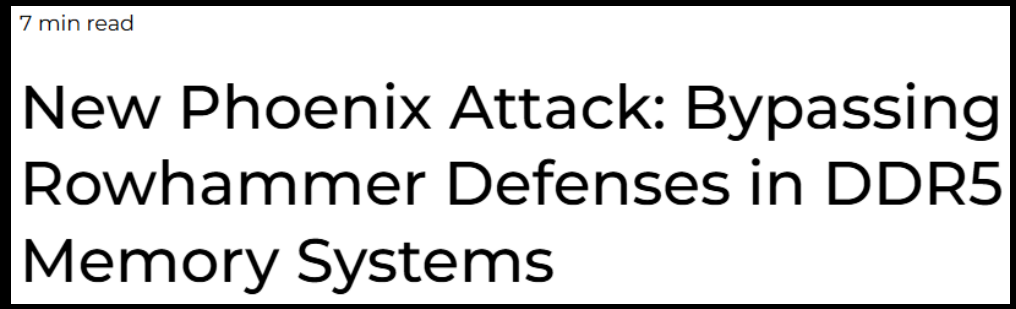
GoldenFuzz: Generative Golden Reference Hardware Fuzzing

Lichao Wu[1,2], Mohamadreza Rostami[1], Huimin Li[1], Nikhilesh Singh[1], and Ahmad-Reza Sadeghi[1]

[1]*Technical University of Darmstadt, Germany*

[2]*University of Bristol, United Kingdom*

Hardware Exploits Causes Damages



Hardware Exploits Causes Damages

Google-Intel Security Audit Reveals Severe TDX Vulnerability Allowing Full Compromise

Dozens of vulnerabilities, bugs, and potential improvements have been identified by the tech giants' security teams.

OCTOBER 8, 2025

How to Build Trust in Hardware?

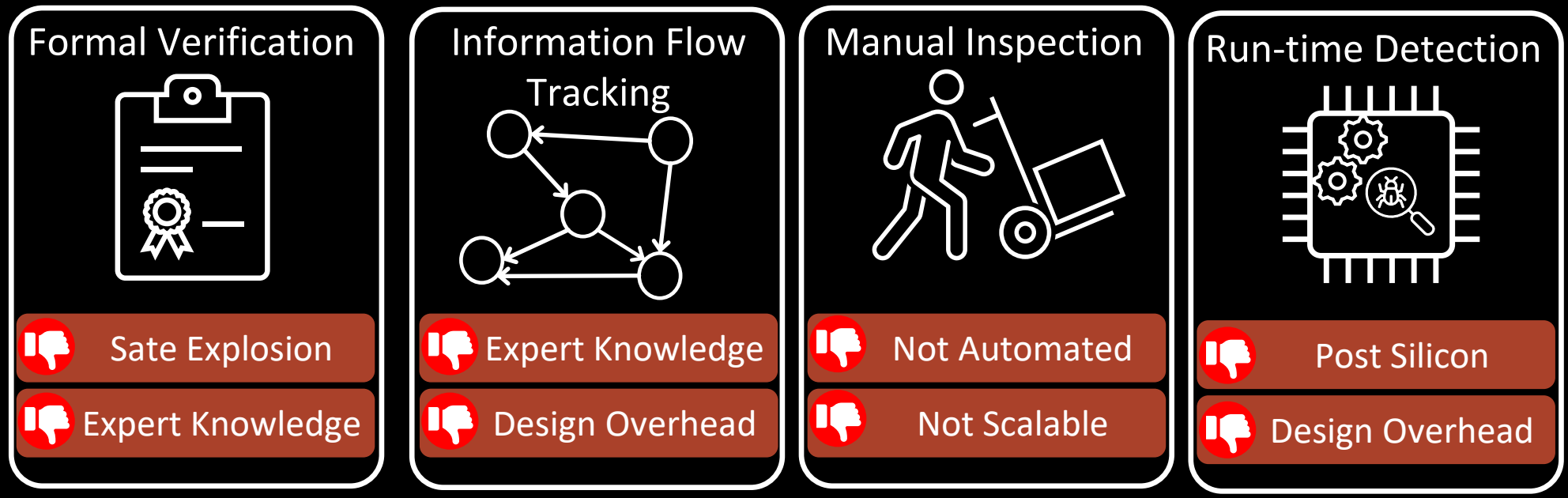
News

Dec 11, 2025 • 3 mins

7 min read

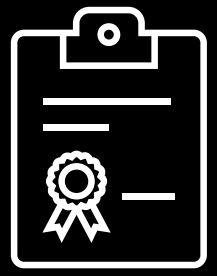
New Phoenix Attack: Bypassing Rowhammer Defenses in DDR5 Memory Systems

