# Fast, Robust and Accurate Detection of Cache-based Spectre Attack Phases

Arash Pashrashid, Ali Hajiabadi, Trevor E. Carlson

*National University of Singapore*

*41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22)*

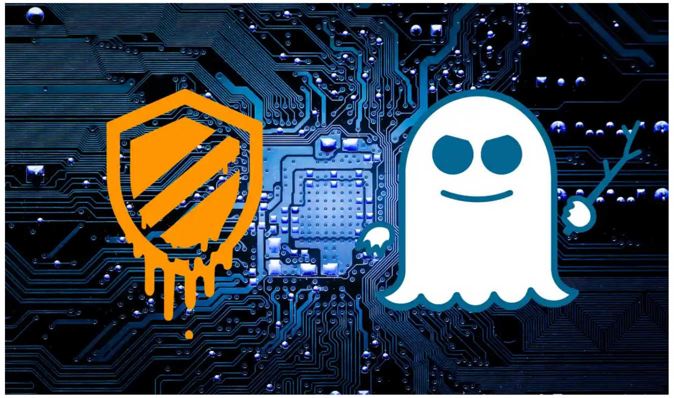
# Breaking News: Fundamental Security Problem of Modern CPUs

- Computer architects' main focus was on CPU performance for decades
- However, modern CPUs can leak sensitive data like passwords, cryptographic keys via unexpected side channels
- Affects all modern processors, servers, smart phones
  - Intel, AMD, ARM, IBM, etc.
  - Affects all operating systems

**Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers** [1]

**Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors – and fix could slow devices**

**Spectre and Meltdown processor security flaws – explained**

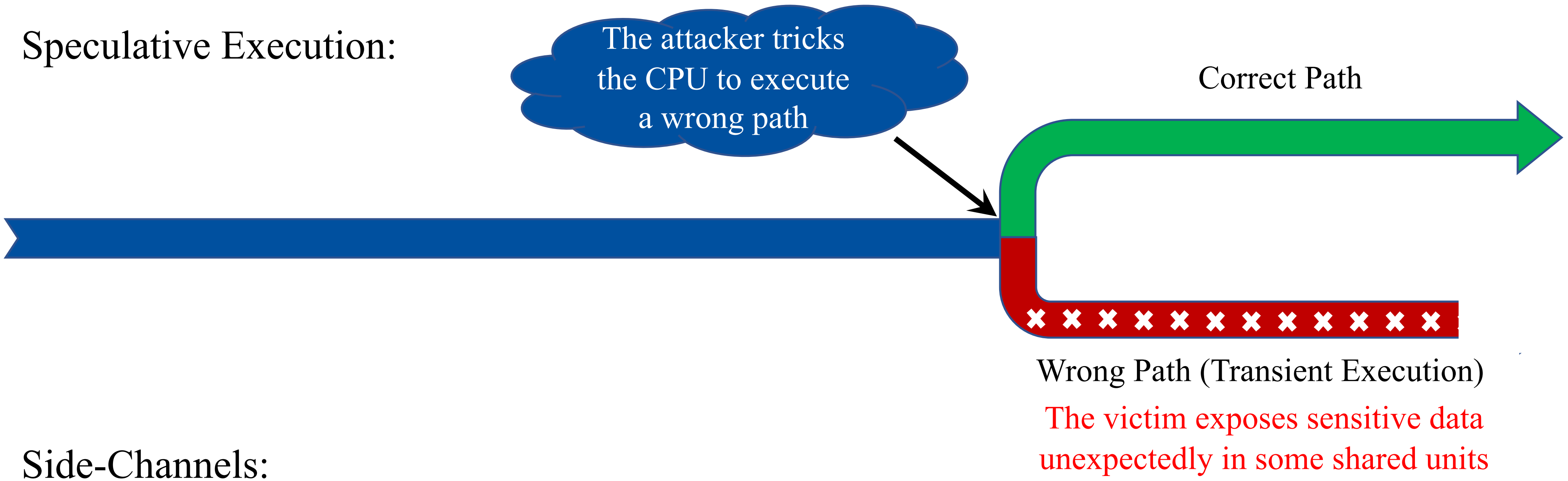


Meltdown and Spectre security flaws: so big they have their own logos. Photograph: tcareob72/Natascha Eibl/Getty Images/iStockphoto

