SYSTEM HANG?

NO RESTART OPTION

μScan: Deep Learning Detection of Faulty Micro-architecture States and Patterns from Scan-Chain Data

Dillibabu Shanmugam, Zhenyuan Liu, Andrew Malnicof, Patrick Schaumont

6th workshop on AIHWS
in conjunction with ACNS 2025

Current Research in fault modeling

Simulation Based Research

- Relies on predefined fault models (bitflips, instruction skips, etc.)
- Heavily dependent on model accuracy and assumptions

Empirical Research

- Observes the real-world effects of faults
- Limited visibility into lowlevel hardware interactions
- Cannot explain by the immediate output

Can Machine Learning enable better fault modeling and analysis?

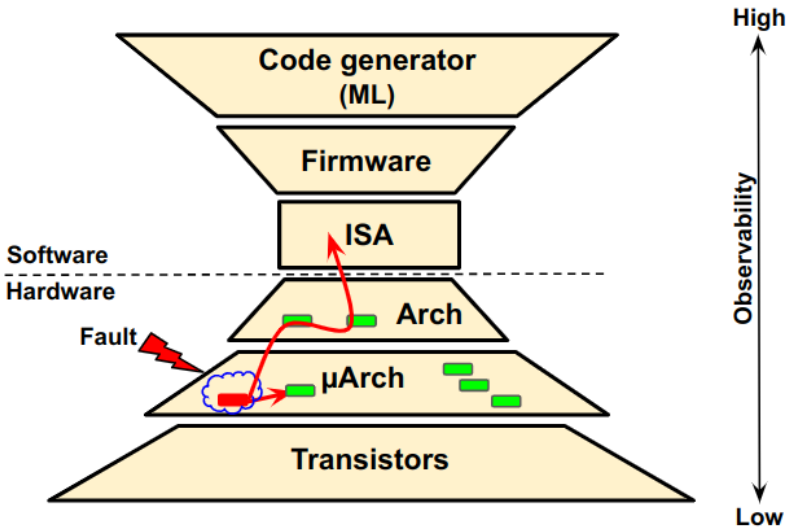
Key Challenges

Fault attacks on crypto engines have three outcomes:

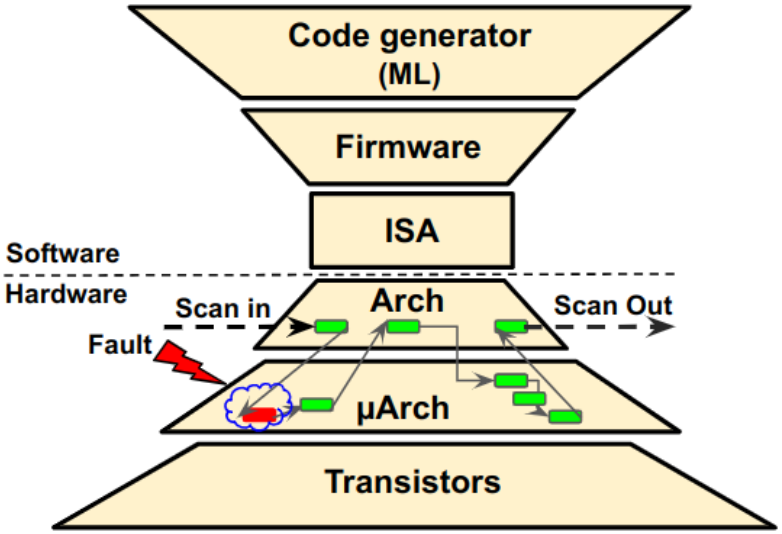
- Correct Ciphertext ✓
- No observable effect ⊖
- Faulty Ciphertext ×

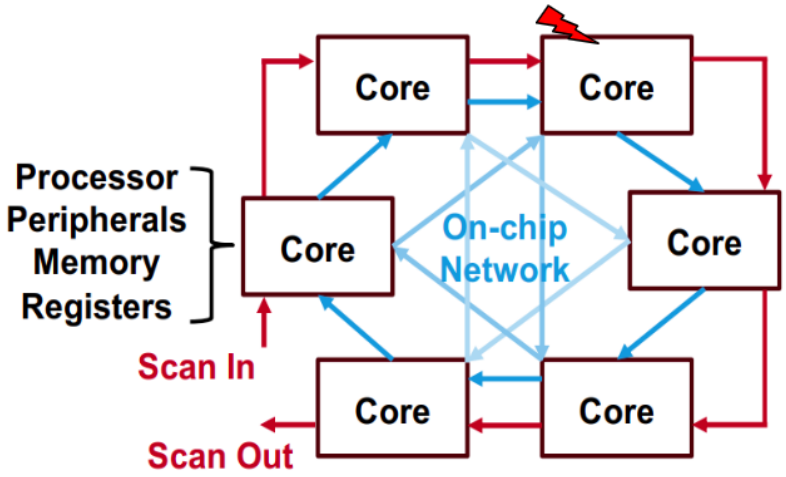
Limited observability across HW/SW stack
Difficult to understand fault behaviour
Pinpointing fault origin is challenging

