GHOST in the Machine: An AI-Assisted Flow for Hardware Trojan Insertion

A Case Study on an AES Cryptographic Core

Clarence Luo, Chirranjeavi Moorthi

ECE-GY 9941, GHOST Lab Report

November 2025

Automating Hardware Vulnerability Insertion is a Grand Challenge

Complexity

Hardware Trojans are subtle and deeply embedded in design logic. Manually crafting them for security testing is difficult and requires deep expertise.

Scale

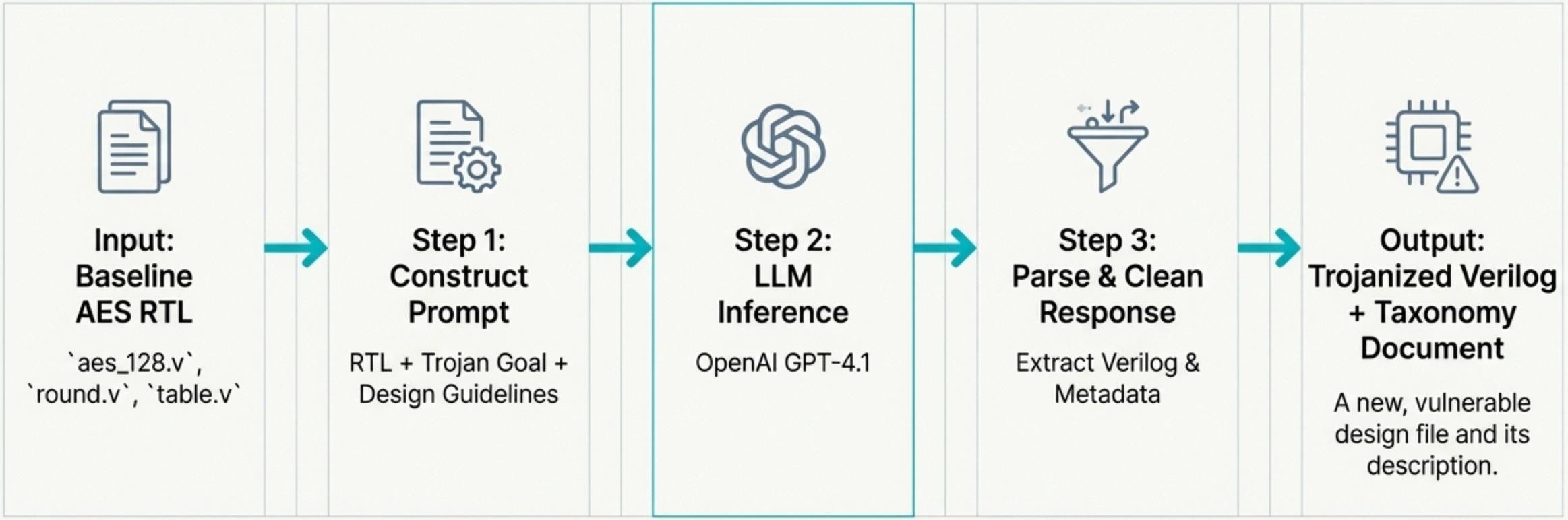
Creating a diverse set of Trojans to robustly test detection tools is a slow, resource-intensive, and moro..., and manual process that limits the scope of "red-teaming" exercises.

The Question

Can we leverage Large Language Models (LLMs) to automate the systematic and creative insertion of hardware Trojans based on a defined taxonomy?

Our Solution: A Fully Automated AI-Driven Pipeline

We built a pipeline to programmatically generate and insert hardware Trojans into baseline AES RTL using the GPT-4.1 model.



The Pipeline's Success Hinges on Three Key Components

Strategic Prompting

Combines base Verilog with specific goals (e.g., "Subtly alter logic...") and a list of strict "Synthesizable Design Guidelines" to constrain the AI.

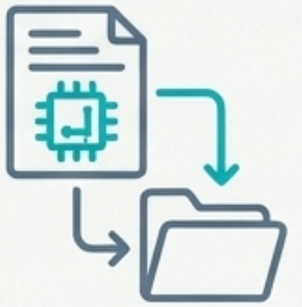
`construct_prompt()`

Robust Parsing

Extracts pure Verilog from markdown wrappers (`` ``` ``) and separates the AI's natural language explanation into structured metadata (Trigger, Payload, Taxonomy).

