








NETLAM: An Automated LLM Framework to Generate and Evaluate Stealthy Hardware Trojans

Tishya Sarma Sarkar¹, Kislay Arya², Siddhartha Chowdhury¹,
Upasana Mandal¹, Shubhi Shukla³, Sarani Bhattacharya¹, and
Debdeep Mukhopadhyay¹

¹ Department of Computer Science and Engineering,
Indian Institute of Technology, Kharagpur

² Department of Electronics and Electrical Communication Engineering,
Indian Institute of Technology, Kharagpur

³ Centre for Computational and Data Sciences,
Indian Institute of Technology, Kharagpur
{tishya, kislaiyarya536}@kgpian.iitkgp.ac.in,
siddhartha.chowdhury92@gmail.com,
{mandal.up98, shubhishukla}@kgpian.iitkgp.ac.in,
{sarani, debdeep}@cse.iitkgp.ac.in

Abstract. Securing externally sourced hardware designs is essential to prevent adversaries from embedding hardware Trojans. Trojans are stealthy modifications that leak data or create backdoors. Existing benchmarks like *Trust-Hub* provide only a limited set of Trojans (106), while the possibilities are virtually infinite. To address this, we propose **NETLAM**, a comprehensive framework utilizing multiple LLM-based tools to generate previously undiscovered Trojans not included in *Trust-Hub*. The first tool converts hardware netlists into Directed Acyclic Graphs (DAGs) to identify vulnerable nets and components in digital designs. Using these insights, the second tool generates stealthy Trojan-infected versions of the original design. To evaluate the stealthiness of these Trojans, we use an LLM-based equivalence checker, where stealthier Trojans pass equivalence checks while others are detected. We evaluate **NETLAM** using the AES dataset from *Trust-Hub* consisting of 28 Trojans. We identified 5 new Trojans, with high Common Vulnerability Scoring System (CVSS) scores, demonstrating their stealthiness. To prove the efficacy of the **NETLAM** generated Trojans, we further utilize an open-source formal equivalence checker to perform a functional equivalence check between the golden and the **NETLAM** generated Trojan-infected circuits. All of the suggested Trojans pass the formal equivalence check. However, the same Trojan-infested circuits fail in the **NETLAM** equivalence test, thus validating the effectiveness of our proposed framework. We show that LLMs and Generative AI models, such as GPT-4o and Gemini, can enhance Trojan detection by using semantic and probabilistic analysis rather than strict logical equivalence⁴.

⁴ GitHub Repository:
<https://github.com/shubhishukla10/NETLAM>

Keywords: Hardware Trojans, Large Language Models, Vulnerability Detector, Equivalence Checker, Directed Acyclic Graphs

1 Introduction

In today’s technology-driven world, hardware devices are everywhere, supporting important applications in communication, healthcare, defense, transportation, and consumer electronics. These devices are the foundation of modern infrastructure, making their security and reliability very important. However, the fast growth of hardware components and their dependence on complex global supply chains have created new risks. One of the most serious threats is hardware Trojans, which are hidden and dangerous modifications that can harm the functionality and trustworthiness of these devices.

Hardware Trojans (HTs) are harmful changes intentionally added to digital designs to disrupt their normal behaviour, steal sensitive information, or disable important functions at the right moment. These changes can take different forms, such as adding extra logic gates, altering connections, or creating small timing changes [15]. Trojans are usually made to stay hidden during normal use and testing, only becoming active under specific conditions, which makes them very hard to detect. The effects of these attacks can be severe: in defense systems, they can threaten national security, while in consumer devices, they can cause serious privacy leaks and financial losses. Consequently, identifying, mitigating, and minimizing these risks has emerged as a crucial research focus.

Existing research in HTs largely focuses on either insertion or detection, often addressing them in isolation. For instance, [1] utilizes large language models (LLMs) to autonomously generate synthesizable HTs from high-level design specifications, enabling rapid exploration of the attack surface. Similarly, [4] applies machine learning techniques to dynamically insert Trojans by analyzing structural and functional features of existing designs. Expanding on these efforts, [10] demonstrates the application of LLMs in offensive hardware security, showcasing their ability to identify vulnerable modules, insert HTs, and craft sophisticated attacks on complex designs like RISC-V CPUs. The study addresses LLM context-length limitations by focusing on register transfer level (RTL) code, validating the inserted HTs through FPGA-based testing on CPU integrity and availability. On the detection front, [3] introduces SPICED, an LLM-based framework for detecting and localizing analog Trojans and syntactical bugs in circuit netlists. By leveraging chain-of-thought reasoning and few-shot learning, SPICED achieves 93.32% average Trojan coverage and 93.4% true positive rate on analog benchmarks without requiring hardware modifications or explicit training, providing an effective software-based solution for analog and mixed-signal circuits. Furthermore, [19] presents a golden reference-free HT detection method using Graph Neural Networks (GNNs). Representing hardware designs as Data Flow Graphs (DFGs), this approach achieves 97% recall for RTL in 21.1ms and 84% recall for gate-level netlists in 13.42s, demonstrating scalability and efficacy. Additionally, [18] proposes a golden-free multidimen-

sional self-referencing technique, which detects hardware Trojans by analyzing side-channel signatures in time and frequency domains, expanding Trojan coverage and reducing process variations. The automated framework includes test generation, signal processing, and decision-making, achieving high detection sensitivity. Evaluated on 96 Trojan-inserted chips, it effectively detects even small, hard-to-detect Trojans. Although these advancements have improved detection techniques, pre-silicon verification continues to face challenges due to security gaps. Such security gaps encompass incomplete threat models, limited detection of hardware Trojans, insufficient formal verification procedures, inadequate test input sets during simulation for rare events, and most importantly, hidden vulnerabilities in the RTL designs. To address this, a formal verification approach [5] has been introduced, enabling exhaustive detection of sequential HTs at the RTL level, independent of golden models or payload behavior, effectively identifying complex HTs in *Trust-Hub* accelerators. Apart from these, Vul-FSM [12] introduces a database of 10,000 vulnerable finite state machine (FSM) designs, generated using the SecRT-LLM framework. Leveraging LLMs like GPT-3.5-turbo, the framework achieves high accuracy in vulnerability insertion and detection, showcasing its potential for efficiently creating and analyzing hardware security benchmarks. *Despite significant advancements, current research lacks a comprehensive tool capable of identifying all potential Trojans that could be inserted into a given design. While existing repositories like Trust-Hub [13], [14] provide only a handful of Trojan examples, and tools like Vul-FSM [12] focus on FSM vulnerabilities, they represent just a fraction of the infinite possibilities. This limitation highlights the need for an automated and versatile solution to systematically discover, analyze, and generate new, previously unknown Trojans across diverse hardware designs, motivating us to develop such a tool.*