HW/SW Abstraction



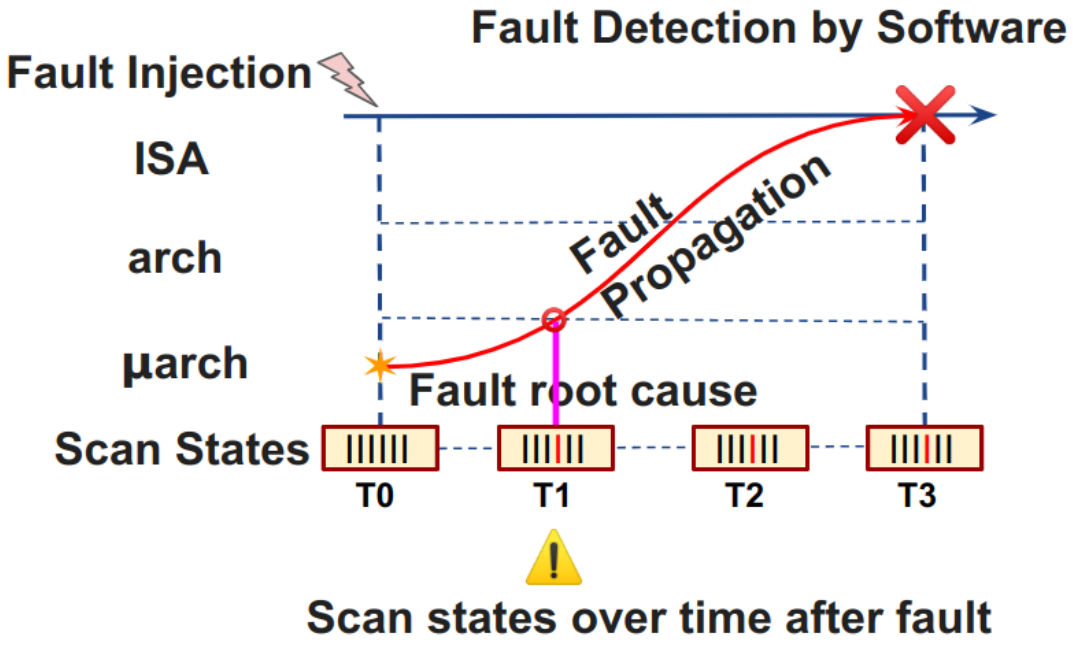
HW/SW Abstraction and Observability





Scan chain + hardware redundancy in lock-step unveils hidden micro-architectural states.

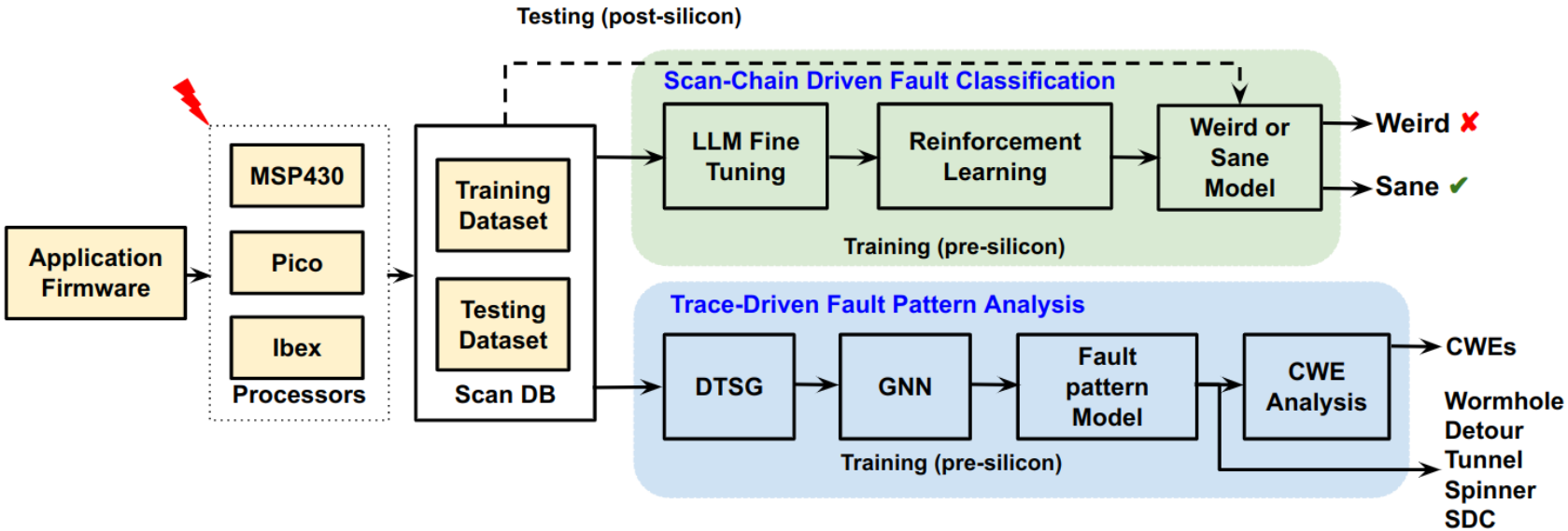
Root-Cause Analysis



- Redundant cores → expensive
- Fullscan at runtime → timing overhead

Our solution: Pre- and post-silicon scan corpus + ML classifiers

μScan Framework Overview



- **ScanChain Driven Classification:** Singlecycle states via LLM
- **TraceDriven Pattern Analysis:** Multicycle DSTGs via GNN
- Identify CWE categories

Scan-Chain Driven Fault Classification

Why use a Large Language Model (LLM)?