`extract_code_and_metadata()`

`clean_content()`



Intelligent Saving

Ensures synthesizability by clipping stray text after the final `` `endmodule` ``. Organises outputs into a logical directory structure with versioning (e.g., `` `round_HT1_gpt-4.1_A1.v` ``).

`save_vulnerable_design()`

The Pipeline Successfully Generated Four Distinct Trojan Taxonomies

T1: Change Functionality

Corrupts AES round output after a predetermined time delay.

T2: Leak Information

Creates a covert channel to exfiltrate the 128-bit encryption key.

T3: Denial of Service

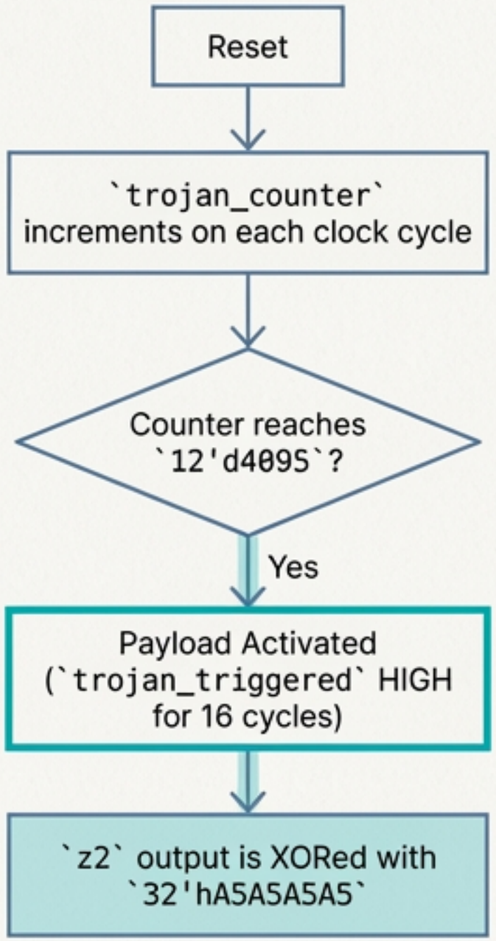
Temporarily freezes module output upon detection of a rare input sequence.

T4: Performance Degradation

Adds parasitic logic to a core helper module to degrade performance and correctness.

Trojan 1: Changing Functionality

Target Module: `one_round`
Objective: Subtly corrupt the AES state after a predetermined number of clock cycles.



T1 In Detail: Code & Verification

Trigger Logic

A simple time-based counter. A `reg [11:0] trojan_counter` increments after reset until it reaches `12'd4095`.

Payload Snippet

A new wire, `z2_trojaned`, replaces the original `z2` logic.

