SPECTERRTL: LLM-ORCHESTRATED HARDWARE TROJAN EVOLUTION FOR AUTOMATED RTL VULNERABILITY INJECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-assisted hardware design creates a new attack surface for automated RTL red teaming. Prior approaches are constrained by limited autonomy, weak adversarial evolution, and hardware security data scarcity. We present **SpecterRTL**, an end-to-end framework for autonomous Hardware Trojan construction and evolution. SpecterRTL establishes a self-sustaining closed loop where design-aware vulnerability mining extracts targeted attack seeds that fuel adversarial evolution to create previously unseen threat variants. Continuously absorbing newly evolved attacks into foundational threat knowledge mitigates data scarcity and drives increasingly sophisticated exploration. Across 137 RTL modules spanning 24 categories, SpecterRTL produces an average of 5.27 verified Trojan seeds per module and raises the proportion of high-risk instances from 12.5% to 92.3% on representative evolution cases. On the clean designs shared with the current SOTA RTL red-teaming benchmark, SpecterRTL produces **42.4**× more verified Hardware Trojan instances and a **35**× higher successful evasion count across three representative detectors, advancing RTL security evaluation from template-constrained generation to self-sustaining threat evolution.

1 INTRODUCTION

The rapid proliferation of Large Language Models (LLMs) has surged demand for customized AI accelerators (Kachris, 2025) and catalyzed an LLM-driven paradigm in agile hardware design (Pan et al., 2025), prominently integrating LLMs into Register-Transfer Level (RTL) design pipelines. In tandem, these new automated entities insert novel security variables into the hardware supply chain, presenting unprecedented challenges to contemporary hardware assurance.

Unlike software security, which benefits from extensive vulnerability databases (Karim & Akter, 2026) and mature automated testing such as fuzzing and symbolic execution (Li et al., 2018; Baldoni et al., 2018), hardware security faces fundamental structural constraints and severe data scarcity. This gap makes **automated red teaming** (Perez et al., 2022; Ganguli et al., 2022; Maloyan et al., 2024; Ferrag et al., 2025) indispensable for proactively stress-testing security boundaries and validating system resilience before deployment.

Among multiple abstraction levels in EDA workflows, RTL has consistently been the principal target (Gohil et al., 2022; Faruque et al., 2025; Sreekumar et al., 2026), most critically embodied by **Hardware Trojans (HTs)** (Chakraborty et al., 2009; Bhunia et al., 2014). An HT pairs a highly evasive activation condition (*trigger*) with a destructive behavior (*payload*). Unlike simple bugs, this structure demands deep semantic understanding of control-data coupling, allowing HTs to remain entirely dormant during functional verification, evade standard security checks, and survive into final silicon, inflicting catastrophic damage only upon deployment. Historically, HT insertion has depended heavily on manual expertise, but LLM-aided design threatens to democratize it. Because the open-source RTL ecosystem is data-constrained and labor-intensive to verify thoroughly, an adversary contaminating even a small fraction of public repositories can poison generative models, embedding persistent vulnerabilities.

Despite the severity of this threat, existing red-teaming work remain fundamentally inadequate due to three challenges. **C1: Autonomy Bottleneck.** Both heuristic and LLM-assisted approaches

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

rely on manually drafted templates or hardcoded insertion patterns, functioning as rigid code generators. Constrained by human oversight or rigid specifications, these systems produce injections that manifest as detectable functional anomalies rather than covert vulnerabilities. **C2: Evolutionary Deficiency.** While defense mechanisms have advanced through multi-stage detection pipelines, LLM-driven red teaming frameworks have remained largely static. Without closed-loop engagement against SOTA detectors, current tools generate Trojans that pass basic verification yet fail under modern security scrutiny. **C3: Data Scarcity and Homogeneity.** As machine learning increasingly dominates hardware security research, model performance is fundamentally limited by dataset quality and diversity. Existing corpora exhibit severe structural homogeneity, while recent augmentation efforts yield primarily syntactic perturbations of known templates rather than architecturally novel threat patterns.

To address these gaps, we introduce **SpecterRTL**, an end-to-end RTL red-teaming framework that shifts hardware security evaluation from template-constrained generation to self-sustaining threat evolution. Design-aware vulnerability mining extracts targeted attack seeds, fueling adversarial evolution to create previously unseen variants. Continuously absorbing these newly evolved attacks to enrich foundational threat strategies empowers the framework to overcome data scarcity and drive increasingly sophisticated exploration, enabling the systematic construction of high-impact threats while expanding hardware threat intelligence for resilience assessment, benchmarking, and defense research.

This research makes the following key contributions:

- We present, to the best of our knowledge, the first end-to-end RTL red-teaming framework that autonomously spans vulnerability mining, exploit realization, and adversarial evolution for Hardware Trojans. **SpecterRTL** performs full red-team reasoning over target RTLs and turns the resulting findings into verifiable Trojan implementations.
- We introduce **Design-Aware Vulnerability Exploration (DAVE)**, an autonomous RTL vulnerability mining paradigm grounded by a Hardware Vulnerability Knowledge Graph (VKG). DAVE establishes a self-sustaining closed loop in which VKG is continuously enriched by emergent attack strategies and novel instances generated during the downstream evolution phase, thereby enabling open-ended threat exploration.
- We introduce **Population-based Hardware Adversarial Genetic Evolution (PHAGE)**, an adversarial evolution algorithm that shifts RTL red teaming from static generation to dynamic threat creation. By orchestrating *iterative mutation* and *cross-context hybridization*, PHAGE not only generalizes existing vulnerability patterns into arbitrary hardware contexts but evolves previously unseen Trojan structures within the RTL semantic space.
- We conduct extensive evaluations on a representative RTL design suite to establish a comprehensive benchmark for RTL red teaming. On clean designs shared with the current SOTA benchmark, SpecterRTL produces **42.4**$\times$ more verified Trojan instances and a **35**$\times$ higher successful evasion count against modern detectors, while elevating the proportion of high-risk threats to 93.1%.

2 BACKGROUND

Distinct Constraints of Hardware Security. Hardware security draws on a far thinner knowledge base than its software counterpart. Vulnerability disclosure density in hardware (Figure 1, left) remains far lower than in software, making hardware vulnerability knowledge not merely scarce but ungroundable in the way software knowledge is, explaining its dependence on domain expertise and labor-intensive manual effort (Zhang et al., 2014; Dharsee & Criswell, 2023; Salmani et al., 2013). Beyond this scarcity, three properties make defense intrinsically harder. First, multi-stakeholder IP coordination and prohibitive post-fabrication remediation costs keep disclosures restricted (MITRE, 2025). Second, verification-stage defense is challenging because a malicious modification can pass functional linting, simulation, and logic synthesis while remaining undetected (Figure 1, right). Third, even systematic testing cannot cover the activation conditions that manifest only at deployment under circumstances the defender cannot enumerate in advance. These properties make a Hardware Trojan a distinct adversarial object, fundamentally harder to prevent, detect, and eliminate than a software vulnerability.

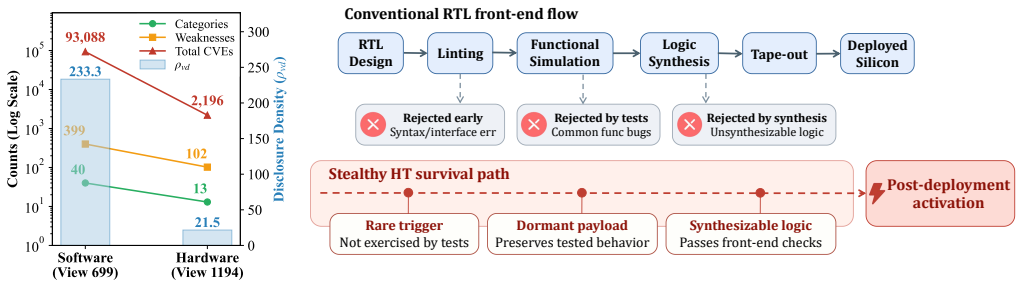

Figure 1: Left: Disparity in vulnerability disclosure density (ρ_{vd}) from NVD CVE data across the Software Development View (CWE-699) and Hardware Design View (CWE-1194). Right: The industrial hardware design and verification pipeline, contrasting verification-stage errors with stealthy hardware Trojans.

LLM-Aided Red Teaming. LLM-based red teaming has undergone a paradigm shift. Early efforts were training-based, using data poisoning or weight perturbation (Aghakhani et al., 2024; Yan et al., 2024; Li et al., 2025). With the emergence of frontier models, the field has moved toward *training-free* strategies, black-box attacks through prompt injection, retrieval-augmented generation, and agentic workflows (Jenko et al., 2025; Cheng et al., 2025; Ouyang et al., 2025; Qu et al., 2025). The paradigms are complementary, since training-based methods excel at *knowledge utilization* by replicating known vulnerability patterns at scale, while training-free methods excel at *knowledge discovery* by synthesizing findings from limited evidence or high-level instructions. This shift toward training-free reasoning makes LLM-aided red teaming viable in hardware, where the limited disclosure base cannot support training-based approaches. Meanwhile, LLMs enter the design flow not only as a red-teaming instrument but as an attack surface. Because they are now embedded in the agile hardware supply chain, from specification translation to RTL and HLS generation (Thakur et al., 2023; Blocklove et al., 2023; Chang et al., 2023; 2024; Wang et al., 2024; Liu et al., 2024; Niu et al., 2025; Xiong et al., 2024), a compromised or manipulated assistant can inject context-aware Trojans that mirror legitimate coding style and pass manual review. This duality is what makes auditing LLM-aided design both necessary and urgent.

Prior Work of Hardware Red Teaming. Existing hardware red-teaming tools share three limitations, as summarized in Table 1. First, they stay tethered to hardcoded payload templates (Cruz et al., 2022; Gohil et al., 2022; Sarihi et al., 2024) or human-crafted trigger specifications (Saha et al., 2024; Faruque et al., 2025; Sreekumar et al., 2026), confining evaluation to simple, homogeneous modules and producing basic bugs rather than stealthy Trojans. Second, offensive development lags the defensive frontier (Tarek et al., 2025; Fu et al., 2023). Only a handful of offensive frameworks directly confront detectors (Gohil et al., 2022; Faruque et al., 2025) and even these are restricted to non-SOTA open-source tools because advanced defenses remain closed-source (Fu et al., 2022; Hayashi & Vicente Ruggiero, 2025; Faruque et al., 2026), so generated samples may evade functional simulation yet collapse under realistic detectors. Third, continued reliance on traditional benchmarks (Yasaei et al., 2025; Hayashi & Vicente Ruggiero, 2025; Faruque et al., 2026; Guo et al., 2019; Sarihi et al., 2024) impairs cross-context generalization and offers no mechanism to accumulate or transfer adversarial knowledge across design contexts.

