2024 IEEE Physical Assurance and Inspection of Electronics (PAINE) | 979-8-3315-4225-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/PAINE62042.2024.10792717

SPICED: **S**yntactical Bug and Trojan **P**attern **I**dentification in A/MS **C**ircuits using LLM-**E**nhanced **D**etection

Jayeeta Chaudhuri†, Dhruv Thapar†, Arjun Chaudhuri‡, Farshad Firouzi†, and Krishnendu Chakrabarty†
†School of Electrical, Computer, and Energy Engineering, Arizona State University, AZ
‡NVIDIA Corporation, CA

Abstract—**Analog and mixed-signal (A/MS) integrated circuits (ICs) are crucial in modern electronics, playing key roles in signal processing, amplification, sensing, and power management. Many IC companies outsource manufacturing to third-party foundries, creating security risks such as stealthy analog Trojans. Traditional detection methods, including embedding circuit watermarks or conducting hardware-based monitoring, often impose significant area and power overheads, and may not effectively identify all types of Trojans. To address these shortcomings, we propose SPICED, a Large Language Model (LLM)-based framework that operates within the software domain, eliminating the need for hardware modifications for Trojan detection and localization. This is the first work using LLM-aided techniques for detecting and localizing syntactical bugs and analog Trojans in circuit netlists, requiring no explicit training and incurring zero area overhead. Our framework employs chain-of-thought reasoning and few-shot examples to teach anomaly detection rules to LLMs. With the proposed method, we achieve an average Trojan coverage of 93.32% and an average true positive rate of 93.4% in identifying Trojan-impacted nodes for the evaluated analog benchmark circuits. These experimental results validate the effectiveness of LLMs in detecting and locating both syntactical bugs and Trojans within analog netlists.**

I. INTRODUCTION

Analog and mixed-signal (A/MS) integrated circuits (ICs) play a critical role in signal processing, amplifiers, sensors, and power management systems. Many IC companies such as Intel, AMD, Qualcomm, and Texas Instruments opt to outsource the manufacturing and fabrication of their analog designs to third-party foundries to avoid expensive capital expenditures as well as substantial costs associated with investing in manufacturing infrastructure. The globalization of the semiconductor industry and the outsourcing of analog ICs to third-party vendors have introduced significant security threats. These threats notably compromise the integrity of ICs, making them vulnerable to analog Trojans [1] [2] [3]. A major risk arises from the possibility of embedding stealthy Trojans that evade detection under normal operating conditions [4] [5].

Trojans occupy minimal area footprints, enabling their easy integration into larger, complex A/MS designs at the netlist level, which includes multiple paths and transistor components. These stealthy components are activated only during specific operating bias voltages and remain dormant otherwise. Prior work on analog Trojan detection utilizes current-sensing amplifiers to identify Trojan activation [6]. A recent study introduces a sensitivity analysis-based framework leveraging analog neural twins to detect stealthy analog Trojans [7]. This approach identifies the critical paths most vulnerable to Trojan insertion [8]. Next, circuit watermarks are embedded to monitor deviations in these paths, triggering an alert when a Trojan is activated. While these methods are effective in detecting Trojans, they do not address the challenge of locating Trojan-impacted nodes within the design netlist, a task that becomes increasingly difficult as circuit complexity grows.

The recent advancements in Large Language Models (LLMs) have showcased their significant capabilities across various tasks including code generation and optimization. Such advancements in the EDA domain naturally position LLMs as highly potent for novel applications in A/MS design. Traditional methods for Trojan detection, including embedding circuit watermarks or conducting hardware-based monitoring, frequently impose significant area and power overheads, especially in large analog designs. To address this issue, we propose SPICED, an LLM-based framework that operates within the software domain, thereby eliminating the need for any hardware modifications to the analog design for Trojan detection and localization. SPICED excels at intelligent parsing and analysis of large volumes of structured data such as SPICE netlists. In addition to Trojan localization, the proposed LLM-based framework provides comprehensive analysis and detailed diagnosis of the detected anomalies. By leveraging a deep understanding of the HSPICE language, simulation logs, and the topological structure of netlists, SPICED can not only distinguish between Trojan-free and Trojan-inserted netlists but also precisely identify the specific Trojan components and the nodes affected by the Trojan. The key contributions of this paper are as follows:

- **Introduction of SPICED**: Presenting SPICED, the first LLM-based framework for Trojan detection in A/MS design that requires no hardware modifications.
- **Syntactical Bug Mitigation:** Leveraging in-context learning and Chain-of-Thought (CoT) prompting to detect and mitigate syntactical bugs in SPICE netlists.
- **Precise Trojan Detection and Localization:** Developing supervised learning rules for the LLM to identify Trojan circuits and the Trojan-affected nodes in an analog design.

The remainder of the paper is organized as follows. Section II provides a comprehensive overview of recent work on LLM-

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Comparison of prior works using LLM-aided techniques.