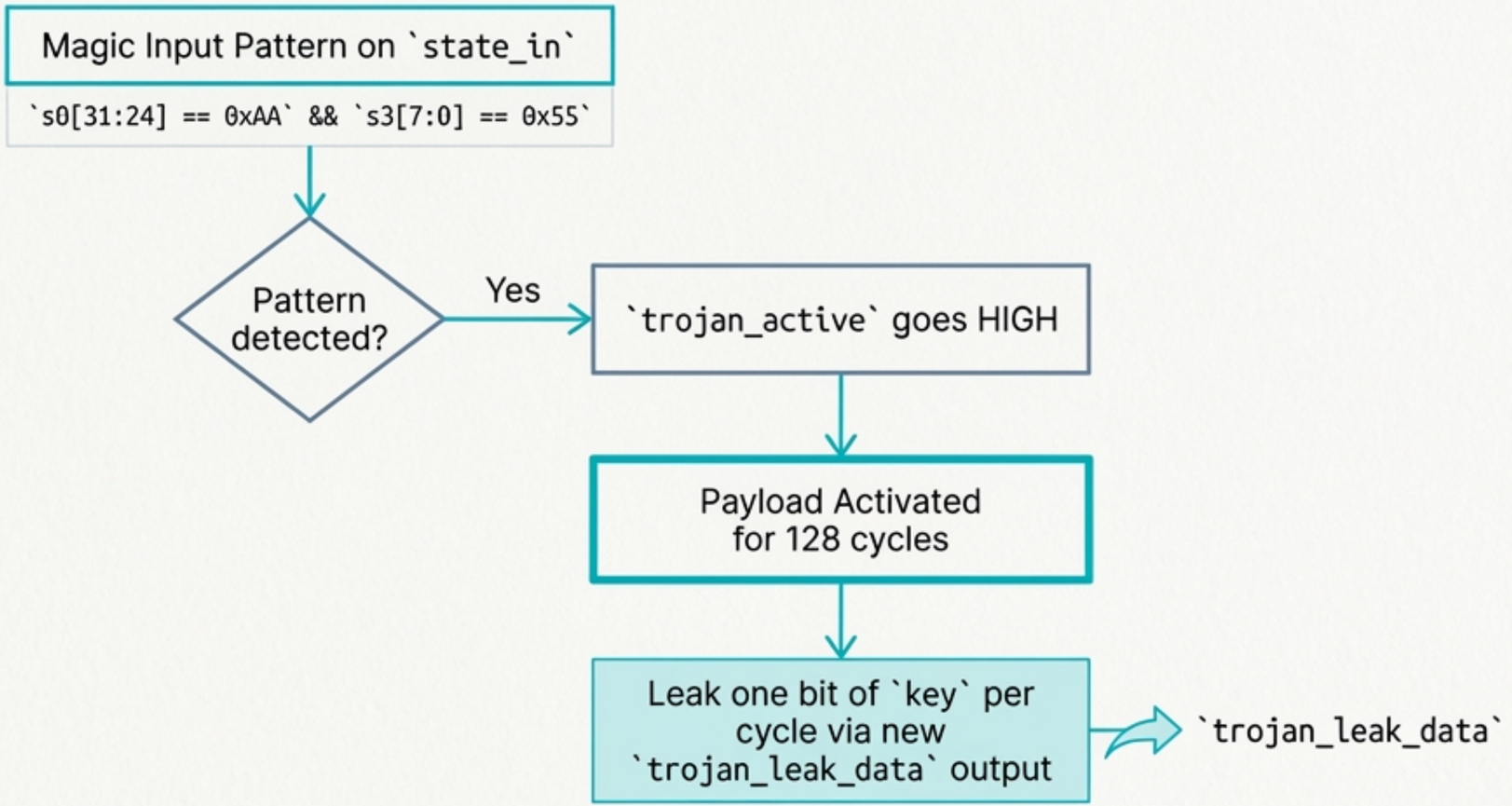
```
verilog verilog (z);
// A new wire, z2_trojaned, replaces the original z2 logic.
assign z2_trojaned = (trojan_triggered && trojan_state < 16) ?
    (p02 ^ p13 ^ p20 ^ p31 ^ k2) ^ 32'hA5A5A5A5 : // Corrupted Path
    (p02 ^ p13 ^ p20 ^ p31 ^ k2);                 // Normal Path
end verilog
```

Verification Plan

- Run with golden vectors; confirm outputs match before 4096 cycles.
- After 4096 cycles, confirm output on `z2` diverges from the golden reference.

Trojan 2: Information Leakage

Target Module: `one_round`
Objective: Create a covert side-channel to leak the 128-bit secret encryption key.



T2 In Detail: Code & Verification

1. Trigger Logic

A data-dependent ‘magic’ pattern on the input state.

```
wire trigger_condition =
  (s0[31:24] == 8'hAA) &&
  (s3[7:0] == 8'h55);
```

2. Payload Snippet

A **new output pin leaks** one **bit** of the secret key each cycle.

```
// A new output pin leaks one bit
of the secret key each cycle.
always @(posedge clk) begin
  if (trojan_active && trojan_cnt
< 128)
    trojan_leak_data <=
key[trojan_cnt];
end
```

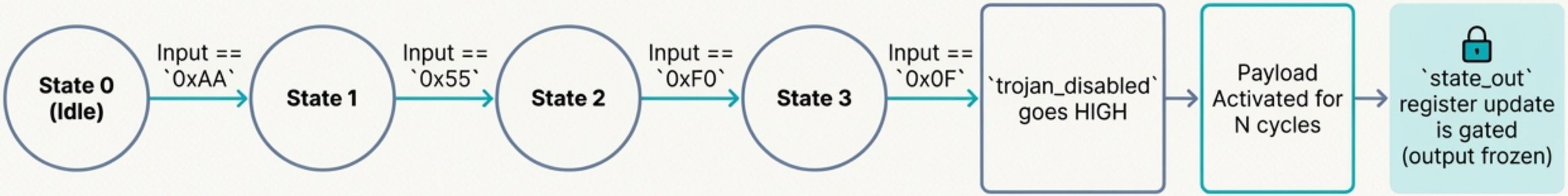
3. Verification Plan

- Apply the trigger pattern to `state_in`.
- Monitor the `trojan_leak_data` output for 128 cycles and reconstruct the full key.

