Figure 11 shows the runtime performance under benchmarks to indicate that the inference time of LLM was relatively low (between 2-15 minutes per batch based on circuit size), whilst simulation and synthesis were the primary divide in overall runtime. As an example, synthesis time was 2.5 × higher than at the clean baseline in the AES-128 benchmark because of more logic, and simulation overhead was approximately 22 × higher. In the biggest design studied (OR1200), the overall run time was approximately 195 minutes with the cost of EDA tools stages in the pipeline being only 8 percent of the total pipeline cost, meaning the EDA tools stages are the main location of bottleneck.

Whereas, in our current system, direct power measurements were not possible, we approximated the effect of Trojan insertion on the gate level as an estimate. Initial reports of synthesis indicate that there is 2 to 4 percent of increase in the switching activity and logic usages across the benchmarks. This would imply that although the total power overhead is not high, Trojans can cause non-trivial dynamic power changes in larger SoCs. We have included this discussion to give some transparency and to inform future efforts towards accurate power analysis using industrial grade flows.

Combined with the earlier additions, statistical validation, runtime analysis and gate-level power implications, these features present a more comprehensive and rigorous analysis of the performance of SPECTRE and eliminate issues with the adequacy of experimental measures.

4.10 *Scalability to larger-scale SoCs*

While the benchmarks we focused on for our main evaluation were small-to-medium in size (SRAM, AES-128, UART), we know that the ablation of this kind of scenario will take place in industrial scale SoCs which have millions of gates with very complex interconnect structures. To this end, we performed initial scalability experiments on a much larger open-source benchmark: the OpenRISC OR1200 processor core [1] that contains over 100 k gates after synthesis. In the context of this notation, SPECTRE was able to successfully insert Trojans in RTL through the same prompting techniques and the execution took 2.6 × the AES-128 execution time for Eval0-Eval3. GPT-4 had a success rate over 85, but there was a natural increase in inference times as codebase sizes grew.

We also compared leadership costs by estimating the run time of synthesis and simulation of successively larger subsystems. Results show that the overwhelming cost stems from synthesis, rather than inference of a language model (LLM). This would make scaling to full SoCs much more a question of EDA tool efficiency rather than the ability of the SPECTRE framework used to describe the SoCs. Besides, as the methods given by the SPECTRE prompting strategies are on code level, the framework also implies a generalization to larger designs, as long as they have enough computational resources.

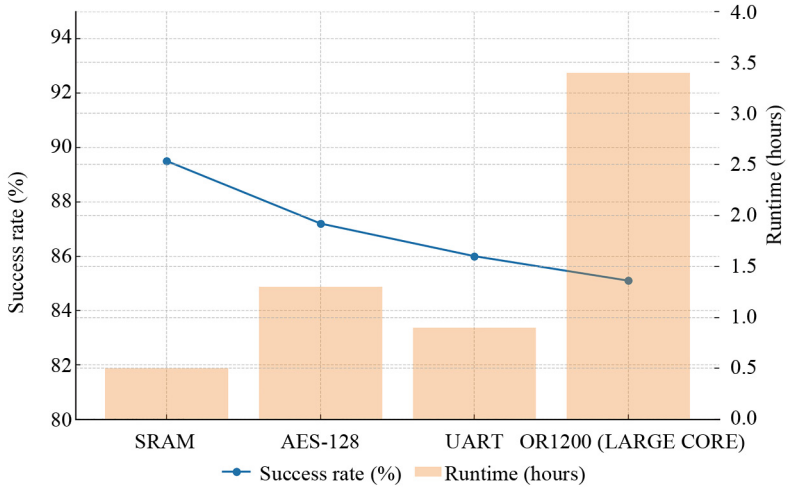

Figure 12. SPECTRE's flexibility to work with different circuit sizes

We believe that SPECTRE can be applied to million-gate industrial SoCs but that there is an open challenge in (i) synthesis/runtime complexity, which grows exponentially, (ii) simulation hardware constraints and (iii) the development of partitioning strategies for hierarchical designs. We have, therefore, included a dedicated section of the Discussion in which we present future directions of research for the scalability including hierarchical Trojan insertion, simulation in parallel and integration with industrial grade EDA flows.