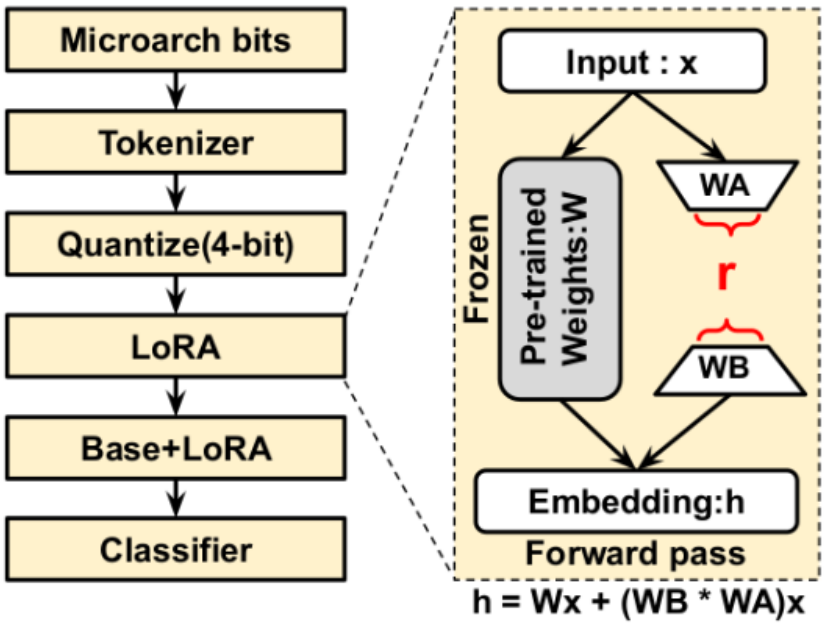
- *Causal system* modeling: captures cause →effect in state bits
- *Order sensitive*: positional encodings preserve bit ordering
- *Parallelization*: attention lets you score all bitpairs at once
- *Longterm dependencies*: can spot distributed, subtle faults

Attention (simplified):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

Contd

- Finetune QLoRA adapters on DeepSeek-R1-Distill-Qwen-1.5B
- Input: quantized microarchitectural bittokens



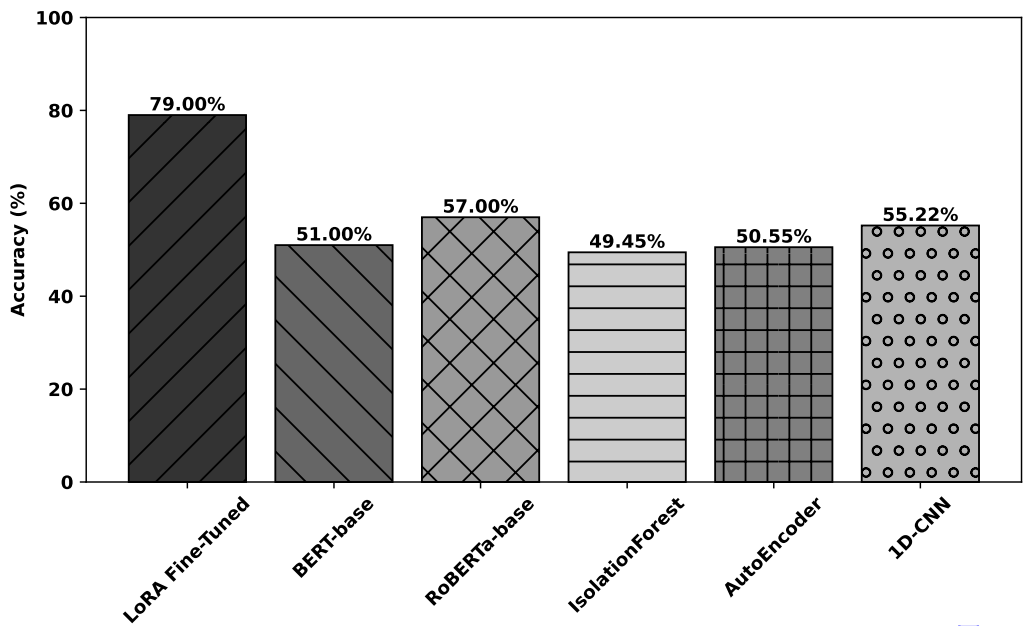
Binary Classification:

$$p(\text{weird} \mid x) = \sigma\big(y(x)\big), \quad \text{Class}(x) = \begin{cases} \text{weird}, & p(\text{weird} \mid x) \geq \tau, \\ \text{sane}, & \text{otherwise.} \end{cases}$$