Method	Domain	Training-Free?	Bug Detection?	Bug Fixing?	Trojan Detection
[9]	Digital	✗	✗	✗	✗
[10]	Analog	✗	✗	✗	✗
[11]	Analog	✓	✗	✗	✗
[12]	Digital	✓	✗	✗	✗
[13]	Digital	✗	✗	✗	✗
[14]	Digital	✗	✗	✗	✗
[15]	Digital	✓	✗	✗	✗
[16]	Digital	✓	✓	✓	✗
SPICED	Analog	✓	✓	✓	✓

aided text generation and how its capability can be leveraged for bug detection tasks in the analog domain. We provide a detailed analysis of syntactical bug detection and correction using LLM in Section III. Section IV presents the framework for Trojan detection and localization using LLM involving both CoT and few-shot prompting examples for generating supervised learning rules. Evaluation results for SPICED with comparisons among several LLM-aided techniques are presented in Section V. Finally, Section VI concludes the paper.

II. BACKGROUND AND MOTIVATION

A. Design Automation using LLMs

LLMs have been integrated into multiple stages of EDA, enhancing processes from code generation and placement and routing to security measures, thereby streamlining and improving the overall design workflow. Recent work used LLM for Verilog code generation by fine-tuning existing LLM with Verilog datasets [9]. Fine-tuned open-source CodeGen LLM outperformed state-of-the-art commercial LLM in generating functionally correct designs [14]. [12] used LLM-based iterative flow to design an 8-bit accumulator-based microprocessor architecture. [11] proposed LLM for generating analog circuits with a feedback-enhanced flow to enable self-correcting design of analog circuits. The feedback allowed generation of circuits without any LLM training involved. Domain-Adaptive Pre-Training (DAPT) followed by Supervised Fine-Tuning (SFT) of foundation LLM models enabled an assistant chatbot for chip design [13]. [17], [18] proposed LLM-based script generation to facilitate the EDA design flow. In the context of security, LLM is shown to be effective in structural generation of digital Trojans [15] as well as fixing syntactical bugs in Verilog codes [16]. These works are focused mainly on code generation tasks and bug detection in the digital domain, as shown in Table I.

B. Nature of Analog Trojans

An analog Trojan consists of two primary components: (1) Trigger circuit, which is conditionally gated with an AND or OR gate to activate the Trojan only upon specific toggling instructions; (2) Detector circuit, which detects the charge buildup of the capacitive component of the Trojan circuit, and when the capacitor voltage reaches a threshold, activates the payload. Recent work shows the impact of an analog Trojan, namely A2 [4] that can be stealthily inserted during the design and fabrication phases of an analog design. Due to its small footprint, it can be maliciously inserted in unused parts during design phase. The trigger for the A2 Trojan is software-controlled i.e., the trigger is activated when a rare instruction is executed. As an extension to the A2 Trojan, authors in [5] shows the implementation of the DELTA Trojan, which uses a glitch generator for the trigger circuit, and can be inserted in any net of a circuit irrespective of it being rarely activated.

In [8], it has been demonstrated that A2 Trojans remain mostly dormant due to their insertion in the less sensitive paths of a circuit. Upon their activation, the primary output voltage behavior is impacted, leading to significant performance degradation. Therefore, it is necessary to detect and localize these malicious circuits within the analog design netlist before the netlist is sent to the fabrication stage.

C. Prior Work on Analog Trojan Detection

In [6], a current sensing-based circuit is inserted in a digital design to detect analog Trojans such as A2 at run-time. A recent work [8] demonstrates the impact of analog Trojans (A2, DELTA, and large-delay Trojans) on A/MS designs as well. In [8], analog neural twins are employed to identify critical paths in an analog circuit netlist through sensitivity analysis. After identifying critical paths, circuit watermarks are inserted to make these paths observable at the circuit output. Sensitizing the least sensitive paths of a circuit makes the detection of stealthy Trojans easier. This ensures that even if the effects of the Trojans are not captured at the primary output of the circuit, they can still be detected through the altered behavior of the sensitized paths.

Although [8] effectively identifies all the Trojan hotspots, there are two significant limitations: (1) the area overhead associated with added watermarks increases with the complexity of the analog design, and (2) the specific Trojan-inserted nodes are not identified, i.e., localization of the detected Trojans is not performed; this limitation makes it difficult to pinpoint the exact nodes affected by the Trojans, making targeted mitigation infeasible.

These limitations motivate the exploration of specialized techniques that leverage the contextual ability of LLMs. LLMs are vastly known for interpreting and generating texts, and are capable of interpreting contexts across various programming languages such as C, C++, and Python. A significant portion of the training data for LLMs is sourced from Github repositories. LLMs such as Llama-2 and Llama-3 are primarily trained on Python datasets [19]. In the realm of digital design, the availability of Verilog code in training datasets is relatively limited compared to other programming languages [11]. The scarcity is even more for analog design data, particularly for the SPICE language, which has less code available in open-source repositories. This poses a challenge for LLMs like Llama 2 and GPT-3.5 to understand SPICE syntax comprehensively. Despite these challenges, GPT-3.5 shows the capability of understanding basic SPICE syntax and type of circuit configuration being implemented in a SPICE netlist. Leveraging LLMs for design analysis offers several advantages.

- **Textual analysis capability**: Aided by the right prompts, LLMs can effectively parse simulation log files without

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.