3 Threat Model

We formalize three primary attack vectors in LLM-aided hardware supply chains.

T1. Compromised Design Automation Tools. Adversaries exploit automated EDA toolchains and LLM-based copilots integrated into agile hardware design workflows. By manipulating prompt contexts or compromising upstream updates, attackers force these assistants to generate context-aware, Trojan-infected RTL during code generation, refactoring, or optimization phases. Because the synthesized vulnerabilities naturally mirror legitimate coding patterns and the surrounding design style, they bypass traditional manual code reviews.

Table 1: Comparison of Hardware Trojan Insertion Tools and Red-Teaming Frameworks. Legend: ● = Full/Comprehensive, ◐ = Partial/Limited, ○ = None/Not Supported.

Tool	Paradigm	Knowledge Source	Threat Reasoning	Verification Compliance	Adversarial Evasion	Guided Evolution
MIMIC (Cruz et al., 2022)	ML	ISCAS-89	○	○	○	○
ATTRITION (Gohil et al., 2022)	RL	Open-source IPs	◐	○	●	○
DTjRTL (Dai et al., 2024)	Config	Human Expert	○	●	○	○
TrojanPlayground (Sarihi et al., 2024)	RL	ISCAS-85	◐	○	●	○
Kokolakis et al. (Kokolakis et al., 2024)	LLM	Instructions	○	○	○	○
SecRT-LLM (Saha et al., 2024)	LLM	Instructions	◐	○	●	○
GHOST (Faruque et al., 2025)	LLM	Instructions	◐	●	◐	○
TrojanGYM (Sreekumar et al., 2026)	LLM	Instructions	◐	●	●	◐
SpecterRTL (this work)	**LLM + Agent**	**Self-Evolving KG**	●	●	●	●

T2. Malicious Third-Party IP Vendors. Untrusted IP suppliers embed covert vulnerabilities into reusable RTL modules before distribution. The infusion of LLMs into hardware design radically amplifies this threat by shifting HT synthesis from manual crafting to automated, high-throughput, and verification-aware generation. LLM-driven adversaries can systematically ensure that poisoned IP passes traditional isolation verification while guaranteeing deterministic activation post-SoC integration, transforming supply chain contamination into an industrialized, polymorphic threat.

T3. Contaminated Training Datasets. Adversaries publish tainted open-source datasets to poison LLMs. RTL constitutes only a small fraction of publicly available corpora, and base models lack the intrinsic capacity to distinguish clean from malicious RTL. No tooling exists to screen for such contamination at scale. Functional verification is labor-intensive, and existing open-source HDL datasets are often modularized and lack full project context, further limiting correctness checks. Attackers can exploit these weaknesses to distribute poisoned datasets on open platforms, embedding errors that evade conventional checks.

4 METHODOLOGY

4.1 OPTIMIZATION MODEL

We first frame automated HT generation as a constrained, multi-objective optimization problem within a high-dimensional, discrete semantic space Ω.

Constraint Definition. Let $\mathcal{V} : \Omega \to \{0, 1\}$ denote the Boolean verification pipeline. A modified design is *verified* if it bypasses all functional and structural checks.

$$\mathcal{V}(x_{HT}) = \begin{cases} 1, & \text{if passes Lint, Func Sim, and Logic Synth} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Objective 1: Vulnerability Rating. We define $f_R(x_{HT})$ as a composite metric that quantifies the stealth severity of the implanted HT, aggregating multiple heuristic dimensions presented in Section 4.2.

Objective 2: Detection Evasion. Let $P_{detect}(x_{HT}) \in [0, 1]$ represent the detection probability or confidence score output by a suite of SOTA HT detectors. We define the evasion capability as $f_E(x_{HT}) = 1 - P_{detect}(x_{HT})$.

Optimization Formulation. The red-teaming task is formalized as

$$\begin{aligned} x^*_{HT} = \arg\max_{x_{HT} \in \Omega} \quad & \Big(\omega \cdot f_R(x_{HT}) + (1 - \omega) \cdot f_E(x_{HT})\Big) \\ \text{s.t.} \quad & \mathcal{V}(x_{HT}) = 1 \\ & x_{HT} \sim \mathcal{T}(x_{clean}) \end{aligned} \tag{2}$$

where $\omega \in [0, 1]$ is a balancing parameter, and $\mathcal{T}(\cdot)$ denotes the semantic transformation function for the clean design.

Optimization Strategy. Solving Equation 2 involves optimizing within a discrete context. Because Ω represents a non-convex, non-differentiable space of code semantics, continuous gradient-based

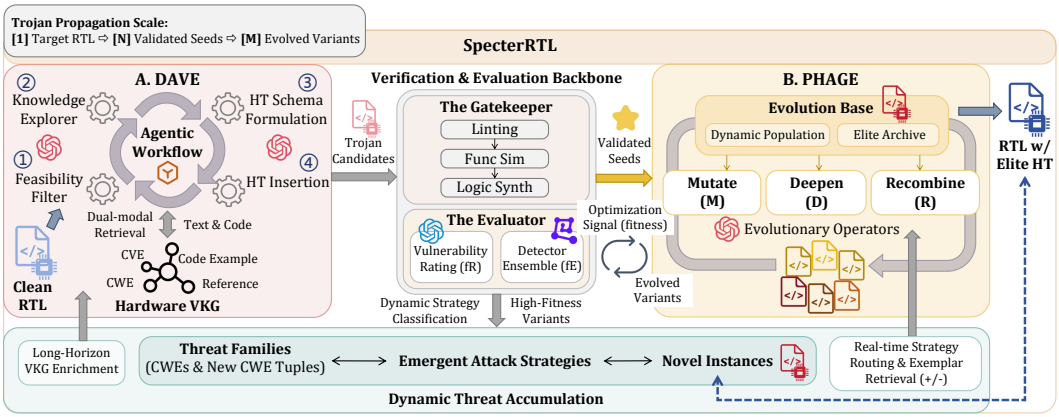

Figure 2: SpecterRTL overview.

updates incur significant approximation overhead and unguided brute-force exploration yields exceptionally low efficiency. To navigate this discrete space efficiently, SpecterRTL orchestrates a synergistic closed-loop of three coupled components, illustrated in Figure 2.

4.2 Framework Components

Design-Aware Vulnerability Exploration (DAVE). DAVE autonomously mines design-specific vulnerabilities by grounding threat reasoning in weakness mechanisms that are physically realizable and semantically feasible for the target hardware. This reasoning is anchored by the Hardware Vulnerability Knowledge Graph (VKG), a dynamic knowledge base that organizes CWE categories alongside concrete CVE evidence, code examples, and reference documents.

An interactive agentic workflow prevents vulnerability research from degrading into standard text-based retrieval. The *Security Auditor* and *Knowledge Explorer* engage in iterative dual-modal text and code retrieval against the VKG to filter infeasible targets and consolidate relevant structural context. The *Defense Architect* then converts this accumulated evidence into a ranked vulnerability schema defining precise activation conditions and payload behaviors. Guided by this schema, the *Red Tester* instantiates complete RTL modifications, and only the verified ones are preserved as seeds. By grounding agentic reasoning in the dynamically accumulated VKG, DAVE bridges the gap between abstract weakness specification and concrete exploit realization, ensuring that downstream adversarial evolution inherits only verified, design-specific attack seeds.

Population-based Hardware Adversarial Genetic Evolution (PHAGE). Verified seeds demonstrate initial feasibility but do not create the sustained offensive pressure required for rigorous adversarial evaluation. PHAGE treats Trojan refinement as population-based genetic evolution over threat families, utilizing LLMs as intelligent operators to drive continuous offspring generation. Each individual carries RTL code, threat metadata, and lineage. PHAGE iteratively selects parents, applies an evolution mode, verifies offspring, and updates the active pool and elite archive. Algorithm 1 in Appendix D summarizes this loop. Each verified child is scored by the objective of Equation (2), namely $fitness(x) = \omega f_R(x) + (1-\omega) f_E(x)$.

Figure 3 illustrates the three LLM-driven evolution modes. *Mutate (M)* preserves the core weakness mechanism while mutating the structural realization to evade pattern-matching detectors. *Deepen (D)* intensifies a given threat family by pushing offspring toward rarer triggers, deeper temporal dependence, and subtler control-data coupling. *Recombine (R)* composes two distinct Trojan families into a unified threat through *chaining* (one establishes a state that activates the other), *masking* (one structurally conceals the other without blocking activation), or *parallelism* (both remain activatable through a shared control resource). These fusion operations expand the reachable semantic space far beyond simple mutation of a single parent. Among generations, PHAGE uses three control mechanisms for convergence and diversity (Appendix D): bucket-aware parent sampling, adaptive operator

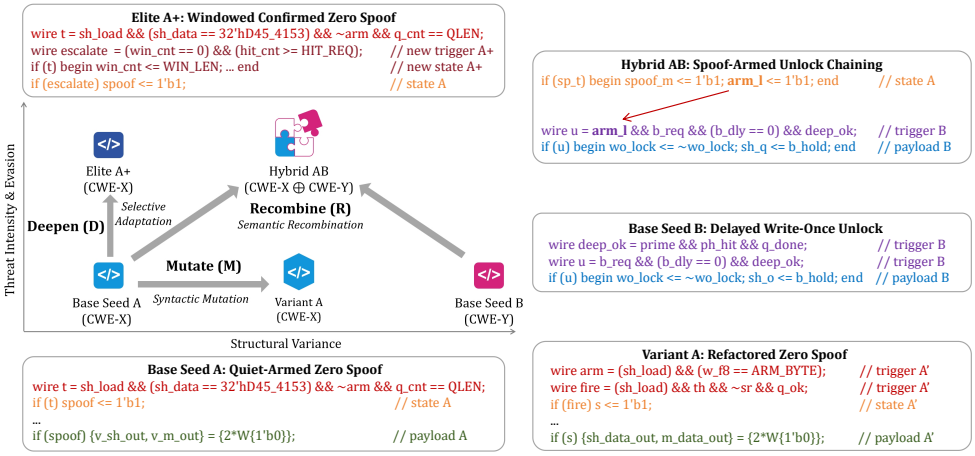

Figure 3: Three evolution operators in PHAGE and their examples.

policy, and separation of exploration (active pool) from exploitation (elite archive). Every offspring is subject to the same verification gate, and every verified offspring receives an optimization signal derived from a vulnerability rating function alongside a detector ensemble. Severity is operationalized through the four-dimensional Verilog Vulnerability Intricacy Score (Appendix F), transforming the exploration from a generic random walk into a directed evolutionary process.

Dynamic Threat Knowledge Accumulation. To overcome the scarcity and homogeneity of hardware security data, the framework dynamically accumulates validated Trojan instances, evaluation feedback, and emergent strategies into a threat repository that provides real-time retrieval for active evolutionary operators and long-horizon enrichment for the foundational VKG.