In this work, we introduce **NETLAM**, a comprehensive framework leveraging multiple LLM-based tools, developed by us, with distinct functionalities to identify, generate, and evaluate Trojans within a given design. First, we develop an LLM-based equivalence checker and demonstrate its effectiveness using Trojan examples from *Trust-Hub*. Next, we create an LLM-based tool to convert digital netlists or Verilog code into directed acyclic graphs (DAGs), enabling the identification of potential Trojan insertion points. This step is crucial because it gives a direct insight into the vulnerable and candidate trigger points of the hardware design under test. Thus, it allows the designer to secure the design to avoid possible threats. Finally, we design an LLM-based Trojan injector capable of generating Verilog code with injected Trojans. Together, these tools form the **NETLAM** framework, which takes a design as input, identifies vulnerabilities by analyzing DAGs generated from the netlist, produces Trojan-injected Verilog code based on these vulnerabilities, and evaluates the stealthiness of the Trojans using the equivalence checker. We specifically evaluated our tool on the AES design from *Trust-Hub*, which includes 28 reported Trojans. Our tool successfully identified and generated 5 new Trojans that were not part of *Trust-Hub*, with high CVSS scores. For ranking the effectiveness of the **NETLAM** generated Trojans, we further test their equivalence with the golden designs using a formal

equivalence checker, EQY, which is included in the Yosys synthesis suite [9], [17]. EQY is an open-source automated tool that performs a logical and functional equivalence between two given designs by exploiting miter circuits. As a result, a formal functional uniformity is examined. Nonetheless, there are innumerable instances of Trojans, which surreptitiously hide within the design and get activated by rare inputs or triggers. In such cases, a functional equivalence might not always provide an intended result and can render a Trojan-infected design equivalent to the golden design due to its sporadic nature. At this point, our LLM-based equivalence checker can come to the rescue by building a graph-based representation of the circuit and detecting redundant or suspicious logic paths. Our LLM-based equivalence checker also provides a semantic analysis of the two designs under test, thus detecting any additional circuitry or variable introduced in the victim modules. This step allows us to claim that our proposed tool, **NETLAM** succinctly generates and evaluates powerful Trojans.

1.1 Contributions:

The contributions of this work are as follows:

- **LLM-Based Equivalence Checker:** We develop an LLM-based equivalence checker capable of verifying the functional equivalence between designs and identifying Trojan injections. Its effectiveness is demonstrated through evaluation on *Trust-Hub* Trojan examples.
- **LLM-Based DAG Generator and Vulnerability Identifier:** We propose a tool that converts hardware netlists or Verilog code into directed acyclic graphs (DAGs) and identifies the most stealthy Trojan-vulnerable points within the design, highlighting potential insertion locations.
- **LLM-Based Trojan Injector:** We introduce an LLM-based Trojan injector that generates Verilog code with stealthily inserted Trojans, creating novel Trojan-infected designs.
- **Comprehensive Framework - NETLAM:** We consolidate the above tools into **NETLAM**, a unified framework that takes a design as input, identifies vulnerabilities via DAG generation, generates Trojan-injected Verilog code from identified vulnerabilities, and evaluates the stealthiness of the generated Trojans using the equivalence checker.

The rest of the paper is organized as follows. Section 2 provides essential background on HTs, LLMs, and equivalence checking, while reviewing existing methods in HT detection and insertion. Section 3 describes the **NETLAM** framework, including its three key components: the LLM-based DAG generator for identifying vulnerabilities, the Trojan injector for creating stealthy HTs, and the equivalence checker for validating stealthiness. Section 4 details the experimental setup and evaluates **NETLAM** on AES designs from *Trust-Hub*, showcasing its ability to generate novel Trojans with high CVSS scores. Finally, Section 5 concludes the paper, summarizing its contributions and emphasizing the need for an integrated framework to advance HT research.

2 Background

2.1 Large Language Models (LLMs)

Large Language Models (LLMs) such as GPT, LLaMa, and Gemini are based on the transformer architecture, which leverages the attention mechanism to compute dynamic relevance scores for input tokens. This enables the model to focus on semantically significant parts of the input. The self-attention module, a core component, computes contextual dependencies across the entire input sequence in parallel, overcoming the sequential limitations of RNNs [11] and LSTMs [7]. Additionally, positional encodings are employed to encode token order, compensating for the model’s inherent lack of sequential processing capability. In this work, we utilize multiple LLMs to develop our tool **NETLAM**, which can generate stealthy and robust trojans injected benchmarks.

2.2 Hardware Trojans

Hardware Trojans (HTs) [16] threaten the security of integrated circuits, especially in third-party design and manufacturing. These stealthy modifications can compromise system confidentiality, integrity, and availability, often lying dormant until triggered. Detection techniques like side-channel analysis, runtime monitoring, and formal verification, alongside advanced methods using machine learning and Hardware Performance Counters (HPCs), help identify anomalies. Prevention strategies include split manufacturing, logic locking, and post-silicon power and delay analysis. As hardware underpins critical systems like smart grids, IoT, and autonomous vehicles, robust defenses against HTs are vital for ensuring security and reliability.