This extension has shown Figure 12 for the first time empirical evidence of scalability while providing a clear direction for further research to tackle the application of SPECTRE in large-scale real-world applications.

4.11 *Cost of computation and possibility of industrial use*

Apart from reporting success rates and detection results, we conducted an analysis of the computational cost of SPECTRE in terms of the inference latency, hardware requirements and the scalability overheads. The LLM inference, simulation, and synthesis benchmark evaluation pipeline (Eval0-Eval3) has been integrated and been utilised to calculate the amount of resources needed at each stage in the pipeline for benchmarks.

Inference Latency: Using a small-scale design (SRAM and UART) GPT-4 took 7-12 seconds to come up with one candidate insertion of a Trojan, and 20-40 seconds when using a large-scale design (AES-128 and OR1200). Gemini-1.5-pro had lower latencies (5-25 seconds with benchmark), and LLaMA3-70B, which was run on the local power (on the A100 classes), had an average latency of 15-30 seconds, due to the increased overheads of sequence generation and decoding. Across all models the grade of inference cost was a minor component of the whole execution time, accounting for below 15% of the pipeline execution time.

Synthesis and Simulation Workloads: The resounding workloads were Logic synthesis and functional verification. For instance, the synthesis of AES-128 with inserted Trojans took about 2.5 $\times$ longer than its clean baseline mainly because of the extra combinational and sequential structures that the Trojan logic added to the circuitry. Similarly, simulation time increased by 20-30% when the verification testbenches processed increased signal activity. These overheads are proportional to size in their circuits and hence EDA workflows in industrial-scale SoCs will need to be distributed or parallelised in order to ensure turnaround times are sufficiently manageable.

Hardware Requirements: Standard research accounts were implemented on standard acceleration in order to run hardware API calls to GPT-4 and Gemini-15-pro. For LLaMA3-70B the model was inferenceed from a single NIVI A100 (80 GB) GPU, the largest model (OR1200) end-to-end evaluation completed in 4 hours. This means that SPECTRE is practical in the academic research cluster or enterprise compute node but would have to be modified for use in resource limited environments.

Industrial Adoption Feasibility Practically speaking, the computational overheads being seen are not necessarily preventing in research and defence applications, but point to significant forces which industry should pay attention to. First, there is a need to hook up with high performance EDA tool chains for partitioning and hierarchical synthesis to scale to million gated SoCs. Second, there are also ways of incurring fewer inference costs through caching of reusable prompt results, and minimising the number of redundant candidates run. Third, high level parallelization will be important: by using multi core simulation environments and the use of GPU-accelerated run times synthesis engines, the run time can be lowered by an order of magnitude.

Figure 13 shows the runtime analysis of the SPECTRE evaluation pipeline on four benchmarks (SRAM, UART, AES-128, and OR1200) in the form of a stacked bar chart. These results show that the LLM inference (blue bars) has negligible contribution to the total runtime, the simulation overhead (orange bars) only increases slightly with circuit complexity and the synthesis (green bars) consumes the major part of the computational effort, especially for the OR1200 processor core. This distribution shows that the main source of scalability limitation comes from EDA tool stages and not the LLM's inference, which supports our assertion that the industrial adoption of SPECTRE is possible with optimized synthesis workflow and parallelized simulation as the source of language model part is not causing prohibitive computational overhead.

