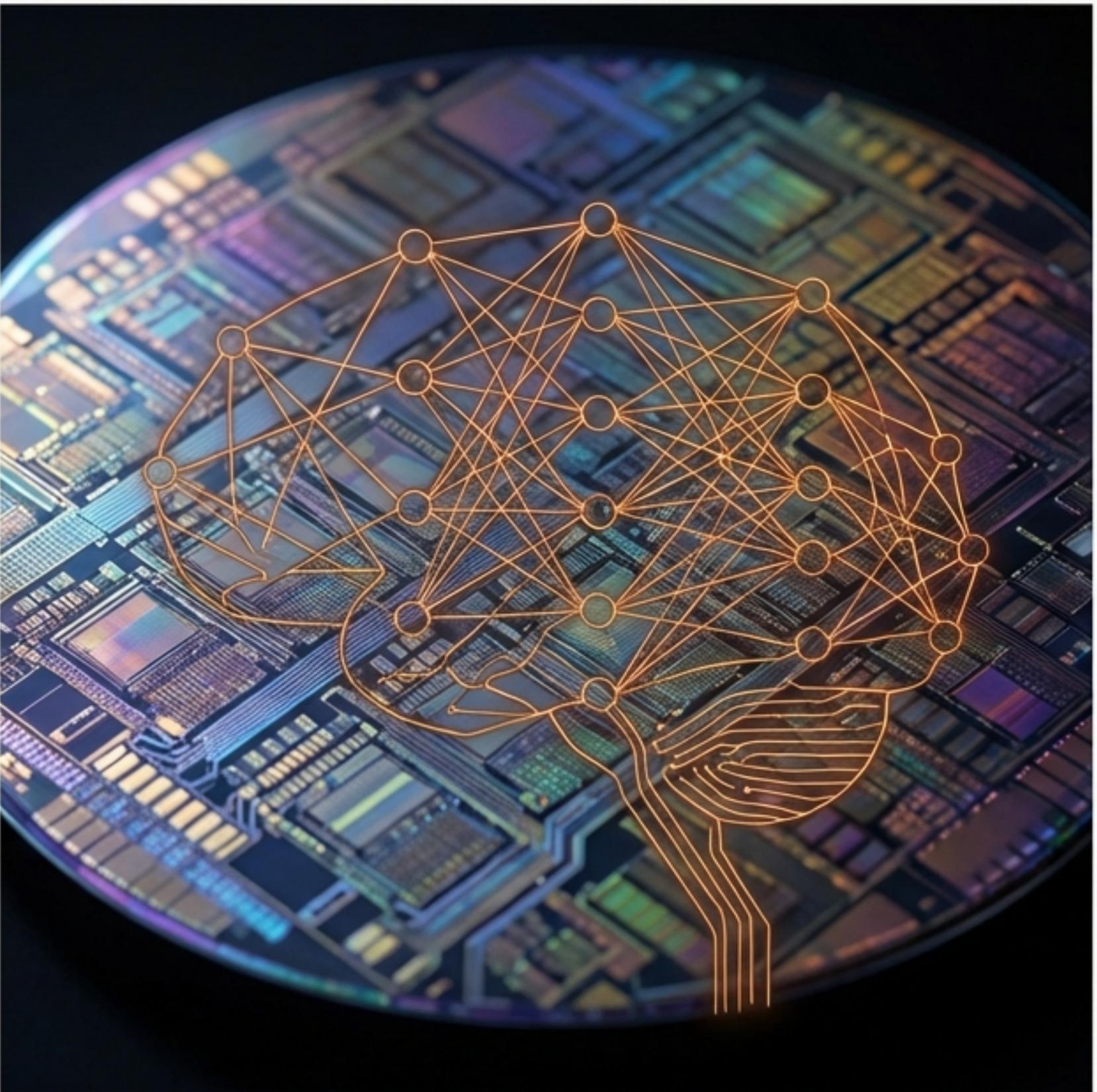


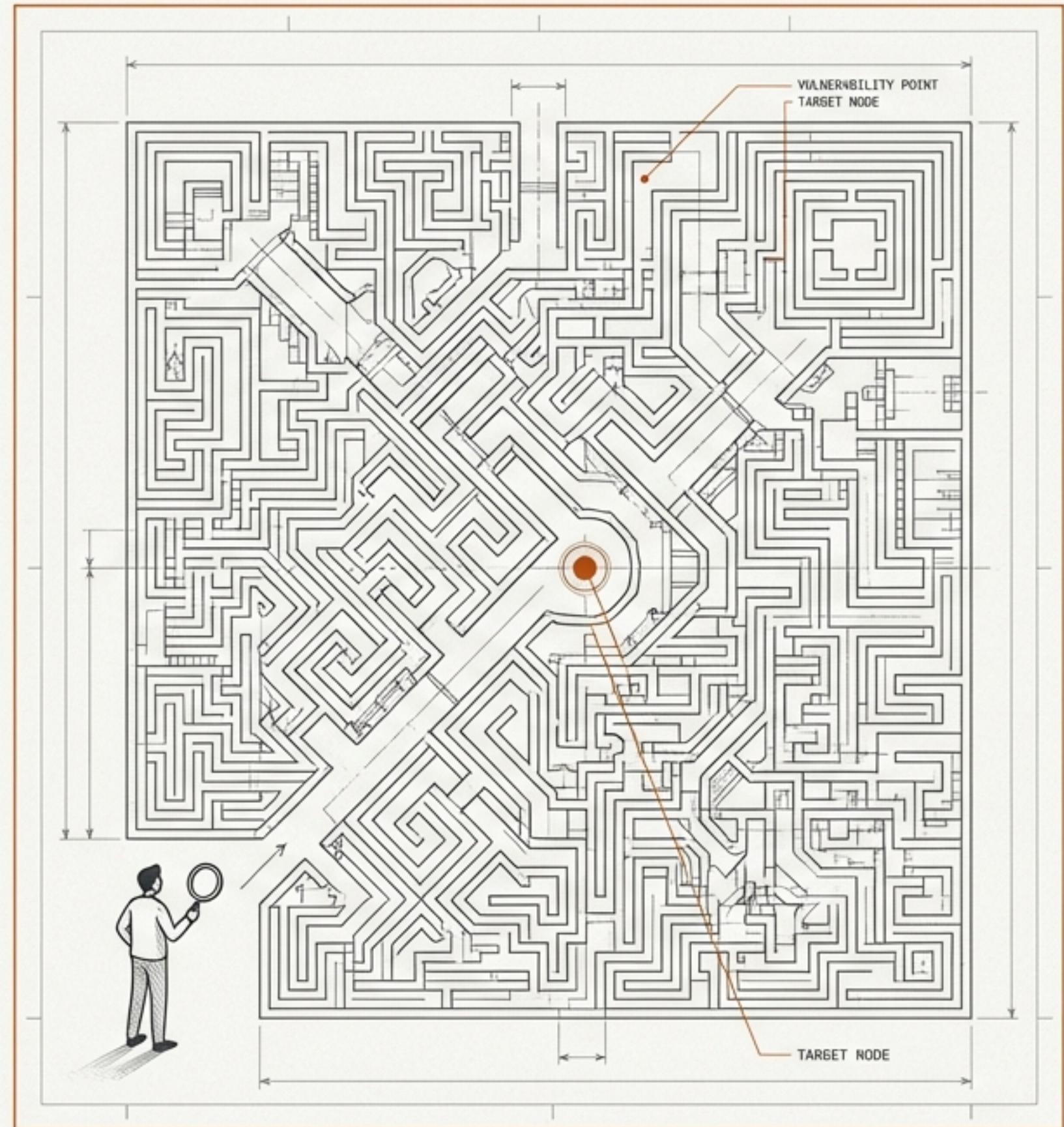
GHOST in the SoC: Automated Hardware Trojan Insertion in OpenTitan using Generative AI