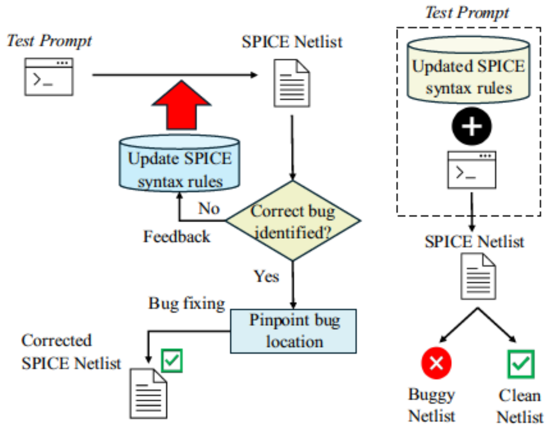
any manual intervention, thereby offering huge productivity boost in terms of reduced engineering hours and log processing runtime. LLMs understand the tabular structure of logged voltage and current values in the logs, making it easier for them to extract the numerical voltage and current data for each circuit node.

- **Netlist Identification and Bug Correction**: Pre-trained LLMs can understand the basic syntax of SPICE netlists, enabling the models to analyze the netlist structure and flag syntax errors.
- **Anomaly detection**: LLMs can learn the node properties and inter-node relationships described by the netlist topology and simulation logs, and identify anomalous patterns among normal operational characteristics of the design via in-context learning.
- **Trojan Detection and Localization**: Equipped with the ability to parse logs, analyze netlists, and identify outlier patterns, LLMs can potentially identify Trojan-affected nodes whose voltage and current profiles manifest as anomalous.
- **No Area and Power Overhead**: Leveraging the computational abilities of LLMs obviates the need for any additional Trojan detection hardware in the circuit.

III. LLM FOR SYNTACTICAL BUG DETECTION

Fig. 1 illustrates the proposed flow of guiding the LLM to detect a wide range of syntactical bugs while reducing the likelihood of incorrectly flagging correct lines as syntax errors. Note that this method does not involve any LLM fine-tuning; instead, it focuses on improving the bug detection capability of LLM through refinement of SPICE syntax rules based on real-time feedback. The steps involved are as follows:

1) Begin with an initial set of SPICE syntax rules and use them to construct the LLM prompt to identify and locate syntactical bugs within a SPICE netlist (shown in Fig. 2(a)).
2) When the LLM identifies the bug correctly, it specifies the type of the detected bug and its location in the netlist (shown in Fig. 2(b)).
3) A false positive occurs when the LLM incorrectly identifies a line as containing a bug. If the LLM incorrectly flags a line as buggy, this feedback is used to manually update and refine the syntax rules in the prompt (shown in Fig. 2(c)).
4) After updating the rules, the refined prompt is applied to the same netlist. The process is repeated until the number of false positives is minimized.



(a) (b)

Fig. 1: (a) Enhancing syntactical bug detection accuracy of LLM by instruction-following approach (b) Identifying and mitigating bugs in a SPICE netlist.

We use the updated set of syntax rules for evaluation. Additionally, we prompt the LLM to generate a structured bug detection report. The report includes the following items:

1) *List of all syntactical bugs in the SPICE netlist*: The bugs may include connection error (missing connections or floating nodes in the netlist), insertion error (unintended or intended insertion of additional transistor components), and incorrect specifications of parameters.
2) *Location of the bugs in the SPICE netlist*: Lines in the SPICE netlist where the bug is located, including the component or connection names.
3) *Suggestions for correction*: Provides a list of recommended actions to correct the bugs and generate a revised netlist.

IV. SUPERVISED LEARNING-BASED FRAMEWORK USING LLM FOR TROJAN DETECTION

While Section III focuses on identifying and correcting syntactical bugs in the SPICE netlist, this section addresses functional bug detection, particularly targeting analog Trojans. A Trojan-impacted node is defined as one of the circuit nodes where the Trojan is inserted or which is part of the trigger node. From prior work on analog Trojan detection [8] [6], the following observations are noted regarding the current and voltage behavior of Trojan-impacted nodes.

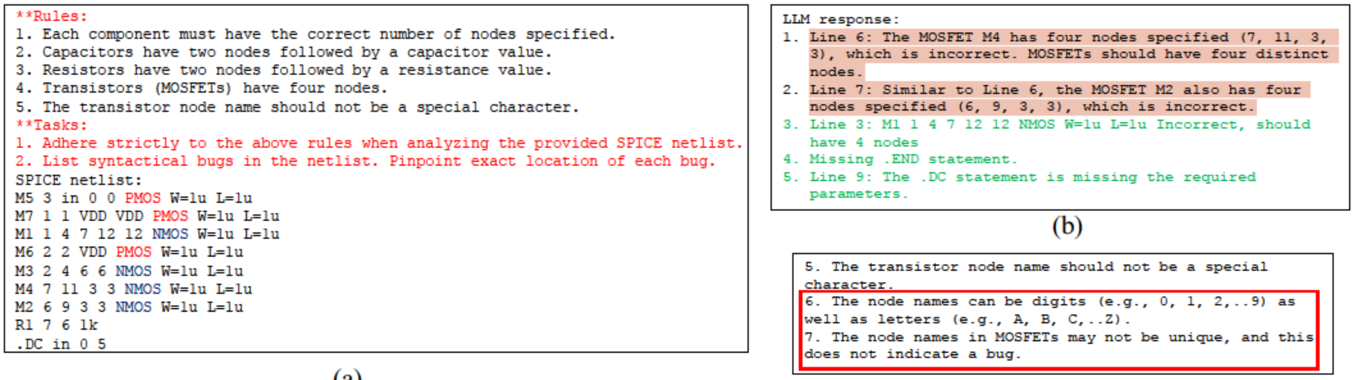
1) Deviation in primary output voltage: As observed in [8], when a Trojan is inserted into one of the sensitive paths of an analog design, its impact is captured at the primary output voltage. Specifically, the output voltage exceeds the desired specifications when the Trojan is activated.
2) Anomalous deviation in circuit intermediate nodes: Nodes impacted by the Trojan, or the neighboring nodes exhibit significant deviations in voltage behavior when the Trojan is activated. This deviation can serve as a critical indicator of the presence of a Trojan.
3) Anomalous surge in MOSFET current: According to [6], upon Trojan activation, some MOSFETs within the analog design draw a substantially higher current. This current behavior deviates significantly from the normal current behavior observed when the Trojan circuit is dormant.