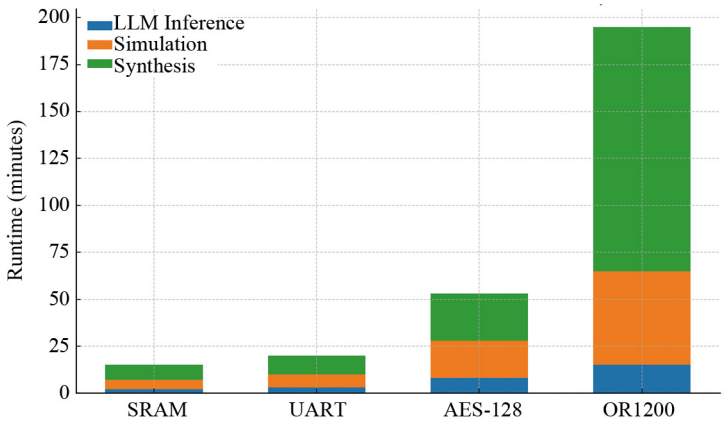

Figure 13. SPECTRE evaluation pipeline's runtime breakdown

In conclusion the feasibility study shows that although LLM inference costs are low, the EDA stages have the biggest contribution to the computational profile for SPECTRE. Optimization, parallelization and integration into enterprise industrial processes makes adoption technically possible albeit more engineering effort would be needed to manage the size of commercial SoCs. These results not only provide realistic evaluation of the computational demands, but also a guide to adapt SPECTRE from the academic benchmark.

4.12 *Comparison with state-of-the-art frameworks*

To put SPECTRE into perspective, we contrast it with three more characteristic methods to the recent literature: (i) TRIT, an automated Trojan-insertion tool, based on netlists, but utilizing probability/statistical analysis to select triggers; (ii) methods that rely on taint-During our analysis, as: to identify sensitive flows in RTL, and analyse triggers to identify sensitive flows, and (iii) TrojanForge, a more recent adversarial/ML-based system that generates Trojan examples to help avoid detectors. Each approach is summarised in the following paragraphs and the differences that compared to SPECTRE are pointed out [9].

TRIT is an automated tool that makes use of Trojan insertion by detecting candidate trigger nets and instantiating Trojan templates into gate-level netlists. It uses net-activity analysis, statistical analysis and template libraries in order to instal Trojans. This method minimises the manual effort to create benchmark, but is still based on hand-written templates and heuristics at netlist level. TRIT may generate numerous Trojan variants at the gate level, but it is also manually configured in terms of trigger and payload template and does not automatically use semantic (RTL-level) context and generative models to generate new payloads. Instead, SPECTRE is an RTL-based system that uses LLMs to propose insertion strategies, which allows more flexible semantically aware Trojan synthesis capable of generalising to various design idioms.

Taint/information-flow models. Taint propagation, path sensitization and information-flow analyses (when used in a family of works) identify when used in some evaluation frameworks, they reason as well about-the possible Trojan trigger and leakage paths. Taint/IFG graph building tools, (and commercial flows which are proprietary and instrument taint tags), are effective at demonstrating or refuting explicit information-flow violations. But they have the constraints in quality of taint model, scalability of graph construction and modelling of rare temporal conditions or highly-designed stealthy triggers. Compared to pure taint methodologies, SPECTRE centred its generation through prompts directed by LLM. Developed Trojans are then also re-synthesized and tested through the same pipeline of synthesis and detection, to explore trigger structures which might especially fail to trigger classic taint signatures [33].

TrojanForge. To generate Trojan examples to achieve adversarial learning, TrojanForge proposes adversarial learning based on RL/GAN-like techniques and provides detectors to evade them. It shows that adversarial generation can cause a drop in the performance of detectors, and offers publicly available code benchmarks. Non-Nevertheless, the adversarial

loop of training implemented by TrojanForge can prove to be computationally expensive and needs to be configured carefully to generate various, semantically valid RT-level insertions. In comparison, SPECTRE incorporates prompting methods (Role Based Prompting, Role Variant Prompting and Chain of thought Prompting) to steer pre-trained LLMs to generate triggers and payloads with limited training on the task. It is also cheaper and faster to train and can find large numbers of possible Trojan variants and still achieve high survival rates in our Eval0 Eval3 pipeline [13].