Knowledge is organized at three levels. *Threat Families* cover both single CWEs and CWE tuples (via hybridization), and are further grouped into *Emergent Attack Strategies* that define pattern-level behaviors and routing statistics, while *Novel Instances* store the grounded RTL implementations, evaluation scores, and detector reasoning behind them. By retaining both abstract patterns and concrete instances, this multi-level representation enables comparison across threat families while preserving the evidence needed for grounded reuse. Retrieval is strategy-aware and returns both strong and weak exemplars, and strategies self-organize in that an offspring whose novelty exceeds a threshold forms a new strategy. Appendix E gives the retrieval weighting and the novelty criterion.

5 Evaluation

5.1 Experiment Setup

Table 2 shows the experimental configuration. We evaluate on two complementary corpora: CORE (Kian Kit Cheah & Li, 2025) for large-scale coverage, and GHOST (Faruque et al., 2025) for external comparison. For evolution we compare against OpenEvolve (Sharma, 2025), an open-source implementation of the AlphaEvolve (Novikov et al., 2025) style code evolution workflow. OpenEvolve evolves each seed independently without repair loop, so offspring that fail verification receive zero evaluation scores while remaining in the evolving population. Every seed is given its own full 200-generation run, which exceeds the budget SpecterRTL allocates per seed. The comparison therefore reflects each system as designed.

The detector ensemble weights GNN4TJ (Yasaei et al., 2021), BugWhisperer (Tarek et al., 2025), and SecRT-LLM (Saha et al., 2024) as 3, 2, and 1, following each detector's scope: continuous graph-level scoring against every design, intermediate semantic checking, and an FSM-specific set-

Table 2: Experimental configuration.

Item	Setting
Main corpus	CORE: 137 clean RTL modules, 24 functionality categories
External comparison	GHOST: 3 shared clean designs, 14 verified Trojan cases
Evolution baseline	OpenEvolve, 200 generations per seed
Evolution budget	20 iterations $\times$ 10 offspring
Agent model	GPT-5.2 (all SpecterRTL agents)
Evaluator	LLM ensemble, median of gradings
Detector ensemble	GNN4TJ (3), BugWhisperer (2), SecRT-LLM (1)
Objective weight	$\omega = 0.7$

ting. Because these detectors are trained on limited hardware security data, we follow Section 4.1 and set $\omega = 0.7$, favoring severity while retaining detector pressure as a secondary objective.

For result reporting, we complement the continuous scores f_R and f_E with hard threshold metrics that make red team success easier to read at a glance. For a verified instance set X, ***High Risk #*** (N_{HR}) counts how many instances cross the severity threshold,

$$N_{\mathrm{HR}} = |\{x \in X \mid f_R(x) \geq 0.7\}| \,. \tag{3}$$

Evasion Success # (N_{ES}) counts how many instances evade *every* detector in the ensemble,

$$N_{\mathrm{ES}} = \left|\{x \in X \mid p^{(d)}_{\mathrm{detect}}(x) < 0.5,\ \forall d \in \mathcal{D}\}\right| . \tag{4}$$

Besides hard counts, we report Pareto-based metrics to capture frontier quality and diversity during evolution. Let $\mathrm{HV}(\cdot)$ denote Pareto hypervolume in the (f_R, f_E) plane. We report the global frontier quality as

$$\mathrm{HV}_{\mathrm{Global}} = \mathrm{HV}(\{(f_R(x), f_E(x)) \mid x \in X\}) \,. \tag{5}$$

We also report a diversity-aware variant over weakness families. Let $\mathcal{T}_*$ be the set of CWE tuples observed in the final verified population of a run, and let $X_\tau = \{x \in X \mid \mathrm{cwe}(x) = \tau\}$. Then

$$\mathrm{HV}_{\mathrm{CWE}} = \frac{1}{|\mathcal{T}_*|} \sum_{\tau \in \mathcal{T}_*} \mathrm{HV}(\{(f_R(x), f_E(x)) \mid x \in X_\tau\}) \,. \tag{6}$$

This metric treats single CWE cases and hybrid CWE tuples as different buckets.

5.2 Main Results

Table 3 and Figure 4 summarize the mining and evolution results. DAVE scales stably across all CORE modules and design categories, yielding 722 verified Trojan seeds in total with an average of 5.27 per module. On 21 representative cases, the 112 verified seeds expand to 4151 verified instances after evolution (98.8% verified yield). Evolution increases the high-risk rate from 12.5% to 92.3% (7.39$\times$) while expanding the absolute number of successful evasion instances by 29.5$\times$. Per-category mining breakdown and per-case evolution traces are in Appendix B.1 and B.3. On the three clean designs of GHOST, SpecterRTL produces 593 verified instances versus 14 (42.4$\times$)

Table 3: Main results.

	(a) Trojan evolution on 21 cases		**(b) Comparison with GHOST**	
Metric	Mining (pre-evolution)	Evolution (post-evolution)	GHOST (released)	SpecterRTL (evolution)
Verified Trojan instances	112	4151	14	**593** (**42.4**$\times$)
High Risk instances ($f_R \geq 0.7$)	14	3832	2	**552** (**93.1%**)
Evasion Success instances	61	1802	4	**140** (**35**$\times$)
High-risk rate	12.5%	**92.3%**	14.3%	**93.1%** (6.52$\times$)
Evasion success rate	54.5%	43.4%	28.6%	23.6%

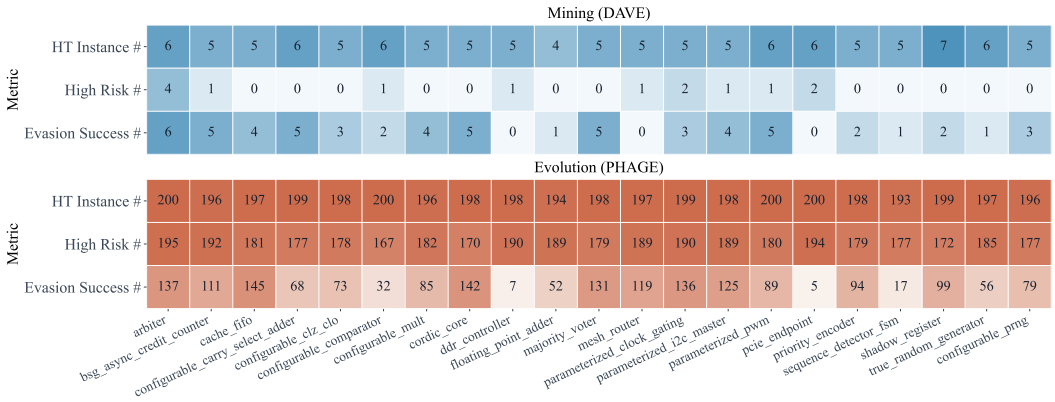

Figure 4: Trojan evolution (PHAGE) on 21 representative cases. Mining: 112 verified seeds, 14 high-risk ones (12.5%). Evolution: 4151 verified instances, 3832 high-risk ones (92.3%).

and achieves 140 detector evasion successes versus 4 (35×). Scale is the discriminating property, since single-shot prompting yields one instance per request while this population-based search keeps returning verified and evasive instances as it grows. GHOST's evasion rate of 28.6% rests on 14 cases and carries no statistical weight, whereas the 23.6% here is estimated over 593.

5.3 Ablation and Comparison Studies

Effectiveness of Adversarial Trojan Evolution. Figure 5 shows that PHAGE improves threat quality consistently over long-horizon evolution. Both metrics rise rapidly in early iterations and then stabilize at a higher plateau, and the narrow standard deviation bands show that the gain is stable across cases rather than driven by isolated lucky mutations. Against OpenEvolve, which evolves each seed in isolation, SpecterRTL reaches a frontier consistently farther toward higher severity and stronger evasion. The figure separates descendants whose CWE tuples contain the seed's CWE from those that do not, showing that PHAGE both strengthens the original threat lineage and diffuses it across a wider threat space through population-level co-evolution and hybridization. Detailed per-case curves including $\mathrm{HV}_{\mathrm{CWE}}$ are in Appendix B.3.

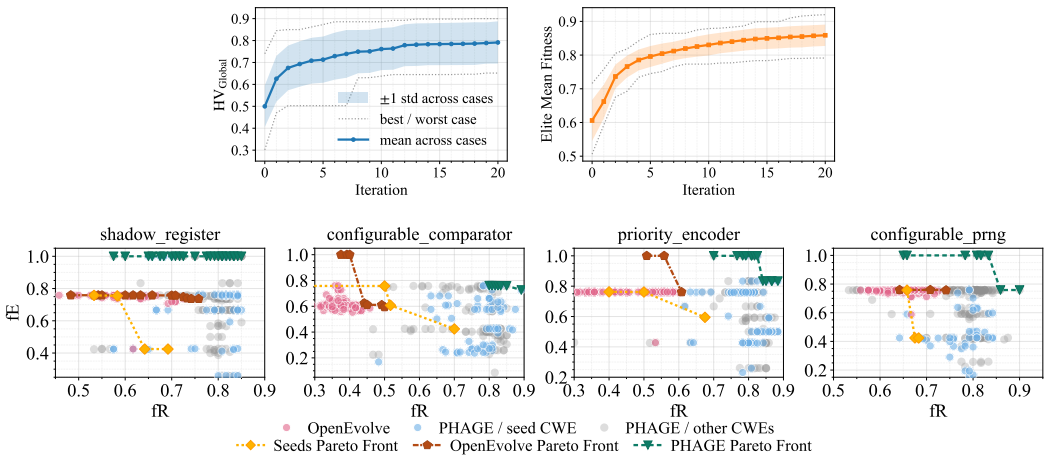

Figure 5: Effectiveness of adversarial Trojan evolution. Top: evolution dynamics aggregated over 20 module cases, showing $\mathrm{HV}_{\mathrm{Global}}$ (left) and elite mean fitness (right). Bottom: comparison with the OpenEvolve baseline on 4 module cases. The pilot case of the 21 is excluded so every panel covers the same protocol.

Ablation Studies. Table 4 and Table 5 isolate the knowledge-side mechanisms of the framework. The first removes either the VKG or the staged vulnerability research workflow from the mining stage. The second disables instant knowledge retrieval or explicit feedback injection during evolution, where *KB Off* cuts retrieval of instant knowledge from the threat corpus and *Feedback Off* removes only the *explicit* evaluation and detector feedback supplied as context to the LLM operators, including numerical scores and reasoning text. It does *not* disable evaluation itself, since evaluation outcomes still guide pool retention, parent sampling, and operator selection through the normal search loop.

Table 4: Ablation on knowledge-guided vulnerability mining, aggregated over all 137 CORE modules. Jailbreak rate is the fraction of cases where the agent engages with the red-team objective. Best per column in bold.