1. https://www.theguardian.com/technology/2018/jan/04/meltdown-spectre-computer-processor-intel-security-flaws-explainer
2. https://arstechnica.com/gadgets/2018/01/meltdown-and-spectre-every-modern-processor-has-unfixable-security-flaws/

# Recipe for Spectre Attack

- Speculative Execution + Side-Channels
  - Speculative Execution:




  - Side-Channels:
    - Attacker analyzes these channels to extract victim's secret dependent activities
    - For example, last level cache is one of the common side-channels

# Example: Speculative Execution Attack via Cache

1. Initialization → 2. Victim Execution → **3. Probe**

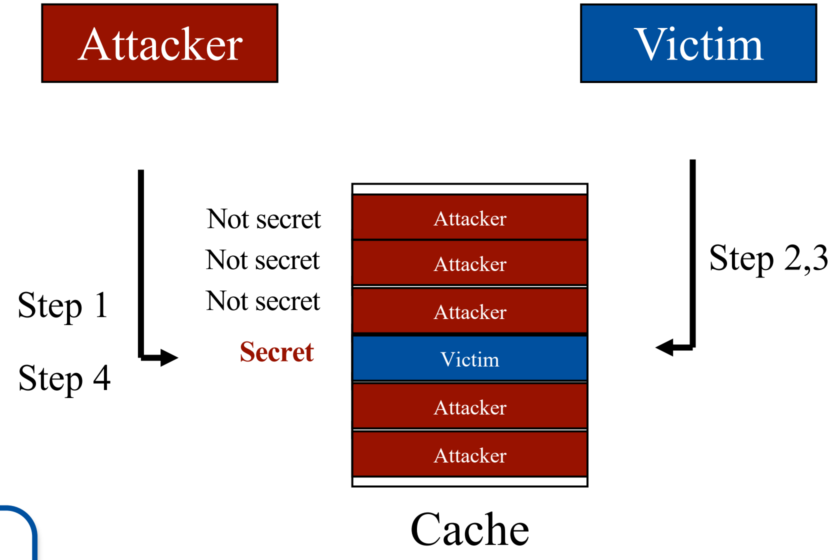
```
Branch_mistraining()
prime_µarch_state()

 if (safety check)
   secret = load(…)
   side-channel(secret)

probe_µarch_state()
```

1. Initializes the system for attack
2. Misspeculation
3. Load Secret — Speculative Access

Secret-dependent trace in the cache

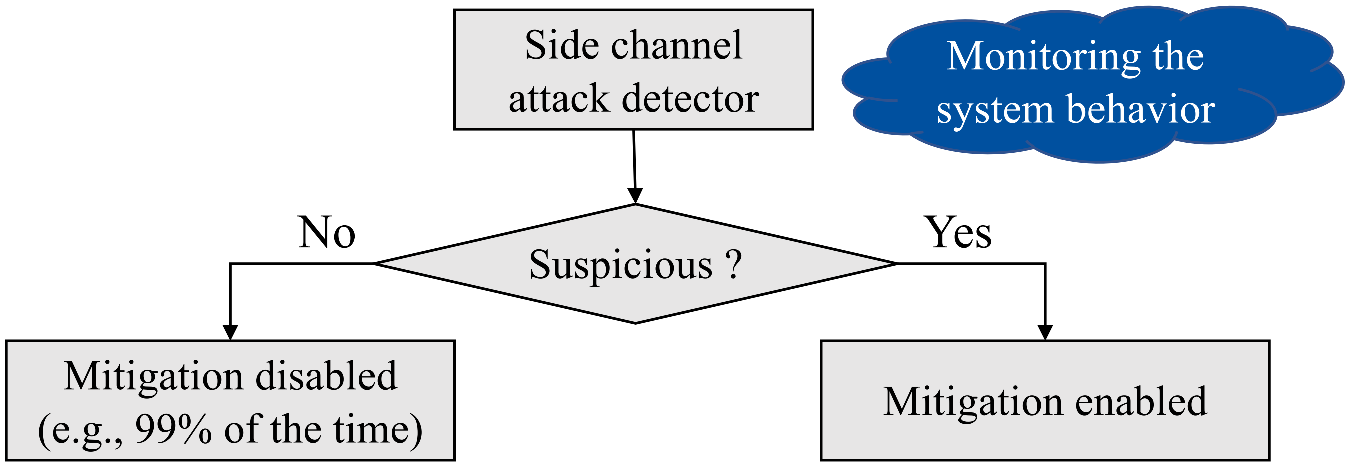
e.g., access specific cache lines based on the secret value