Detecting Hardware Vulnerabilities



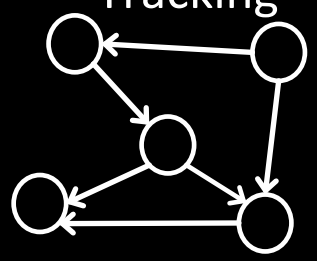
Detecting Hardware Vulnerabilities

Formal Verification



- Sate Explosion
- Expert Knowledge

Information Flow Tracking

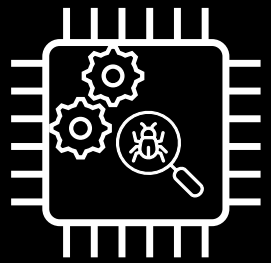


- Expert Knowledge
- Design Overhead

Manual Inspection

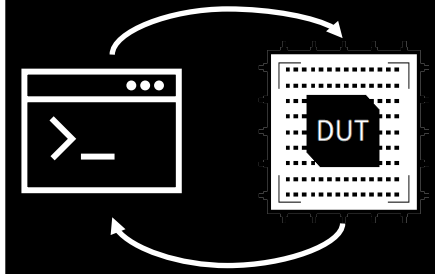
- Not Automated
- Not Scalable

Run-time Detection



- Post Silicon
- Design Overhead

Hardware Fuzzing



- Scalable
- No state explosion

Detecting Hardware Vulnerabilities

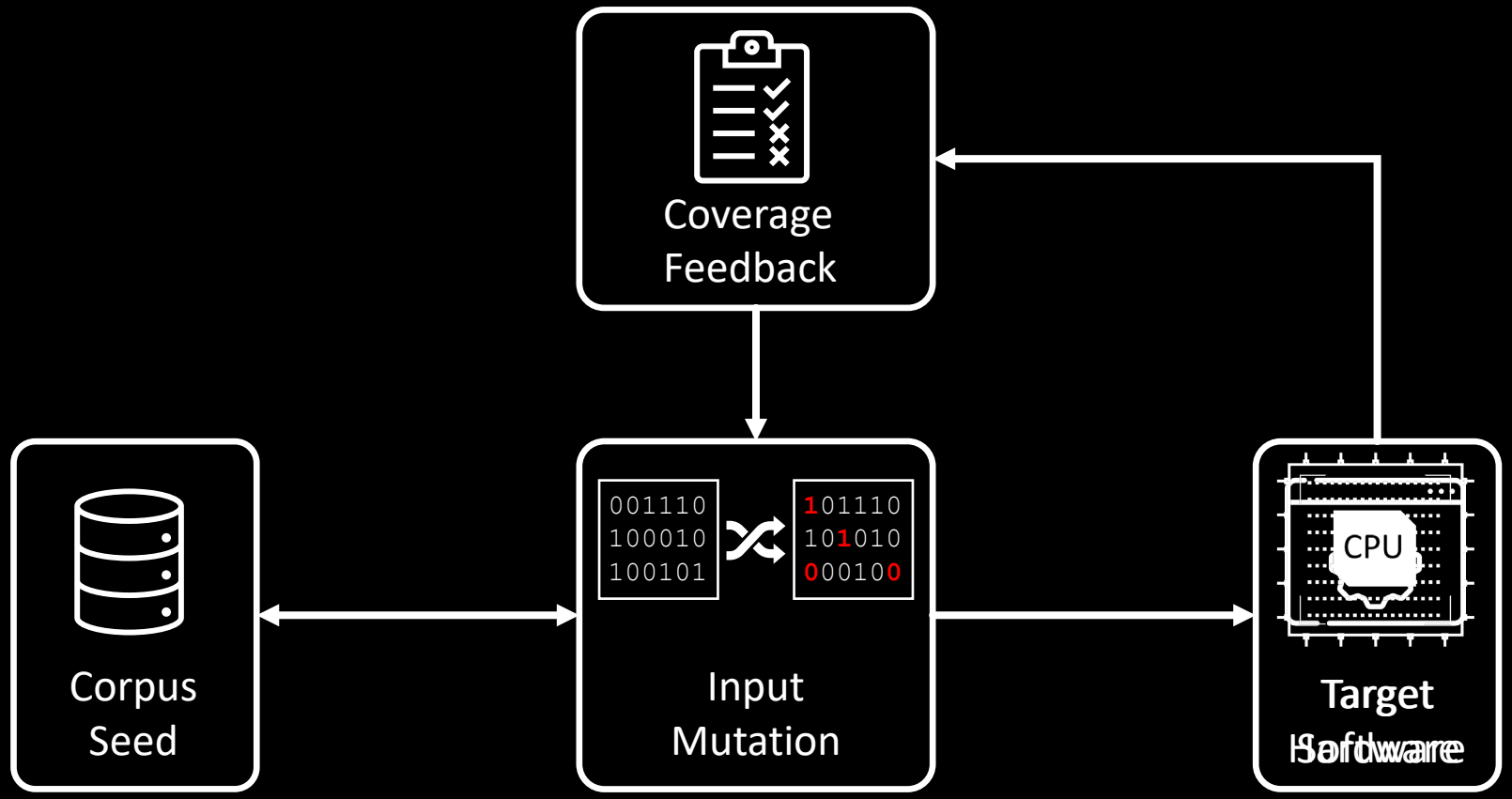
Fuzzing has shown promising results

+27,000
Bugs

+1,800
Bugs

+11,000
Bugs

How to Do Hardware Fuzzing?



Limitations of Current Fuzzing Techniques

Random Mutation

- **Low Semantic Validity**
  Generates many invalid instructions that are rejected early by the decoder.
- **Shallow Exploration**
  Struggles to pass complex checks or reach deep logic states.

Limitations of Current Fuzzing Techniques

Random Mutation

- **Low Semantic Validity**
  Generates many invalid instructions that are rejected early by the decoder.
- **Shallow Exploration**
  Struggles to pass complex checks or reach deep logic states.

Grammar-Based

- **Manual Effort**
  Requires tedious, expert-driven definition of grammar rules and templates.
- **Rigid Structure**
  Limited by the defined rules; misses "corner cases" outside the grammar.

Limitations of Current Fuzzing Techniques

Random Mutation

- **Low Semantic Validity**
  Generates many invalid instructions that are rejected early by the decoder.
- **Shallow Exploration**
  Struggles to pass complex checks or reach deep logic states.

Grammar-Based

- **Manual Effort**
  Requires tedious, expert-driven definition of grammar rules and templates.
- **Rigid Structure**
  Limited by the defined rules; misses "corner cases" outside the grammar.

The Missing Piece: Automated Semantic Awareness

Need a method that learns valid semantics automatically while exploring diverse states.

GoldenFuzz: A Novel Generative Fuzzing Framework

1. Generation

Customized language model generates semantically valid instruction blocks for RISC-V.

GoldenFuzz: A Novel Generative Fuzzing Framework

1. Generation

Customized language model generates semantically valid instruction blocks for RISC-V.