Mode	Identified CWE # ↑	Valid Instance # ↑	No-Fix Verified # ↑	Fix Verified # ↑	Jailbreak Rate ↑
Full	**743**	**743**	**281**	**722**	**100.0%**
w/o VKG	181	143	50	139	89.8%
Monolithic	249	223	95	220	90.5%

Table 5: Ablation on instant knowledge accumulation and explicit feedback, averaged over four representative cases. **A**: final $\mathrm{HV}_{\mathrm{Global}}$ (endpoint height). **B**: area under the $\mathrm{HV}_{\mathrm{Global}}$ trace divided by the iteration count (growth speed and endpoint height jointly). **D**: earliest iteration reaching $0.95\times$ its own final $\mathrm{HV}_{\mathrm{Global}}$ (convergence speed). Best per column in bold.

Mode	A: Final ↑	B: AUC ↑	D: First-hit ↓	$\mathrm{HV}_{\mathrm{CWE}}$ ↑	Elite ↑	Fix # ↓
Full	**0.818**	**0.741**	**9.2**	**0.568**	**0.858**	**371**
KB Off	0.695	0.631	9.5	0.524	0.826	362
Feedback Off	0.787	0.689	11.0	0.545	0.846	435
Both Off	0.662	0.603	13.2	0.516	0.794	1331

The *Full* pipeline dominates both tables. Removing the VKG leaves only 24% of identified weaknesses and 19% of post-repair verified instances, while the monolithic alternative recovers 34% and 30% respectively, and the case-level jailbreak rate falls from 100.0% to 89.8% and 90.5%. Task decomposition and tool-mediated context gathering therefore make agents more willing to engage with the red-team objective, and both the VKG and staged vulnerability research are necessary for stable large-scale seed construction. Under evolution, *Full* ranks best on every metric in Table 5, reaching the highest final frontier quality, growing fastest on average, converging soonest, and spreading most widely across weakness families. Its repair burden is the most telling signal of the same effect, since *Both Off* needs 1331 coder fixes, more than three times the *Full* count, while the single-mechanism ablations stay close to *Full* on repair cost yet fall short on frontier quality. Instant knowledge accumulation and explicit feedback thus improve both frontier quality and the practical efficiency of keeping offspring repairable and evolvable.

6 Conclusion

We presented SpecterRTL, an end-to-end RTL red-teaming framework for autonomous Hardware Trojan construction and evolution. It unifies knowledge-guided vulnerability mining and adversarial evolution within a self-sustaining closed-loop paradigm, where newly evolved attacks continuously enrich the foundational threat knowledge. On the clean designs shared with the current SOTA RTL red-teaming benchmark, SpecterRTL produces **42.4**$\times$ more verified Hardware Trojan instances and a **35**$\times$ higher successful evasion count across three representative detectors. Beyond evolving previously unseen adversarial variants, SpecterRTL progressively expands hardware threat intelligence for future resilience assessment, benchmarking, and defense research.

AI USE STATEMENT

Generative AI tools were used in this work in two ways. First, the framework studied in this paper orchestrates large language models as its core mechanism, so the methodological description necessarily concerns model behavior, and the experiments were run with GPT-5.2 as reported in Table 2 and Appendix I. Second, generative AI tools were used as writing assistance for language editing of the manuscript. They were not used to generate research ideas, to design the method, to analyze results, or to produce the claims made in this paper.

All AI-assisted output was reviewed by the authors. The framework's generated RTL is validated by the verification pipeline described in Appendix I rather than accepted on the model's assertion, so every reported instance has passed an executable check. We take responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work studies hardware Trojans from the defender's perspective, with the goal of stress-testing RTL security evaluation before deployment. All experiments are conducted exclusively on the public datasets described in Appendix I, and no real product, vendor design, or deployed system is targeted or tested. Generated cases are research artifacts intended for analysis, and the framework operates on designs already available to any researcher. We consider the work to raise no ethical concerns beyond those inherent to security research, and Appendix J records the safeguards adopted.

REPRODUCIBILITY STATEMENT

The experimental configuration is given in Table 2, with the full protocol in Appendix I, which covers the datasets and their preprocessing, the three verification checks, the detector ensemble and its score fusion, all hyperparameters, and the toolchain. The method is specified in Section 4, with the complete evolution loop in Algorithm 1 and the operator definitions in Appendix G. The knowledge graph construction and retrieval are described in Appendix H, and per-category and per-case results are given in Appendix B and Appendix C. Upon publication, we intend to release the framework and the benchmark cases through a public repository, and to provide the complete configuration and data processing steps alongside them so that the reported results can be reproduced end to end.

REFERENCES

Hojjat Aghakhani, Wei Dai, Andre Manoel, Xavier Fernandes, Anant Kharkar, Christopher Kruegel, Giovanni Vigna, David Evans, Ben Zorn, and Robert Sim. Trojanpuzzle: Covertly poisoning code-suggestion models. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 1122–1140, 2024. doi: 10.1109/SP54263.2024.00140.

Roberto Baldoni, Emilio Coppa, Daniele Cono D'elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3), May 2018. ISSN 0360-0300. doi: 10.1145/3182657. URL https://doi.org/10.1145/3182657.

Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014. doi: 10.1109/JPROC.2014.2334493.

Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-chat: Challenges and opportunities in conversational hardware design. In *Proceedings of the 5th ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6. IEEE, 2023.

Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware trojan: Threats and emerging solutions. In *2009 IEEE International High Level Design Validation and Test Workshop*, pp. 166–171, 2009. doi: 10.1109/HLDVT.2009.5340158.

Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. Chipgpt: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019*, 2023.

Kaiyan Chang, Kun Wang, Nan Yang, Ying Wang, Dantong Jin, Wenlong Zhu, Zhirong Chen, Cangyuan Li, Hao Yan, Yunhao Zhou, et al. Data is all you need: Finetuning llms for chip design via an automated design-data augmentation framework. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Wen Cheng, Ke Sun, Xinyu Zhang, and Wei Wang. Security attacks on llm-based code completion tools. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 23669–23677, 2025.

Jonathan Cruz, Pravin Gaikwad, Abhishek Nair, Prabuddha Chakraborty, and Swarup Bhunia. A machine learning based automatic hardware trojan attack space exploration and benchmarking framework. In *2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6, 2022. doi: 10.1109/AsianHOST56390.2022.10022234.

Ruochen Dai, Zhaoxiang Liu, Orlando Arias, Xiaolong Guo, and Tuba Yavuz. Dtjrtl: A configurable framework for automated hardware trojan insertion at rtl. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, GLSVLSI '24, pp. 465–470, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706059. doi: 10.1145/3649476.3658759. URL https://doi.org/10.1145/3649476.3658759.

Komail Dharsee and John Criswell. Jinn: Hijacking safe programs with trojans. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA, 2023. USENIX Association. ISBN 978-1-939133-37-3.

Md Omar Faruque, Peter Jamieson, Ahmad Patooghy, and Abdel-Hameed A Badawy. Unleashing ghost: An llm-powered framework for automated hardware trojan design. *Electronics*, 14(23): 4745, 2025.

Md Omar Faruque, Peter Jamieson, Ahmad Patooghy, and Abdel-Hameed A. Badawy. Trojanwhisper: Evaluating pre-trained llms to detect and localize hardware trojans. *IEEE Access*, 14: 30965–30983, 2026. doi: 10.1109/ACCESS.2026.3663556.

Mohamed Amine Ferrag, Abderrahmane Lakas, Norbert Tihanyi, and Merouane Debbah. Securing llm agents: From prompt sanitization to autonomous red teaming and beyond. *Internet of Things and Cyber-Physical Systems*, 5:185–209, 2025. ISSN 2667-3452. doi: https://doi.org/10.1016/j.iotcps.2026.03.001. URL https://www.sciencedirect.com/science/article/pii/S2667345226000015.

Weimin Fu, Honggang Yu, Orlando Arias, Kaichen Yang, Yier Jin, Tuba Yavuz, and Xiaolong Guo. Graph neural network based hardware trojan detection at intermediate representative for soc platforms. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI '22, pp. 481–486, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393225. doi: 10.1145/3526241.3530827. URL https://doi.org/10.1145/3526241.3530827.

Weimin Fu, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu. Llm4sechw: Leveraging domain-specific large language model for hardware debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6, 2023. doi: 10.1109/AsianHOST59942.2023.10409307.

Deep Ganguli, Liane Lovitt, Jackson Kernion, Amanda Askell, Yuntao Bai, Saurav Kadavath, Ben Mann, Ethan Perez, Nicholas Schiefer, Kamal Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

Vasudev Gohil, Hao Guo, Satwik Patnaik, and Jeyavijayan Rajendran. Attrition: Attacking static hardware trojan detection techniques using reinforcement learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pp. 1275–1289, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. doi: 10.1145/3548606.3560690. URL https://doi.org/10.1145/3548606.3560690.

Xiaolong Guo, Raj Gautam Dutta, Jiaji He, Mark M. Tehranipoor, and Yier Jin. Qif-verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 91–100, 2019. doi: 10.1109/HST.2019.8740840.

Victor Takashi Hayashi and Wilson Vicente Ruggiero. Hardware trojan detection in open-source hardware designs using machine learning. *IEEE Access*, 13:37771–37788, 2025. doi: 10.1109/ACCESS.2025.3546156.

Slobodan Jenko, Niels Mündler, Jingxuan He, Mark Vero, and Martin Vechev. Black-box adversarial attacks on llm-based code completion. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Christoforos Kachris. A survey on hardware accelerators for large language models. *Applied Sciences*, 15(2), 2025. ISSN 2076-3417. doi: 10.3390/app15020586. URL https://www.mdpi.com/2076-3417/15/2/586.

Tasmin Karim and Mst Shapna Akter. Vulnerability datasets for software security: A survey of existing resources, challenges, and future directions. *Computers Security*, 167:104926, 2026. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2026.104926. URL https://www.sciencedirect.com/science/article/pii/S0167404826001021.

Yunxiang Zhang Zhuofan Lin Yuxin Ji Xinfei Guo Harikrishnan Ramiah Kian Kit Cheah, Fu Qi Chua and Yongfu Li. Descriptor: A corpus of synthesizable verilog rtl modules dataset for eda research (core), 2025. URL https://github.com/SJTU-YONGFU-RESEARCH-GRP/core.

Georgios Kokolakis, Athanasios Moschos, and Angelos D. Keromytis. Harnessing the power of general-purpose llms in hardware trojan design. In *Applied Cryptography and Network Security Workshops*, pp. 176–194, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-61485-9. doi: 10.1007/978-3-031-61486-6_11. URL https://doi.org/10.1007/978-3-031-61486-6_11.

Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: A survey. *Cybersecurity*, 1(1):6, 2018.

Yige Li, Hanxun Huang, Yunhan Zhao, Xingjun Ma, and Jun Sun. BackdoorLLM: A comprehensive benchmark for backdoor attacks and defenses on large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Tianyang Liu, Qi Tian, Jianmin Ye, LikTung Fu, Shengchu Su, Junyan Li, Gwok-Waa Wan, Layton Zhang, Sam-Zaak Wong, Xi Wang, et al. Chatchisel: Enabling agile hardware design with large language models. In *Proceedings of the 2nd International Symposium of Electronics Design Automation (ISEDA)*, pp. 710–716, Xi'an, Shanxi, China, 2024. IEEE.

Narek Maloyan, Ekansh Verma, Bulat Nutfullin, and Bislan Ashinov. Trojan detection in large language models: Insights from the trojan detection challenge. *arXiv preprint arXiv:2404.13660*, 2024.

MITRE. 2025 cwe most important hardware weaknesses. Technical report, MITRE Corporation, 2025. URL https://cwe.mitre.org/topHW/archive/2025/2025_CWE_MIHW.pdf.

Juxin Niu, Xiangfeng Liu, Dan Niu, Xi Wang, Zhe Jiang, and Nan Guan. Rechisel: Effective automatic chisel code generation by llm with reflection. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference*, pp. 1–6, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Sheng Ouyang, Yihao Qin, Bo Lin, Liqian Chen, Xiaoguang Mao, and Shangwen Wang. Smoke and mirrors: Jailbreaking llm-based code generation via implicit malicious prompts. *arXiv preprint arXiv:2503.17953*, 2025.

Jingyu Pan, Guanglei Zhou, Chen-Chia Chang, Isaac Jacobson, Jiang Hu, and Yiran Chen. A survey of research in large language models for electronic design automation. *ACM Trans. Des. Autom. Electron. Syst.*, 30(3), February 2025. ISSN 1084-4309. doi: 10.1145/3715324. URL https://doi.org/10.1145/3715324.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3419–3448, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.225. URL https://aclanthology.org/2022.emnlp-main.225/.

Yubin Qu, Song Huang, and Peng Nie. A review of backdoor attacks and defenses in code large language models: Implications for security measures. *Information and Software Technology*, 182:107707, 2025. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2025.107707. URL https://www.sciencedirect.com/science/article/pii/S0950584925000461.

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)*, pp. 233–243. IEEE, 2024.

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)*, pp. 471–474, 2013. doi: 10.1109/ICCD.2013.6657085.

Amin Sarihi, Ahmad Patooghy, Peter Jamieson, and Abdel-Hameed A. Badawy. Trojan playground: A reinforcement learning framework for hardware trojan insertion and detection. *J. Supercomput.*, 80(10):14295–14329, March 2024. ISSN 0920-8542. doi: 10.1007/s11227-024-05963-8. URL https://doi.org/10.1007/s11227-024-05963-8.

Asankhaya Sharma. Openevolve: An open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Saideep Sreekumar, Zeng Wang, Akashdeep Saha, Weihua Xiao, Minghao Shao, Muhammad Shafique, Ozgur Sinanoglu, Ramesh Karri, and Johann Knechtel. Trojangym: A detector-in-the-loop llm for adaptive rtl hardware trojan insertion. *arXiv preprint arXiv:2601.17178*, 2026.

Shams Tarek, Dipayan Saha, Sujan Kumar Saha, and Farimah Farahmandi. Bugwhisperer: Fine-tuning llms for soc hardware vulnerability detection. In *2025 IEEE 43rd VLSI Test Symposium (VTS)*, pp. 1–5, 2025. doi: 10.1109/VTS65138.2025.11022958.

Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6. IEEE, 2023.

Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. Chatcpu: An agile cpu design and verification platform with llm. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Chenwei Xiong, Cheng Liu, Huawei Li, and Xiaowei Li. Hlspilot: Llm-based high-level synthesis. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024.

Shenao Yan, Shen Wang, Yue Duan, Hanbin Hong, Kiho Lee, Doowon Kim, and Yuan Hong. An LLM-Assisted Easy-to-Trigger backdoor attack on code completion models: Injecting disguised vulnerabilities against strong detection. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 1795–1812, Philadelphia, PA, August 2024. USENIX Association. ISBN 978-1-939133-44-1. URL https://www.usenix.org/conference/usenixsecurity24/presentation/yan.

Rozhin Yasaei, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1504–1509, 2021. doi: 10.23919/DATE51398.2021.9474174.

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*, 44(1):25–38, 2025. doi: 10.1109/TCAD.2022.3178355.

Jie Zhang, Feng Yuan, and Qiang Xu. Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pp. 153–166, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329576. doi: 10.1145/2660267.2660289. URL https://doi.org/10.1145/2660267.2660289.

A Appendix Index

This appendix provides material referenced from the main text. Every claim in the main text is supported within the main text itself, and the sections below supply the underlying detail.

- **Appendix B** — Mining and evolution details: per-category mining breakdown, per-case evolution summary, detailed evolution curves, and per-case ablation detail.
- **Appendix C** — Per-design detail for the comparison with GHOST.
- **Appendix D** — The full PHAGE loop as an algorithm listing, together with its control mechanisms.
- **Appendix E** — Threat accumulation details: retrieval weighting and the novelty criterion for strategy formation.
- **Appendix F** — The vulnerability rating specification: the four scored dimensions, their anchors, and the scoring and normalization procedure.
- **Appendix G** — Operator case study: one concrete RTL instance per evolution mode (Mutate, Deepen, Recombine) traced along a single real lineage.
- **Appendix H** — Construction of the Hardware Vulnerability Knowledge Graph (VKG) and the taxonomy underlying threat families.
- **Appendix I** — Full experimental configuration: datasets, detector ensemble, hyperparameters, and the agentic workflow.
- **Appendix J** — Extended discussion of responsible disclosure and dual-use considerations.

B Mining and Evolution Details

B.1 Per-Category Mining Breakdown

Table 6 reports the complete mining result over all 137 CORE modules, broken down by the 24 functionality categories. Figure 6 visualizes the distribution.

DAVE scales stably across all 137 CORE modules and 24 design categories, yielding 722 verified Trojan seeds in total with an average of 5.27 per module, slightly exceeding the nominal mining target of 5. The category level distribution is uneven due to varying numbers of designs per category, yet per-module output consistently concentrates around five to six instances, confirming that the mining process generalizes across design types rather than exploiting a narrow subset.

Table 6: Vulnerability mining (DAVE) across all 137 CORE modules and 24 functionality categories. HT: verified Trojan instances. HR: high-risk instances ($f_R \geq 0.7$). ES: instances evading every detector in the ensemble.

Category	Case #	HT #	HR #	ES #
adders	6	31	1	6
alu	2	12	4	4
arbiters	4	21	11	19
cdc	5	26	18	21
codings	7	38	10	28
comms	12	65	33	47
cordic	3	15	5	13
counters	15	82	30	46
debug	2	10	3	2
encryption	1	6	2	3
fifos	20	107	56	67
filters	2	10	6	6
fsm	1	5	0	5
interfaces	4	21	9	12
io	2	11	6	3
math	3	14	4	9
mems	4	21	8	11
multipliers	3	15	2	10
noc	5	19	8	10
power	5	28	19	20
registers	19	105	38	77
signals	9	44	17	32
test	2	11	10	6
voters	1	5	0	5
Total	**137**	**722**	**300**	**462**

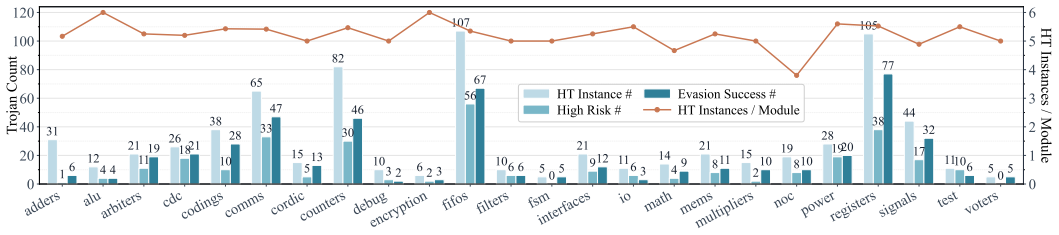

Figure 6: Per-category mining breakdown visualized. DAVE scales stably across all 24 design categories, yielding 722 verified Trojan seeds with an average of 5.27 per module.

B.2 Per-Case Evolution Summary

Figure 4 summarizes how the mining seeds expand under PHAGE on 21 representative cases. These cases provide broad category coverage while avoiding extreme seed quality outliers. At the end of mining, the 21 cases contain 112 verified seeds, of which 14 are high risk and 61 evade all detectors, yielding a high-risk rate of 12.5% and an evasion success rate of 54.5%. After evolution, the population grows to 4151 verified instances out of 4200 offspring-generation attempts, for an overall verified yield of 98.8%. Among these instances, 3832 are high risk and 1802 achieve detector-wide evasion, corresponding to a high-risk rate of 92.3% and an evasion success rate of 43.4%. Thus, evolution increases the high-risk rate by 7.39× while expanding the absolute number of successful evasion instances by 29.5×. The evasion success rate is not directly comparable across the two stages, since the pre-evolution count rests on 112 seeds and the post-evolution count on 4151 instances. What the absolute counts show is that spending more generation budget on this search returns more evasive instances at scale, which is precisely the property that template-driven and single-parent generation lack.

B.3 Per-Case Evolution Curves

Figure 7 reports the complete per-case evolution traces on all 20 representative cases, extending the aggregated view discussed in the main text. One of the 21 cases was run first as a pilot to exercise the full loop, and it is omitted here so that the grid shows a uniform set of cases that all completed the identical protocol. Each panel tracks $\mathrm{HV}_{\mathrm{Global}}$, $\mathrm{HV}_{\mathrm{CWE}}$, and elite mean fitness over the full iteration horizon.