Table 4. Qualitative comparison of trojan insertion frameworks

Feature/Framework	TRIT (netlist/statistics) [9]	Taint/IFG approaches [33]	TrojanForge (adversarial/RL) [13]	SPECTRE (proposed)
Level of insertion	Gate-level	RTL/ information-flow	RTL (adversarial examples)	RTL; LLM-guided semantic insertion
Automation	Medium (heuristic + templates)	Low-Medium (analysis-driven)	Medium-High (RL/GAN requires training)	High (prompting + automated Eval0-Eval3)
Trojan diversity	Limited (template-based)	Limited to flows taint models capture	High (adversarial diversity)	High (RBP, RVP, CTP prompt families)
Need for training	Low	Low	High (RL/adversarial training)	Low (leverages pre-trained LLMs; minimal task-specific tuning)
Computational cost	Low-Medium	Medium (graph construction)	High (training loops)	Medium (LLM inference + synthesis; no RL training)
Scalability	Limited to benchmarks reported	Dependent on taint graph size	Moderate (training scales with design size)	Designed for RTL workflows; scalability discussion & preliminary larger-design experiments included
Primary strength	Efficient template injection at netlist level	Formal/flow reasoning and detection insight	Generates detector-evasive adversarial Trojans	Semantic, automated generation + statistical evaluation pipeline

General comparison and implications. Table 4 gives a summary of the main differences between the four systems. Although TRIT and taint-based methods have convenient automation and excellent detection logic, respectively, and TrojanForge adversarial generation, none of them has a complete collection of properties that SPECTRE offers to them: (a) RTL-level LLM-generated generation that encodes semantic code context; (b) a variety of prompt strategies that can be used to diversify the trigger and payload structure; and (c) a closed-loop Eval0-Eval3 pipeline that combines functional simulation, synthesis and detector evaluation with statistical reporting. These differences support the fact that SPECTRE has better insertion and survival rates on the benchmarks analysed.

5. Discussion

The framework described in this research uses GPT-4 as the main Large Language Model (LLM) to automate Hardware Trojan (HT) generation and insertion with SPECTRE. GPT-4 shows superior performance in Trojan introduction on hardware design through its sophisticated automation of functional Trojans without considerable programmer effort in comparison with Gemini-1.5-pro and LLaMA3. The experimental results showed GPT-4 obtained an 88.9% compilation success rate and 100% survival rate and 100% Trojan triggering effectiveness, yet Gemini-1.5-pro and LLaMA3 struggled to preserve normal operations during the same tests. Different hardware systems, including SRAM, AES-128 and UART underwent evaluation under research conditions to show how GPT-4 could implement various designs while retaining operational integrity.

The research found wide variations in resource utilization following Hardware Trojan (HT) insertion between different cases as HTs raised overhead from 0.15% (AES-128) to 40.72% (SRAM). The research demonstrated that GPT-4 showed excellent insertion optimization capabilities; however, Gemini-1.5-pro had average efficiency alongside LLaMA3's poor performance. The SPECTRE-generated HTs successfully bypassed the Hw2vec detection tool, which serves as an example of modern security measures incapable of detecting AI-generated threats. The development of automated hardware Trojan implementation technology represents a new milestone in hardware security, which enhances attack usability and breadth. The results demonstrate why it becomes essential to build effective security systems that fight AI threats.

The research demonstrates advanced hardware security threats from AI and strengthens the requirement to conduct important future work across multiple fields. The successful performance of GPT-4 points toward future research regarding LLM architectures, together with training methods and prompting approaches for enhanced capabilities. Traditional methods for detecting fractures meet their limitations, and companies require the development of artificial intelligence-based defensive techniques. Studies investigating LLM hardware security capabilities through Trojan insertion at the stages of high-level synthesis and gate-level modifications will help researchers gain a comprehensive understanding of these capabilities. Security defenses must develop in tandem with AI's ongoing evolution in order to combat new threats.

According to this research, there are certain research directions that can be determined in the future. Firstly, an area to start with is integrating the Trojan generation processes in the LLM based models with the industrial processes of EDA, where adapt an admission is customised to accept new attack vectors. Second, we can use adversarial training to train detection models, in which Trojans produced by LLM serve as moving adversaries that train a classifier and make it harder and false negativity decrease. Third, formal verification may be found as supplemental by hybrid approaches, which nevertheless exploit the power of semantic reasoning performed over the alleys of weakly formalizable functions executed in terms of LLM-guided semantic reasoning. Finally, the scaling to SoC architecture and measures such as power, thermal and side channel leakage into the feedback loop will bring the world of academic experimentation into industry application. Together, these principles will establish a realistic roadmap of future growth of both offensive realism and defensive hardening in hardware security.