▸ Next: Scan-chain Values

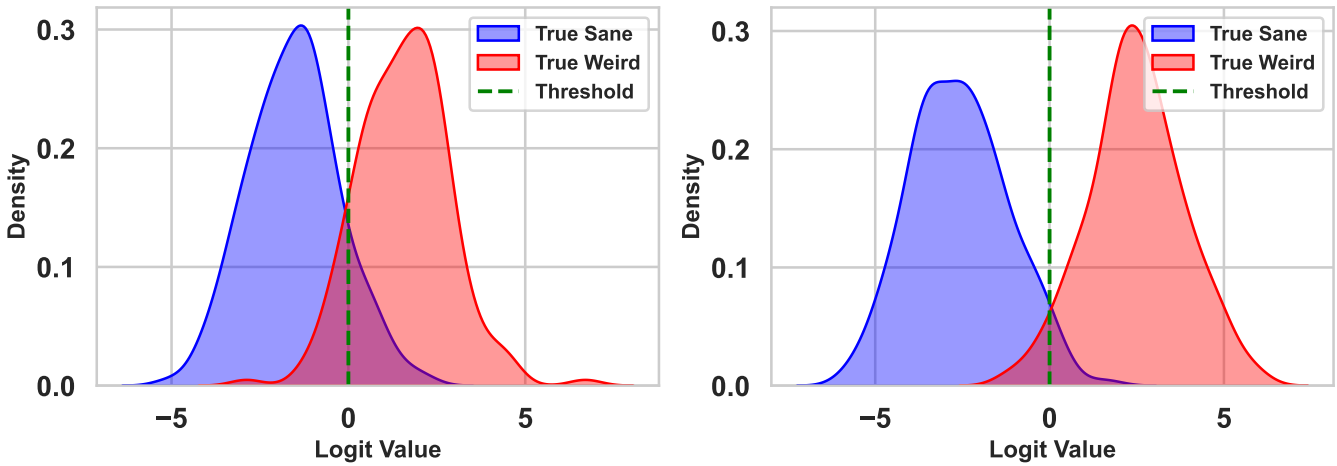
Performance

Test Case	Target	FSMRed	ASCON Sbox	VP5	Random Inst	MOV/ADD/MUL	Train Sane	Train Weird	Test Sane	Test Weird	Acc. (%)	Time (min)
1	MSP430	•	•	•			183	183	182	182	79	6+1
2	IBEX		•		•		486	1,013	402	1,000	97	370+15
3	CAPRI6					•	292	21,019	517	517	90	460+6

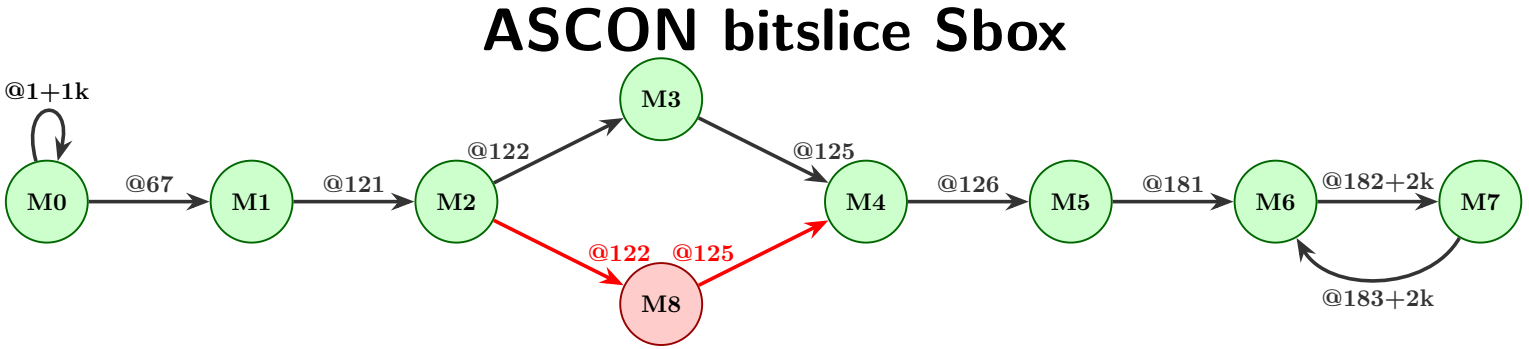


Refining with Reinforcement Learning

- RL policy $\pi_\theta(a \mid x)$ on top of SFT model
- Reward $r(x, a) = \begin{cases} p(a \mid x) & \text{if correct} \\ -p(a \mid x) & \text{if wrong} \end{cases}$
- Results: SFT 79% $\rightarrow$ SFT+RL 90% on Case 1



Trace-Driven Fault Pattern Analysis (GNN)



- **Dynamic State Transition Graph**
  - Node `M2` has neighbors $\{M1, M8/M3\}$
  - Node `M8` has neighbors $\{M2, M4\}$
- **Graph** $\mathcal{G} = (V, E)$ $V = \{\text{micro-architectural states}\}$, $E = \{\text{state transitions}\}$
- **Generic GNN update (layer l):**

$$h_i^{(l)} = \sigma\Big(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} \, W^{(l)} \, h_j^{(l-1)} \Big)$$

Trace-Driven Fault Pattern Analysis (GNN)