For identifying analog Trojans and the Trojan-impacted nodes, we used these observations to craft supervised-learning rules. We then created prompts based on these rules to teach the LLM to detect anomalies in the current and voltage simulation logs. The simulation logs are obtained by running HSPICE simulation on analog netlists. The procedure of teaching the supervised learning approach (shown in Fig. 3 (a)) to the LLM is shown in Fig. 3(b). We link each Trojan characteristic to a specific supervised learning rule as follows:

1. Deviation in primary output voltage

■ **Supervised Learning Rule 1:** We craft prompts that

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.



```
**Rules:
1. Each component must have the correct number of nodes specified.
2. Capacitors have two nodes followed by a capacitor value.
3. Resistors have two nodes followed by a resistance value.
4. Transistors (MOSFETs) have four nodes.
5. The transistor node name should not be a special character.
**Tasks:
1. Adhere strictly to the above rules when analyzing the provided SPICE netlist.
2. List syntactical bugs in the netlist. Pinpoint exact location of each bug.
SPICE netlist:
M5 3 in 0 0 PMOS W=1u L=1u
M7 1 1 VDD VDD PMOS W=1u L=1u
M1 1 4 7 12 12 NMOS W=1u L=1u
M6 2 2 VDD PMOS W=1u L=1u
M3 2 4 6 6 NMOS W=1u L=1u
M4 7 11 3 3 NMOS W=1u L=1u
M2 6 9 3 3 NMOS W=1u L=1u
R1 7 6 1k
.DC in 0 5
```

(a)

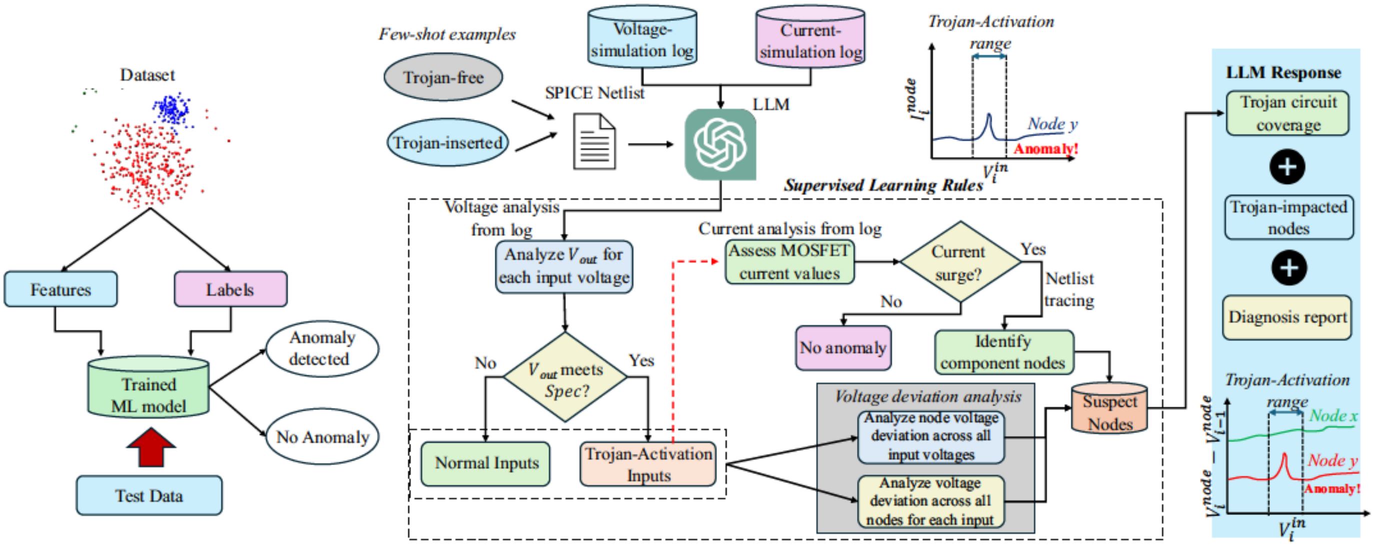
```
LLM response:
1. Line 6: The MOSFET M4 has four nodes specified (7, 11, 3,
   3), which is incorrect. MOSFETs should have four distinct
   nodes.
2. Line 7: Similar to Line 6, the MOSFET M2 also has four
   nodes specified (6, 9, 3, 3), which is incorrect.
3. Line 3: M1 1 4 7 12 12 NMOS W=1u L=1u Incorrect, should
   have 4 nodes
4. Missing .END statement.
5. Line 9: The .DC statement is missing the required
   parameters.
```

(b)

```
5. The transistor node name should not be a special
character.
6. The node names can be digits (e.g., 0, 1, 2,..9) as
well as letters (e.g., A, B, C,..Z).
7. The node names in MOSFETs may not be unique, and this
does not indicate a bug.
```

(c)

Fig. 2: (a) An example prompt highlighting the SPICE syntax rules for bug detection and localization (b) LLM correctly detects the bugs injected in the netlist; however, it incorrectly flags Lines 6 and 7 as bugs (highlighted in red) (c) Explicitly updating the rules in the prompt to reduce false positives.