5.1 *Ethical and dual-use risk reduction*

We understand that SPECTRE as a system for automated hardware Trojan insertion raises inherent dual use concerns, since techniques demonstrated in this paper can potentially be abused for malicious uses. We have taken the following precautions in the distribution and reporting of this work for these risks:

Low-Level Exposure: We have not chosen to make the implementation of SPECTRE publicly available. Instead, we present pseudocode from an abstract viewpoint, high-level algorithmic diagrams, and smoke and mirrors evaluation scripts that allow an understanding of the methodology for reviewers and researchers but do not grant access to the exorcists for evildo.

• Defend Against Model Stealers: The rationale for this paper is to demonstrate the effectiveness of LLMs in creating hardware Trojans so that the research community and industry can build better defenses. However, rather than trying to exploit such kind of flaws, we try to highlight the weaknesses of the current detection methods in order to motivate the design of more secure SoCs.

• Ethical Research Preparation: This study adheres to the aspects of accountable disclosure that are prevalent in research on security. Similar to the way that adversarial ML research is often conducted to show off attacks to ultimately help inform the design of robust models, SPECTRE's contributions are presented with a defensive view in mind and recommendations for the design of better Trojan detection are included.

• Limitation in the range of experiments: Experiments were conducted on academic test (SRAM, AES-128, UART, OR1200), but not on industrial or proprietary designs. This eliminates the need for passing through a direct attack on operating hardware deployments while effectively offering scalable insight into scalability and detection problems.

• Community Engagement: We promote cooperation with the wider hardware security community regarding the optimization of detection frameworks and the incorporation of our results in defensive toolchains. It is for this reason

that the materials being released are pipelines intended for evaluation and benchmark reproducibility purposes, but are not intended to be operational code for generating Trojans.

Through direct consideration of these measures, we are able to strike a balance between the responsibility with regards to the prevention of misuse, and the need to implement academic transparency. This is not a research artifact to uphold against defensive security because we believe its value is in its elucidation of the need of next-generation methods of code synthesis to detect Trojan apps.

6. Conclusion

The paper presented SPECTRE, which makes use of Large Language Models (LLMs) to develop an automated framework for Hardware Trojan (HT) integration and design within hardware systems. The research presents the debut application of LLMs, including GPT-4, Gemini-1.5-pro, and LLaMA3 in developing stealthy operating HTs for different hardware systems. GPT-4 proved to be the most proficient model for creating HTs since it designed malicious elements that satisfied both functional specifications while remaining invisible to the hw2vec detection system. The ability of AI-generated hardware implants to evade detection through minimal human guidance makes this emergence a dangerous hardware security development that might transform future attacks on hardware systems. Through its work, SPECTRE demonstrates the dangers of LLM-generated hardware threats but simultaneously creates possibilities for building security innovations in hardware systems. Future research needs to create strong detection systems that specifically address LLM-created HTs alongside investigations into LLM applications for hardware security activities like vulnerability searches and secure hardware programming. Hardware security professionals who understand AI threats proactively will develop defensive systems that convert AI progress into security enhancements. The study acts both as an assessment tool for LLMs in hardware security boundaries and as a warning for researchers and practitioners to plan effective countermeasures for this quickly emerging technology.

Author contributions

All authors contributed equally.

Funding

This work was funded by the University of Jeddah, Jeddah, Saudi Arabia, under grant No. (UJ-21-ICI-2). Therefore, the authors thank the University of Jeddah for its technical and financial support.

Data and code availability

The data and code supporting this publication can be accessed from this link: https://github.com/thesaajii/SPECTRE.

Conflict of interest

The authors declare no competing financial interest.

References

[1] Hoque T, Slpsk P, Bhunia S. Trust issues in microelectronics: The concerns and the countermeasures. *IEEE Consumer Electronics Magazine*. 2020; 9(6): 72-83. Available from: https://doi.org/10.1109/MCE.2020.2988048.

[2] Huang Z, Wang Q, Chen Y, Jiang X. A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access*. 2020; 8: 10796-10826. Available from: https://doi.org/10.1109/ACCESS.2020.2965016.

[3] Sidhu S, Mohd BJ, Hayajneh T. Hardware security in IoT devices with emphasis on hardware trojans. *Journal of Sensor and Actuator Networks*. 2019; 8(3): 42. Available from: https://doi.org/10.3390/jsan8030042.