- **Graphlevel embedding:**

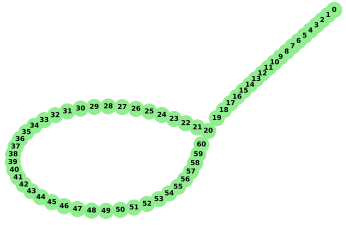
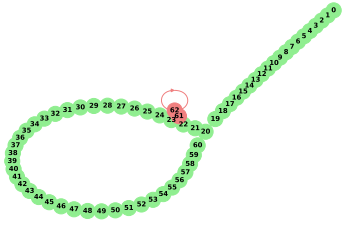
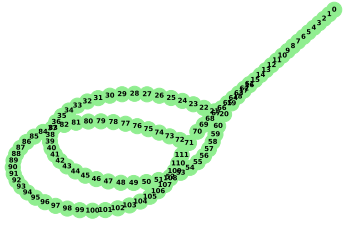
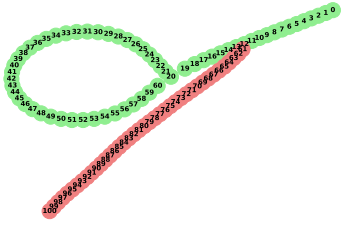
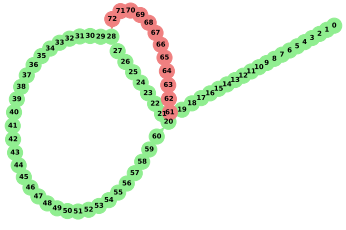
$$z = \mathrm{Pool}(\{ h_i^{(L)} : i \in V \})$$

- **Classifier:**

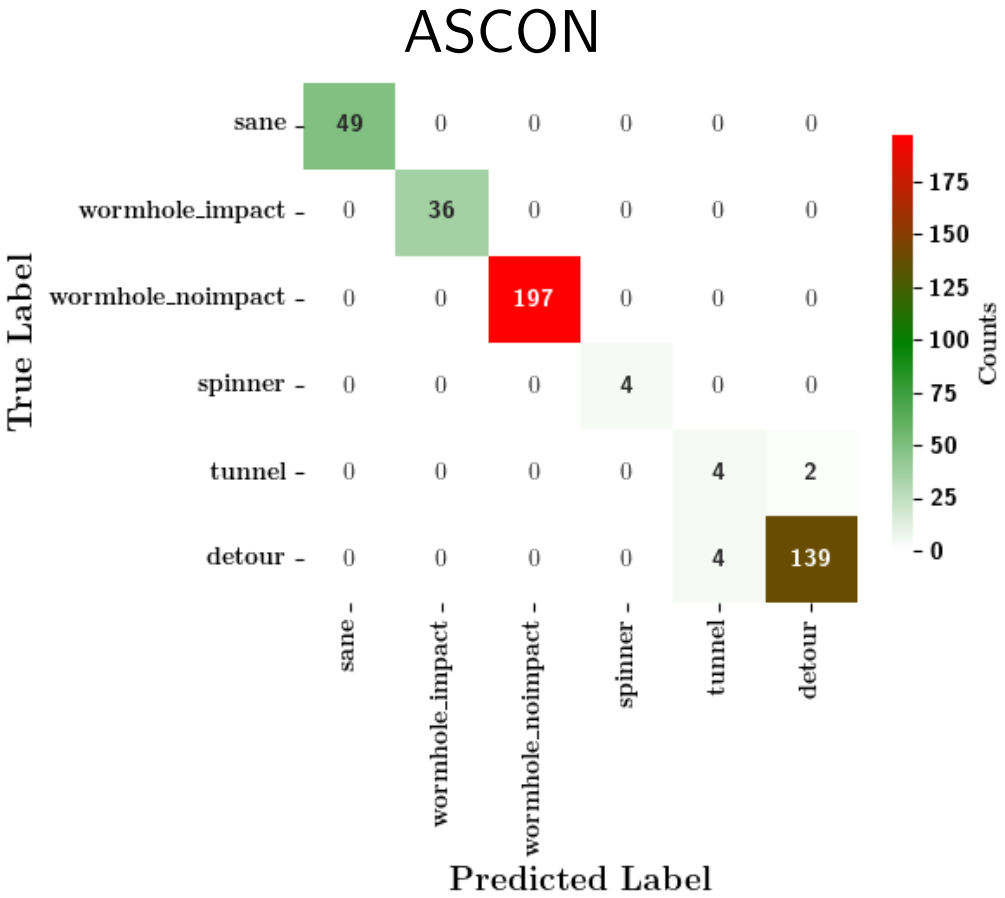
$$\mathrm{MLP}(z) : z \longmapsto \left\{ \begin{array}{l} \text{Sane, Spinner, Tunnel,} \\ \text{Wormhole\_impact, Detour,} \\ \text{Wormhole\_noimpact} \end{array} \right\}$$

- **Training:** cores 0, 1, 2, 4, 5
- **Test:** core 3

Fault Patterns

-  Sane: Normal execution sequence
- Spinner: Stuck in infinite loop, 
-  Tunnel: Skipped instruction transitions
- Wormhole: Abrupt state transition jump, 
-  Detour: Temporary control flow deviation