2. Execution

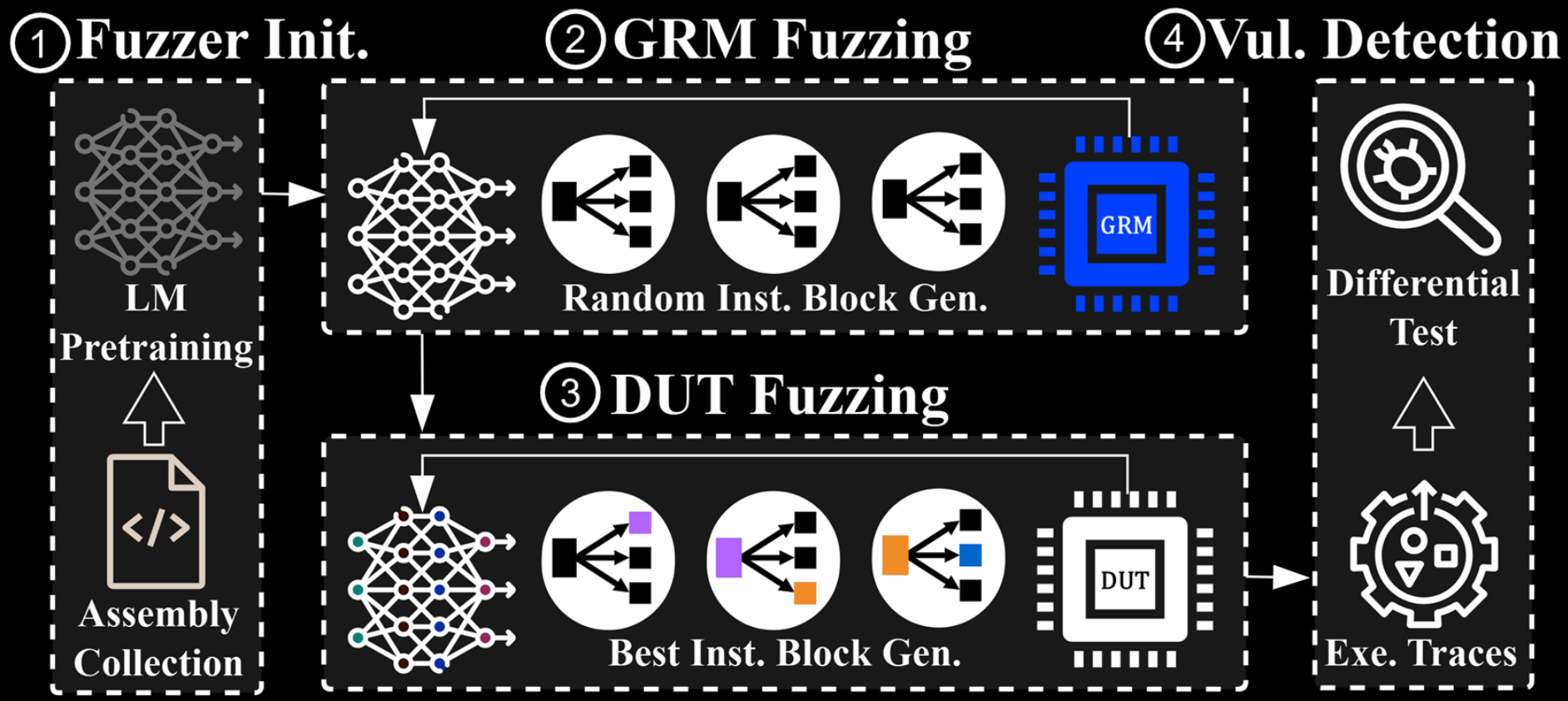
Dual execution on DUT and GRM to identify discrepancies via differential testing.

GoldenFuzz: A Novel Generative Fuzzing Framework

GoldenFuzz: A Novel Generative Fuzzing Framework

Bridging the gap between Semantic Awareness and Hardware Coverage.

GoldenFuzz: A Novel Generative Fuzzing Framework



Fuzzer Initialization

GPT-2

0.8 Billion Params

Training Corpus

10 Million

Randomly generated RISC-V assembly instructions covering all standard extensions.

Training Duration

50,000 Epochs

Extensive training ensures the model captures deep syntactic and semantic patterns.

Infrastructure

NVIDIA A6000

High-performance GPU used for both training and rapid inference during fuzzing.

Objective

Causal LM

Next-token prediction task optimized for generating valid instruction sequences.

Hardware-Specific Tokenization Strategy

Input: "addi x1, x0, 10"

Standard BPE

ad	di	x	1	,	x	0	,	10

Fragmented Semantics

Hardware-Specific Tokenization Strategy

Input: "addi x1, x0, 10"

Standard BPE ad | di | x | 1 | , | x | 0 | , | 10 — **Fragmented Semantics**

GoldenFuzz addi | x1 | x0 | 10 — **Atomic Preservation**

Atomic Units

Treats Opcodes and Operands as distinct, indivisible units, preserving the structural integrity of assembly.

Semantic Learning

Enables the LLM to learn architectural rules and dependencies rather than just character patterns.

Efficiency

Reduces sequence length and vocabulary noise, leading to faster training and generation.

Leveraging the Golden Reference Model (GRM)

What is a GRM?

A high-level, trusted software model (e.g., Spike for RISC-V) that strictly adheres to the ISA specification.

Leveraging the Golden Reference Model (GRM)

What is a GRM?

A high-level, trusted software model (e.g., Spike for RISC-V) that strictly adheres to the ISA specification.

The Oracle
Provides the "ground truth" for architectural state (registers, memory).

High Speed
Simulates instructions orders of magnitude faster than RTL.

The Guide
Used by GoldenFuzz to steer generation, not just check correctness.