4. Recovering the Secret — e.g., load latency



# Why Using Side-Channel Attacks Detectors?

- Problem: Comprehensive Spectre mitigation incurs significant performance overhead (Up to 2x)[1,2]
- Not all these overheads are necessary to provide the secure system
- One possible solution: If not attacks are present, no expensive mitigations needed
- Our goal: Addressing the limitations of existing detectors






1. Loughlin, Kevin, et al. "{DOLMA}: Securing Speculation with the Principle of Transient {Non-Observability}." 30th USENIX Security Symposium (USENIX Security 21). 2021.
2. Weisse, Ofir, et al. "NDA: Preventing speculative execution attacks at their source." Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. 2019.

# Insight: Is Machine Learning the Solution?

- The ideal detector should be **fast, accurate**, **robust** and **efficient**
- Machine learning is widely deployed for the SCA detectors[1,2,3]

Question: Are ML-based methods robust to evasive attacks or benign applications?

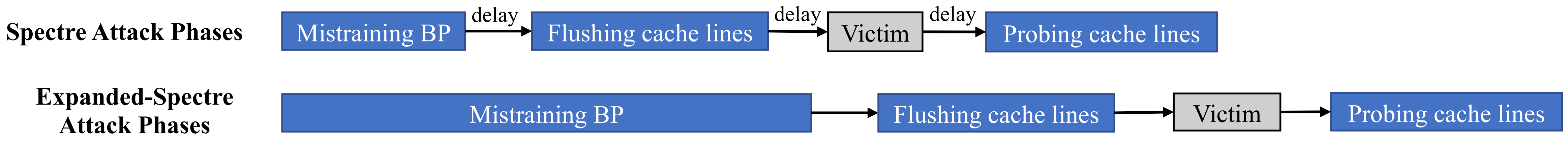
- In this work, we propose the evasive attacks to break ML-based detectors

1- Nguyen, Luong N., et al. "Creating a backscattering side channel to enable detection of dormant hardware trojans." *IEEE transactions on very large scale integration (VLSI) systems* (2019)
2- Mirbagher-Ajorpaz, Samira, et al. "Perspectron: Detecting invariant footprints of microarchitectural attacks with perceptron." IEEE/ACM International Symposium on Microarchitecture (MICRO). 2020
3- Mushtaq, Maria, et al. "WHISPER: A tool for run-time detection of side-channel attacks." IEEE Access 8 (2020): 83871-83900.

# Evasive Spectre Attacks

**Expanded-Spectre** | Benign-Program-Spectre

- **Goal:** To change the footprint of a Spectre attack without compromising the attack's success
- **How:** Expanding the branch mistraining without disrupting the attack's success



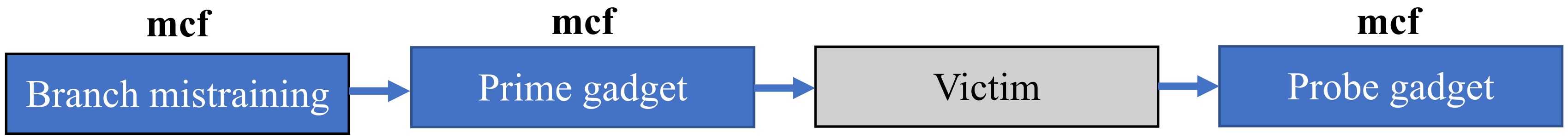

- Two variants of this attack:
  - insertion of NOPs to the branch mistraining part of the original Spectre
  - insertion of memory delay instructions
- Accuracy of PerSpectron, the SOTA ML-Based detector, drops from 99% to 14%

# Evasive Spectre Attacks

Expanded-Spectre | **Benign-Program-Spectre**

- **Goal:** Performing all the essential steps of attack from benign programs
- **How:** Finding similar behavior inside benign programs for each step

1. Branch mistraining (Attacker): A loop with a large number of iterations.
2. Side-channel initialization phase (Attacker): Initialization of a large array
3. Secret recovering phase (Attacker): Same with Phase 2
4. By linking the selected slices that represent each step, a full attack can be launched



