APRIL 2024

TEXAS
The University of Texas at Austin

LEARNING-BASED DETECTION OF MICROARCHITECTURAL ATTACKS

ASPLOS 2024 tutorial

Mulong Luo, Ayush RoyChowdhury, Mohit Tiwari

https://ut-ldma.github.io

SPARK Lab
The University of Texas at Austin
https://spark.ece.utexas.edu

Topic to be Covered

- What are microarchitectural attacks?
- Why and how to prevent and detect attacks?
- Why use learning-based detection?

Organizers

- Dr. Mulong Luo, postdoc researcher, ECE, UT Austin
- Ayush RoyChowdhury, senior undergrad, ECE, UT Austin
- Prof. Mohit Tiwari, associate professor, ECE, UT Austin
- Technical Contributors
  - Austin Harris, PhD candidate, ECE, UT Austin
  - Shijia Wei, PhD candidate, ECE, UT Austin
  - Prateek Sahu, PhD candidate, ECE, UT Austin
  - Jiaxun Cui, PhD candidate, CS, UT Austin
  - Prof. Wenjie Xiong, ECE, Virginia Tech
  - Prof. Edward Suh, Meta / Cornell University
  - Dr. Yuandong Tian, Meta AI

Dealing with Attacks in HW

- Attack is from system imperfection
  - Cache timing attacks
  - Spectre/Meltdown
- Two approaches
  - **Attack prevention**
    - design the HW or SW so that the attack is not feasible
  - **Attack detection**
    - Allow the attack to happen but build detectors to alarm the attack, and perform the corresponding responses
  - Both can be improved with **machine learning**

Structure of the Tutorial

Microarchitecture attacks and defense

ML algorithms for microarchitecture attack detection

Architecture and infrastructure for ML detection

Contest for ML detection

Timeline

time	content
1:30-1:35	Opening remarks
1:35-2:10	Microarchitecture attacks and defenses
2:20-3:00	Machine learning algorithms
3:00-3:30	Coffee breaks
3:30-4:00	Machine learning-based microarchitecture architectures
4:00-5:00	Mini lab on reinforcement learning-based microarchitecture attack detection

APRIL 2024

TEXAS
The University of Texas at Austin

MICROARCHITECTURE ATTACKS AND DEFENSES

ASPLOS 2024 tutorial

https://ut-ldma.github.io

MULONG LUO
Postdoctoral researcher
SPARK Lab, The University of Texas at Austin
https://spark.ece.utexas.edu

Security Attacks

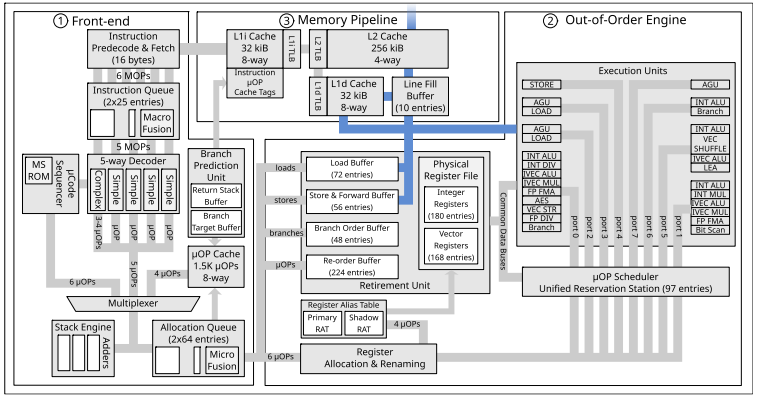
- Vulnerability is everywhere
  - Stuxnet: nuclear power plant
  - Cambridge Analytica: social networks
  - Log4j: web infrastructure
  - Spectre/Meltdown: computer hardware
- Huge dollars spent
  - 2.5 trillion USD = GDP of UK (5th largest country)
  - finding and resolving vulnerabilities

Source: fair institute

Microarchitectural Attacks

- adversaries exploit the microarchitecture vulnerabilities in microprocessors
- steal information, damage the information integrity, makes the processor unavailable

- Examples:
  - Cache side channel attacks
  - Speculative execution attacks
  - RowHammer attacks



OOO architecture

Meltdown/Spectre

Side Channel Attacks

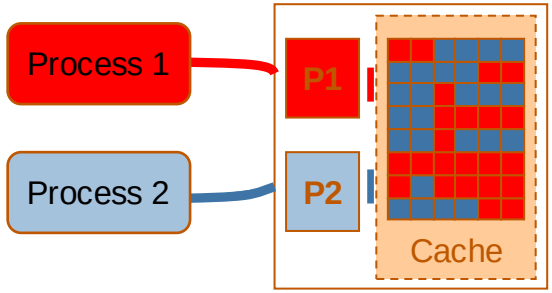
- What is a side channel attack
  - Side channel attack
    - Secret stealing via peripheral information (without directly accessing the information)
    - “peripheral information”: information that is not a direct representation of the secret
  - Examples:
    - Guessing the age of a person
    - In computer science domain:
      - Secret: a piece of information (bits) potentially in different format and physical residence (storage device, memory, register, etc)
      - Peripheral information: any information that is not in the original representation AND format, or encodings not known to the observers
        - » E.g., power consumption
        - » E.g., cipher text

Side Channel vs Covert Channel

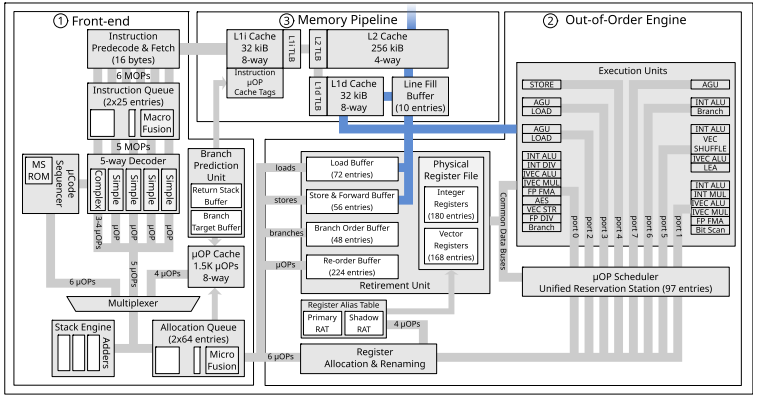
- Similarities
  - Both involve two parties
    - Attacker/victim, sender/receiver
  - Can usually leverage the same mechanism
    - E.g., cache side channel vs cache covert channel
- Differences
  - Side channel: none cooperative, cover channel: cooperative
  - Covert channel is usually used to measure characterize the bandwidth and error rate of a channel
- Used interchangeably in many cases

How uarch side channel is created?