Leveraging the Golden Reference Model (GRM)

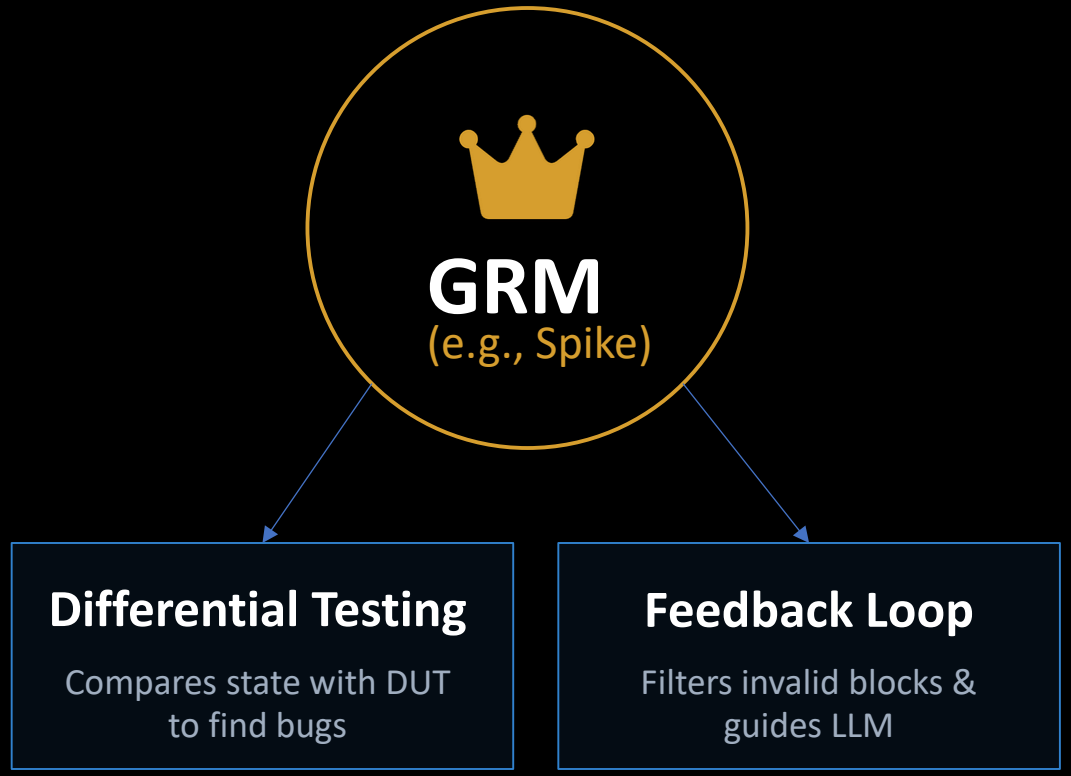
What is a GRM?

A high-level, trusted software model (e.g., Spike for RISC-V) that strictly adheres to the ISA specification.

The Oracle
Provides the "ground truth" for architectural state (registers, memory).

High Speed
Simulates instructions orders of magnitude faster than RTL.

The Guide
Used by GoldenFuzz to steer generation, not just check correctness.



Iterative Instruction Block

Instruction Block (IB) Strategy

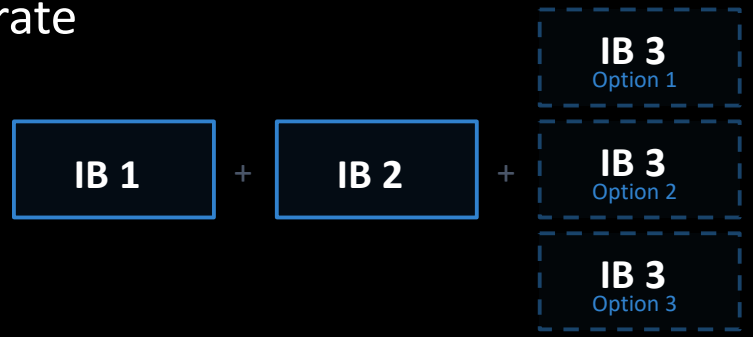
Generates small groups of instructions (blocks) rather than full test cases.

Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

01 Generate

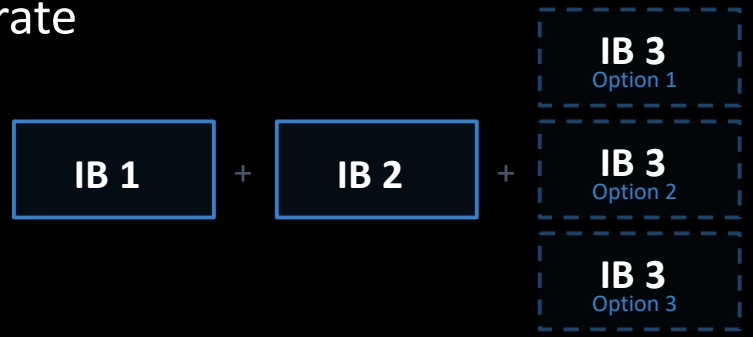


Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

01 Generate



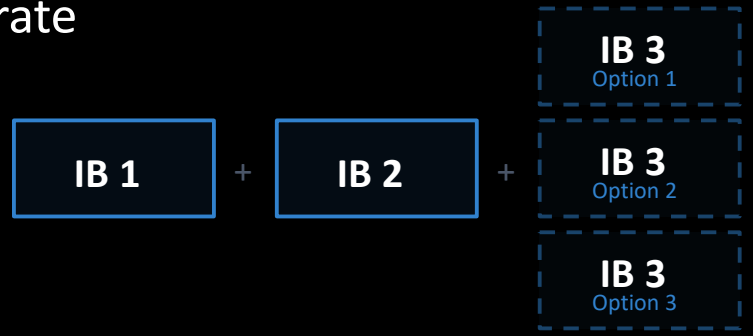
02 Select & Score

Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

01 Generate



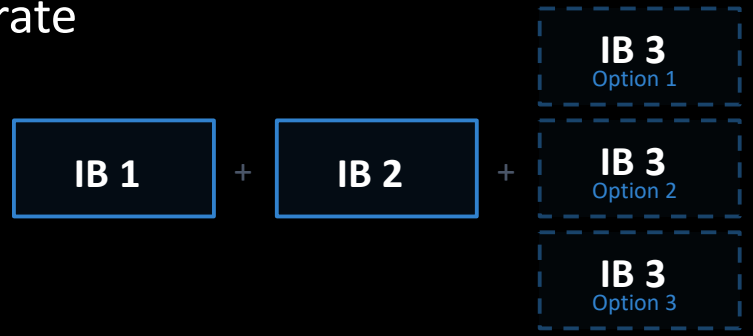
02 Select & Score

Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

01 Generate



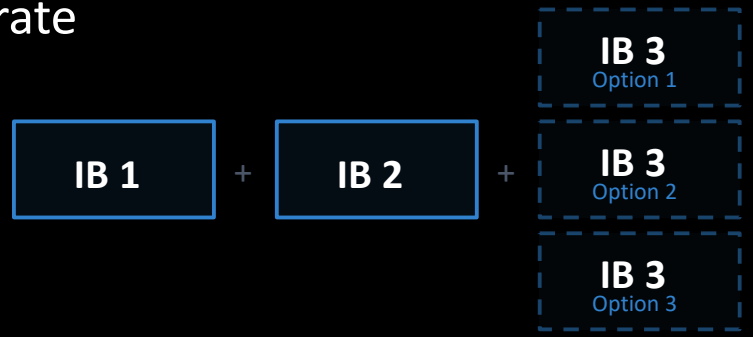
02 Select & Score

Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

01 Generate



02 Select & Score

03 Concatenate

Iterative Instruction Block

Instruction Block (IB) Strategy

Generates small groups of instructions (blocks) rather than full test cases.

Anchors Execution

Starts from known hardware states for focused exploration.

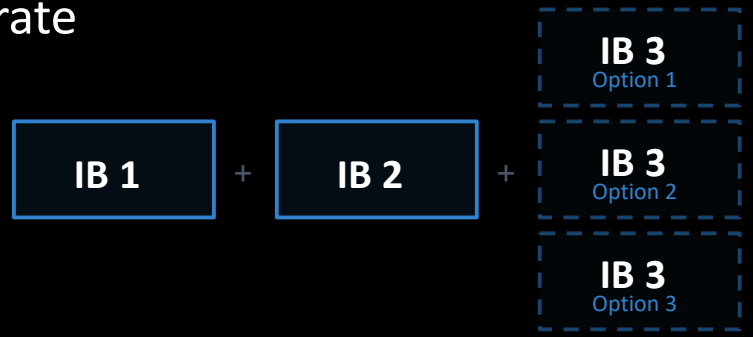
Simplifies Learning

Model learns patterns for small blocks instead of long sequences.

Diverse Exploration

Repeated sampling of new IBs covers different state spaces.

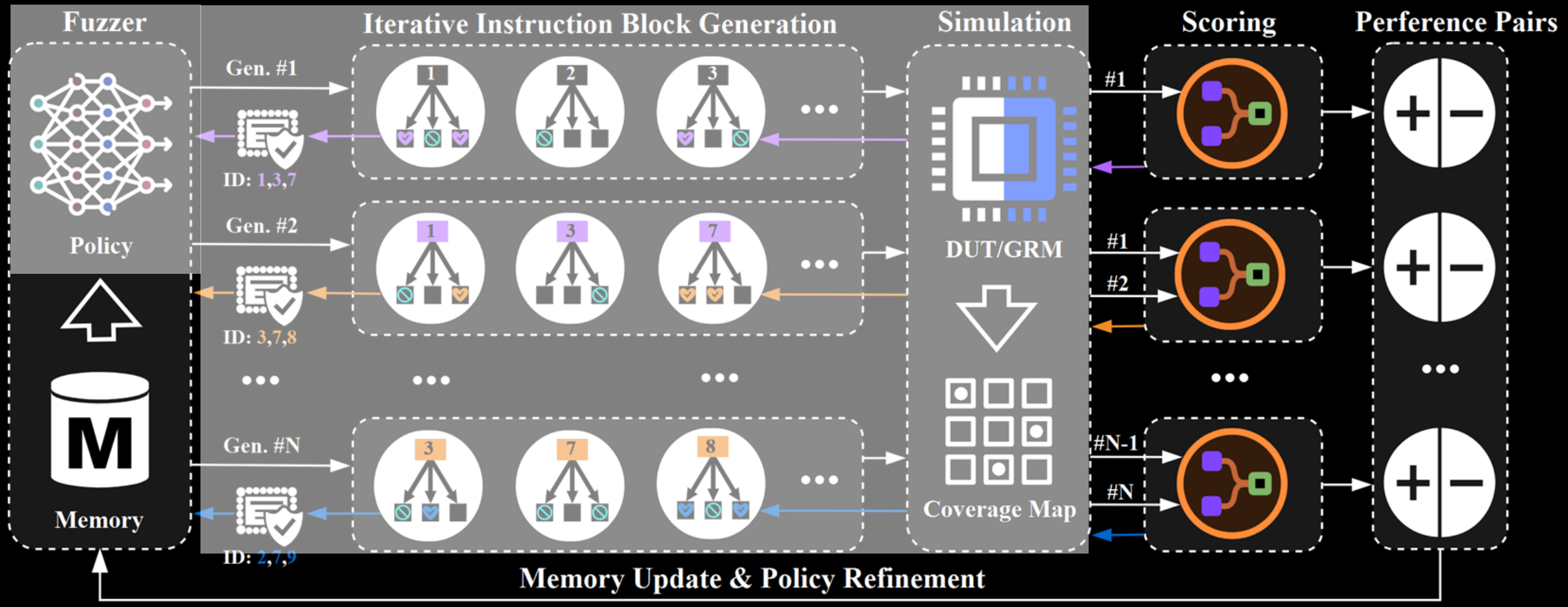
01 Generate



02 Select & Score

03 Concatenate

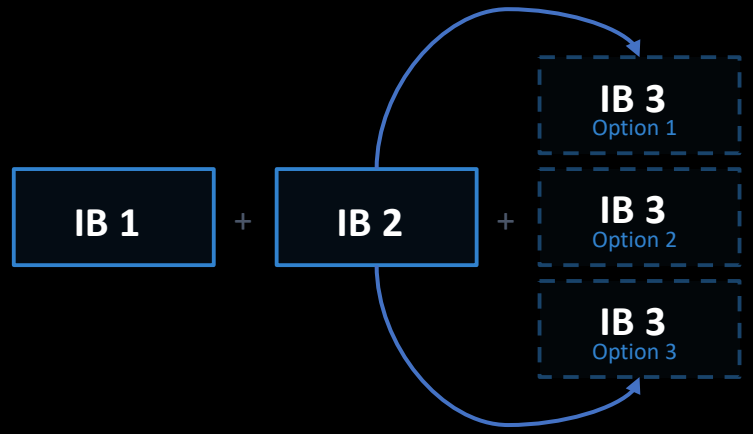
GoldenFuzz Pipeline



Test Case Scoring

Intra-Test Scoring

Encourage the model to discover new coverage points (edges, lines) within the current test case, prioritizing exploration over repetition.