2.3 Register Transfer Level (RTL) and Gate-Level Netlists

In digital design, netlists serve as textual descriptions of circuits, detailing their components and interconnections. These representations exist at varying levels of abstraction, with the most common being the Register-Transfer Level (RTL) and gate-level. RTL netlists depict a design at a high level of abstraction, focusing on the flow of data and control signals between registers. Written in hardware description languages (HDLs) such as Verilog or VHDL, RTL netlists emphasize functionality over implementation specifics. They describe algorithmic behavior using constructs like conditional statements, loops, and procedural blocks, enabling designers to focus on the logic and operation of the circuit.

Gate-level netlists, on the other hand, offer a detailed view of the circuit after synthesis. These netlists consist of interconnected logic gates, flip-flops, and hardware primitives, representing the actual implementation of the design based on a specific technology library. Gate-level netlists are essential for detailed analysis tasks such as timing verification, power estimation, and functional equivalence checking. Bridging the abstraction gap between RTL and gate-level netlists is a critical aspect of hardware verification. Ensuring that the synthesized gate-level

netlist faithfully implements the intended behavior of the RTL design is key to detecting functional errors and identifying potential hardware Trojans.

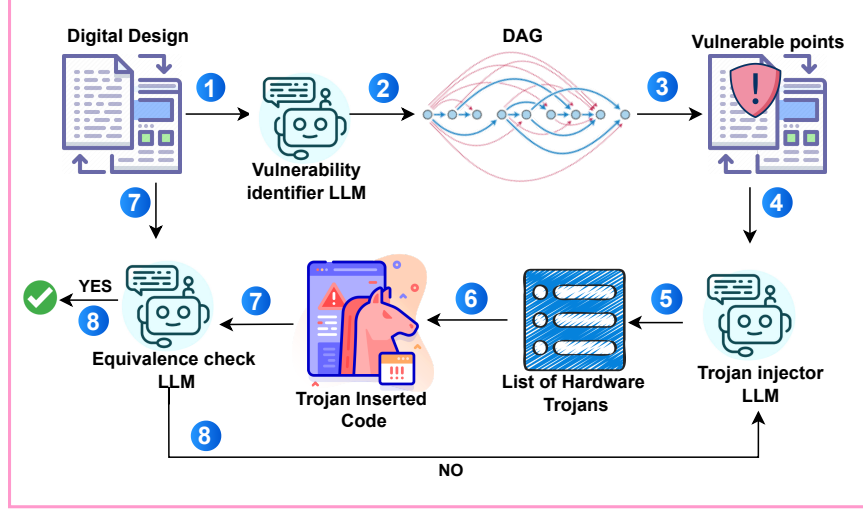


Fig. 1. Workflow of the Proposed **NETLAM** Framework. The three LLM modules, labeled as Vulnerability Identifier LLM, Trojan Injector LLM, and Equivalence Check LLM, are three separate LLM instantiations. The DAG generated by the Vulnerability Identifier is an adjacency list representation of the complex netlist. The vulnerable points include the probable trigger nets in the design. The Trojan inserted code is a maliciously modified version of the target Verilog module.

3 NETLAM Framework

We propose an automated framework, **NETLAM**, that integrates three powerful LLM-based tools: a DAG generator and vulnerability identifier, a Trojan injector, and an equivalence checker. Together, these tools systematically identify vulnerabilities, generate stealthy hardware Trojans, and evaluate their impact, providing a comprehensive solution for uncovering new Trojans and enhancing existing benchmarks like *Trust-Hub*, which is currently limited to 106 Trojan examples. **NETLAM** aims to create a significantly larger set of Trojans while ensuring their stealthiness through advanced equivalence evaluation. Simultaneously, our methodology aims to assist a designer to assess the susceptibility of the RTL design under test to probable threats by exploring the vulnerable nets and unknown hardware Trojans that can be implanted in those points. This methodology is compatible with two state-of-the-art LLM models, Gemini-1.5-flash00

and ChatGPT-4, to enable robust and automated hardware security analysis. The workflow for **NETLAM** is illustrated in Figure 1, wherein the input is an RTL design or a gate-level netlist and the output is a set of undiscovered hardware Trojans that pass the equivalence check.

1. Instruction for Vulnerability Identification and DAG Creation

The following are two synthesized gate-level netlists. You are tasked with the following:

1. DAG Conversion and Analysis: Give a small description of the circuit, what it does and what are the components involved. Convert the Verilog code to a Directed Acyclic Graph (DAG) to map all signals, registers, and logic blocks. Provide the DAG in a clear adjacency list format for readability and the nodes in the DAG must be -
 - a. input and output ports and registers
 - b. wires
 - c. input and output ports of the components.
2. Identify Vulnerable Points: Detect vulnerable locations (e.g., specific registers, data paths, logic gates) in the first netlist that could be exploited for hardware Trojans utilizing the DAG. Evaluate vulnerabilities based on placement, signal flow, and sensitivity to conditions. For each vulnerable point, recommend the most stealthy Trojan types that should pass any functional equivalence check with the original design, including: Trigger mechanisms (e.g., specific input sequences, clock cycles) Payload actions (e.g., bit-flipping, signal tampering) Stealth features (e.g., random activation, intermittent triggering) Rank the top 5 Trojans with descriptions that align with the vulnerabilities identified. Print the DAG as an adjacency list. List the top 5 Trojans with targeted vulnerable points in the design. Rank the Trojans using the CVSS framework, evaluating its stealth, impact, and exploitability.