- Resource sharing
  - Cache, TLB, BTB, etc.
- Design optimizations
  - OOO executions



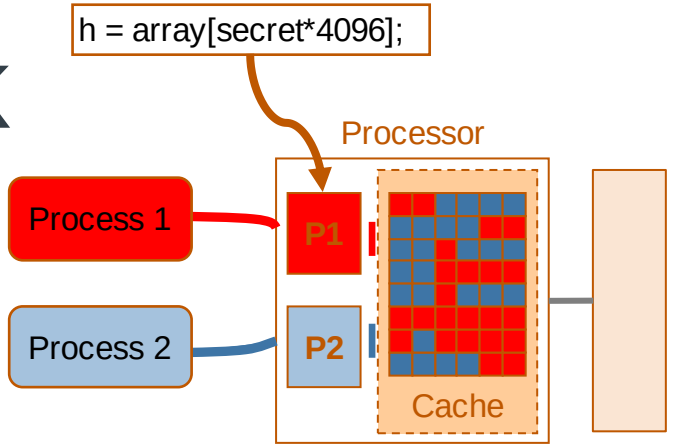
Cache sharing

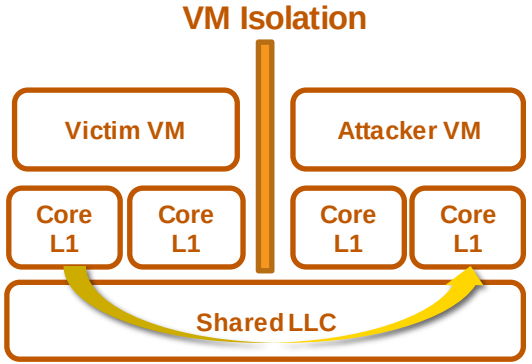


OOO architecture

Cache-Timing Attack

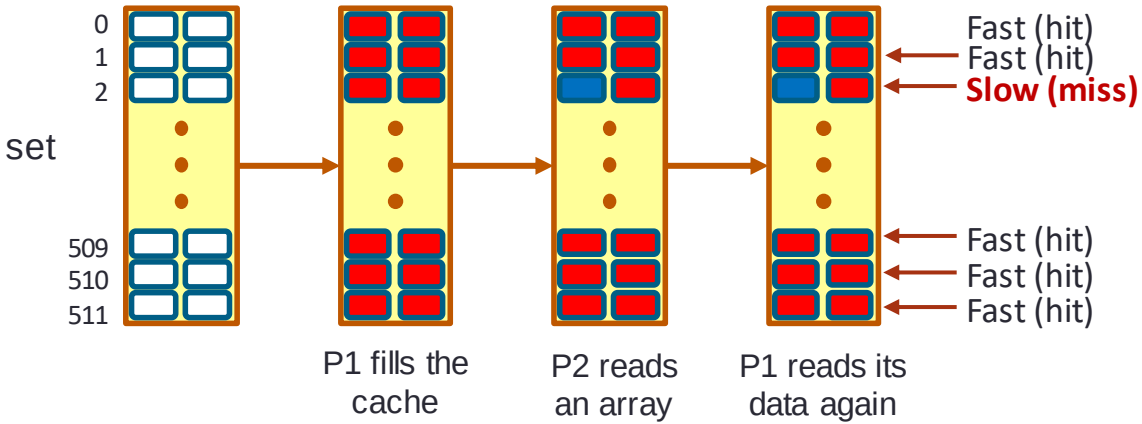
- Mechanism
  - sharing of caches by different processes
  - infer secret by observing cache timing
- Advantages
  - attacker is just a program, no physical access
  - does not violate any OS-level access control
- Leak important assets
  - cryptographic keys
  - VM/browser isolation
  - building blocks for Spectre/Meltdown





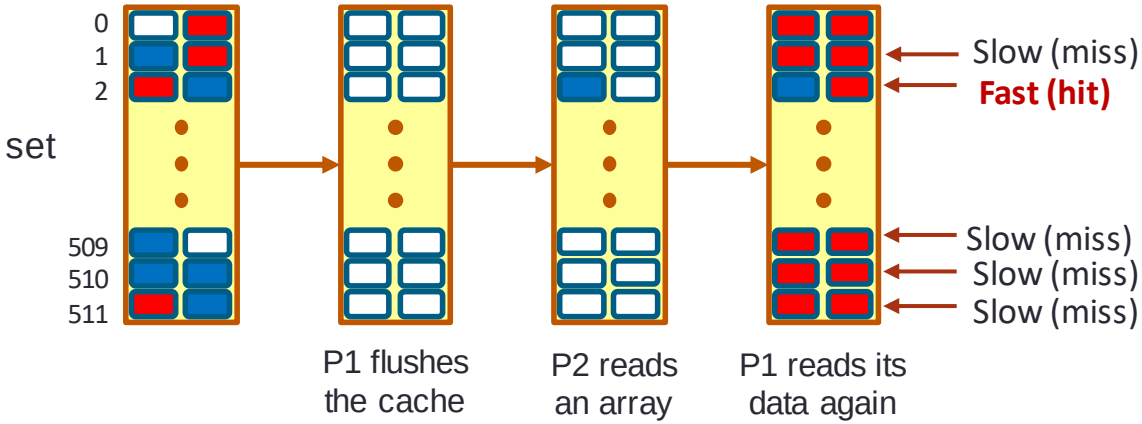
Prime+Probe Covert channels

- P1 and P2 have different address space



Flush+Reload covert channels

- P1 and P2 share the same address space

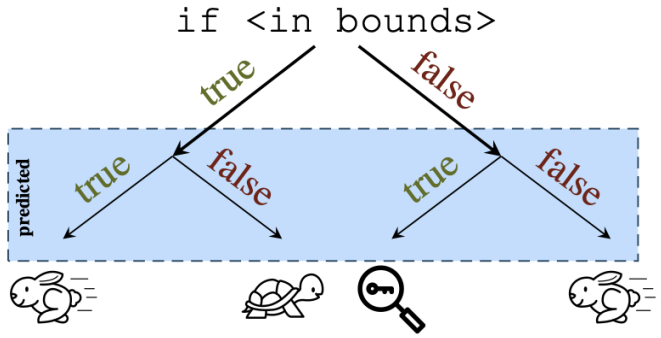


Speculative/Meltdown

- Idea:
  - OOO execution may access secret by passing the domain isolation
  - OOO does not leave architectural traces
  - OOO's microarchitecture impact can be committed and used to extract the secret

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

Listing 1: Conditional Branch Example



Dealing with Attacks in HW

Strategy	Approach	How ML can help
Attack prevention	Strategies: randomization, isolation architecture Tools: Fuzzing and formal verification	Improving the coverage and find hard-to-discover attacks
Attack detection	HW architecture for detection and response	Improve the detection rate

Defense Strategies

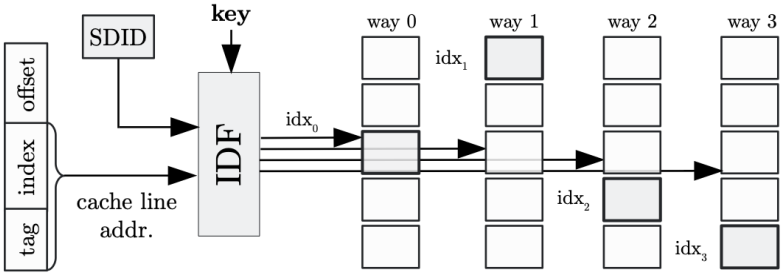
- Partition
  - Separate different domains, eliminate interference
- Randomization
  - Randomize the actual interference, making it hard to guess the secret based on interference

Cache Partition

- Form:
  - Way partition, set partition.
- Static partition: eliminate the interference
  - Inflexible to the workload performance needs
- Dynamic partition
  - Adjustable to performance needs of the applications
  - Potential leakage