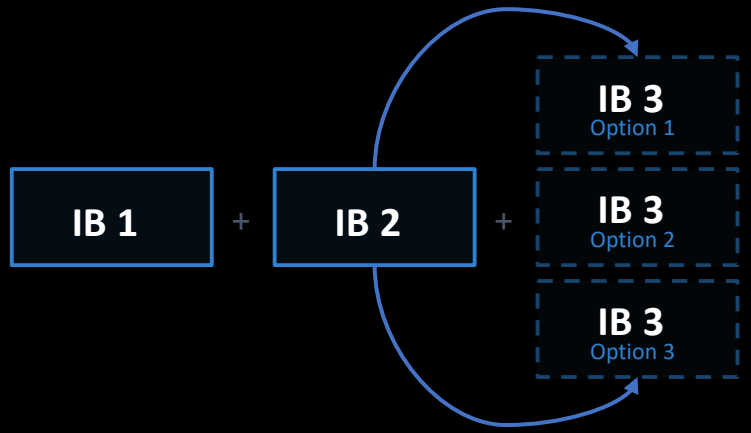


“Try to find something new."

Test Case Scoring

Intra-Test Scoring

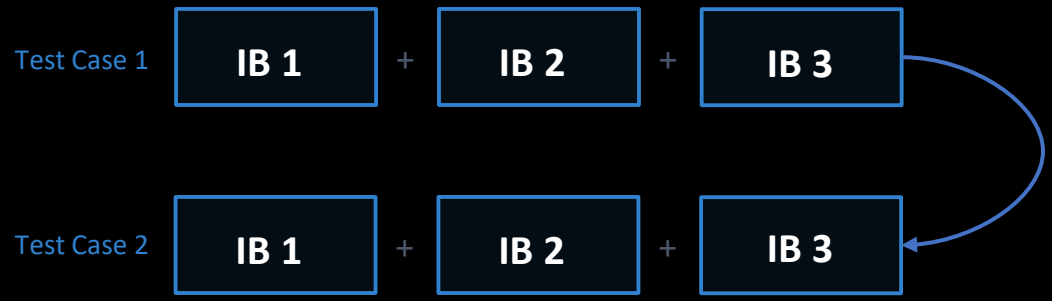
Encourage the model to discover new coverage points (edges, lines) within the current test case, prioritizing exploration over repetition.



“Try to find something new."

Inter-Test Scoring

Reduce the reward of high-reward test cases in value over time. The model is motivated to seek new, less-visited regions of the state space.



"If you've seen it before, it's worth less now."

Direct Preference Optimization (DPO) Refines Generation

Preference Learning

Instead of complex Reinforcement Learning (RL), DPO directly optimizes the model to prefer "winning" blocks over "losing" ones based on coverage data.