(a)

(b)

Fig. 3: (a) Flow of supervised-learning approach using machine learning (ML) models (b) Using the supervised-learning analogy to locate Trojan-impacted nodes using LLM (V_{out}: primary output voltage, $Spec$: desired output voltage specifications, $V_i^{node}(I_i^{node})$: node voltage (current) corresponding to i^{th} input voltage sample V_i^{in}).

highlight output voltages that exceed the desired circuit specifications. Based on these output voltages, we ask the LLM to identify the corresponding input voltages. These input voltages are labeled as 'Trojan-Activation Inputs', while inputs where the Trojan is inactive are labeled as 'Normal Inputs.'

2. Anomalous deviation in circuit intermediate nodes

■ **Supervised Learning Rule 2:** Fig. 4 shows the voltage deviations in intermediate nodes of a Trojan-inserted circuit across different input voltages and how their voltage deviations compare across nodes for anomaly detection. The LLM identifies the nodes that exhibit (a) a significant voltage deviation between 'Trojan-Activation Input' and 'Normal Input' ranges, and (b) the highest deviation among all the intermediate nodes for each 'Trojan-Activation Input'. Based on these observations, the LLM applies one of the following rules to determine the Trojan-impacted nodes.

- Rule #1: Union of nodes identified in steps (a) and (b).
- Rule #2: Intersection of nodes obtained in steps (a) and (b).

3. Anomalous surge of MOSFET current

■ **Supervised Learning Rule 3:** We design prompts to examine current simulation logs and identify MOSFETs that generate unusual current spikes under 'Trojan-Activation Inputs' compared to the baseline of 'Normal Inputs'. The nodes corresponding to the MOSFETs with detected current anomalies are flagged as suspect nodes.

Finally, combining the above supervised learning rules, we prompt the LLM to detect the Trojan circuit within an analog netlist and generate the final set of Trojan-impacted nodes. The supervised learning rules are defined at the start of the prompt. Next, we provide the LLM with few-shot examples, where it sees a limited number of example netlists, labeled as 'Trojan' or 'Trojan-Free'. Additionally, we provide explanations related to the supervised-learning rules and the simulation behavior of current and voltage from the log files to justify deviations that

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.

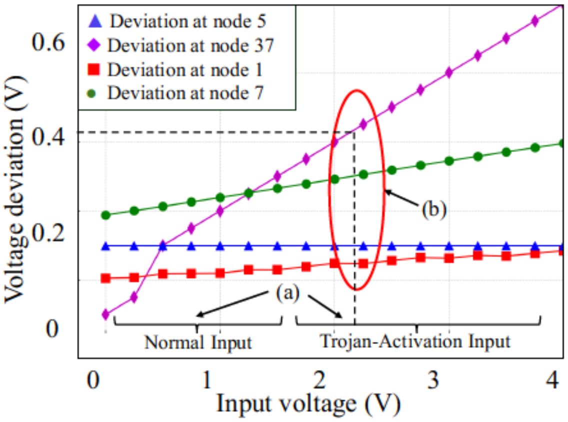


Fig. 4: Voltage deviation analysis for A2 Trojan-inserted netlist of circuit ‘642’ from AMSNet [10]. Voltage deviation of a node x is given by $V_i^x - V_{i-1}^x$, where $i-1$ and i are consecutive input voltage samples. (a) Analyze voltage deviation of each node across the ‘Normal Input’ and ‘Trojan-Activation Input’ ranges, (b) analyze voltage deviation across nodes in the ‘Trojan-Activation Input’ range. Combining (a) and (b), we observe that node 37 is a Trojan-impacted node.

indicate potential Trojan behavior.

V. EXPERIMENTAL RESULTS

A. *Experimental Setup*

For both bug detection and Trojan detection experiments, we evaluate a wide variety of analog benchmark circuits that are selected from the OpenSource netlist dataset from AMSNet [10] and Github [20]. These circuits include differential amplifier, inverter, OPAMP, and bandpass filter, thus providing a comprehensive dataset for evaluating both syntactical bug and Trojan detection capabilities of the LLM. We have compiled a total of 18 syntactical bugs from [21]. We choose varying complexity of bugs to test the effectiveness of LLM in identifying bug-inserted SPICE netlists. The bug complexity is classified as: easy, medium, and difficult. The bug benchmark used for LLM evaluation is shown in Table II. The benchmark includes 4 easy, 6 medium, and 8 difficult syntactical bugs that can be present in a SPICE netlist.