3.1 Vulnerability Identifier and DAG Generator

The first tool in **NETLAM** systematically identifies vulnerabilities in Verilog codes, RTL, and gate-level netlists. It converts the design into a DAG representation, revealing the logical flow of operations and structural dependencies. We provide specific preliminary instructions to the LLM which broadly include (i) Understanding the functionality of each module of the provided design, (ii) Constructing a DAG specific to each module in the form of an adjacency matrix, (iii) Identifying all the vulnerable points or modules from the DAG where critical Trojans can be inserted. A snippet of the predefined instruction is provided in the listing **1. Instruction for Vulnerability Identification and DAG Creation**. We provide complete instructions in the GitHub repository. The DAG serves as a blueprint for identifying exploitable points in the design. The LLM evaluates the DAG using a context-aware analysis, considering critical factors such as timing, data dependencies, and logic sensitivity. This analysis identifies

the top five most stealthy Trojan-vulnerable points, which may include specific registers, signals, or logic blocks. These points represent locations where Trojans can be inserted with minimal functional disturbance, providing a detailed vulnerability map to guide the Trojan injection phase.

2. Instruction for Trojan Generation

The following are the Verilog codes of an AES block cipher design. You are tasked with the following:

1. Identify the ‘S’ module in the provided Verilog files.
2. Implement a Gate-Level Trojan only in the ‘S’ module which gets triggered by specific input combinations.
 - Do not modify any other modules.
 - Ensure there are no apparent hints of the Trojan inclusion.
 - Make the Trojan as stealthy as possible.
3. Modify the original Verilog code by inserting the newly suggested Trojan into the identified vulnerable points. Ensure that the modified code integrates seamlessly into the design without introducing apparent anomalies. Test the new Trojan-injected design for operational correctness while ensuring the Trojan remains functional. After creating the new Trojan, rank it using the CVSS framework, evaluating its stealth, impact, and exploitability. Provide a justified CVSS score and a comparison with the original Trojan to highlight improvements in stealth and effectiveness.
4. Provide the entire modified Verilog code with well-labeled comments where the Trojan is included.

3.2 Trojan Injector and Trojan Generation

The second tool focuses on generating Trojan-injected versions of the original design based on the vulnerabilities identified in the DAG analysis. We provide specific preliminary instructions to the LLM to build this tool which broadly include (i) Identifying the top 5 stealthy Trojans that can be inserted in the specific vulnerable modules, (ii) Ranking the suggested Trojans based on their CVSS scores, and (iii) Modifying the existing module and generating the Trojan-infected module. A snippet of the detailed instruction is provided in the listing **2. Instruction for Trojan Generation**. We provide complete instructions in the GitHub repository. This Trojan injector creates modified Verilog code, embedding stealthy hardware Trojans that exploit the identified vulnerabilities while ensuring operational correctness. The tool classifies the generated Trojans into categories, such as key-dependent, payload-triggered, or side-channel-based, and evaluates their potential impact on the design’s behavior and security. To enhance stealth, techniques such as delayed activation or seamless integration into legitimate components are employed. The Trojan injector generates a range of Trojans, from simple to highly stealthy, broadening the scope of Trojan possibilities for hardware designs.

3. Instruction for Equivalence Check

The following are two synthesized gate-level representations of two Verilog designs. You are tasked with the following:

1. Construct a miter circuit to check the functional equivalence of the two provided gate-level netlists. Provide a detailed analysis of their functional behavior and determine if they are equivalent at the RTL-level, despite having varying top modules or additional sub-modules. Focus on input-output behavior and logical equivalence, ignoring optimizations or naming differences or gate implementations or additional Verilog modules. Is there any additional circuitry or malicious alteration in any of the two files?

2. If a Trojan is detected in either netlist, analyze its characteristics and impact on the design. Start by identifying the Trojan type, such as key-dependent, payload-triggered, or side-channel-based, through structural and behavioral analysis of the netlist. Next, assess how the Trojan affects the overall design, focusing on vulnerabilities introduced, disruptions caused to timing or functionality, and potential security implications. Provide a thorough explanation of the Trojan's behavior and its operational consequences.

3. Based on the equivalence check outcome, determine whether the Trojan passes or fails the check. If the Trojan remains undetected during functional equivalence testing, evaluate its stealth and impact using the Common Vulnerability Scoring System (CVSS). Rank the Trojan and justify its score with detailed observations. If the Trojan is easily detected during equivalence testing, proceed to create a more stealthy alternative. Identify vulnerable points or modules in the design, such as unused registers, infrequent logic, or weak constraints, and propose a new Trojan designed to blend into the original structure using techniques like delayed activation, payload encryption, or integration into legitimate components.

3.3 Equivalence Checker and Stealth Assessment

The procedure of building the third tool, an equivalence checker, involves converting a complex digital design into gate-level netlists using *Yosys* [17], an open-source Verilog RTL synthesis tool, to synthesize Verilog files for functional equivalence analysis. The golden reference and Trojan-infected Verilog source codes are sourced from the AES-based Trojan attack variants provided in *Trust-Hub* [13], [14]. Each Verilog file undergoes a sequence of transformations in *Yosys*, including reading the design, mapping it to standard cells using a provided library file, optimizing the design, and generating mapped netlists.

We use the *freePDK45 45 nm* [8] variant open-source Process Design Kit (PDK) to map the synthesized Verilog designs into the gate-level netlists. These netlists are then fed into a Generative AI model as part of a carefully constructed and pre-defined prompt, requesting an analysis to determine their functional equivalence based on logical behavior and structural characteristics. This approach avoids the need to upload files to external services by directly embedding the netlist content into the prompt, thereby streamlining the process. The AI's response provides a comprehensive comparison and insight into the equivalence or differences between the two Verilog designs.