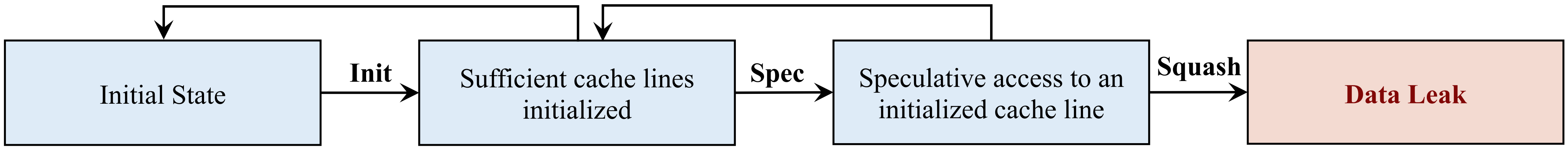

- Accuracy of PerSpectron drops from 99% to 12%

# Motivation

- Limitations of state-of-the-art ML-based detectors:
  - They are fragile to our Expanded-Spectre
  - Also, they can be fooled by our Benign-Program-Spectre
- We need to design an SCA detector to overcome these shortcomings
  - To be ***robust*** to our evasive Spectre attacks
  - And to be ***accurate, fast***, and ***efficient***
- We design Spectify to get closer to an ideal detector

# Spectify Detection Methodology

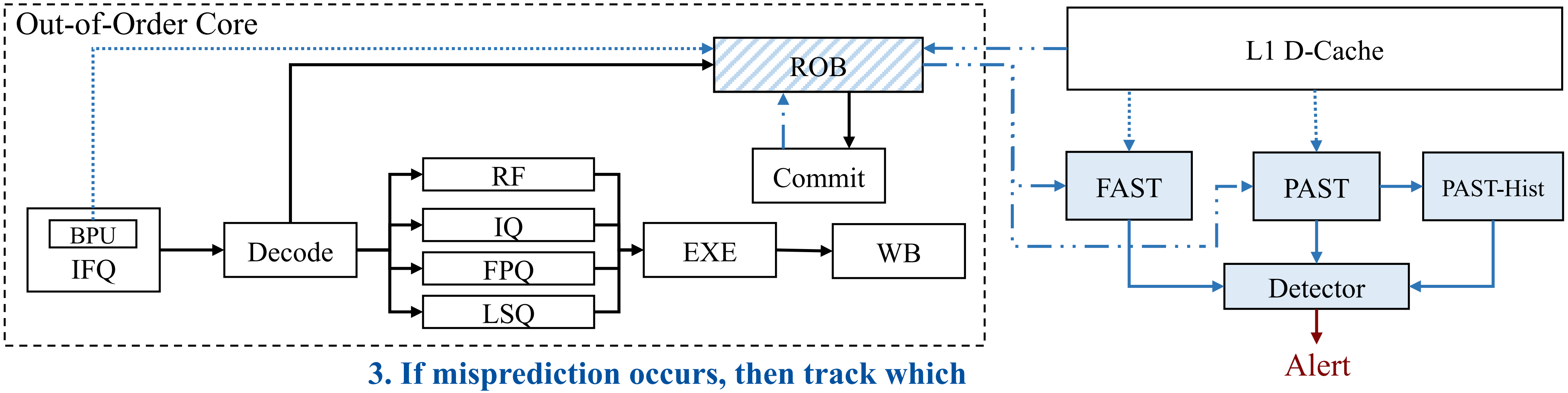
- We aim to track the sequence of attack phases
- Using a direct-analysis approach to monitor microarchitectural state changes
- **Init transition:** If enough number of cache lines are initialized
- **Spec transition:** If a sufficient number of cache lines are initialized by previous processes and the current process speculatively accesses one of initialized cache lines
- **Squash transition:** If misprediction happens and the state of only one of the initialized cache lines is changed



# Spectify Microarchitecture

**1. Modifying the ROB to trach unresolved branches**

**2. Adding new tables (PAST and FAST) to track the accessed/flushed locations**




**3. If misprediction occurs, then track which locations accessed during speculation window**

**4. If context-switch occurs, then the new tables checked for a data leak. Also, taking a checkpoint before CS in a new table (PAST-Hist)**

# Experimental Setup

- Simulation:
  - gem5 in syscall emulation mode
  - CACTI 6.5 for power and area overheads
- Benchmarks:
  - **Benign programs:** SPEC CPU2006 benchmark suite
  - **Malicious programs:** Spectre V1, Spectre V2, different cache attacks, and our evasive Spectre attacks
  - **Representatives:** ELFies as executable representative with a region size of 100M instructions
- PerSpectron Experimental Setup
  - FANN C library for the implementation of neural networks
  - 10k instruction sampling rate
  - Single-layer perceptron neural network
  - 66% of the data is used for training and the rest for testing