Randomized Caches

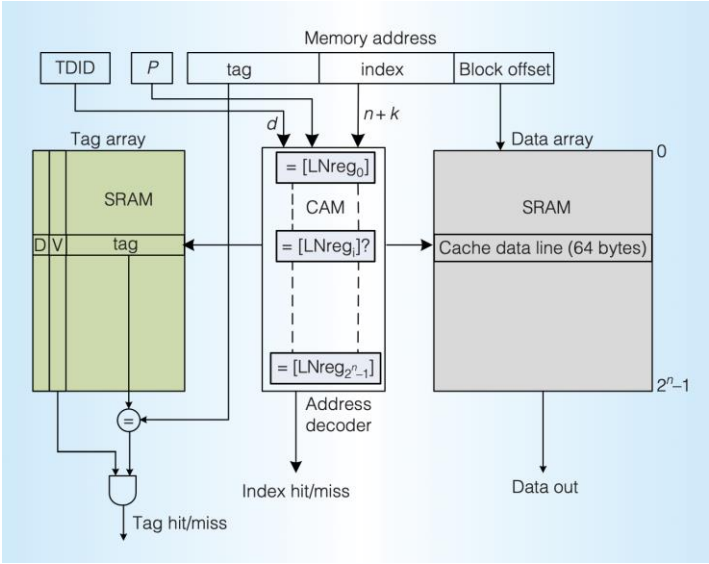
- Forms:
  - Static randomization
  - Dynamic randomization
- Implementation
  - Table-based
  - Cipher-based



ScatterCaches

Cache Randomization: NewCache

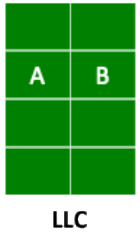
- Table-based solution to specify the randomized mapping
  - Fine-grained control of the randomization
  - Uses lots of storage space
  - Low latency
  - Suitable for L1 cache

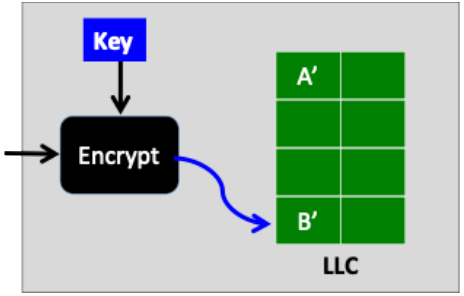


NewCache

CEASER: Cipher-based Randomization with Remapping

- Cipher-based randomization
  - Almost no storage space needed
  - Cipher has high latency
  - Suitable for LLC randomization
- Dynamic Remapping
  - Increase the difficulty for attacker to find the eviction set





CEASE

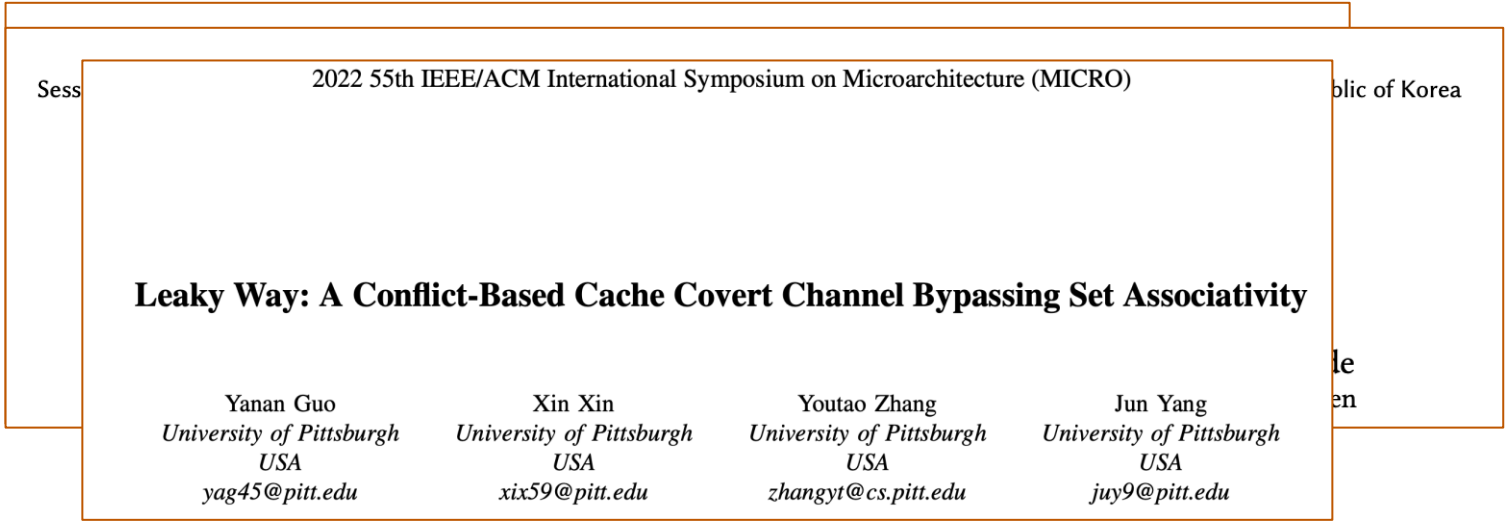
Limitation

- Isolation
  - Potential performance overhead
  - Scalability (millions of domains)
- Randomization
  - Still leaks (occupancy channel)
  - Performance overhead (CEASER, ScatterCache)

Are these schemes secure?

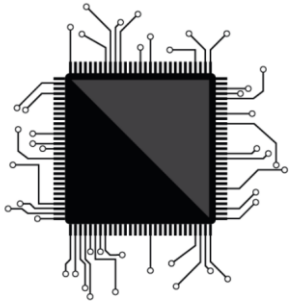
- PLCache
  - A cache set is partitioned for different domains
  - However, metadata can still leak information
- CEASER
  - Originally thought to be secure
  - Later turned out to be vulnerable to new eviction set finding algorithms

Lots of New Attacks!



Why so Many Attacks in Systems?

- System is too complex
  - laptop processors have ~ 20,000,000,000 transistors
- Undefined system behavior
  - timing of a memory read is unspecified
  - speculative execution that are not committed
- Humans are slow and make mistakes
  - can we use **machine intelligence** to replace them?



Meltdown/Spectre

Dealing with Attacks in HW

Strategy	Approach	How ML can help
Attack prevention	Strategies: randomization, isolation architecture Tools: Fuzzing and formal verification	Improving the coverage and find hard-to-discover attacks
Attack detection	HW architecture for detection and response	Improve the detection rate

Question: How are we sure if these schemes are really secure?

- Formal verification
- Fuzzing

FV Works

CheckMate: Automated Synthesis of Hardware Exploits and Security Litmus Tests

Caroline Trippel
Princeton University
ctrippel@princeton.edu

Daniel Lustig
NVIDIA
dlustig@nvidia.com

Margaret Martonosi
Princeton University
mrm@princeton.edu

dzag@cs.cornell.edu cs897@cornell.edu andru@cs.cornell.edu suh@csl.cornell.edu

Formal Verification

- Process
  - Build a spec that models expected behaviors
  - Proof the actual behavior matches the spec in all cases
    - Theorem proving
    - Enumerate all cases
- Limitations
  - Modeling accuracy
  - Scalability
  - Verification of toolchain
  - Theorem proving still need humans

Fuzzing Works

ARTIFACT
EVALUATED

Revizor: Testing Black-Box CPUs against Speculation Contracts

Oleksii Oleksenko*
Christof Fetzer
TU Dresden
Dresden, Germany

Boris Köpf
Microsoft Research
Cambridge, UK

Mark Silberstein
Technion
Haifa, Israel

[1]Worcester Polytechnic Institute, Worcester, MA, USA
[2]Graz University of Technology, Graz, Styria, Austria

Fuzzing

- Taxonomy
  - Mutation-based fuzzing
    - Require previous knowledge of attacks
      - Need to know one type of Spectre to find other types of Spectre
  - Generative fuzzing
    - Require knowing the basic steps
      - Huge search space
  - **ML can help improve search strategy to find vulnerabilities**