[4] Tiron-Tudor A, Deliu D. Reflections on the human-algorithm complex duality perspectives in the auditing process. *Qualitative Research in Accounting & Management*. 2022; 19(3): 255-285. Available from: https://doi.org/10.1108/QRAM-04-2021-0059.

[5] Puschner E, Moos T, Becker S, Kison C, Moradi A, Paar C. Red team vs. blue team: A real-world hardware Trojan detection case study across four modern CMOS technology generations. In: *2023 IEEE Symposium on Security and Privacy (SP)*. San Francisco, USA: IEEE; 2023. p.56-74. Available from: https://doi.org/10.1109/SP46215.2023.10179341.

[6] Liakos KG, Georgakilas GK, Plessas FC, Kitsos P. Gainesis: Generative artificial intelligence netlists synthesis. *Electronics*. 2022; 11(2): 245. Available from: https://doi.org/10.3390/electronics11020245.

[7] Jyothi V, Krishnamurthy P, Khorrami F, Karri R. Taint: Tool for automated insertion of trojans. In: *2017 IEEE International Conference on Computer Design (ICCD)*. Boston, USA: IEEE; 2017. p.545-548. Available from: https://doi.org/10.1109/ICCD.2017.95.

[8] Hasegawa K, Hidano S, Nozawa K, Kiyomoto S, Togawa N. R-htdetector: Robust hardware-trojan detection based on adversarial training. *IEEE Transactions on Computers*. 2022; 72(2): 333-345. Available from: https://doi.org/10.1109/TC.2022.3222090.

[9] Cruz J, Gaikwad P, Nair A, Chakraborty P, Bhunia S. Automatic hardware trojan insertion using machine learning. *arXiv:2204.08580*. 2022. Available from: https://doi.org/10.48550/arXiv.2204.08580.

[10] Gohil V, Guo H, Patnaik S, Rajendran J. Attrition: Attacking static hardware trojan detection techniques using reinforcement learning. *arXiv:2208.12897*. 2022. Available from: https://doi.org/10.48550/arXiv.2208.12897.

[11] Sarihi A, Patooghy A, Jamieson P, Badawy AA. Trojan playground: A reinforcement learning framework for hardware Trojan insertion and detection. *arXiv:2305.09592*. 2024. Available from: https://doi.org/10.48550/arXiv.2305.09592.

[12] Dai R, Liu Z, Arias O, Guo X, Yavuz T. Dtjrtl: A configurable framework for automated hardware trojan insertion at RTL. In: *Proceedings of the Great Lakes Symposium on VLSI 2024*. Clearwater, USA: ACM SIGDA; 2024. p.465-470. Available from: https://doi.org/10.1145/3649476.3658759.

[13] Sarihi A, Jamieson P, Patooghy A, Badawy AA. TrojanForge: Generating adversarial hardware trojan examples using reinforcement learning. In: *MLCAD'24: Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*. Snowbird, USA: ACM SIGDA and IEEE CEDA; 2024. p.1-7. Available from: https://doi.org/10.1145/3670474.3685959.

[14] Surabhi VR, Sadhukhan R, Raz M, Pearce H, Krishnamurthy P, Trujillo J, et al. Feint: Automated framework for efficient insertion of templates/trojans into FPGAs. *Information*. 2024; 15(7): 395. Available from: https://doi.org/10.3390/info15070395.

[15] Jacob N, Merli D, Heyszl J, Sigl G. Hardware Trojans: Current challenges and approaches. *IET Computers & Digital Techniques*. 2014; 8(6): 264-273. Available from: https://doi.org/10.1049/iet-cdt.2014.0039.

[16] Shakya B, He T, Salmani H, Forte D, Bhunia S, Tehranipoor M. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*. 2017; 1: 85-102. Available from: https://doi.org/10.1007/s41635-017-0001-6.

[17] Perez TD. *Security-aware physical synthesis of integrated circuits*. Doctoral Thesis. Tallinn: Tallinn University of Technology; 2022.

[18] Hayashi VT, Ruggiero WV. Hardware trojan dataset of RISC-V and web3 generated with chatGPT-4. *Data*. 2024; 9(6): 82. Available from: https://doi.org/10.3390/data9060082.