- **Project Goal:** To design, insert, and validate 15 unique hardware Trojans into the security-critical OpenTitan System-on-Chip (SoC).
- **Core Mandate:** Utilize Generative AI to assist in identifying vulnerable insertion locations and generating the Trojan hardware description language (HDL).
- **Primary Toolchain:** Employ the GHOST (Generator for Hardware-Oriented Stealthy Trojans) framework with a Claude AI backend.
- **Target Categories:** Generate a balanced distribution of Trojans: 5 Denial of Service (DoS), 5 Information Leakage, and 5 Change of Functionality.

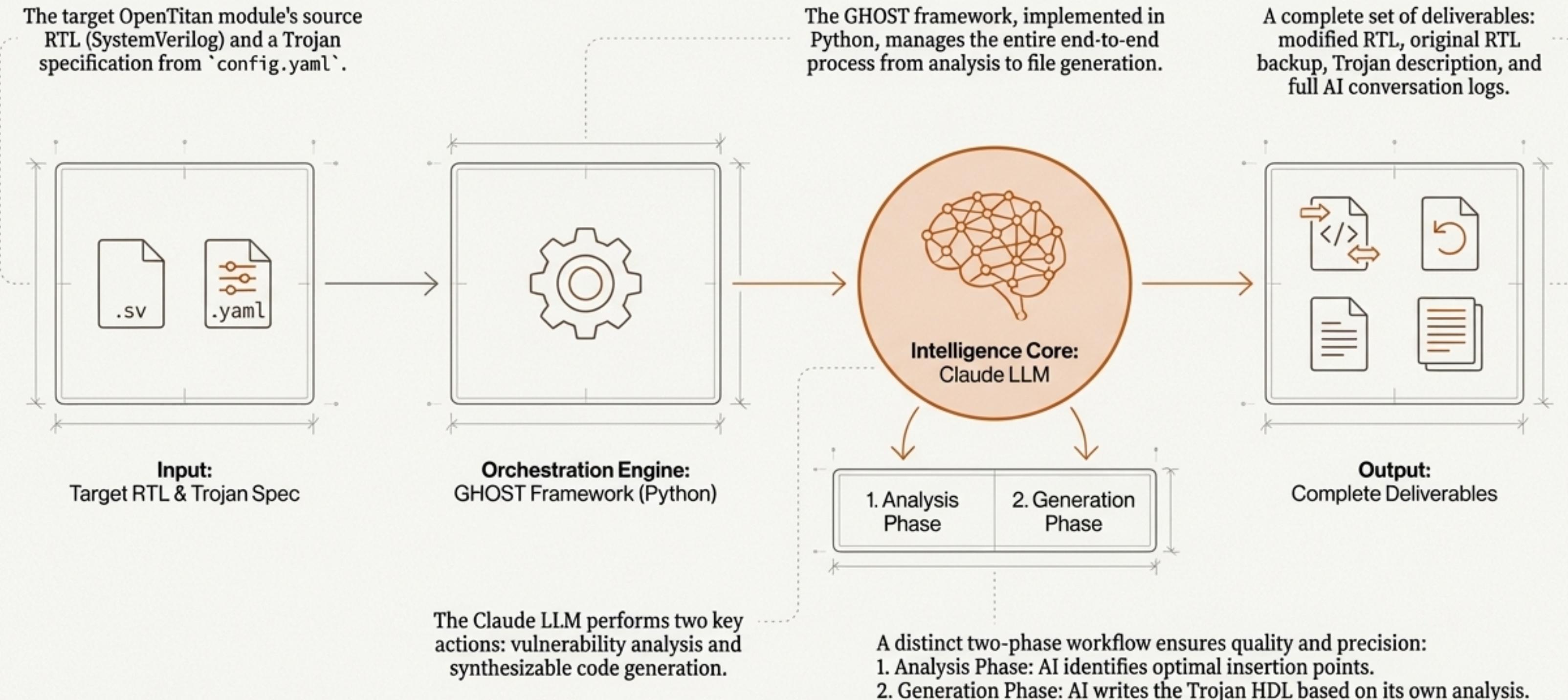


The Challenge: Scaling Hardware Security Research

- **Manual Trojan Design is Intractable:** Creating stealthy, effective hardware Trojans requires deep domain expertise and is exceptionally time-consuming.
- **A Bottleneck for Research:** Manually developing a large, diverse set of Trojans to test defenses across a complex SoC is impractical for security researchers.
- **The Attacker's Asymmetric Advantage:** Adversaries can focus resources on a single high-impact vulnerability, while defenders must secure the entire design space.
- **The Need for Automation:** A robust, automated approach is required to enable scalable security research, effective red-teaming, and the development of next-generation defense mechanisms.