Dealing with Attacks in HW

Strategy	Approach	How ML can help
Attack Prevention	Strategies: randomization, isolation architecture Tools: Fuzzing and formal verification	Improving the coverage and find hard-to-discover attacks
Attack Detection	HW architecture for detection and response	Improve the detection rate

Attack Detection – Detect and Response

- Attack prevention disadvantages
  - There are always new attacks, we cannot try to eliminate all of them
  - Prevention comes with run time and hardware/software overhead
- Attack detection advantages
  - Develop response strategies (e.g., fences) when new attack emerges without changing the hardware
  - Only pay for the overhead when needed

Examples of Detect and Response

domains	Enterprise security	Software security	Microarchitecture security
Related work	CrowdStrike Inc.	Morpheus [ASPLOS'19]	EVAX [MICRO'22]
Attack considered	Data breaches, IP theft, etc	Memory attack	Spectre attack
detection	Rule-based detection, ML-based detection	Conditions	Perceptron detector
response	Reboot, data recovery, etc.	Abort/Churn	Insert fences

Detection and Response

- Response – action taken when attack detected
  - Consumes system resources
- Detection accuracy
  - False positive: wasted resource
  - False negative: undetected attack
- **ML can help with improving detection accuracy**

Defense: Runtime Detection

2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

Cyclone: Detecting Contention-Based Cache Information Leaks Through Cyclic Interference

Austin Harris*
austinharris@utexas.edu
The University of Texas at Austin

Shijia Wei*
shijiawei@utexas.edu
The University of Texas at Austin

Prateek Sahu
prateeks@utexas.edu
The University of Texas at Austin

Pranav Kumar
pranavkumar@utexas.edu
The University of Texas at Austin

Todd Austin
austin@umich.edu
University of Michigan

Mohit Tiwari
tiwari@austin.utexas.edu
The University of Texas at Austin

Computer Science and Engineering
Texas A&M University
College Station, USA
elba@tamu.edu

Computer Science and Engineering
University of California, Riverside
Riverside, USA
naelag@ucr.edu

Computer Science and Engineering
Texas A&M University
College Station, USA
djimenez@acm.org

ML for Attack Detection

- Human rules are not comprehensive
- Threshold is arbitrary
- High false positive rate
- Achieves better detection rate: ML
  - How to formulate, what feature to use
  - How to train: RL, unsupervised, supervised

APRIL 2024

TEXAS
The University of Texas at Austin

ML ALGORITHMS FOR ATTACK DETECTION AND PREVENTION

ASPLOS 2024 tutorial

https://ut-ldma.github.io

MULONG LUO
Postdoctoral researcher, The University of Texas at Austin

What is Machine Learning

- Tom Mitchell
  - Machine learning is the study of algorithms that
    - Improve the performance P
    - At some task T
    - With experience E
  - A well-defined learning task is given by < P, T, E>

- Optimization
  - Objective O
  - Parameter P
  - Constraints C

  Many machine learning algorithms can be understood as an optimization over some tasks,

When do we use machine learning

- ML is used when
  - Human expertise does not exist
  - Human can’t explain their expertise
  - Models must be customized
  - Models are based on huge amounts of data

Why ML is Good for Attack Detection?

- Attack traces are huge, hard to develop rules for detection
- Different attacks correspond to different rules
- Lots of attack traces

Taxonomy of ML

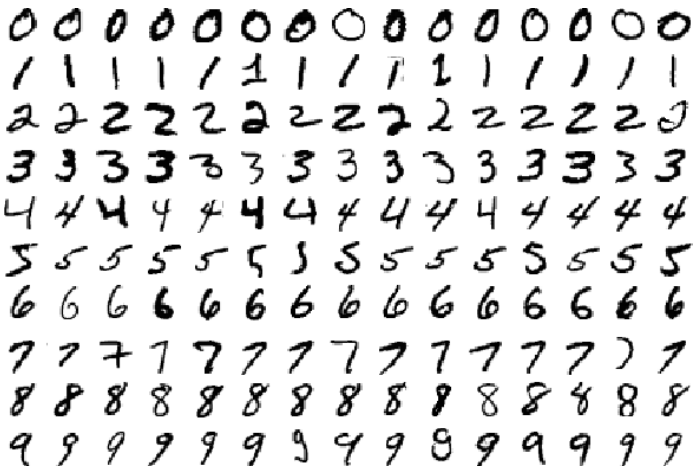
- Supervised learning
  - Regressions
  - Classification
- Unsupervised learning
- Reinforcement learning

Supervised Learning

- Supervised learning refers to the machine learning scenario in which the ground truth labels are presented
  - Literally – learn something with a supervisor
  - The supervisor tells you what is right and what is wrong
  - The most used example when people talks about machine learning (if not specified, it is usually referred to as supervised learning)
- Advantages
  - Learning is efficient
- Disadvantages
  - Needs a supervisor (labeled dataset)
  - Overfitting and bias

Supervised Learning Example

- Hand-written digit recognition



data

0
1
2
3
4
5
6
7
8
9

label

Labeled dataset

Supervised learning: regressions

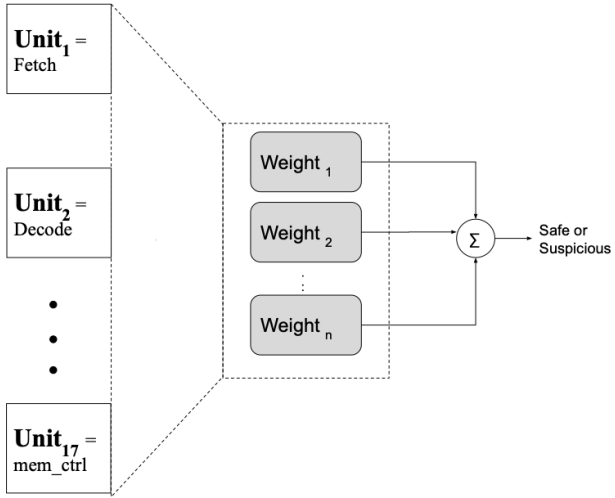
- Regression: predicting a number
  - E.g., learns a model that predicts the temperature at San Diego
- Classification: predict a class/category
  - E.g., learns a model that predicts whether it rains today
- Regression and classification are sometimes covertible
  - e.g., predicting the probability of rain (regression) vs. predicting whether it rains today (classification)
- Run time detection is usually a classification task

Typical supervised learning models (for classification)

- Decision tree/random forest kNN
- Support vector machine
- Neural networks

Supervised Learning in Security

- Side Channel Attack Detection with PerSpectron
  - Data: performance counters
  - Label: Safe/Suspicious