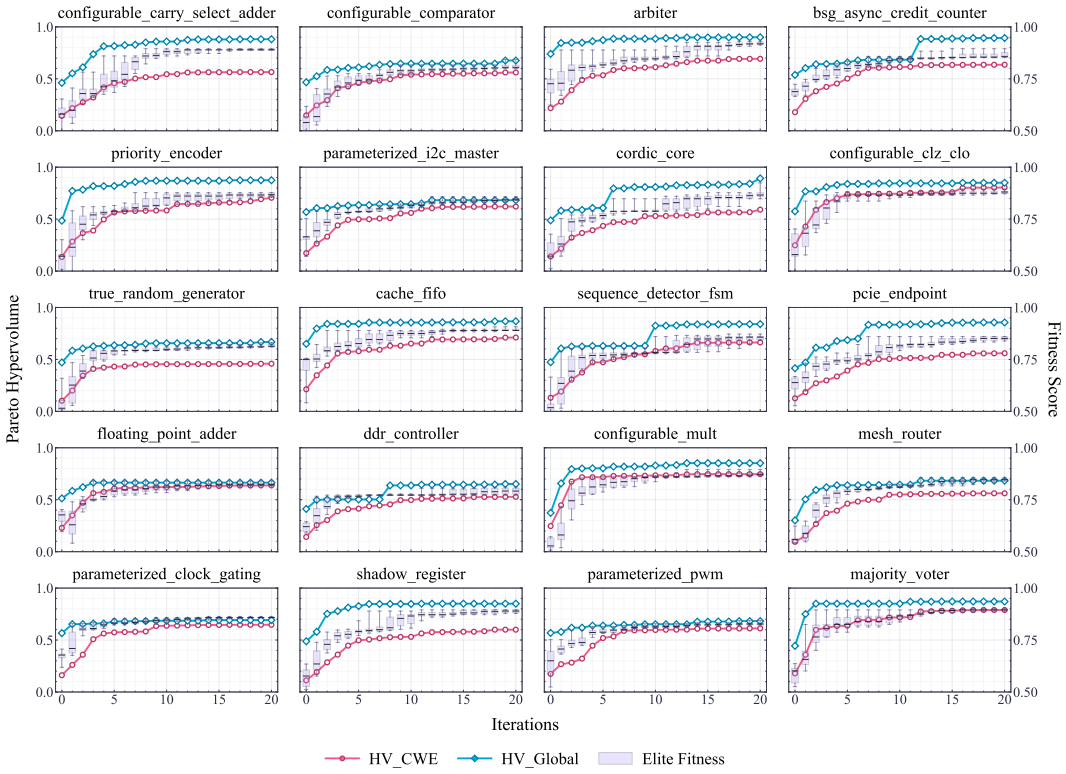

Figure 7: Per-case evolution over 20 representative module cases.

Many cases exhibit late-stage jumps rather than a flat terminal plateau, showing that dynamic RAG exemplars and diversity-preserving search control help the population escape local optima even after initial rapid gains. $\mathrm{HV}_{\mathrm{CWE}}$ increases across all cases, showing that the search not only strengthens existing lineages but also expands across distinct CWE tuples, including hybrid tuples produced by Recombine. Elite fitness distributions move upward in parallel, confirming sustained gains in the strongest verified individuals.

B.4 Per-Case Ablation Detail

Table 4 in the main text reports the mining-stage ablation in aggregate. Figure 8 gives the corresponding per-category totals and frontier scatter, so that the effect of removing the VKG or the staged research workflow can be inspected individually rather than only in aggregate.

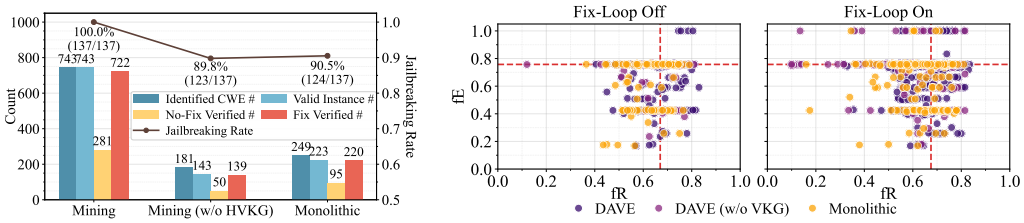

Figure 8: Red teaming without VKG or the whole vulnerability mining pipeline.

Beyond quantity, DAVE improves quality. Without the fix loop, the complete mining pipeline attains a stronger elite frontier, whereas both ablations fall short at the top end of the (f_R, f_E) plane. The fix loop narrows this gap because repair partially compensates for weaker seeds, but the complete pipeline still produces far more usable seeds under comparable quality levels. Knowledge-guided mining improves both the initial elite ceiling and the supply of evolvable seeds, directly shaping the downstream evolution stage.

Table 5 in the main text reports the evolution-stage ablation averaged over four representative cases. Figure 9 gives the corresponding per-case traces, so that the behavior of each ablated variant can be inspected individually rather than only in aggregate.

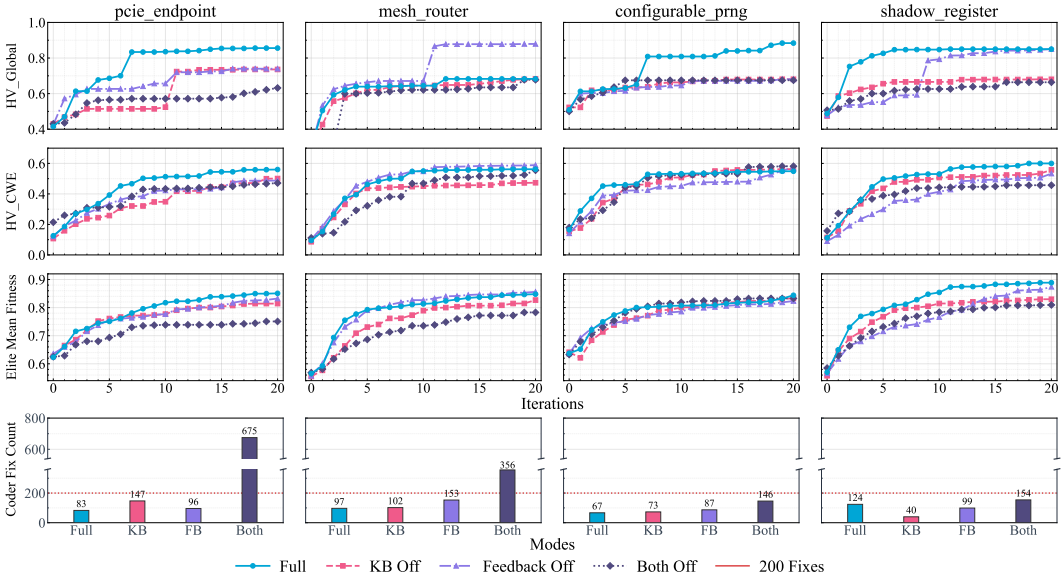

Figure 9: Per-case ablation traces without instant knowledge reference and evaluation feedback.

Reading the panels individually shows the same ordering that the averaged table reports, and it also exposes where an individual run departs from it. In the *Feedback Off* panel for one case, $\mathrm{HV}_{\mathrm{Global}}$ makes a late jump above the other settings. This remains a reasonable observation, since PHAGE preserves room for exploration rather than focusing only on exploitation. The local gain does not translate into a broader advantage. In that same run $\mathrm{HV}_{\mathrm{CWE}}$ and elite mean fitness stay comparable to the *Full* setting, so the jump comes from a favorable but narrow frontier excursion rather than a systematic improvement in overall solution quality. The repair burden separates the settings more clearly than the frontier alone. Runs that occasionally discover favorable Pareto points do not maintain the low repair pressure of *Full*, while runs that occasionally reduce repair cost do not match its frontier quality.

C PER-DESIGN DETAIL FOR THE GHOST COMPARISON

Table 3(b) in the main text reports the GHOST comparison in aggregate. Table 7 gives the same comparison per design, so that the contribution of each of the three shared clean designs can be inspected individually. Figure 10 shows that the gain over GHOST is not only quantitative, as the evolved Pareto frontier consistently moves toward the upper right region of the (f_R, f_E) plane, improving threat intensity and detector resistance simultaneously.

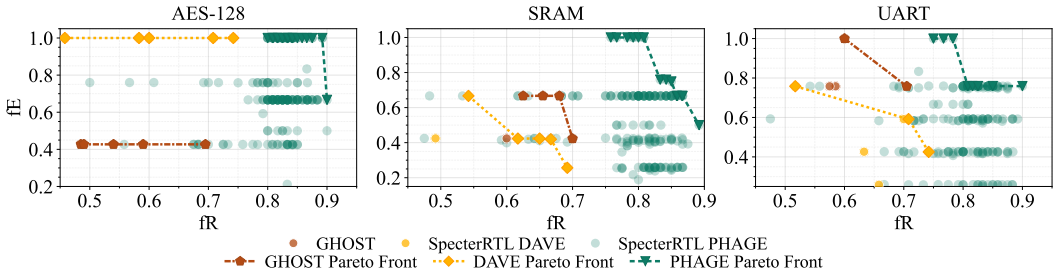

Figure 10: Comparison with GHOST baseline.

Table 7: Per-design comparison with GHOST on the three shared clean designs. HT: verified Trojan instances. HR: high-risk instances ($f_R \geq 0.7$). ES: instances evading every detector in the ensemble.

Design	Stage	HT #	HR #	ES #
AES-128	GHOST	5	0	0
AES-128	Mining	5	2	5
AES-128	Evolution	197	187	72
SRAM	GHOST	5	1	0
SRAM	Mining	6	0	0
SRAM	Evolution	199	178	10
UART	GHOST	4	1	4
UART	Mining	6	3	1
UART	Evolution	197	187	58
Total	**GHOST**	**14**	**2**	**4**
Total	**Mining**	**17**	**5**	**6**
Total	**Evolution**	**593**	**552**	**140**

D PHAGE ALGORITHM AND CONTROL MECHANISMS

Algorithm 1 gives the full PHAGE loop referenced from Section 4.2.

Control Mechanisms. PHAGE uses three control mechanisms for convergence and diversity. First, *parent sampling* is bucket-aware, partitioning individuals by stealth severity, evasion profile, and weakness identity. Selection samples a region of the evolutionary landscape and performs local competition inside that region, preserving exploration across heterogeneous threat families while biasing reproduction toward strong candidates. Second, *operator policy* is adaptive. PHAGE updates operator preference from downstream outcomes such as survival into the active pool or admission into the elite archive, allocating more budget to operators that are empirically productive for a given family without collapsing onto a single mode. Third, *pool and archive update* separates exploration from exploitation. The archive stores the globally strongest validated Trojans, while the active pool retains high-quality representatives across different severity and evasion regions. In practice, the diversity-preserving assignment can be viewed as a partition map

$$b(x) = \big(\lfloor B_R f_R(x) \rfloor,\ \lfloor B_E f_E(x) \rfloor,\ \mathrm{cwe}(x)\big), \tag{7}$$

which allows PHAGE to maintain competitive representatives in multiple threat regions at once.