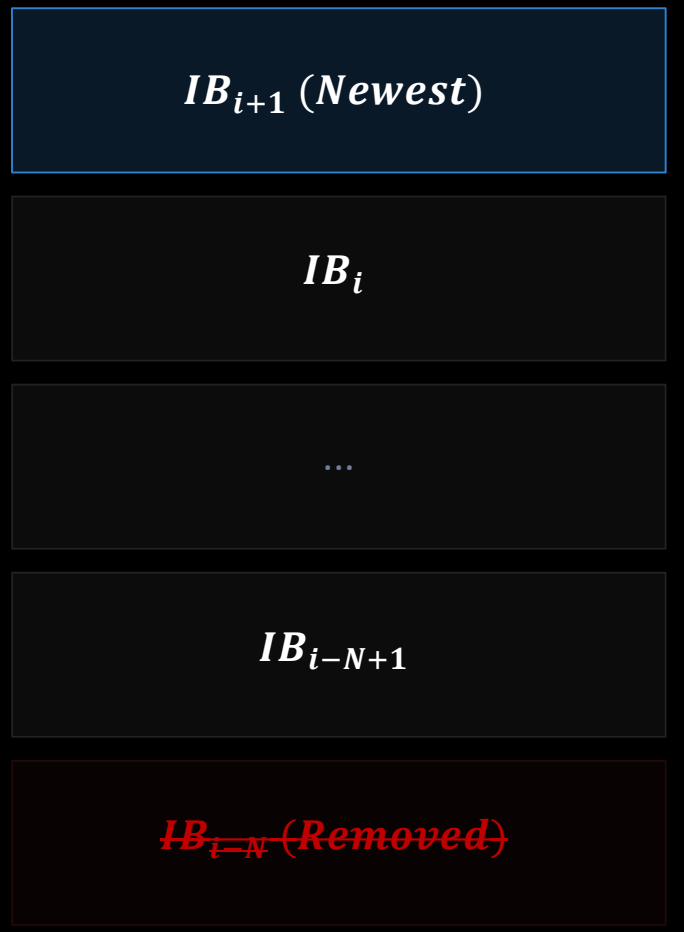
Fuzzing Memory

Rolling Window

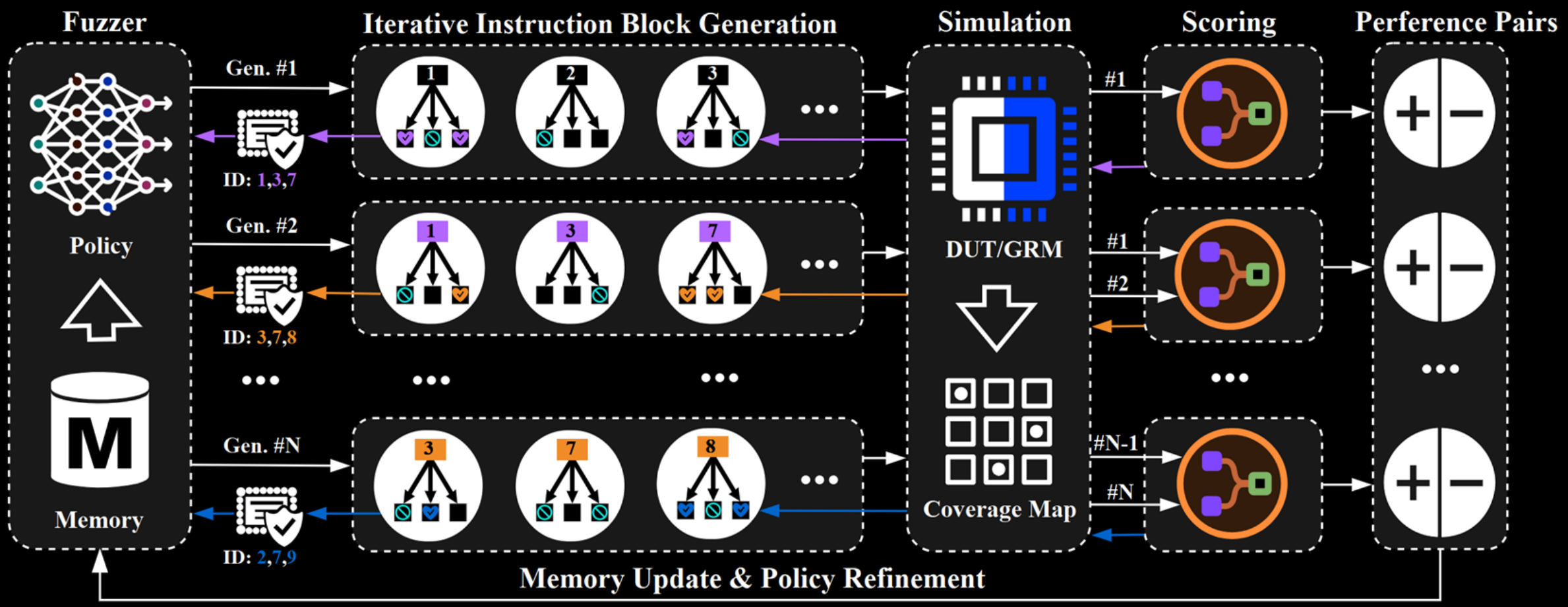
Follows a "First-In-First-Out" (FIFO) principle to continuously refresh the pool of high-performance IBs

Exponential Weighted Sampling

Newer samples are prioritized to exploit recent discoveries, while older samples remain accessible to prevent overfitting to local maxima.



GoldenFuzz Pipeline



Experimental Setup and Baselines

Designs Under Test (DUTs)

RocketChip

In-Order Core

5-stage pipeline, single-issue processor. Represents standard embedded RISC-V implementations.

BOOM

Out-of-Order

Superscalar, out-of-order execution. High-performance core with complex branch prediction.

CVA6 (Ariane)

Out-of-Order

6-stage pipeline, out-of-order execution. Industrial-grade core used in security-critical contexts.

Coverages

FSM Coverage

Measures the percentage of visited states and transitions within the design's Finite State Machines.

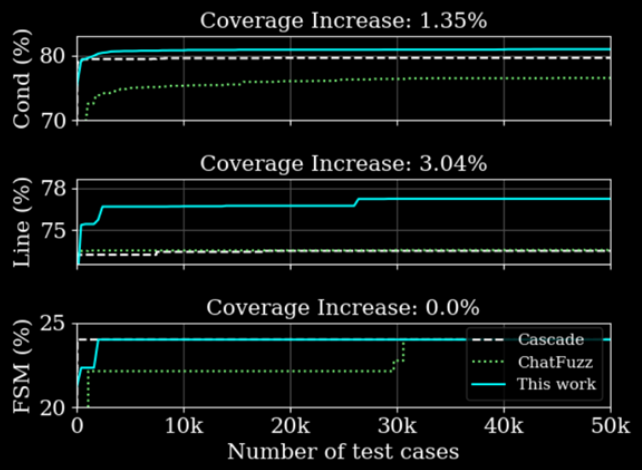
Condition Coverage

Verifies that every boolean sub-expression has been evaluated to both True and False.

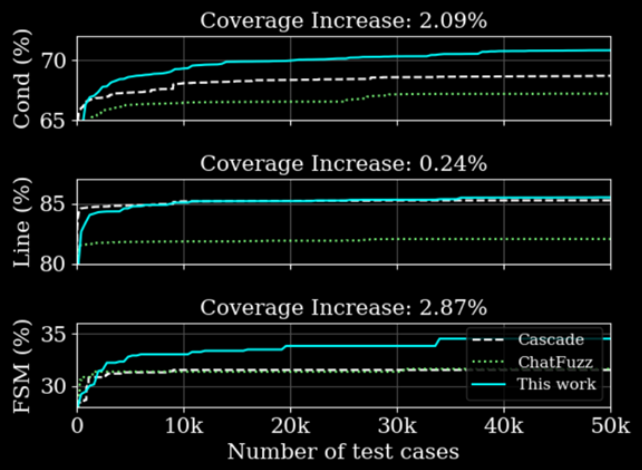
Line Coverage

Tracks the percentage of executable RTL code lines that have been exercised during simulation.