PerSpectron detector

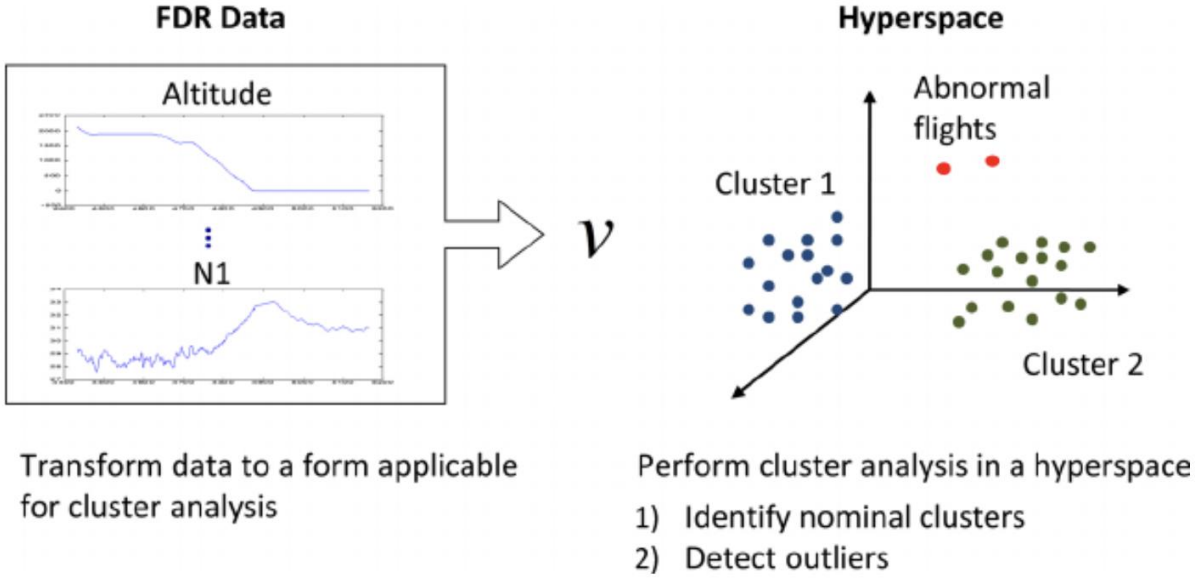
Unsupervised Learning

- Unsupervised learning – literally, learning without a supervisor (labeled data)
  - "I do not know their names but I can see the difference"
  - Clustering
- Advantage
  - No labels are needed, suitable for scenario where the labels are hard to get
- Disadvantage
  - May learn slower compared to supervised learning (requiring more data to achieve similar performance/accuracy)

- Anomaly detection is a security application of unsupervised learning
  - Assumption
    - Attack and benign are different
    - Most of samples are benign

Unsupervised Learning Example

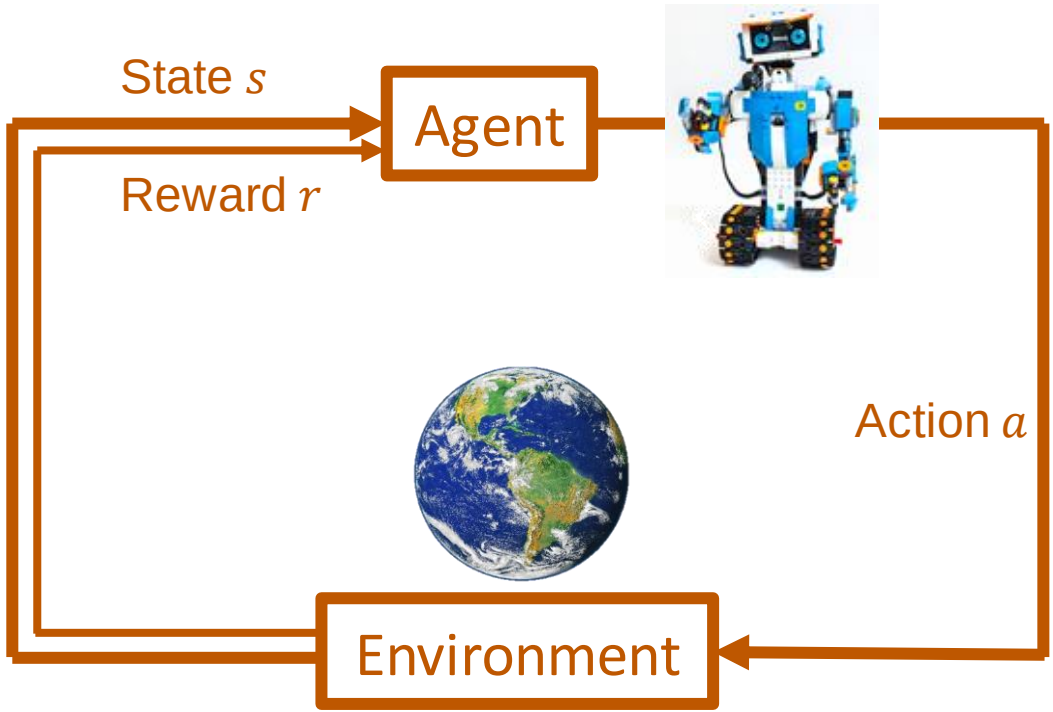
- Anomaly Detection



Reinforcement Learning

- In many cases even a dataset is not available
  - E.g., an interactive scenario, playing a game, etc.
  - use reinforcement learning

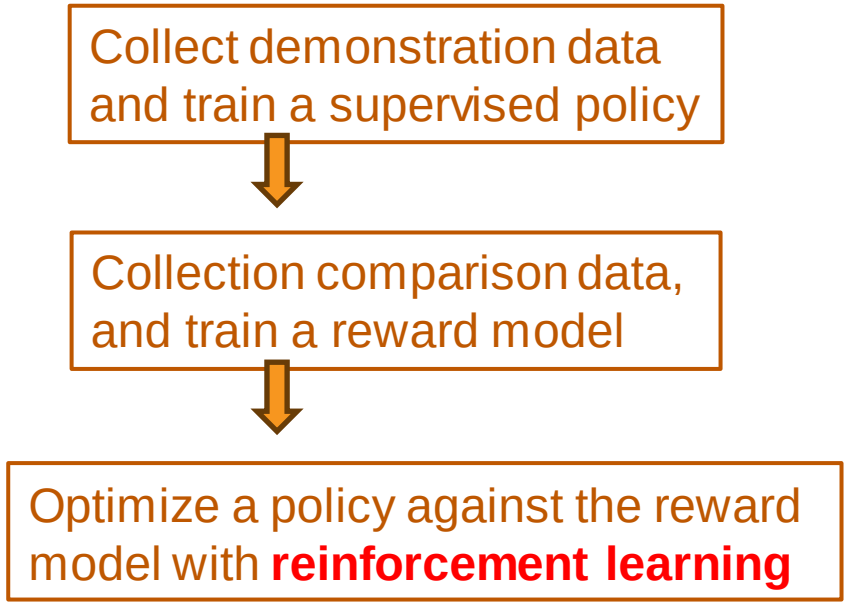
What is Reinforcement Learning?



Slides modified from Yuandong Tian, 2023.

Reinforcement Learning

ChatGPT / GPT4

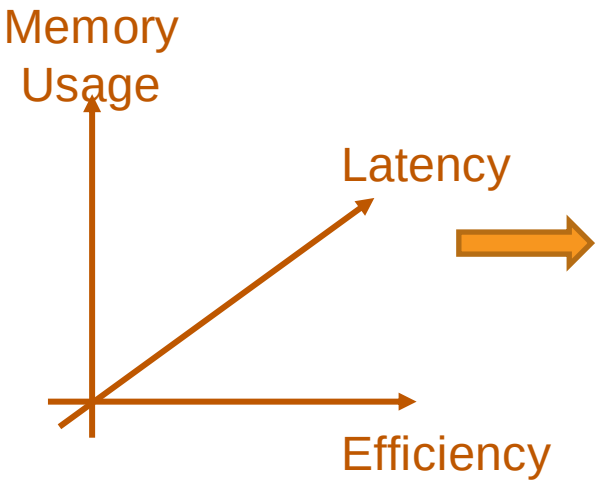


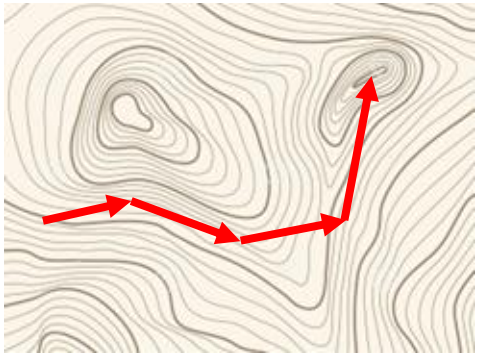
Reinforcement Learning from Human Feedbacks (RLHF)

[from OpenAI: https://openai.com/research/instruction-following]

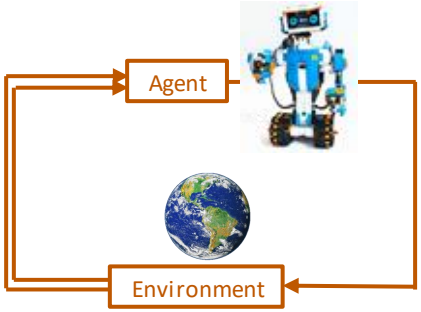
Slides modified from Yuandong Tian, 2023.