Algorithm 1 PHAGE

Require: validated Trojan seeds $\mathcal{S}$ from DAVE, verification backbone $\mathcal{V}$, evaluation backbone $\mathcal{E}$, threat corpus $\mathcal{K}$, batch size K
Ensure: elite archive $\mathcal{A}$ and updated threat corpus $\mathcal{K}$

$\mathcal{P} \leftarrow \textsc{InitializePopulation}(\mathcal{S}, \mathcal{E})$
$\mathcal{A} \leftarrow \textsc{InitializeArchive}(\mathcal{P})$
while termination criterion is not met **do**
  $\mathcal{C} \leftarrow \emptyset$
  for $i = 1$ to K **do**
    $p \leftarrow \textsc{BucketAwareParentSampling}(\mathcal{P})$
    $o \leftarrow \textsc{AdaptiveOperatorPolicy}(\mathcal{P}, p)$
    if $o = \textsc{Mutate}$ **then**
      $c \leftarrow \textsc{Mutate}(p)$
    else if $o = \textsc{Deepen}$ **then**
      $r \leftarrow \textsc{RetrieveContext}(\mathcal{K}, \{p\})$
      $c \leftarrow \textsc{Deepen}(p, r)$
    else
      $q \leftarrow \textsc{SelectHybridParent}(\mathcal{P}, p)$
      $r \leftarrow \textsc{RetrieveContext}(\mathcal{K}, \{p, q\})$
      $c \leftarrow \textsc{Recombine}(p, q, r)$
    end if
    if $\mathcal{V}(c) = 1$ **then**
      $(f_R, f_E, m) \leftarrow \textsc{Evaluate}(c, \mathcal{E})$
      $c.\text{fitness} \leftarrow \omega f_R + (1 - \omega) f_E$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
      $\mathcal{K} \leftarrow \textsc{UpdateCorpus}(\mathcal{K}, c, m)$
    end if
  end for
  $\mathcal{A} \leftarrow \textsc{UpdateHallOfFame}(\mathcal{A}, \mathcal{C})$
  $\mathcal{P} \leftarrow \textsc{DiversityPreservingPoolUpdate}(\mathcal{P} \cup \mathcal{C})$
  $\textsc{UpdateOperatorPreference}(\mathcal{P}, \mathcal{A}, \mathcal{C})$
end while
return $\mathcal{A}, \mathcal{K}$

E Threat Accumulation Details

The accumulation mechanism provides both real-time retrieval for active evolutionary operators and long-horizon enrichment of the foundational VKG.

Retrieval must return context that is relevant, diverse, and informative. This is achieved through strategy-aware routing, positive and negative exemplar selection, and anti-repetition retrieval. Routing restricts retrieval to strategies aligned with the active weakness context, where strong and weak examples are both returned so evolutionary operators learn what survives selection and what structural choices lead to failure. Recent overused exemplars are down-weighted to preserve diversity without over-pruning in low data regions. This weighting is formalized as

$$w_i^+ \propto e^{f_i/\tau} \rho_i, \qquad w_i^- \propto e^{(1-f_i)/\tau} \rho_i, \tag{8}$$

where f_i is the exemplar fitness, τ controls exploration, and ρ_i is a freshness term that decreases with recent reuse count.

The mechanism also autonomously decides when a validated offspring belongs to an existing strategy and when it should form a new one. Each strategy maintains an online novelty distribution over its assigned instances, updating the mean and variance incrementally throughout evolution. Once a strategy accumulates sufficient instances, offspring are evaluated relative to that historical profile

using a novelty z-score

$$z(x, s) = \frac{\text{novelty}(x, s) - \mu_s}{\sigma_s}, \tag{9}$$

where $\text{novelty}(x, s)$ measures how far offspring x departs from strategy s, and μ_s and σ_s are the online mean and standard deviation. Concretely, $\text{novelty}(x, s)$ is the median embedding distance from x to its k nearest neighbours inside strategy s, so a high value means the offspring sits far from the instances that strategy already covers. The strategy's mean and second moment are updated incrementally as instances are assigned, and the decision is taken against the nearest strategy only, which keeps a large strategy from absorbing an offspring that belongs elsewhere. If $z(x, s)$ exceeds a threshold, the offspring forms a new strategy. This autonomous categorization prevents knowledge fragmentation while continuously feeding newly discovered threat tuples back into the epistemic core of DAVE, establishing a self-sustaining loop.

F Vulnerability Rating Specification

The severity term f_R of Equation 2 is the four-dimensional Verilog Vulnerability Intricacy Score (V-VIS). Each dimension is scored on a 1–10 scale against the anchors below, and the score of an instance is the mean of the four. The specification is written as an expert standard, in the sense that the anchors describe properties of the RTL that a hardware security reviewer can identify and rank, and the score is produced by applying that standard to the candidate.

Syntactic Plausibility. How closely the modification resembles valid, professionally written logic. Low scores cover code that reads as a mistake or as blatant sabotage, including empty procedural blocks, constants driven onto critical outputs with no supporting logic, and unintended latches. Middle scores cover syntactically correct code that violates common practice, such as mixed blocking and non-blocking assignment in one block or redundant logic. High scores cover code indistinguishable from the surrounding design, such as an off-by-one boundary in a counter, a subtle change to one transition of a complex state machine, or a conditional write to a single status bit under a plausible condition.

Trigger Rarity. The size of the input space or the length of the event sequence required to activate the modification. Low scores cover conditions active immediately after reset or on nearly any valid input, such as inverted reset logic or a permanently inverted output. Middle scores cover conditions needing a specific combination of control and data values, such as a write enabled only for a particular address range or a FIFO that must be exactly full. High scores cover events of very low probability, such as a comparison against a specific 32-bit constant, a handshake sequence repeated a large number of times, or a race requiring precise cycle alignment.

Sequential Depth. The temporal distance between trigger and effect, and the complexity of the state involved. Low scores cover purely combinational modifications whose effect appears in the same cycle as the input change. Middle scores cover simple register updates and short pipelines, where the effect propagates within a few cycles. High scores cover deep state machine behavior, including transition to an undefined state from which the design does not recover, deadlock in a request-grant protocol, and corruption that persists after the trigger has passed.

Control-Data Coupling. Whether the modification corrupts only data values or also hijacks control. Low scores cover data-path effects such as arithmetic errors, bit flips on a data bus, and truncation. Middle scores cover miscomputed control signals that indirectly corrupt data, such as an incorrectly generated full flag that permits overwrite or a write enable that toggles at the wrong time. High scores cover direct manipulation of control structure, including redirection of a state transition, bypass of a privilege comparison, and alteration of the reset sequence.

Higher scores correspond to modifications that are harder to expose by review, simulation, or formal analysis, so the four dimensions jointly index the difficulty of uncovering an implanted modification rather than the breadth of the damage it can do.

F.1 Scoring Procedure and Normalization

Each dimension is scored on the integer scale of 1–10 by a language model evaluator that is given the specification above as its system prompt together with the candidate and, in diff mode, the original

module. The evaluator returns one integer per dimension and a short justification. The $\mathrm{total_score}$ is the arithmetic mean of the four integers, so it lies on the same 1–10 scale with one decimal of resolution.

Because a single judgement is sensitive to sampling, each instance is scored by an ensemble of independent judgements and the per-dimension scores are combined by their median, reported to two decimals. The default ensemble size is five. The ensemble is configured through the same model interface as the rest of the framework, and it can be run either as repeated judgements from one model at a per-run seed derived from the prompts, or as one judgement from each of several models. Aggregation is per dimension rather than over totals, so a judgement that is extreme on one dimension cannot move the other three. The justification reported alongside a score is taken from the judgement closest to the aggregated vector in L1 distance, which keeps the explanation consistent with the reported numbers rather than with an arbitrary member of the ensemble.

The severity threshold $f_R \geq 0.7$ used for N_{HR} (Equation 3) is the same quantity rescaled. The score enters the objective as $f_R = \mathrm{total_score}/10$, so the threshold corresponds to a mean dimension score of 7, and an instance counts as high risk when the ensembled verdict places it in the upper third of the scale on average across all four dimensions. Because aggregation takes a median, this threshold is reached only when a majority of the ensemble places the instance at or above 7 on enough dimensions to pull the mean there, which makes the count robust to a single outlying judgement in either direction.

G Operator Case Study

This section traces one real `shadow_register` lineage so that the abstract operator definitions of Section 4.2 can be read against concrete RTL. Figure 3 shows the lineage, and the paragraphs below explain what each operator changed and why that change matters.

G.1 The Seed Mechanism

Base Seed A is a guarded zeroing trojan. A single conjunction, combining a data comparison against a constant with an arming guard and a quiet-cycle counter, sets a latch named `spoof`, and the latch drives a payload that zeroes two output ports. The threat therefore consists of three roles in a fixed order, namely a trigger that decides when the mechanism arms, a state element that records the decision, and a payload that acts on it. Everything the three operators do is defined relative to this structure, and the roles stay legible throughout the lineage because each operator preserves the trigger-state-payload chain even when it rewrites the code that realizes it.

G.2 Mutate

Mutate leaves the mechanism intact and changes how it is written. The single inline conjunction becomes two separately named conditions, `arm` and `fire`, the latch is renamed, and the payload is rewritten against renamed output ports while keeping its form. No functional step is added or removed, and the destructive behavior is unchanged, yet almost no surface feature of the original statement survives. This is what makes Mutate effective against detectors that recognize structural patterns rather than behavior, and it is the reason the operator is defined as preserving the mechanism while mutating the realization.

G.3 Deepen

Deepen keeps the trigger exactly as it was and inserts a stage between the trigger and the latch. The original condition no longer sets the latch directly. It now opens a window counter, and a separate condition named `escalate` combines the window state with a confirmation count before the latch is set. The threat therefore requires a sequence of events spread over time rather than a single instant, which is the deeper temporal dependence the operator is defined to produce. The two roles stay separable within the snippet, since the inherited condition is unchanged at the head and the latch statement at the tail is the same statement as in the seed, now reached through a narrower predecessor.

G.4 RECOMBINE

Base Seed B supplies the second family. Its guard combines a delayed request with an internal ready signal, and its payload toggles a write-once lock, so the family differs from A in both what activates it and what it does. The recombination inherits A's state statement and extends it. The inherited statement now latches an additional signal, and that signal becomes the leading term of B's guard. The consequence is that B's payload can only fire once A's condition has been satisfied and latched, so the two mechanisms are joined in series rather than placed side by side. This is the chaining form of recombination, where one parent establishes an internal state that enables the other. The hybrid thus inherits a trigger from one family and a payload from the other, and the join between them is the new latched signal, which exists in neither parent.

G.5 WHAT THE LINEAGE SHOWS

Read together, the three branches act on one mechanism in three different ways. Mutate changes every surface feature while holding the mechanism fixed, so it attacks detectors that recognize structure. Deepen holds the trigger fixed and replaces a single-shot activation with a windowed one, so it attacks detectors that look for rare but instantaneous conditions. Recombine takes a guard from one family and a payload from another and joins them through a latched signal that neither parent contains, so it produces a composite threat whose parts are each individually familiar. The lineage also shows why the three operators are not substitutes, since each raises a different property of the threat while leaving the verification pipeline intact, and none of them raises all three at once.