In scenarios where a Verilog designer needs to verify whether an outsourced design has been tampered with or contains malicious alterations, the equivalence checker proves invaluable. The designer inputs the golden reference design alongside the manufactured IC design into the checker, which analyzes the two netlists for additional logic blocks, structural modifications, or any alterations that could pose security threats. The equivalence checker operates based on pre-defined instructions to detect such malicious changes, without any prior knowledge of the presence, absence, or potential types of Trojans, or which netlist is Trojan-infected in the provided designs. We provide detailed preliminary instructions to the LLM to build our tool, which broadly includes (i) Performing a rigorous functional equivalence checking of the two gate-level netlists, ignoring any additional modules or input/output variables. (ii) Justifying whether the modified module passes or fails the checking procedure, (iii) If any Trojan is detected, then rank the same using CVSS score and provide a more stealthy Trojan, which should have a higher CVSS score than the provided one. A snippet of the detailed instruction is provided in the listing **3. Instruction for Equivalence Check**. We provide complete instruction in the GitHub repository. Importantly, the checker requires gate-level netlists for accurate analysis, as Verilog source codes often include procedural blocks that are challenging to model effectively.

If the equivalence checker identifies any such malicious alterations in either of the netlists in comparison to the other, it proceeds to determine the type of Trojan that has been inserted in the victim netlist. The stealth of the Trojan is decided based on a hypothetical Common Vulnerability Scoring System (CVSS) score. The CVSS score quantifies the severity of security vulnerabilities. It considers factors like exploitability and impact to assign a numerical rating between 0 and 10, where 0 indicates no severity and 10 indicates critical severity. This process also enables the designer to assess the impact of the inserted Trojan and identify vulnerable points within the design. The equivalence checker’s output is further leveraged by the **NETLAM** tool to pinpoint vulnerabilities and explore the possibility of inserting alternative Trojans with higher CVSS scores into the same design. To further validate the credibility of our tool and the stealth of the generated Trojans, we evaluate the equivalence between the golden design and the Trojan-infected design using the formal equivalence checker (EQY) included within the Yosys suite [9]. This is a miter circuit-based equivalence checker, which strategically checks the functional equivalence between two designs by comparing the outputs of the two designs. If the outputs vary then there is a potential modification in either of the designs.

We now evaluate this comprehensive framework in the next section by uncovering new stealthy Trojans that were previously unknown.

4 Experimental results

In this section, we discuss the comprehensive results of our proposed tools. We specifically target the AES-based Trojan attacks included in *Trust-Hub* for analyzing the security-critical aspect of our proposed tool. The authors propose an

Table 1. Comparison of **NETLAM** with state-of-the-art tools for hardware Trojan generation.

Feature	NETLAM	[1]	[12]	[2]	[4]	[3]
Approach	Uses LLMs for vulnerability identification, Trojan insertion, and equivalence checking	Uses LLMs to analyze and generate Trojans based on <i>Trust-Hub</i> benchmarks	Uses LLMs for vulnerability insertion, detection, and validation using fidelity checks	Automated framework for generating and benchmarking Trojans on PCB designs	ML-based Trojan detection using trigger-net classification	Uses LLMs to detect and localize stealthy Trojans in circuit netlists using chain-of-thought reasoning
Equivalence Checking	LLM-based equivalence checker with functional analysis	Not mentioned	Hybrid verification approach combining static analysis, formal verification, and manual review	Not a primary focus	Not mentioned	Not applicable (focused on analog circuit analysis)
Vulnerability Analysis	Identifies and ranks vulnerable points in RTL/netlists using DAG representation	Cannot identify vulnerable points in the design	Uses LLMs for vulnerability detection, categorization, and validation	Analyzes trigger and payload selection for board-level Trojans	Uses ML models to classify trigger nets based on probability estimates	Cannot identify vulnerable points in the design
Trigger Net Selection	Identifies stealthy nets using DAG-based vulnerability mapping	User prompt-based trigger net selection	Uses guided prompting for vulnerability selection	Uses signal probability analysis for rare node detection	Uses ML-based classification to predict high-probability trigger nets	Extracts trigger locations based on LLM prompts
Learning Time	Uses pre-trained LLMs, reducing training overhead	Uses pre-trained LLMs, reducing training overhead	Uses GPT models with iterative refinement to improve accuracy	No explicit learning; relies on automation heuristics	Requires training time for ML models	Uses pre-trained LLMs, reducing training overhead
Scalability	Can be extended to any RTL designs	Platform-independent but focused on FPGA/ASIC designs	Evaluates 10,000 FSM-based designs for security weaknesses	Benchmarked multiple PCB designs with different Trojan instances	Scales well with ML training but requires dataset availability	Analog/mixed signal circuits only
Automation Level	Fully automated	Fully automated	Semi-automated, requiring manual validation in some cases	Fully automated	Semi-automated	Fully automated
Integration with EDA Tools	Uses Yosys for netlist processing and formal verification	Works with FPGA tools (Xilinx Vivado)	Integrates JasperGold Superlint and ARC-FSM for verification	Primarily uses a custom Trojan insertion tool	Integrates with Synopsys and Cadence ATPG tools for Trojan validation	No integration with traditional EDA tools

LLM-based framework capable of generating stealthy Trojans in [1]. However, they do not assess the vulnerabilities of the victim RTL designs. On the other hand, [3] proposes a tool that identifies stealthy analog Trojans, but does not insert them. Moreover, this work does not explore the vulnerabilities in a digital design. The authors of [12] propose a comprehensive benchmark dataset based on finite state machines (FSMs) and proposes a semi-automated tool to evaluate 10,000 FSM-based designs. However, this work uses guided prompting for vulnerability selection. As our tool comprehensively discovers the probable trigger nets/payloads in the design from the generated DAGs, it provides a better understanding of the design. Specifically, a designer can also use the vulnerability detector before manufacturing to discover any design shortcomings. A detailed state-of-the-art survey is provided in Table 1. During equivalence checking, our tool has effectively detected several highly stealthy Trojans from the benchmark dataset provided by *Trust-Hub*, which are otherwise challenging to identify during the standard operation of an AES encryption process. The Trojan-infected designs comprise additional modules that make the detection of Trojans easier. Moreover, the synthesized netlists of the same comprise additional circuitry as compared to the original design, further altering the structural uniformity. Few of the state-of-the-art Trojans such as the side-channel-based or payload-triggered Trojans do not alter the normal functionality, thus making detection difficult.