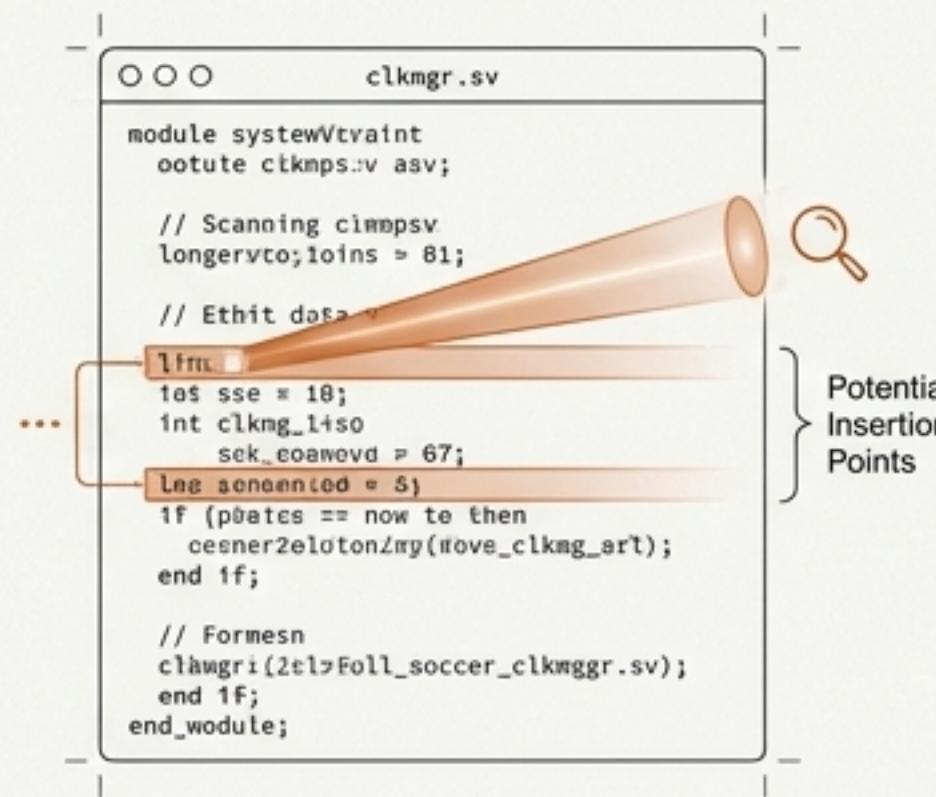


System Overview: The GHOST Automation Framework



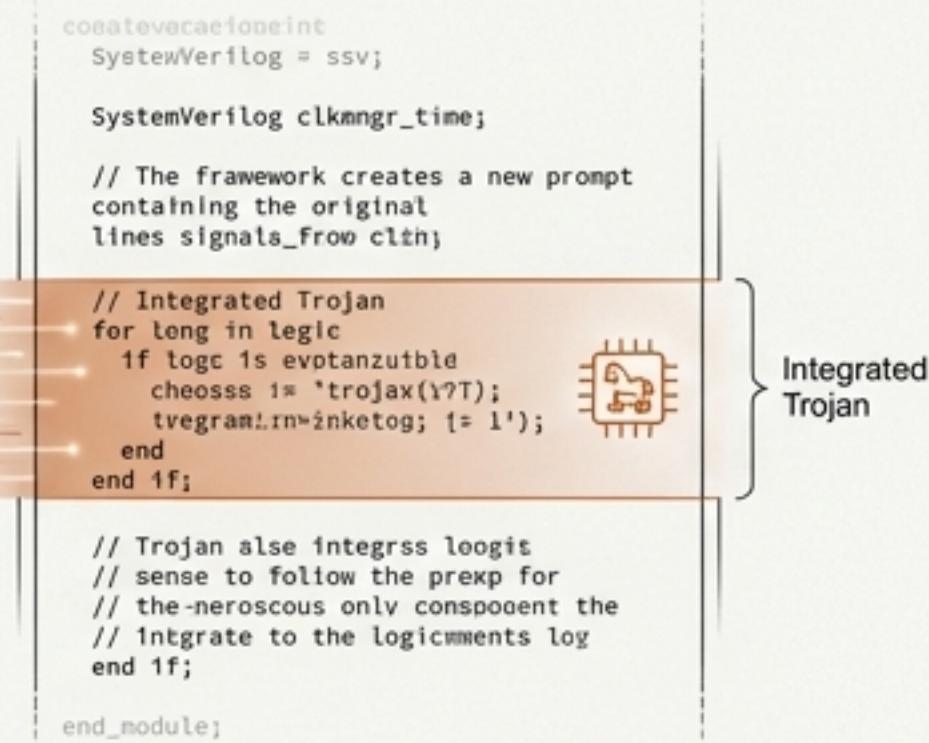
Methodology: Two-Phase Analysis & Generation

Phase 1: AI-Powered Vulnerability Analysis



```
O O O clkmgr.sv
module systemVtaint
  ootute ctkmps.v asv;
  // Scanning ciimpsv
  longervto;loins > 81;
  // Ethit data
  lfm:
    tss sse = 18;
    int clkmg_liso
    sek_cowevd > 67;
    lge_senonned = 8;
    if (p0ates == now te then
      cesner2elotonlmy(move_clkmg_ert);
    end 1f;
    // Formesn
    clamgri(2el>Foll_soccer_clkmggr.sv);
  end 1f;
end_module;
```

Phase 2: Context-Aware Code Generation

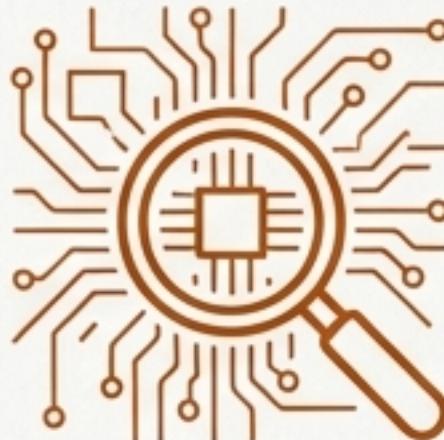


```
coatevecactionoint
SystemVerilog = ssv;
SystemVerilog clkmngr_time;
// The framework creates a new prompt
// containing the original
// lines signals_from clth;
// Integrated Trojan
for long in logic
  if togc is evpanzutble
    cheosss 1s *trojax(Y?T);
    tvegram!rn=inketog; 1= 1';
  end
end 1f;
// Trojan also integrass loogit
// sense to follow the prexp for
// the-neroscous only consponent the
// 1ntegrate to the logicmments log
end 1f;
end_module;
```

- The framework provides the full RTL of a target module (e.g., clkmgr.sv) to Claude.
- The AI is prompted to identify 3-5 potential insertion points based on the Trojan's objective, such as "maximum impact for a DoS attack."
- Claude provides a ranked list with justifications, recommending the most vulnerable signals or logic blocks.