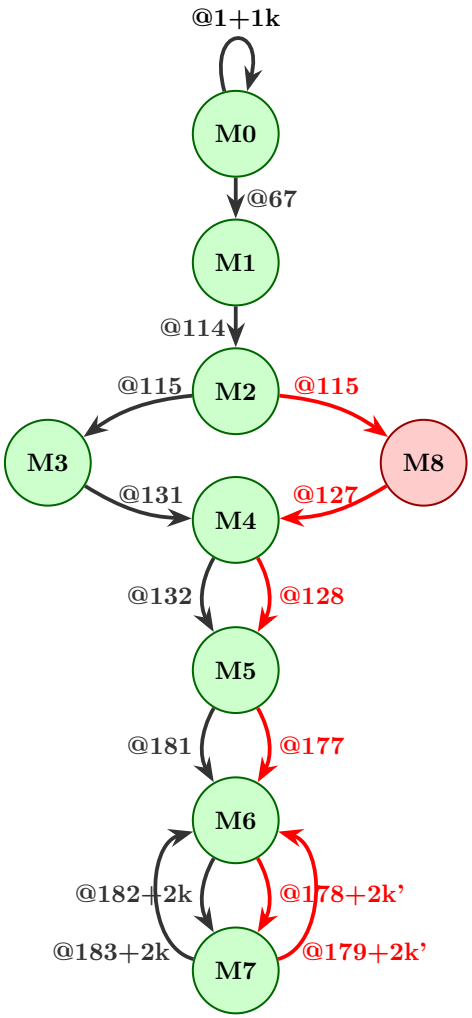
GNN Classification Results



Injection	Firmware	Train	Test	Cycles	GNN Acc.
Laser	ASCON	2171	435	89	98.62%
Glitch	ASCON	480	96	89	96.88%

Mapping Fault Patterns to CWEs

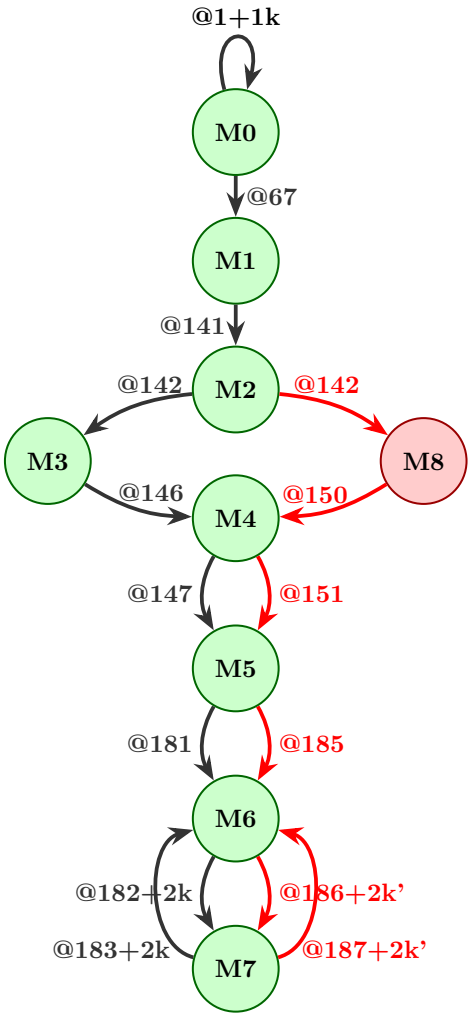
Repeated fault patterns aligns with CWE
Exposing control-flow vulnerabilities



PC Advance

- Force program counter to skip three instructions
  f0ac →f0b4
- Data register holds previous value missing sbox update.
- **Tunnel (PC advance):**
  CWE-1332 Skipped Valid Transitions

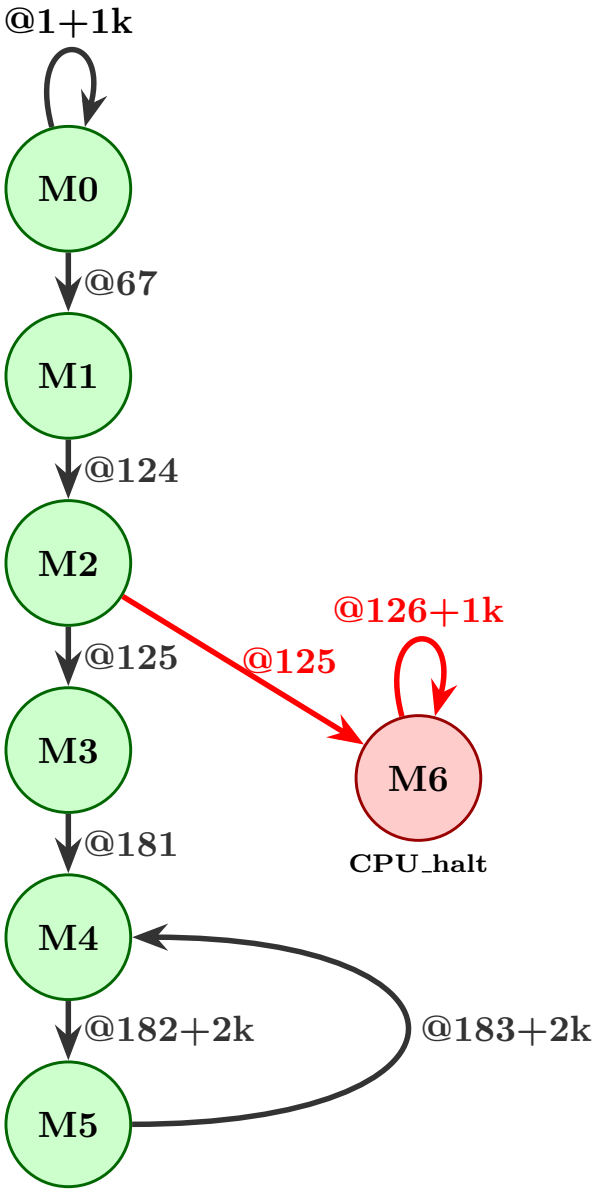
Contd,.