RL Usage in Computer Systems





Sequential optimization



Reinforcement Learning

Slides modified from Yuandong Tian, 2023.

RL Usage in Computer Systems

AlphaTensor (DeepMind)

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

$m_1 = (a_1 + a_4)(b_1 + b_4)$

$m_2 = (a_3 + a_4)b_1$

$m_3 = a_1(b_2 - b_4)$

$m_4 = a_4(b_3 - b_1)$

$m_5 = (a_1 + a_2)b_4$

$m_6 = (a_3 - a_1)(b_1 + b_2)$

$m_7 = (a_2 - a_4)(b_3 + b_4)$

$c_1 = m_1 + m_4 - m_5 + m_7$ $\quad c_3 = m_2 + m_4$

$c_2 = m_3 + m_5$ $\quad c_4 = m_1 - m_2 + m_3 + m_6$

A. Fawzi et al, *Discovering faster matrix multiplication algorithms with reinforcement learning*, Nature'22

(Adding one low-rank approximation at a time)

AlphaDev (DeepMind)

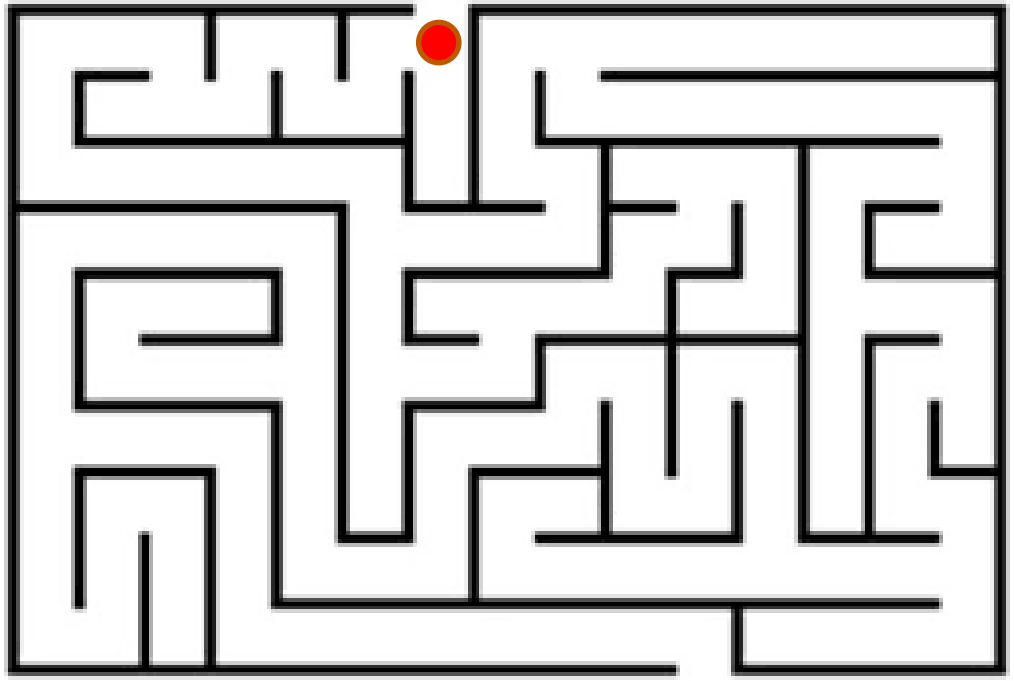
mov	R	S
cmp	P	R
cmovl	P	S
~~mov~~	~~S~~	~~P~~
cmp	S	Q
cmovg	Q	P
cmovg	S	Q

D. Mankowit et al, *Faster sorting algorithms discovered using deep reinforcement learning*, Nature'23

(Adding one instruction at a time)

Slides modified from Yuandong Tian, 2023.

What is Reinforcement Learning?



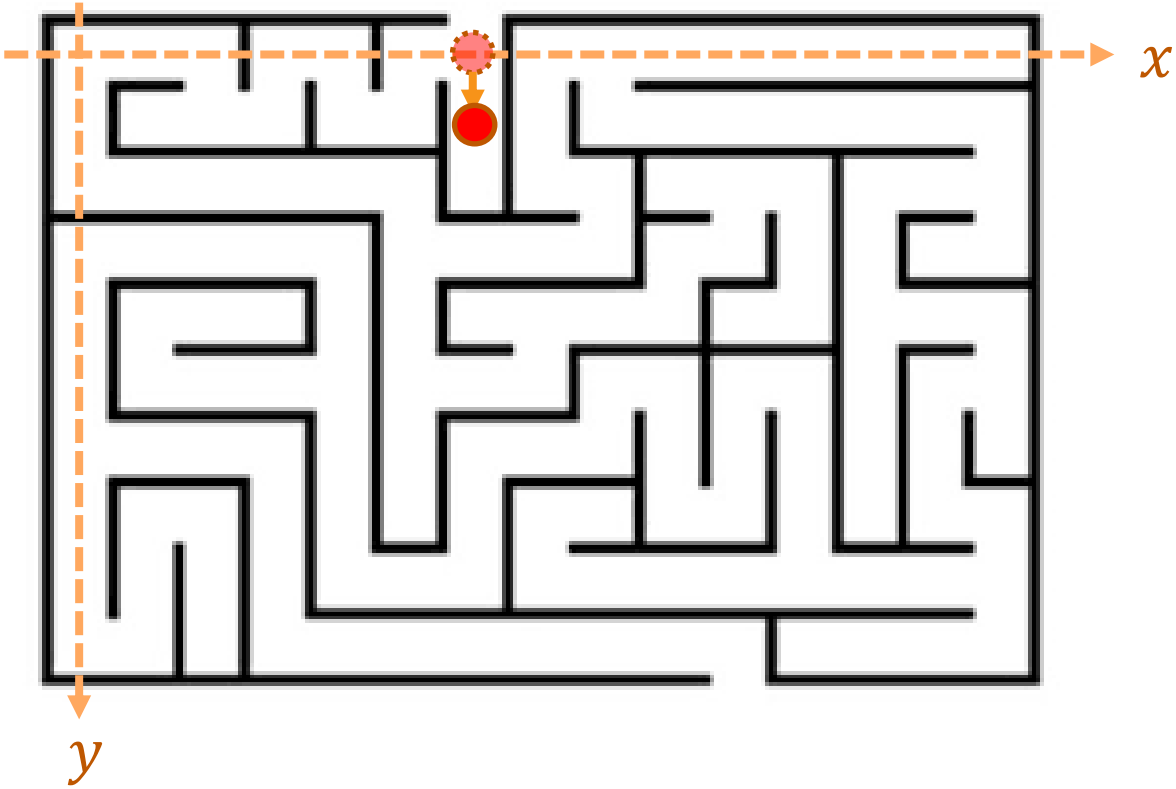
State:
where you are?