Table 2. Comprehensive Results of Equivalence Checking

Sl. No.	AES Variant	Equivalence Check (EQY)	Equivalence Check (NETLAM)	Trojan Provided	CVSS Score
1	T100	Fail	Fail	Side-channel, key-dependent	7.5
2	T1000	Fail	Fail	Payload-triggered	7.8
3	T1100	Fail	Pass	Side-channel-based	8
4	T1200	Fail	Fail	Side-channel-based	7.5
5	T1300	Pass	Fail	Data-dependent	9
6	T200	Fail	Fail	Data-dependent	8.5
7	T300	Pass	Pass	Side-channel-based	9.6
8	T900	Fail	Pass	Payload-triggered, key-dependent, side-channel Trojan	9.8
9	T1500	Pass	Pass	Side-channel-based	9.8
10	T700	Fail	Fail	Payload-triggered	7.8

Among the 28 instances of AES-based Trojan attacks, Table 2 showcases 10 of the most powerful Trojan-infected AES designs. We carry out the equivalence in two methods: preliminarily, we utilize the formal equivalence checker of Yosys, EQY, to find the functional equivalence between the golden design and the Trojan-infested designs. This gives us a comprehensive idea of the functional and logical similarities/dissimilarities. Finally, we check the equivalence by ex-

exploiting our LLM-based equivalence checker. This allows us to further extend the investigation if the Trojan is not visible by EQY. Out of the 10 variants, 7 fails the functional equivalence check using EQY, while the remaining 3 passes due to their high level of stealth. Although most of the Trojans are very difficult to trace, a considerable number of the Trojans are identifiable by our LLM-based equivalence checker. **NETLAM** equivalence checker can successfully detect 6 Trojans. The Trojans of the attack variants that pass the equivalence test are mainly side-channel-based, data or key-dependent and payload-triggered Trojans.

4. Functional Equivalence Check

A direct comparison of File 1 and File 2 reveals significant differences. While both files implement an AES-128 encryption algorithm with similar sub-modules (S, S4, T, expand_key_128, final_round, one_round, table_lookup), File 2 contains an additional module: 'TSC' and 'Trojan_Trigger'. File 1's 'top' module only includes an AES-128 instance, while File 2's 'top' module includes the AES-128 instance, a 'Trojan_Trigger' module, and a 'TSC' module. The 'TSC' module acts as a Trojan. It takes the key and data as input and, when triggered by 'Tj_Trig', produces an output ('Capacitance') that is derived from the input data and XORed with portions of the key. This output is not part of the legitimate AES-128 functionality. The Trojan_Trigger module generates the trigger signal 'Tj_Trig' based on a large counter, making its activation seemingly random and hard to predict during normal operation. The counter's logic seems convoluted and designed to make it difficult to analyze at a glance. Therefore, the two netlists are **not functionally equivalent**. File 2 includes additional circuitry in the form of a Trojan that leaks information.

Side-channel-based Trojans are difficult to detect through functional equivalence since they do not alter the primary operation of the circuit. Data or Key-dependent Trojans, on the other hand, remain dormant for extended periods, becoming active only when the target key or data pattern appears, making them extremely hard to detect during standard testing. Payload-triggered Trojans are triggered only when the rare signal pattern occurs. The corresponding CVSS scores of the undetected Trojans are considerably high, with the side-channel-based Trojans being the most critical ones. The outputs of the equivalence checker, provided in the listings **4. Functional Equivalence Check** and **5. Trojan Detection and Stealth Evaluation** portray the structural as well as functional discrepancies among the Trojan-free and Trojan-infected designs of the **AES-T1200** variant. Additionally, the listing **5. Trojan Detection and Stealth Evaluation** presents the strength of the particular Trojan. Furthermore, our proposed **NETLAM** tool performs an overview of the different sub-modules present in the design, describing the functionality of each of them to better understand the vulnerable sites where Trojans can be inserted. Such an output is provided in the listing **6. NETLAM output for Circuit Description**.

5. Trojan Detection and Stealth Evaluation

The Trojan fails the simple functional equivalence check because of the added modules and outputs. However, detecting it requires a more sophisticated side-channel analysis method rather than a simple input-output comparison.

To evaluate the stealth of the original Trojan, we can use the CVSS framework. A possible CVSSv3 score for the original Trojan could be:

Attack Vector (AV): Local (L) – Assuming the attacker has physical access to the device. Attack Complexity (AC): High (H) - Requires specific side-channel analysis techniques.

Privileges Required (PR): None (N) - No specific privileges are needed beyond physical access.

User Interaction (UI): None (N) - The Trojan activates autonomously.

Scope (S): Unchanged (U) - The Trojan doesn't affect other system parts.

Confidentiality Impact (C): High (H) – Potential data and key leakage.

Integrity Impact (I): None (N) - No data modification.

Availability Impact (A): None (N) - No denial of service.

This yields a CVSSv3 base score of approximately **7.5 (High)**. This reflects the high confidentiality impact, but the high attack complexity slightly reduces the score.

6. NETLAM output for Circuit Description:

The top-level module top instantiates three main modules:

aes_128: This module performs the core AES-128 encryption process. It takes a 128-bit state (plaintext) and a 128-bit key as input and produces a 128-bit ciphertext as output. The encryption involves multiple rounds, each consisting of several sub-operations (SubBytes, ShiftRows, MixColumns, AddRoundKey).

Trojan_Trigger: This module acts as a trigger for a hardware Trojan. It's a counter that generates a trigger signal ('Tj_Trig') based on a specific internal counter value. The trigger condition is not explicitly defined in the provided code but is implicitly controlled by the internal state of the counter.

TSC(Trojan State Controller): This module is the Trojan itself. It takes the trigger signal ('Tj_Trig'), the AES key, and the AES state (plaintext) as input. When the 'Tj_Trig' signal is asserted, it loads a potentially modified version of the state into the 'Capacitance' output, which is not part of the original AES functionality and could be used to leak information or manipulate the output.