For the Trojan-detection experiments, we evaluate the well-known analog Trojan explored in recent literature - A2 [4]. This type of Trojan occupies small footprint and hence, can be easily embedded in the netlist stage by an untrusted foundry. We embed A2 in the SPICE netlist of an analog benchmark circuit to generate a Trojan-inserted netlist. The Trojan is considered to be internally triggered by an intermediate node of the analog design, thus emulating a realistic attack scenario shown in [22]. For our experiments, we use the GPT-3.5-turbo API. The experiments are carried out on an NVIDIA A100 GPU. Table III lists the information available to the LLM for performing bug and Trojan detection tasks.

TABLE II: List of syntactical bugs with varying complexity for evaluating LLM performance.

Description of Bug	Example	Complexity
Missing node of transistor	M1 2 3 0 PMOS	E
Missing .END statement	Netlist without terminating .END	E
Missing transistor model name	M20 2 3 0 0 (model undefined)	E
Floating node	in	E
Extra node in transistor definition	M20 2 3 2 0 **4** PMOS	M
Incorrect resistor value format	R1 in out **1K** (should be 1k)	M
Incorrect subcircuit definition	.SUBCKT a b (instance name undefined)	M
Missing capacitor value	C12 in out	M
Incorrect usage of transient analysis	.tr 100p (simulation duration undefined)	M
Missing voltage value	VB 5 0	M
Special characters in node names	M**!2** 3 5 0 0 NMOS	D
Incorrect .PRINT statement	.PRINT TRAN **in** (should be **V(in)**)	D
Incorrect current source definition	Ib 1 0 1M	D
Incorrect transistor name	M1 2 3 0 0 **NMOSC**	D
Missing .END in subcircuit	Subcircuit without .ENDS	D
Incorrect parameter definition	.PARAM **R1=**	D
Incorrect .OPTIONS definition	**.OPTION** POST (should be .OPTIONS)	D
Missing value in .DC	.DC vin 0.1 5 (missing increment value)	D

The syntactical bugs in the SPICE examples are indicated in **bold**. Different bug complexities are indicated by easy (E), medium (M), and difficult (D).

TABLE III: Information provided to the LLM for syntactical bug and Trojan detection tasks.

Task	Information provided
Bug Detection	SPICE netlist + syntax rules
Trojan Detection	SPICE netlist + corresponding simulation log files (current and voltage) + circuit specification

B. *Evaluation Metrics*

We use the following metrics to evaluate the effectiveness of the LLM in syntactical bug and Trojan detection.

- Bug coverage (in %): This metric represents the percentage of syntactical bugs detected by the LLM out of the total number of bugs embedded within a SPICE netlist.
- Bug resolved (in %): It represents the percentage of detected syntactical bugs that are correctly fixed by LLM.
- Trojan identified: It indicates whether the LLM has correctly detected at least one Trojan component in the netlist.
- Trojan coverage (in %): This metric calculates the ratio of the number of malicious components (transistor, resistor, or capacitor) embedded in the SPICE netlist that are correctly identified by the LLM and the total number of Trojan-injected components in the netlist.
- Precision (in %): It indicates the percentage of correctly predicted Trojan-impacted nodes out of all the predicted impacted nodes. It is denoted by $Precision = \frac{TP}{TP+FP}$, where TP and FP indicate the true positive and false positive counts, respectively.
- Recall (%): It is indicated by the percentage of correctly predicted Trojan-impacted nodes out of the actual number of Trojan-impacted nodes. It is denoted by: $Recall = \frac{TP}{TP+FN}$, where TP and FN indicate the true positive and false negative counts, respectively.

C. *Performance Evaluation of GPT Model*

1) *Syntactical Bug Detection*

The bug detection results are shown in Table IV. We observe that GPT-3.5 demonstrates higher accuracy and broader bug coverage across all evaluated SPICE netlists. Additionally, the proposed instruction-following approach results in fewer false positives for the evaluated bugs compared to the scenario when instructions were not explicitly provided in the prompt.

2) *Functional Bug (Trojan) Detection*

To test the inherent capability of LLM to analyze structured prompts and to further improve its accuracy through well-crafted prompts based on supervised-learning rules and few-shot examples, we evaluate the following four test cases:

1) FS: We evaluate the LLM by providing only few-shot

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.

TABLE IV: Performance of LLMs in identifying syntactical bugs across different circuit types.

Case	Circuit Type	Bugs inserted			Bugs Detected (%)			Bug Coverage (%)	FPR (%)		Bug resolved (%)
		Easy	Medium	Difficult	Easy	Medium	Difficult		Without Instructions	**With Instructions**	
1	Common source amplifier (Resistive load)	2	5	5	100	100	80	91.6	38.8	0	100
2	Common source amplifier (Resistive and capacitive loads)	3	4	6	100	75	66.7	76.9	28.5	7.1	90
3	NMOS Transistor	4	5	6	100	100	83.3	93.3	16.6	0	100
4	Switched capacitor	4	4	7	100	100	71.4	86.6	23.5	5.8	100
5	Inverter	4	4	7	100	75	71.4	80	14.2	0	83.3
6	Differential amplifier	3	6	8	100	100	87.5	94.1	15.7	0	81.2
7	Current mirror (Both NMOS and PMOS)	4	6	8	100	83.33	75	83.3	21	6.25	86.6
8	Current mirror (Differential pair)	3	5	7	100	100	71.4	86.6	18.75	0	76.9
9	OPAMP	4	5	7	100	100	100	100	31.2	6.25	87.5
10	Bandgap filter	3	6	5	100	100	80	92.8	31.5	13.3	92.3

Using a refined set of SPICE syntax rules results in a significantly lower FPR compared to the scenario when a basic set of syntax rules is prompted to the LLM.