Action:
left/right/up/down

Next state:
where you are after the action?

Slides modified from Yuandong Tian, 2023.

What is Reinforcement Learning?



State:

$$s = (x, y) = (6,0)$$

Actions:

Left: $x \leftarrow x - 1$

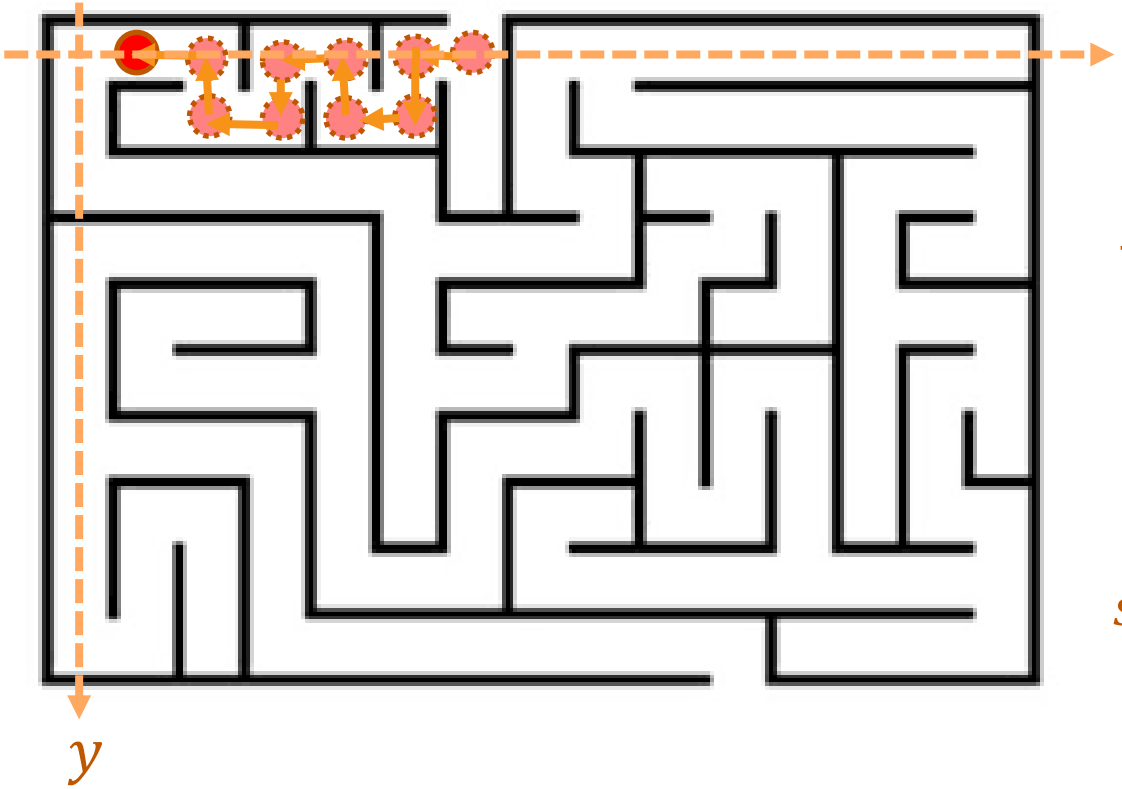
Right: $x \leftarrow x + 1$

Up: $y \leftarrow y - 1$

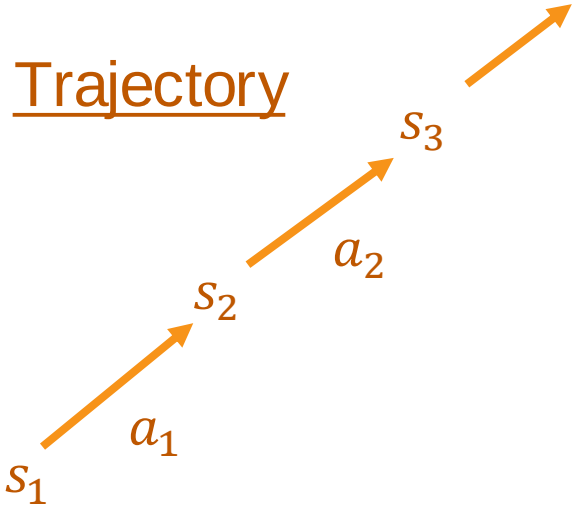
Down: $y \leftarrow y + 1$

Slides modified from Yuandong Tian, 2023.

What is Reinforcement Learning?

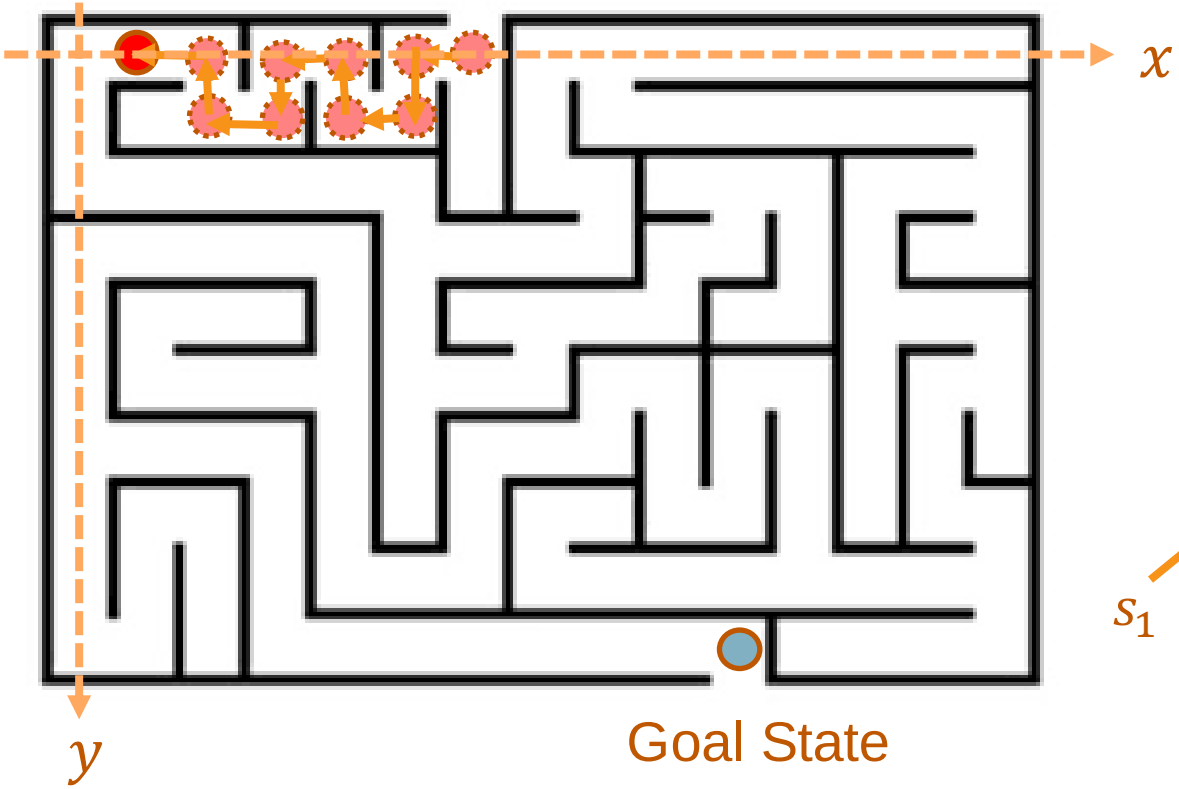


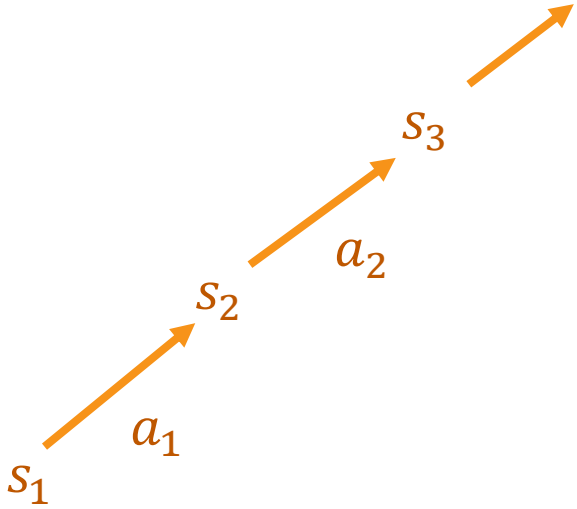
Trajectory



Slides modified from Yuandong Tian, 2023.