PC Lag

- Force program counter to re-execute instructions **f0b4 →f0b0**
- Results in repeated execution of instructions, SIFA.
- **Tunnel :** CWE-691 Insufficient Control-Flow Management

Contd,.



CPU halt

- Reset forcing PC registers to zero.
- **Spinner :** CWE-835 Loop with Unreachable Exit Condition

Summary

- **Full visibility** via cyclebycycle scan chains
- **LLM + QLoRA + RL**: highaccuracy state classification
- **GNN**: robust multicycle pattern detection
- **CWE mapping**: standardize hardware fault categories

Thank you for your attention!

dshanmugam@wpi.edu
zliu12@wpi.edu
pschaumont@wpi.edu

References

- Fault Detective : Explainable to a Fault, from the Design Layout to the Software : TCHES 2024
- Weird machines, exploitability, and provable un exploitability Thomas Dullien
- The Forgotten Threat of Voltage Glitching: A Case Study on Nvidia Tegra X2 SoCs : FDTC
- Glitched on Earth by Humans: A Black-Box Security Evaluation of the SpaceX Starlink User Terminal : Black hat 2022
- 1 in 1000 chips produce Silent Data Corruptions: https://www.sigarch.org/sdcs-a-b-c/

N/W Architecture & Hyperparameters

LLM Fine-Tune Hyperparameters

Hyperparameter	Value
EPOCHS	4 / 16
Learning Rate	1×10^{-5} / 2×10^{-5}
LoRA Rank	4 / 8 / 16
LoRA Alpha	64 / 128
LoRA Dropout	0.1 / 0.2
Batch Size	4 / 8 / 16
Warmup Steps	50 / 200
Threshold	0.48 / 0.5 / 0.52

Graph Transformer (GAT) Hyperparameters

Parameter	Value
`input_dim`	5
`node_features`	{deg, cluster, node_labels, cyc_paths}
`hidden_layer`	64
`output_dim`	5
`num_layers`	2
`heads`	4
`dropout`	0.25

Scan-chain Values by Clock Cycle

Clockcycles	Clock	SFR	Frontend	Executionunit	dbg	cpu_status	Classification
1	1	1	0	0	1	0	Sane
2	1	1	0	0	1	0	Sane
3	1	1	0	0	1	0	Sane
.	.	.	.	.	.	.	.
27	1	1	0	0	1	0	Sane
28	1	1	0	0	1	0	Sane

Scan-chain Values by Clock Cycle

Clockcycles	Clock	SFR	Frontend	Executionunit	dbg	cpu_status	Classification
1	1	1	0	0	1	0	Sane
2	1	1	0	0	1	0	Sane
3	1	1	0	1	1	0	Weird
.	.	.	.	.	.	.	.
27	1	1	0	0	1	0	Sane
28	1	1	0	0	1	0	Sane

- **FSMRed : 6 cores * 28 clock cycles * 660 bits**

Embedding & Positional Embedding

- **Input State:** For simplicity, consider a 6-bit state: 110110.
- **Tokenization:** Split into two 3-bit tokens:

$$\text{Token 1: } 110, \quad \text{Token 2: } 110.$$

- **Token Embedding:** Assume token 110 is assigned

$$e_{\text{token}} = [0.5, 0.8].$$

- **Positional Embedding:** For its position, add

$$e_{\text{pos}} = [0.1, 0.2].$$

- **Final Embedding:**

$$e = e_{\text{token}} + e_{\text{pos}} = [0.6, 1.0].$$

Self-Attention

- **Assume Two Token Embeddings:**

$$t_1 = [0.6,\ 1.0], \quad t_2 = [0.2,\ 0.4].$$

- **Projection (for simplicity):**

$$Q = K = V = t_i.$$

- **Dot Product:**

$$QK^T = \begin{bmatrix} 0.6 \cdot 0.6 + 1.0 \cdot 1.0 & 0.6 \cdot 0.2 + 1.0 \cdot 0.4 \\ 0.2 \cdot 0.6 + 0.4 \cdot 1.0 & 0.2 \cdot 0.2 + 0.4 \cdot 0.4 \end{bmatrix} = \begin{bmatrix} 1.36 & 0.52 \\ 0.52 & 0.20 \end{bmatrix}.$$

- **Scaling:** Divide by $\sqrt{2}$ (since $d_k = 2$):

$$\frac{QK^T}{\sqrt{2}} \approx \begin{bmatrix} 0.962 & 0.367 \\ 0.367 & 0.141 \end{bmatrix}.$$

- **Softmax (row-wise):**
  - Token 1: $\mathrm{softmax}([0.962, 0.367]) \approx [0.644, 0.356]$
  - Token 2: $\mathrm{softmax}([0.367, 0.141]) \approx [0.556, 0.444]$
- **Attention Outputs:**

$$\mathrm{Output}_1 = 0.644\, t_1 + 0.356\, t_2 \approx [0.4576,\ 0.7864],$$
$$\mathrm{Output}_2 = 0.556\, t_1 + 0.444\, t_2 \approx [0.4224,\ 0.7336].$$