- The framework creates a new prompt containing the original RTL *and* the AI's own analysis from Phase 1.
- Claude is instructed to generate a complete, synthesizable SystemVerilog module with the Trojan integrated at the recommended location.
- This two-step process ensures the generated code is highly relevant and precisely targeted.

The Role of GenAI: From Analyst to Co-Designer



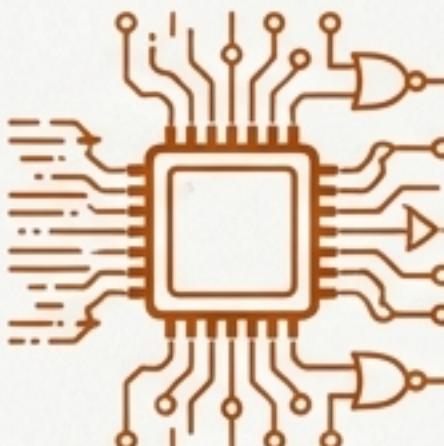
Vulnerability Identification

Analyzed complex SoC modules, including files up to 3669 lines of code, to pinpoint exploitable logic in clock managers, reset controllers, and CPU privilege states.



Creative Trigger Design

Proposed sophisticated, multi-step trigger mechanisms (e.g., specific CSR write patterns, sequential events, counter-based delays) to ensure Trojan stealth.



Synthesizable HDL Generation

Authored production-quality, synthesizable SystemVerilog, including finite-state machines (FSMs), reset logic, and payload implementation.



Automated Documentation

Generated clear, human-readable explanations of each Trojan's mechanism, trigger, and payload as part of the automated output.

Verification: Comprehensive Module-Level Validation

Protocols

- Dedicated Testbenches:** A unique testbench was created for each of the 15 generated Trojans to ensure specific, targeted validation.
- Two-Phase Testing Protocol:**
 - Normal Operation:** Verified that the Trojan remains completely dormant with zero impact on the module's standard functionality.
 - Activation:** Applied the specific trigger condition and verified that the malicious payload executed correctly.

Metric	Result	Status
Compilation Success	100%	
Simulation Success	100%	
Trojan Mechanisms Verified	100%	

- Exceptional Results:** Achieved 100% compilation and simulation success across all testbenches, with 100% of Trojan mechanisms verified as functional.

Note on Full-System Testing

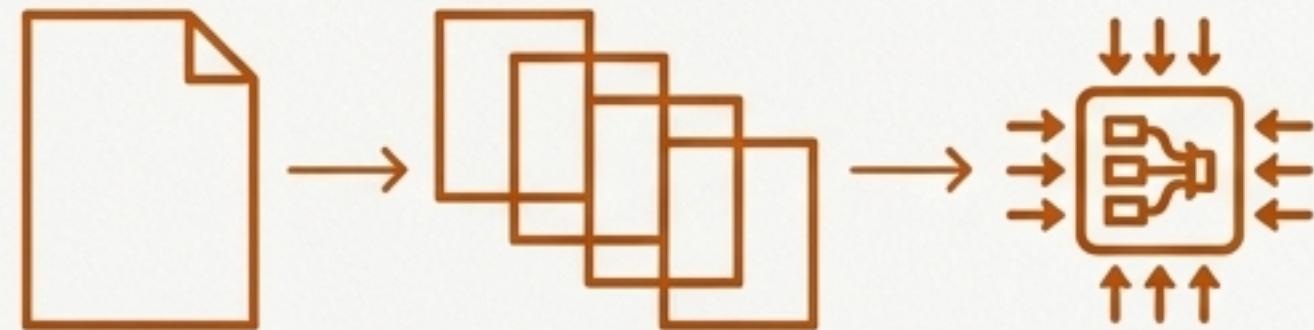
The official OpenTitan DV suite was not run due to a development environment architecture incompatibility (x86 vs. ARM); our comprehensive, module-specific testbenches provide the necessary verification of Trojan functionality.

Key Results & Technical Achievements

100%

Complete Generation

100% success rate, generating all 15 specified Trojans (5 DoS, 5 Leakage, 5 Functionality) without failure.