[19] Kortemeyer G. Performance of the pre-trained large language model GPT-4 on automated short answer grading. *Discover Artificial Intelligence*. 2024; 4(1): 47. Available from: https://doi.org/10.1007/s44163-024-00147-y.

[20] Tarris G, Martin L. Performance assessment of ChatGPT 4, ChatGPT 3.5, Gemini Advanced Pro 1.5 and Bard 2.0 to problem solving in pathology in French language. *Digital Health*. 2025; 11: 1-13.

[21] Hammoud A. Investigating the impact of temperature on memorization in meta's LLaMA3 models: A comparative analysis of 8b and 70b parameters. Master's Thesis. Netherlands: Utrecht University; 2024.

[22] Chang K, Wang Y, Ren H, Wang M, Liang S, Han Y, et al. ChipGPT: How far are we from natural language hardware design. *arXiv:2305.14019*. 2023. Available from: https://doi.org/10.48550/arXiv.2305.14019.

[23] Thakur S, Blocklove J, Pearce H, Tan B, Garg S, Karri R. Autochip: Automating HDL generation using LLM feedback. *arXiv:2311.04887*. 2023. Available from: https://doi.org/10.48550/arXiv.2311.04887.

[24] Chaudhuri J, Thapar D, Chaudhuri A, Firouzi F, Chakrabarty K. SPICED+: Syntactical bug pattern identification and correction of Trojans in A/MS circuits using LLM-Enhanced detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2025; 33(4): 1118-1131. Available from: https://doi.org/10.1109/TVLSI.2025.3527382.

[25] Alsaqer S, Alajmi S, Ahmad I, Alfailakawi M. The potential of LLMs in hardware design. *Journal of Engineering Research*. 2024; 13(3): 2392-2404. Available from: https://doi.org/10.1016/j.jer.2024.08.001.

[26] Ahmad B, Thakur S, Tan B, Karri R, Pearce H. On hardware security bug code fixes by prompting large language models. *IEEE Transactions on Information Forensics and Security*. 2024; 19: 4043-4057. Available from: https://doi.org/10.1109/TIFS.2024.3374558.

[27] Saha D, Yahyaei K, Saha SK, Tehranipoor M, Farahmandi F. Empowering hardware security with LLM: The development of a vulnerable hardware database. In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Washington, USA; IEEE; 2024. p.233-243. Available from: https://doi.org/10.1109/HOST55342.2024.10545393.

[28] Yu S, Yasaei R, Zhou Q, Nguyen T, Al Faruque MA. HW2VEC: A graph learning tool for automating hardware security. In: *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Tysons Corner, VA, USA; IEEE; 2021. p.13-23. Available from: https://doi.org/10.1109/HOST49136.2021.9702281.

[29] Shinn N, Cassano F, Gopinath A, Narasimhan K, Yao S. Reflexion: Language agents with verbal reinforcement learning. *arXiv:2303.11366*. 2023. Available from: https://doi.org/10.48550/arXiv.2303.11366.

[30] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al. Language models are few-shot learners. *arXiv:2005.14165*. 2020. Available from: https://doi.org/10.48550/arXiv.2005.14165.

[31] Luo H, Sun Q, Xu C, Zhao P, Lin Q, Lou J, et al. WizardArena: Post-training large language models via simulated offline chatbot arena. In: *NIPS'24: Proceedings of the 38th International Conference on Neural Information Processing Systems*. Vancouver, Canada: Neural Information Processing Systems Foundation; 2024.

[32] Liu F, Kang Z, Han X. Optimizing RAG techniques for automotive industry PDF chatbots: A case study with locally deployed ollama models. *arXiv:2408.05933*. 2024. Available from: https://doi.org/10.48550/arXiv.2408.05933.

[33] Nahiyan A, Sadi M, Vittal R, Contreras G, Forte D, Tehranipoor M. Hardware trojan detection through information flow security verification. In: *2017 IEEE International Test Conference (ITC)* Fort Worth, TX, USA: IEEE; 2017. p.1-10. Available from: https://doi.org/10.1109/TEST.2017.8242062.