H KNOWLEDGE GRAPH AND THREAT TAXONOMY

The Hardware Vulnerability Knowledge Graph (VKG) is the knowledge core that grounds DAVE's threat reasoning. It is built from authoritative weakness and vulnerability records rather than from model priors, so that every retrieved item can be traced back to a public disclosure.

H.1 CONSTRUCTION

The graph is extracted from the hardware weakness view of the Common Weakness Enumeration and enriched with concrete vulnerability evidence, in four steps.

Hierarchy extraction. The hardware design view (CWE-1194) is parsed into a structured hierarchy of categories and weaknesses, preserving the parent-child and peer relations that define how hardware weaknesses are organized.

Vulnerability download. A local snapshot of the public CVE corpus is obtained, so that enrichment does not depend on live API availability.

Evidence enrichment. Each weakness is matched against the CVE snapshot, and the matching records together with their reference documents are attached to the corresponding weakness node. This is the step that turns an abstract weakness definition into a weakness with documented real-world instances.

Import. The enriched hierarchy is loaded into a graph database, where weaknesses, categories, vulnerabilities, references, and code examples become nodes and the relations among them become typed edges. Loading the same source reproducibly rebuilds the same graph, which is what makes the knowledge state a fixed experimental artifact rather than a moving target.

H.2 RETRIEVAL

Retrieval is dual-modal. Every description in the graph, at the level of categories, weaknesses, vulnerabilities, and examples, is embedded and indexed for text search, and every code example is embedded and indexed separately for code search. A query therefore retrieves both the conceptual definition of a weakness and the concrete RTL that instantiates it, which is what allows the mining agents to move from a weakness pattern to a realizable modification rather than stopping at a textual description. Retrieval is routed by the active weakness context, so the agents see evidence relevant to the mechanism under investigation instead of the whole graph.

H.3 Threat Taxonomy

The taxonomy that organizes accumulated threat knowledge has three levels, as described in Section 4.2. A *threat family* is identified by the weakness it exploits, which is either a single CWE or, for recombined Trojans, a tuple of CWEs joined by the hybridization. Within a family, an *emergent attack strategy* groups instances that share a pattern-level behavior, including how the trigger is constructed and how the payload is routed to an observable effect. A *novel instance* is the concrete RTL implementation at the leaf level, stored with its evaluation scores and the reasoning returned by the detector ensemble. The three levels are linked rather than duplicated, so a strategy can be compared across families through its instances and an instance can be traced back to the pattern it realizes.

I Experimental Configuration

Table 2 in the main text gives the condensed configuration. This section expands each entry and records the protocol details needed to reproduce the results.

I.1 Datasets and Preprocessing

CORE (Kian Kit Cheah & Li, 2025) supplies the main evaluation corpus. It contains 137 clean RTL modules organized into 24 functionality categories, ranging from arithmetic units and arbiters to communication and memory components. Every module is a self-contained design with its own testbench and synthesis target, which lets the verification backbone run unmodified across the whole corpus. The 21 representative cases used for the evolution study are selected from this corpus for broad category coverage while avoiding extreme seed-quality outliers, so that the reported dynamics are not dominated by a handful of unusually easy or unusually hard designs.

GHOST (Faruque et al., 2025) supplies the external comparison. It releases three clean designs together with 14 publicly verified Trojan cases, and it is the only prior benchmark that publicly exposes verified LLM-generated RTL Trojan cases on shared clean designs. Because the same clean designs are available to both SpecterRTL and GHOST, the comparison isolates the generation procedure rather than confounding it with the choice of target hardware.

I.2 Verification Backbone

A candidate is accepted only if it passes three sequential checks, all executed against the original module and its unmodified testbench.

Lint. Verilator runs in lint-only mode with all warnings enabled. The testbench is treated as read-only and immutable throughout, so a candidate cannot pass by weakening the check.

Functional simulation. The original module is replaced by the candidate and the full testbench is re-run. The check requires both a zero exit status and the absence of failure markers in the simulation log, since a testbench can report a failed comparison while still exiting cleanly. Runs that exceed the time limit are treated as failures, which also catches deadlocks and infinite loops introduced by the candidate.

Logic synthesis. The candidate is synthesized through the same target as the original module, and the check requires a clean exit status. This rejects constructs that simulate correctly but cannot be realized in hardware, which is the case for many superficially evasive trigger designs.

A candidate that fails any check is routed to a repair loop with a bounded number of attempts. The loop returns a diagnostic to the generating agent, which applies a minimal refinement and re-enters the pipeline from the first stage. Candidates that exhaust the attempt budget are discarded and are not counted as verified seeds.

A verified instance is therefore admissible in the target flow. It preserves the module interface, disrupts no behavior that the original testbench observes, and remains realizable in hardware. The three checks jointly establish this, and they establish it by executing the candidate rather than by accepting a model's assertion. This is the standard the threat model of Section 3 sets. An adversary who poisons an LLM copilot or ships a contaminated third-party IP block must survive exactly these

checks to reach silicon, and the operative threat is an instance that passes them while still carrying a trigger the defenders never observe.

I.3 Detector Ensemble and Score Fusion

The evasion term $f_E = 1 - P_{\text{detect}}$ is obtained from three detectors combined with weights 3, 2, and 1 for GNN4TJ (Yasaei et al., 2021), BugWhisperer (Tarek et al., 2025), and SecRT-LLM (Saha et al., 2024). The weights follow the scope of each detector. GNN4TJ produces a continuous graph-level score and applies to every design, so it carries the highest weight. BugWhisperer performs intermediate semantic checking and receives an intermediate weight. SecRT-LLM is restricted to FSM settings and therefore applies to the narrowest subset, so it receives the lowest weight. Detector reasoning text is retained alongside the fused score and is supplied to the agents as explicit feedback during evolution.

I.4 Hyperparameters

Evolution runs for 20 iterations with 10 offspring per iteration, so each run attempts 200 offspring generations. The active pool is capped at 20 individuals with a capacity of 2 per behavior cell, and the elite archive holds at most 10 individuals. A behavior cell is the grid position of an individual in the objective plane, formed by binning f_R and f_E and pairing the result with the individual's CWE tuple, so capping occupancy per cell prevents the pool from being overwhelmed by near duplicates of one strong solution. All three are upper bounds rather than guarantees, and the realized values are reported with the results. The objective weight is $\omega = 0.7$, which favours severity while retaining detector pressure as a secondary objective, as discussed in Section 4.1. Severity is scored by the four-dimensional Verilog Vulnerability Intricacy Score, whose scoring and ensemble aggregation are specified in Appendix F.

I.5 Evolution Baseline

OpenEvolve (Sharma, 2025) is an open-source implementation of the AlphaEvolve (Novikov et al., 2025) style code evolution workflow and serves as the evolution baseline. It evolves each seed in an independent run of 200 generations, so the baseline receives a larger per-seed generation budget than SpecterRTL, which evolves all seeds jointly inside a single population of 20 iterations and 10 offspring per iteration. OpenEvolve maintains no shared population across seeds and supports no repair loop, so offspring that fail verification are not fixed and instead receive zero evaluation scores while remaining in the evolving population. Both differences follow from the baseline's own architecture rather than from a configuration choice made here, and the comparison therefore reflects each system as designed.

I.6 Agent Prompts and Workflow

Every agent runs as a bounded tool-use loop over the same interface, and each role differs in its system prompt, its retrieval permissions, and the tools it may call. The workflow is implemented in the supplementary code released with this paper.

Mining roles. The *Security Auditor* holds the target RTL and the weakness context, and drives the search by issuing retrieval queries. The *Knowledge Explorer* owns the VKG tools, resolving a query into CVE evidence, CWE descriptions, and code examples, and returning both text and code matches. The two iterate until the evidence supports a single weakness specification. The *Defense Architect* then receives the accumulated evidence and emits a ranked schema item fixing the activation condition, the payload behavior, and the affected signals. The *Red Tester* receives that schema together with the original module and produces a complete RTL modification. The generating prompt requires the modification to preserve the module interface and to condition the payload on a trigger drawn from the schema, and it supplies the module's own signal names so that the trigger references real logic rather than invented identifiers.

Evolution operators. Each operator has its own system prompt and its own transformation handbook. The handbook for *Mutate* enumerates semantics-preserving restructuring patterns, covering parameterization of literal constants, single-process and dual-process state machine forms, state en-

codings, conditional and case-form control flow, procedural and dataflow styles, port declaration and sensitivity-list forms, signed arithmetic, generate blocks, operator equivalences, naming conventions, and loop unrolling. The handbook for *Deepen* states the four dimensions of Appendix F as targets and lists directions for each. The handbook for *Recombine* defines the three fusion modes described in Section 4.2. Operator prompts receive the parent RTL or parents, the retrieved exemplars, and the explicit evaluation and detector feedback described in the ablation study of Section 5.2.

Repair. A failed candidate re-enters the workflow with the diagnostic from the failing check, and the repair prompt instructs the agent to make the smallest change that resolves the diagnostic without disturbing the trigger or the payload logic.

I.7 TOOLCHAIN

All agents run on GPT-5.2. Verification uses Verilator for linting and functional simulation, and Yosys for logic synthesis, each invoked against the module's own synthesis target. The knowledge graph is stored in Neo4j, and retrieval uses vector search over embedded text and code representations. The complete configuration, including the data processing steps, is provided in the supplementary material.

J RESPONSIBLE DISCLOSURE AND DUAL-USE CONSIDERATIONS

Research that generates evasive hardware Trojans carries an obvious dual-use dimension, and the safeguards below were adopted for that reason.

Scope of evaluation. Every experiment runs on the public corpora described in Appendix I, namely CORE and the clean designs released with GHOST. No proprietary design, commercial IP, or fabricated device was involved at any stage, and no result in this paper describes the behavior of any specific product. The threat model of Section 3 is therefore studied at the level of RTL structure rather than at the level of a deployed system.

Nature of the generated artifacts. The Trojans produced by the framework are synthesized test cases rather than exploits. They are built to satisfy a verification pipeline on benchmark modules that are already public, and they depend on the module's own ports and signals, so a generated case does not transfer to a real product without the same access an attacker would already need in order to insert a Trojan there at all. The framework does not discover or exploit a defect in any third-party design.

Disclosure. No vulnerability in a third-party product was discovered during this work, so no vendor disclosure is applicable. Degradation of the defensive tools evaluated here is a property measured within our own experimental setting, not a reported defect in a fielded system.

Future release. Upon publication we intend to release the framework and the benchmark cases for research use, accompanied by the detection results reported in Section 5.2. The intent is that the same cases serve both directions of work, so that defensive research can use them to measure and improve detection rather than the offensive direction being the only beneficiary. The released material is the evaluated set rather than an unbounded generator aimed at any particular design.