After a comprehensive understanding of the functionalities of the different modules in the design, the LLM tool creates a DAG in the form of an adjacency list, which aids in determining the interconnections and intermediate registers and signals of the design. A snippet of the query output by the LLM is provided in the listing **7. NETLAM output for DAG Creation**. Then the LLM identifies the most vulnerable nets or modules in the design using the DAG. An instance of the query output is illustrated in the listing **8. Identifying Vulnerable Points**.

7. NETLAM output for DAG Creation

Creating a complete DAG for such a large circuit would be extremely cumbersome to represent as a simple adjacency list. However, I can illustrate the DAG concept with a simplified example focusing on a small section and showing the general approach.

Let’s consider a part of the ‘S’ module:

```
module S(...);
input [7:0] in;
wire _081_ = ~ in[7];
wire _092_ = ~ in[6];
INV_X1_492_ (.A(in[7]), .ZN(_081_));
INV_X1_493_ (.A(in[6]), .ZN(_092_));
endmodule
```

The DAG for this snippet would be like this (adjacency list representation):

```
in[7]: _081_
in[6]: _092_
_081_:
_092_:
```

8. Identifying Vulnerable Points

Given the structure, vulnerable points in the TjFree netlist are likely within the aes_128 module’s internal registers and the data paths leading to the out ports. The Trojan could be stealthy if it manipulates signals only when specific conditions are met by the trigger.

Vulnerable Point 1: Registers within the aes_128 module (Specifically the DFFs storing intermediate results of the AES rounds). A Trojan could subtly alter data stored in these registers under specific conditions, without affecting the overall functionality during normal operation.

Vulnerable Point 2: The data paths between combinational logic blocks in aes_128 (between SubBytes, ShiftRows, MixColumns, AddRoundKey operations). Trojans could insert small, conditional changes within these data paths, especially around the AddRoundKey operation that combines round keys and state.

Vulnerable Point 3: The output registers of the aes_128 module (DFFs feeding the out port). A Trojan could flip bits conditionally before the final output is latched.

After pinpointing the vulnerable points in the design, the LLM is asked to propose the top 5 Trojans specific to those vulnerable sites. The top 10 Trojans suggested by the LLM across various AES-based Trojan attack variants are listed in Table 3. These Trojans have significantly high CVSS scores, and notably, 5 of them have not been identified earlier and are not part of the *Trust-Hub* benchmark dataset. We perform the two-stage equivalence checking of these 5 newly suggested Trojans, which are not included in *Trust-Hub*, to justify the credibility of our proposed tool. Surprisingly, all of the suggested Trojans pass

Table 3. Stealthy Trojans suggested by **NETLAM** and their corresponding target modules, trigger mechanism, payload action, stealth features and CVSS Scores. The proposed Trojans are applicable across various AES-based Trojan attack variants.

Rank	Trojan Description	Vulnerable Point	Trigger Mechanism	Payload Action	Stealth Features	Included in <i>Trust-Hub</i>
1	Data-Dependent Bit-Flip	'aes_128' registers	Specific input patterns, clock cycle	Flip a bit in a selected register	Random activation, low probability	Yes
2	Conditional Key Modification	'expand_key_128' XOR gates	Specific input patterns	Modify a few bits in a round key	Intermittent triggering	Yes
3	Intermittent Signal Tampering	Data path in 'aes_128'	Trigger count, specific data	Add small bias to an intermediate value	Rare activation	No
4	Clock Gating Trojan	Clock signal in 'aes_128'	Counter value, external signal	Temporarily halt clock signal to block	Random activation	No
5	Data Tampering Trojan	'xS' module	Complex trigger	Modifies specific bits of the intermediate state in each encryption round	Data-dependent, complex trigger	No
6	Data Dependent Trojan	Module final_round	Conditional trigger	Affects the final output directly	Data-dependent, complex trigger	Yes
7	Logic-level Trojan	'table_lookup' module	Always active	Modifies lookup table values	Small, localized modification	No
8	Gate-level Trojan	'S' module gates	Specific input combinations	Bit-flipping	Random activation, low frequency	No
9	Clock Glitch Trojan	Clock input to 'S'	Random clock glitches	Data corruption	Very low frequency	Yes
10	Delay-based Trojan	'aes_128' module critical path	Always active	Introduces intentional delay	Change in delay not visible during functional tests	Yes

the EQY functional equivalence check. However, 4 of them fail the **NETLAM** equivalence checker, thus strengthening our claim of evaluating such surreptitious Trojans. The Trojans fail in the equivalence test mainly because it incurs additional circuitry in the victim modules. A brief analysis is provided in Table 4. As the LLM-based equivalence checker performs a one-to-one matching between the circuitry as well as the hierarchy of the designs, it is easier to detect stealthy Trojans using such tools.

Table 4. Comprehensive Results of Equivalence Checking for **NETLAM** suggested Trojans

Rank	Trojan Description	CVSS Score	Equivalence Check (EQY)	Equivalence Check (NETLAM)
1	Intermittent Signal Tampering	9	Passed	Failed due to presence of additional circuitry
2	Clock Gating Trojan	8.8	Passed	Failed due to presence of the additional clock-gating circuitry
3	Data Tampering Trojan	8.5	Passed	Failed due to difference in the 'xS' module's logic
4	Logic-level Trojan	8	Passed	Passed
5	Gate-level Trojan	7.8	Passed	Failed due to additional circuitry in 'xS' module

Nevertheless, seldom in formal equivalence and verification tools, the user does not have access to the golden reference design [6]. In such cases, it is hard for the verification tool to detect the discrepancies in the Trojan-infected designs. Thus, a more reliable metric in such cases is the CVSS score. Moreover, this work is a preliminary exploration of the capabilities of LLM tools in detecting hardware Trojans in digital designs. We aspire to enhance the capacity of our tool further and uncover more stealthy Trojans capable of even passing the golden-input inclusive equivalence check to broaden the scope of benchmarks like *Trust-Hub* further. The ultimate instruction given to the LLM is to devise a Trojan specific to the vulnerable sites that is more stealthy than the one initially provided. This is illustrated in the listing **9. Trojan Inserted Benchmark**.