# Comparison of PerSpectron and Spectify

- Both PerSpectron and Spectify show high accuracy for Benign, Spectre V1 and V2
- While PerSpectron accuracy falls from 99% to 14% for Expanded-Spectre attack, there is no accuracy reduction for Spectify
- While PerSpectron accuracy falls from 99% to 12% for Benign-Program-Spectre attack, there is no accuracy reduction for Spectify
- Even retraining PerSpectron with our evasive Spectre doesn't give acceptable accuracy to the PerSpectron
- Even the false positive rate in Spectify is less than PerSpectron and is around 0.02%

| | *PerSpectron* | *Spectify* | |
|---|---|---|---|
| **Test Scenario** | **Accuracy** | **Avg. #Alerts** | **Accuracy** |
| Benign | 99.10% | 11 | 99.98% |
| Spectre V1 | 99.61% | 1 | 100.0% |
| Spectre V2 | 98.67% | 1 | 100.0% |

# Running Spectify with SPEC CPU2006

Number of times at least 2 sets are primed

Actual memory data leaks that potentially can be exploited

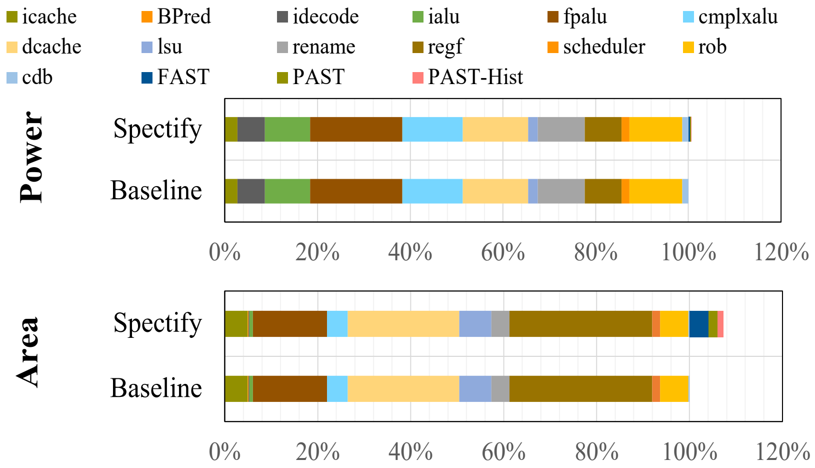
| Application | #frames | #min 2 sets primed | #Data Leaks |
|---|---|---|---|
| 401.bzip2 | 38136 | 6667 | 4 |
| 403.gcc | 151771 | 53186 | 11 |
| 410.bwaves | 55255 | 46278 | 3 |
| 416.gamess | 26720 | 1205 | 10 |
| 429.mcf | 217673 | 106053 | 52 |
| 434.zeusmp | 32763 | 19760 | 40 |
| 436.cactusADM | 60729 | 7407 | 0 |
| 444.namd | 277321 | 244 | 0 |
| 445.gobmk | 48742 | 4074 | 1 |
| 450.soplex | 128519 | 39411 | 10 |
| 462.libquantum | 72327 | 26315 | 0 |
| 471.omnetpp | 85982 | 22715 | 2 |

Demonstrates the possibility of initialization for Benign-Program-Spectre from the SPEC programs

Demonstrates that our Benign-Program-Spectre is possible

# Efficiency Analysis of Spectify

- No performance overhead: Operates in parallel with the main processor core, off the critical path
- Power overhead: 0.66% over the baseline core
  - Most overheads come from FAST, PAST, and PAST-Hist
  - Direct-mapped cache structures are relatively efficient
- The area overhead: 7.3% over the baseline core



# Conclusion

- We break the state-of-the-art detector, PerSpectron, by our evasive Spectre
  - Expanded-Spectre
  - Benign-Program-Spectre
- We propose a new detector to satisfy ideal detector conditions
  - 100% accuracy for our tested applications ✓
  - Detection before attack completion ✓
  - Robust to our evasive Spectre attacks ✓
  - No performance overhead, 0.66% power overhead, and 7.3% area overhead ✓

*Thanks for your attention!*

NUS National University of Singapore

# Fast, Robust and Accurate Detection of Cache-based Spectre Attack Phases

Arash Pashrashid, Ali Hajiabadi, Trevor E. Carlson

*National University of Singapore*

*41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22)*