TABLE V: Performance of LLMs in detecting analog Trojans and locating Trojan-impacted nodes for several analog designs.

Case	Netlist	Trojan identified?				Trojan Coverage (%)				Precision (%)				Recall (%)			
		FS	R	R_{FS}	R^*_{FS}	FS	R	R_{FS}	R^*_{FS}	FS	R	R_{FS}	R^*_{FS}	FS	R	R_{FS}	R^*_{FS}
1	642_troj_1	✓	✓	✓	✓	57.14	14.28	**100**	**100**	67	67	**100**	**100**	67	67	**100**	**100**
2	642_troj_6	✓	✗	✓	✓	28.57	0	**100**	**100**	43	0	**100**	**100**	100	0	**100**	**100**
3	642_troj_7	✗	✓	✓	✓	0	14.28	100	100	0	25	67	67	0	33.3	67	100
4	642_troj_20	✓	✓	✓	✓	71.5	57.14	100	85.7	43	12.5	100	100	67	33.3	100	67
5	642_troj_34	✓	✓	✓	✓	42.8	42.8	100	100	100	12.5	100	67	100	33.3	100	100
6	642_troj_36	✓	✗	✓	✓	57.14	0	100	100	43	67	100	67	33.3	67	100	100
7	642_troj_38	✓	✓	✓	✓	57.14	14.28	100	100	50	67	67	75	67	67	67	100
8	642_troj_39	✗	✓	✓	✓	0	28.57	85.7	85.7	0	33.3	100	100	0	33.3	100	100
9	755_troj_1	✓	✗	✓	✓	14.28	0	100	100	50	0	100	67	33.3	0	100	100
10	755_troj_2	✓	✗	✓	✓	14.28	0	100	85.7	43	0	75	75	67	0	100	100
11	755_troj_3	✓	✓	✓	✓	42.8	42.8	100	85.7	43	12.5	100	67	67	33.3	67	67
12	755_troj_5	✗	✓	✓	✓	0	42.8	71.4	100	25	12.5	100	100	67	33.3	100	67
13	755_troj_6	✗	✓	✓	✓	0	42.8	100	100	0	40	100	75	0	67	100	100
14	755_troj_9	✓	✓	✓	✓	57.14	57.14	100	100	50	33.3	100	100	67	33.3	100	67
15	755_troj_10	✓	✓	✓	✓	71.4	42.8	100	100	100	25	100	75	100	33.3	100	100
16	755_troj_11	✓	✓	✓	✓	57.14	42.8	85.7	100	71.4	0	100	100	33.3	0	100	100
17	755_troj_12	✓	✓	✓	✓	28.57	14.28	85.7	100	50	33.3	75	75	67	33.3	67	100
18	755_troj_16	✓	✓	✓	✓	57.14	14.28	71.4	85.7	67	0	75	100	67	0	100	100
19	755_troj_19	✓	✗	✓	✓	28.57	0	**100**	**100**	25	0	**100**	**100**	33.3	0	**100**	**100**
20	755_troj_23	✓	✓	✓	✓	28.57	28.57	71.4	71.4	50	33.3	75	100	67	33.3	100	100
21	755_troj_24	✗	✗	✓	✓	0	0	100	100	0	0	75	75	0	0	100	100
22	738_troj_3	✓	✓	✓	✓	14.28	14.28	85.7	100	67	40	100	100	67	67	67	100
23	738_troj_4	✓	✓	✓	✓	42.8	28.57	100	85.7	67	33.3	75	100	67	33.3	67	67
24	738_troj_7	✓	✗	✓	✓	57.14	0	100	100	25	0	100	67	33.3	0	100	67
25	738_troj_12	✓	✗	✓	✓	57.14	0	71.4	100	67	20	100	100	67	33.3	100	100
26	738_troj_13	✓	✗	✓	✓	57.14	0	85.7	71.4	40	0	75	75	67	0	100	100
27	738_troj_16	✓	✓	✓	✓	28.57	28.57	85.7	71.4	40	40	100	67	67	33.3	67	100
28	738_troj_17	✓	✗	✓	✓	71.4	0	100	85.7	0	0	75	100	0	0	100	100
29	738_troj_23	✓	✓	✓	✓	28.57	28.57	**100**	**100**	25	33.3	**100**	**100**	33.3	33.3	**100**	**100**
30	738_troj_26	✓	✓	✓	✓	14.28	28.57	100	85.7	40	33.3	75	100	67	33.3	100	100
Average						36.18	20.93	**93.32**	**93.32**	43.04	22.47	**90.3**	**86.46**	52.39	27.81	**92.3**	**93.4**

The proposed framework detects Trojan-impacted nodes as well as the Trojan circuit with 100% accuracy and zero false positives for the **highlighted** cases. Across all evaluated scenarios, applying supervised-learning rules combined with few-shot learning yields higher average Trojan coverage, precision, and recall (marked in **bold**).

examples.