Trojan 3: Denial of Service

Target Modules: `round` & `final_round`

Objective: Temporarily freeze the module's output, halting the encryption process, when a rare input sequence is detected.



T3 In Detail: Code & Verification

1. Trigger Logic

A 4-state FSM (`trojan_seq_state`) detects a specific byte sequence (`T_SEQ_0` to `T_SEQ_3`) over multiple cycles.

2. Payload Snippet

Gated logic in the main clocked block freezes the output.

```
// Gated logic in the main
clocked block freezes the output.
always @(posedge clk or negedge
rst_n) begin
  if (trojan_disabled)
    state_out <= state_out; //
  else
    state_out <= {z0, z1, z2, z3}
};
end
```

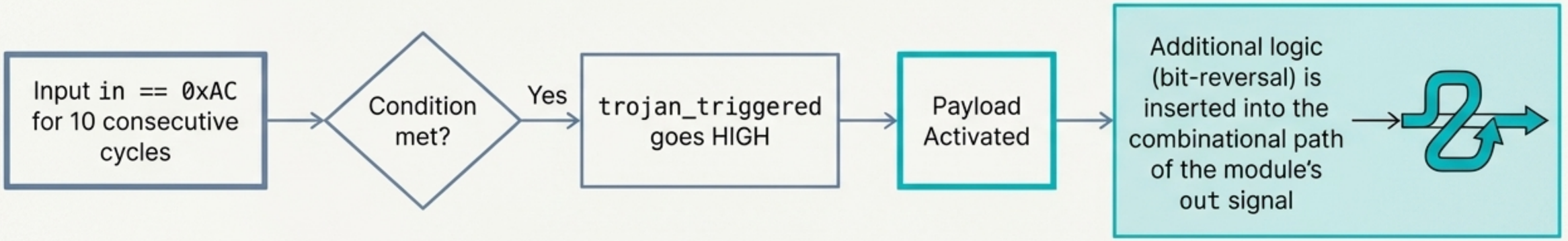
3. Verification Plan

- Apply the four-byte trigger sequence.
- Confirm in a waveform that `state_out` stops changing for the duration defined by `trojan_disable_cnt`.

Trojan 4: Performance Degradation

Target Module: T (S-box / MixColumns helper in `table.v`)

Objective: Add parasitic logic to a frequently used block, subtly degrading performance and correctness.



T4 In Detail: Code & Verification

1. Trigger Logic

A ‘magic’ repeating value (`8'hAC`) must be present on the input for 10 consecutive clock cycles, tracked by `trojan_counter`.

2. Payload Snippet

Parasitic logic path is activated inside a combinational block.

```
// Parasitic logic path is
activated inside a combinational
block.
if (trojan_triggered) begin
  temp_out[23:16] = {s0_out[0],
s0_out[7:1]}; // Bit-reversed byte
  ...
end else begin
  temp_out[23:16] = s0_out; //
Normal path
  ...
end
```

3. Verification Plan

- Apply the trigger input (`0xAC` for 10+ cycles).
- Confirm corrupted output. Optionally, use timing analysis to show increased path delay.

Key Learnings and Implications for AI-Driven Hardware Security

1. Prompt Design is Paramount

Success is highly dependent on clear, detailed prompts with strict 'Synthesizable Verilog' guidelines. LLMs require expert guidance to produce valid hardware.

2. The Post-Processor is Non-Negotiable

A robust parsing and cleaning layer (`extract_code_and_metadata`, `save_vulnerable_design`) is as critical as the AI model. It converts raw text into usable, error-free HDL.

```
// Example post-processor action
if (ai_generated_code matches "synthesizable") begin
  // Extract valid module definition
  vulnerable_design = extract_module(ai_generated_code);
end
```

3. A New Frontier for Red-Teaming

AI-assisted flows can rapidly generate a diverse corpus of vulnerabilities. This enables more robust and comprehensive testing of next-generation hardware next-generation hardware security and verification tools.

An Automated Pipeline for Hardware Sabotage

We successfully demonstrated an end-to-end, automated pipeline capable of inserting four distinct classes of hardware Trojans into an AES cryptographic core using a Large Language Model.

This work shows that AI-driven flows are a viable method for systematically exploring and generating hardware vulnerabilities, providing a powerful new tool for security research and verification.