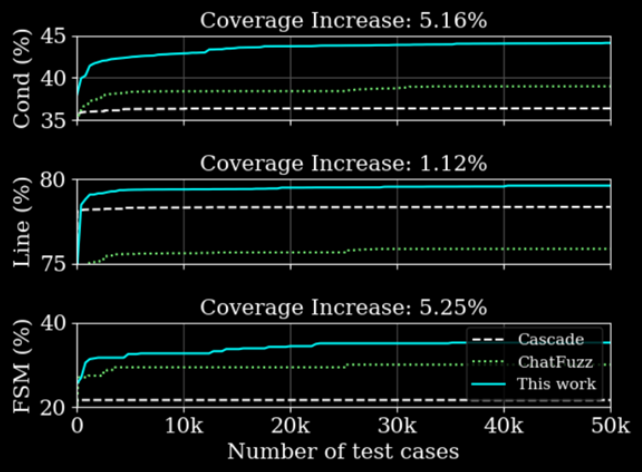
Experimental Results



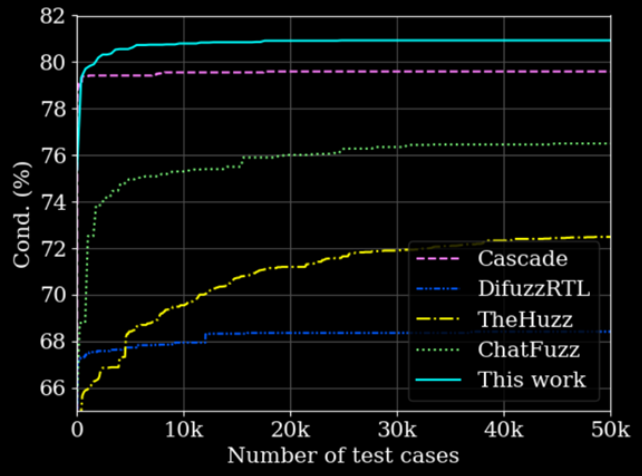
(a) RocketChip



(b) Boom



(c) CVA6



(d) All Fuzzers on RocketChip

Overview of Discovered Vulnerabilities

7 New Vulnerabilities Confirmed in Open-source and Proprietary RISC-V Cores

V1 Incorrect Store Endianness

CVA6 · Logic Error

V2 Incorrect Load Endianness

CVA6 · Logic Error

V3 Interrupt Masking Failure

CVA6 · Security

V4 Exception Priority Inversion

CVA6 · Control Flow

V5 CSR Access Control Bypass

CVA6 · Privilege Escalation

V6 Invalid Instruction Decoding

BOOM · Decoder

V7 Undisclosed Commercial Bug

BA51-H · Proprietary

Case Study: Endianness Vulnerability (V1 & V2)

Target Core

CVA6 (Ariane)

A 6-stage, out-of-order RISC-V core capable of running Linux.

Root Cause

LSU Logic Error

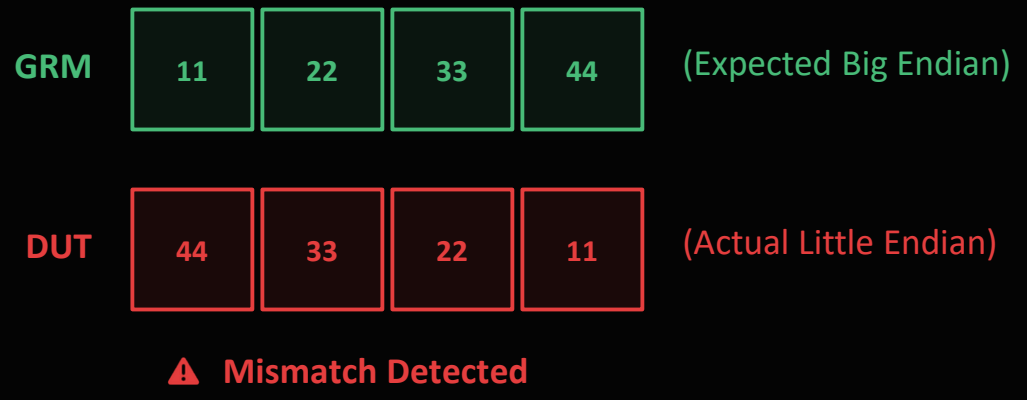
The Load-Store Unit failed to correctly check the mstatus.MBE (Memory Big Endian) bit during specific unaligned access scenarios.

Impact

Data Corruption

Critical system data could be read in reverse byte order, leading to potential crashes or exploitable logic errors.

```
1  li    t0, 0x11223344
2  sw    t0, 0(a0)      // Store to memory
3  csrs  mstatus, 0x20  // Set Big-Endian Mode
4  lw    t1, 0(a0)      // Load back
```



Case Study: Commercial Core

BA51-H Core

A commercial, high-performance RISC-V processor used in industrial applications.

The Challenge

Rigorous Verification

Unlike academic cores, commercial IPs undergo extensive internal testing and validation before release.

Closed Ecosystem

Limited visibility into internal microarchitecture makes "grey-box" fuzzing significantly harder.

Case Study: Commercial Core

BA51-H Core

A commercial, high-performance RISC-V processor used in industrial applications.

The Challenge

Rigorous Verification

Unlike academic cores, commercial IPs undergo extensive internal testing and validation before release.

Closed Ecosystem

Limited visibility into internal microarchitecture makes "grey-box" fuzzing significantly harder.

GoldenFuzz Results

- **Proven Effectiveness:** Successfully identified a critical logic bug that bypassed internal regression tests.

Vulnerability V7

Confirmed Logic Error in Privileged Mode Handling

Summary of Contributions

Generative Golden Reference

Pioneered the use of Large Language Models (LLMs) as an executable oracle for hardware verification, eliminating the need for manual reference models.

Feedback-Driven Refinement

Introduced a novel feedback loop using Direct Preference Optimization (DPO) to align the LLM's generation with valid hardware behavior.

7 New Vulnerabilities

Discovered and confirmed critical bugs in widely used RISC-V cores (CVA6, BOOM) and a commercial IP, demonstrating real-world effectiveness.

High Efficiency

Achieved 28x speedup via GPU acceleration, enabling high-throughput fuzzing that outperforms traditional mutation-based approaches.

Future Directions

Cross-ISA Expansion

Extending the Golden Reference Model and LLM tokenizer to support other major architectures, enabling universal hardware fuzzing beyond RISC-V.

ARM x86 POWER

RAG Integration

Incorporating Retrieval-Augmented Generation to dynamically fetch relevant specification details and errata, reducing hallucination and improving test case precision.

Spec PDF → Vector DB → Prompt

GoldenFuzz: Generative Golden Reference Hardware Fuzzing

Lichao Wu[1,2], Mohamadreza Rostami[1], Huimin Li[1], Nikhilesh Singh[1], and Ahmad-Reza Sadeghi[1]

[1]*Technical University of Darmstadt, Germany*

[2]*University of Bristol, United Kingdom*