2) R: We provide supervised-learning rules without any few-shot examples.
3) R_{FS}: We provide supervised-learning rules followed by few-shot examples such that LLM can correlate the examples with the established rules, and use these examples to determine the Trojan-impacted nodes for a new test netlist based on these rules.
4) R^*_{FS}: We incorporate Rule #2 instead of Rule #1 (see Supervised Learning Rule 2), keeping the other rules unchanged, followed by few-shot examples.

To increase the complexity for LLM evaluation, we scrambled the Trojan components and nodes within the netlist as well as changed the parameters, such as the width-to-length (W/L) ratios of transistors and the capacitor values. The Trojan-inserted netlists used for the experiments are labeled as '$netlist\_troj\_n$', where $netlist$ is the specific circuit chosen from the AMSNet repository [10] and n indicates the netlist node where the Trojan payload is activated. Evaluation results are shown in Table V. We observe that incorporating the supervised-learning rules lead to a higher precision as well as overall accuracy in identifying the Trojan-impacted nodes. Additionally, the combination of few-shot examples with these rules enables the LLM to successfully identify the Trojan circuit in the benchmark netlists. The proposed method shows superior Trojan coverage and accuracy of Trojan-impacted nodes compared to scenarios where only few-shot examples are used without providing the supervised-learning rules.

The maximum number of tokens fitting within the context window for the GPT-3.5-turbo model is 16385. Average LLM inferencing time is 9.2 seconds across all cases of Trojan identification, thereby confirming the runtime efficiency of SPICED as it streamlines the process of analog design analysis and bug (both syntactical and functional) localization.

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.

VI. CONCLUSION

We have explored both syntactical and functional bug detection capabilities of the LLM. The proposed instruction-following approach significantly reduces the number of false positives while achieving high bug coverage. Additionally, by curating prompts with few-shot examples and CoT, LLM efficiently localizes the Trojan-impacted nodes for a range of Trojan-insertion scenarios while incurring zero area and power overheads. By incorporating the supervised learning rules in the prompt, we achieve an average Trojan coverage of 93.32% and an average true positive rate of 93.4% in identifying Trojan-impacted nodes for the evaluated analog benchmark circuits. This opens up new directions for securing analog designs from threats arising anywhere between design and fabrication stages of the chip.

REFERENCES

[1] M. M. Alam *et al.*, "Challenges and opportunities in analog and mixed signal (AMS) integrated circuit (IC) security," *Journal of Hardware and Systems Security*, vol. 2, pp. 15–32, 2018.

[2] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, 2014.

[3] B. Bilgic *et al.*, "Performance Degradation Monitoring for Analog Circuits Using Lightweight Built-in Components," in *VTS*, 2022, pp. 1–7.

[4] K. Yang *et al.*, "A2: Analog Malicious Hardware," in *IEEE SP*, 2016.

[5] N. Gupta *et al.*, "DELTA: DEsigning a Stealthy Trigger Mechanism for Analog Hardware Trojans and Its Detection Analysis," in *DAC*, 2022.

[6] M. Abedi *et al.*, "High-Precision Nano-Amp Current Sensor and Obfuscation based Analog Trojan Detection Circuit," in *ISCAS*, 2022.

[7] J. Talukdar *et al.*, "Automatic Structural Test Generation for Analog Circuits using Neural Twins," in *ITC*, 2022, pp. 145–154.

[8] J. Chaudhuri *et al.*, "DAWN: Efficient Trojan Detection in Analog Circuits using Circuit Watermarking and Neural Twins," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[9] S. Thakur *et al.*, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[10] Z. Tao *et al.*, "AMSNet: Netlist Dataset for AMS Circuits," *arXiv preprint arXiv:2405.09045*, 2024.

[11] Y. Lai *et al.*, "AnalogCoder: Analog Circuit Design via Training-Free Code Generation," *arXiv preprint arXiv:2405.14918*, 2024.

[12] J. Blocklove *et al.*, "Chip-chat: Challenges and opportunities in conversational hardware design," in *ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.

[13] M. Liu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[14] S. Thakur *et al.*, "Benchmarking large language models for automated verilog RTL code generation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[15] J. Bhandari *et al.*, "SENTAUR: Security EnhaNced Trojan Assessment Using LLMs Against Undesirable Revisions," *arXiv preprint arXiv:2407.12352*, 2024.

[16] Y. Tsai *et al.*, "Rtlfixer: Automatically fixing RTL syntax errors with large language models," *arXiv preprint arXiv:2311.16543*, 2023.

[17] H. Wu *et al.*, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[18] R. Zhong *et al.*, "Llm4eda: Emerging progress in large language models for electronic design automation," *arXiv preprint arXiv:2401.12224*, 2023.

[19] B. Roziere *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[20] Infineon, "adsbenchmark," https://rb.gy/1m2263.

[21] A. Vladimirescu, *The Spice Book*. USA: John Wiley & Sons, Inc., 1994.

[22] M. Tehranipoor *et al.*, "Trustworthy Hardware: Trojan Detection and Design-for-Trust Challenges," *Computer*, vol. 44, no. 7, pp. 66–74, 2011.

Authorized licensed use limited to: Arizona State University. Downloaded on September 26,2025 at 03:19:05 UTC from IEEE Xplore. Restrictions apply.