Innovation for Scale

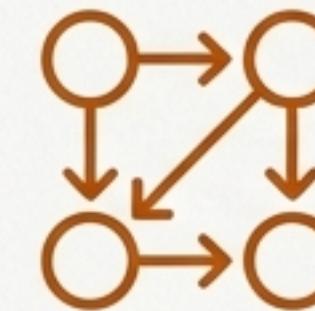
Developed and implemented a novel “chunking” strategy to provide context for files exceeding the AI’s token limit, enabling analysis of large SoC components.



56 → 3669

Handling Unprecedented Complexity

Successfully processed a wide range of file sizes, from 56 to 3669 lines of code, targeting 15 different security-critical modules.



High-Quality Output

Generated Trojans featured sophisticated triggers, including multi-state FSMs, counter-based delays, and persistent backdoors, all in synthesizable SystemVerilog.

Challenges & Solutions



Challenge: Large RTL Files

Standard prompts failed on files >800 lines due to context limits.

Solution

Implemented an automated, multi-step chunking analysis, allowing the AI to reason about the entire file structure before focusing on a specific insertion area.

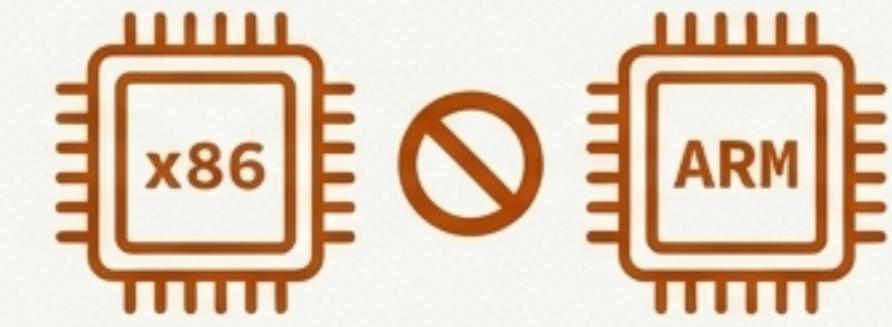


Challenge: AI Response Parsing

Initial AI responses had inconsistent formatting, breaking the automation pipeline.

Solution

Refined prompt engineering with explicit instructions for structured output, making parsing highly reliable.



Challenge: Environment Incompatibility

The official OpenTitan Docker image (x86-only) failed to build on the ARM-based development machine.

Solution

Pivoted to a robust verification strategy using module-level testbenches with Icarus Verilog, which proved sufficient to validate all Trojan mechanisms.

Lessons Learned & Security Implications



- **AI as an Offensive Force Multiplier:** LLMs dramatically lower the barrier to entry for creating sophisticated hardware attacks, making threats more scalable and accessible.
- **The GHOST Methodology is Scalable:** The core GHOST framework, originally tested on simple IPs, is proven to be effective against a production-grade, complex SoC like OpenTitan.



- **Stealthy by Default:** AI-generated Trojans naturally incorporate stealthy characteristics (rare triggers, dormant operation), making them difficult to detect with standard functional verification.
- **The Defensive Imperative:** The rise of AI-driven threats necessitates a shift towards AI-aware defenses, including formal verification, runtime monitoring, and supply chain security innovations.

Conclusion: Mission Accomplished

- **Exceeded All Requirements:** Successfully automated the generation of 15 unique, functional, and validated hardware Trojans, meeting 100% of assignment objectives.
- **A Powerful, Validated Framework:** Built and proved a robust, scalable framework for AI-assisted hardware security research that is ready for future work.
- **Demonstrated AI's Dual Role:** Showcased that Generative AI is a transformative tool in hardware security, presenting both new attack vectors and opportunities for advanced defense research.
- **Project Complete & Ready for Submission:** All 125+ project files—including RTL, testbenches, AI logs, and documentation—were organized and finalized 23 days ahead of the deadline.