Pooling, Logits & Inference

- **Attention Outputs (from previous):**

$$\text{Output}_1 = [0.4576,\ 0.7864], \quad \text{Output}_2 = [0.4224,\ 0.7336].$$

- **Pooling:** Mean pooling

$$\text{pooled} = \frac{\text{Output}_1 + \text{Output}_2}{2} = \left[\frac{0.4576 + 0.4224}{2},\ \frac{0.7864 + 0.7336}{2}\right] = [0.44,\ 0.76].$$

- **Logits Computation:** Linear layer $y = \text{pooled} \cdot w + b$, with $w = [1.0, -0.5]$, $b = 0.2$:

$$y = 0.44 \times 1.0 + 0.76 \times (-0.5) + 0.2 = 0.44 - 0.38 + 0.2 = 0.26.$$

- **Probability Conversion:**

$$p(\text{weird} \mid x) = \sigma(0.26) \approx 0.565.$$

- **Inference:** With threshold 0.5, classify as *weird*.

▸ Next: QLORA

Dataset

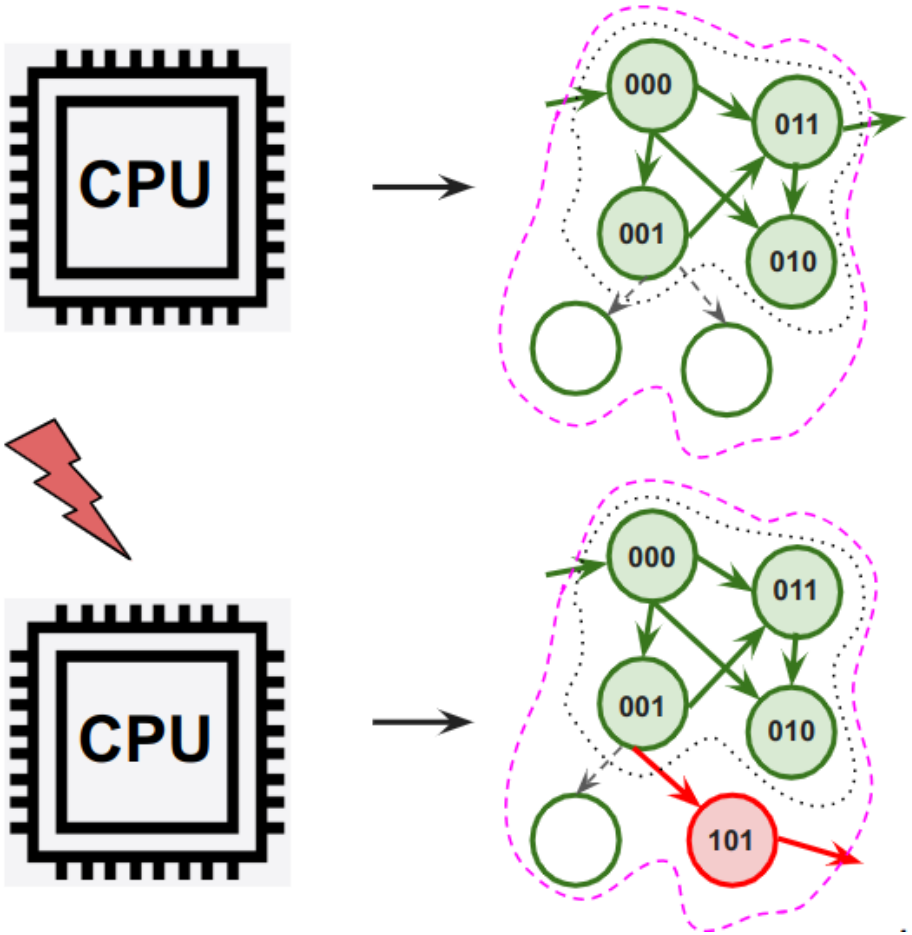
Processors	Arch. Reg	μarch. Reg	Firmware	Sane	Weird
MSP430	385	275	FSMRed	28	627
			ASCON Sbox	89	3120
			PinVerify5	248	22576
PICO	1024	1265	Random Instr.	80	480
			ASCON Sbox	24	350
IBEX	993	1013	Random Instr.	486	1013
			ASCON Sbox	402	1000
Post-Silicon (CAPRI6)			MOV/ADD/MUL	517	517

Threat Model

A 1000-bit processor has 2^{1000} possible statesimpossible to exhaustively model.

Intended FSM: sane states only

Unintended FSM: weird states emerge under faults



$$\textbf{model}(\alpha) = \big((\textit{Fault}(\theta),\ \textbf{ProcessorISA}(\gamma),\ \textbf{WeirdState}(\beta)\big)$$

$$\beta_{\textbf{weird}} = \big(Q_{\textbf{cpu}}^{\textbf{weird}},\ q_{\textbf{init}},\ Q_{\textbf{cpu}}^{\textbf{IFSM}},\ \Sigma',\ \Delta',\ \delta',\ \sigma'\big)$$

Concept of weird states from Thomas Dullien, http://www.dullien.net/thomas/weird-machines-exploitability.pdf