9. Trojan Inserted Benchmark

A more stealthy Trojan could be integrated within the existing AES-128 module, specifically within the ‘expand_key_128’ module. This module has many unused intermediate wires and registers, which present perfect camouflage.

This example adds a simple XOR operation to modify a single bit in the key expansion. Note: Actual implementation would involve a more sophisticated method.

```
module expand_key_128(clk, in, out_1, out_2, rcon);
// ... existing code ...
// Trojan: XOR with a bit of the input data, conditionally
always @(posedge clk) begin
if (Counter[0] & Counter[1]) begin // Conditional activation, even more
subtle
k0a[0] <= k0a[0] ^ in[0]; //Modifying the first bit of k0a
end
// ... existing code ...
endmodule

module Trojan_Trigger(clk, rst, out, Tj_Trig);
//The counter can now use outputs of the key expansion instead of the state
// ... modified counter logic ...
always @(posedge clk) begin
if (rst) begin
Counter <= 0;
end else begin
Counter <= Counter + 1;
end
end
assign Tj_Trig = Counter[0] & Counter[1]; // A simple, but subtle trigger
endmodule
```

5 Conclusion

In this work, we present **NETLAM**, a comprehensive framework leveraging multiple LLM-based tools to tackle the challenge of identifying and generating hardware Trojans in digital designs. By combining an LLM-based equivalence checker, a DAG generator with vulnerability identification capabilities, and a Trojan injector, **NETLAM** provides a systematic approach to uncovering new, previously undiscovered Trojans. Our evaluation of *Trust-Hub*'s AES dataset demonstrates the framework's ability to generate novel Trojans with high CVSS scores. We demonstrate that LLM models, such as GPT-4o and Gemini, can improve Trojan detection by leveraging semantic and probabilistic analysis instead of relying solely on strict logical equivalence. The functional, logical, and structural equivalence test is critical in the domain of hardware Trojan detection, thus necessitating the need for more open-source and accessible tools for performing such routine tests. Our proposed tool **NETLAM** paves the way for such design-critical validation by using a hybrid model for inspecting digital designs that include a formal verification tool for deterministic logic checks and a Generative AI for anomaly detection, trigger analysis, and structural mapping. Importantly, **NETLAM** not only identifies vulnerabilities but also contributes to enhancing Trojan research by uncovering more potential Trojans, thus paving the way for expanding benchmarks like *Trust-Hub*.

Acknowledgments. This work is supported by the Information Security Education and Awareness (ISEA) project under the Ministry of Electronics and Information Technology (MeitY).

References

1. Jitendra Bhandari, Rajat Sadhukhan, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. SENTAUR: security enhanced trojan assessment using llms against undesirable revisions. *CoRR*, abs/2407.12352, 2024.
2. Aritra Bhattacharyay, Shuo Yang, Jonathan Cruz, Prabuddha Chakraborty, Swarup Bhunia, and Tamzidul Hoque. An automated framework for board-level trojan benchmarking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(2):397–410, 2023.
3. Jayeeta Chaudhuri, Dhruv Thapar, Arjun Chaudhuri, Farshad Firouzi, and Krishnendu Chakraborty. SPICED: syntactical bug and trojan pattern identification in A/MS circuits using llm-enhanced detection. *CoRR*, abs/2408.16018, 2024.
4. Jonathan Cruz, Pravin Gaikwad, Abhishek Nair, Prabuddha Chakraborty, and Swarup Bhunia. Automatic hardware trojan insertion using machine learning. *CoRR*, abs/2204.08580, 2022.
5. Anna Lena Duque Antón, Johannes Müller, Lucas Deutschmann, Mohammad Rahmani Fadiheh, Dominik Stoffel, and Wolfgang Kunz. A golden-free formal method for trojan detection in non-interfering accelerators. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.
6. Anna Lena Duque Antón, Johannes Müller, Lucas Deutschmann, Mohammad Rahmani Fadiheh, Dominik Stoffel, and Wolfgang Kunz. A golden-free formal method

- for trojan detection in non-interfering accelerators. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.
7. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory (lstm). *Neural Computation*, 9(8):1735–1780, 1997.
 8. *The freePDK45 process design kit (PDK)*.
 9. Lucas Klemmer, Dominik Bonora, and Daniel Grosse. Large-scale gatelevel optimization leveraging property checking. In *DVCon Europe 2023; Design and Verification Conference and Exhibition Europe*, pages 86–93, 2023.
 10. Georgios Kokolakis, Athanasios Moschos, and Angelos D. Keromytis. Harnessing the power of general-purpose llms in hardware trojan design. In Martin Andreoni, editor, *Applied Cryptography and Network Security Workshops*, pages 176–194, Cham, 2024. Springer Nature Switzerland.
 11. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Recurrent neural network (rnn). *Nature*, 323:533–536, 1986.
 12. Dipayan Saha, Katayoon Yahyaei, Sujan Kumar Saha, Mark Tehranipoor, and Farimah Farahmandi. Empowering hardware security with llm: The development of a vulnerable hardware database. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 233–243, 2024.
 13. Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 471–474, 2013.
 14. Bicky Shakya, Miao Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Mohammad Tehranipoor. Benchmarking of hardware trojans and maliciously affected circuits. *Journal of Hardware and Systems Security*, 1:85 – 102, 2017.
 15. Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 23–40, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
 16. Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27(1):10–25, 2010.
 17. Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. 2013.
 18. Shuo Yang, Tamzidul Hoque, Prabuddha Chakraborty, and Swarup Bhunia. Golden-free hardware trojan detection using self-referencing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(3):325–338, 2022.
 19. Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. Hardware trojan detection using graph neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2022.