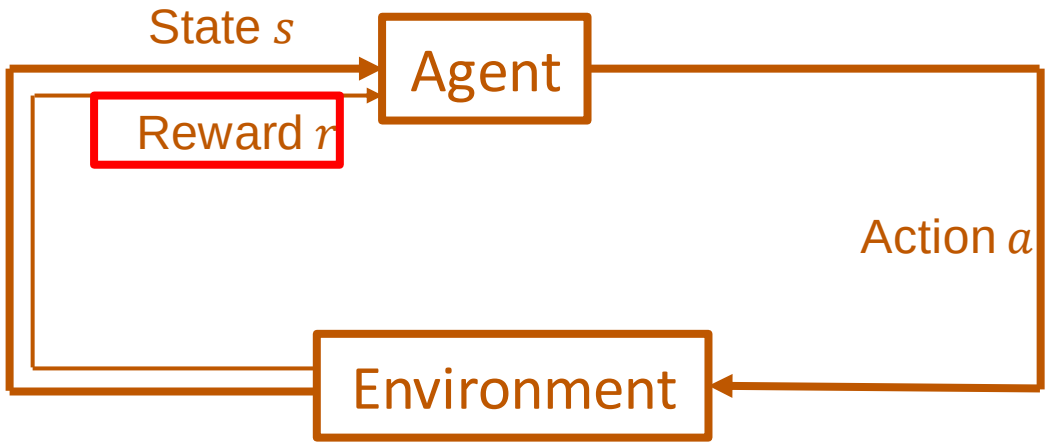
Goal of Reinforcement Learning

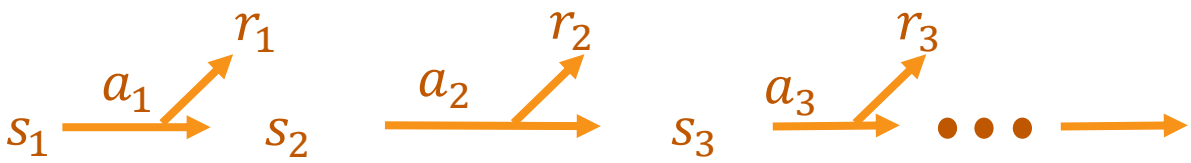




Slides modified from Yuandong Tian, 2023.

Goal of Reinforcement Learning





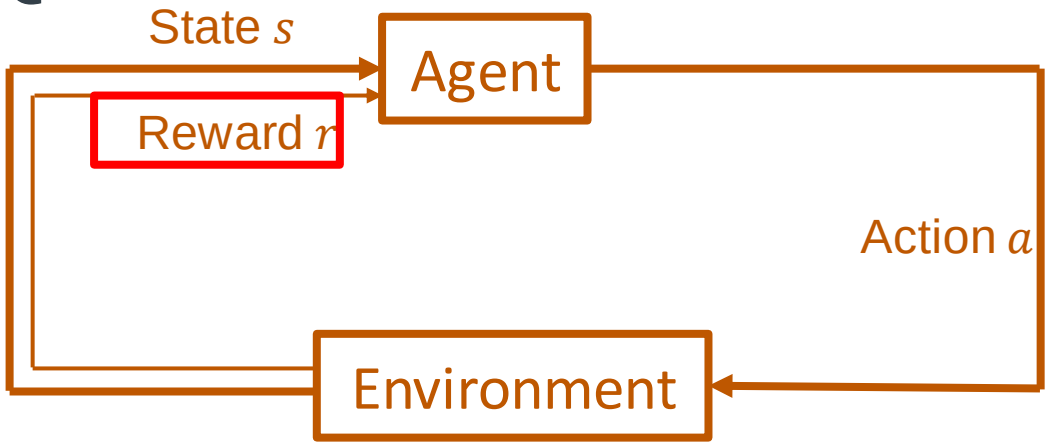
Maximize long-term reward:

$$R = \sum_{t=1}^{+\infty} \gamma^{t-1} r_t$$

γ: Discount factor

Slides modified from Yuandong Tian, 2023.

Key Quantities



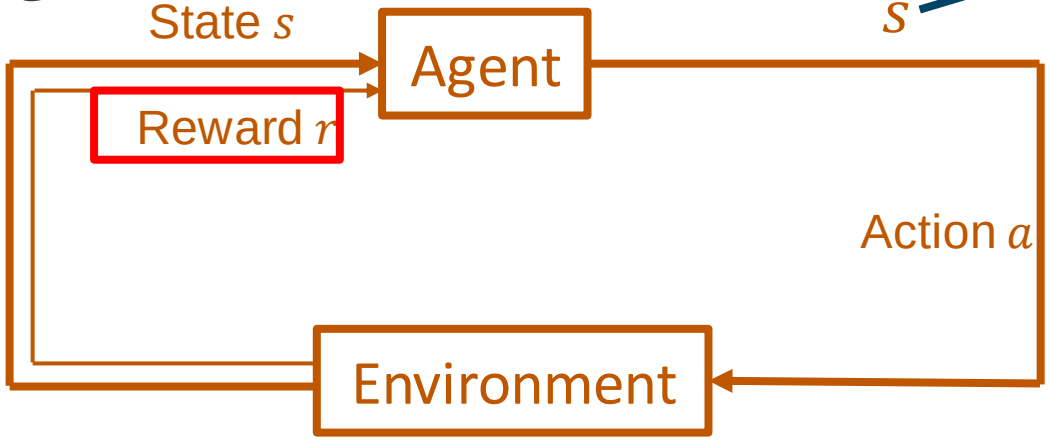
$V^*(s)$: Maximal reward you can get starting from state

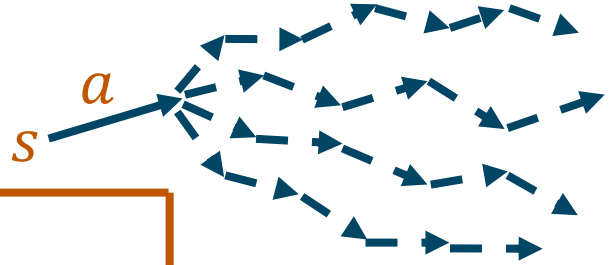
$Q^*(s, a)$: Maximal reward starting from s after taking action a

$\pi(a|s)$: Probability of taking action a given state s

Slides modified from Yuandong Tian, 2023.

Key Quantities





$V^{\pi}(s)$: Reward you can get, starting from s following policy π

$Q^{\pi}(s,a)$: Reward starting from s after taking action a and following π

Slides modified from Yuandong Tian, 2023.

RL Learning Algorithms

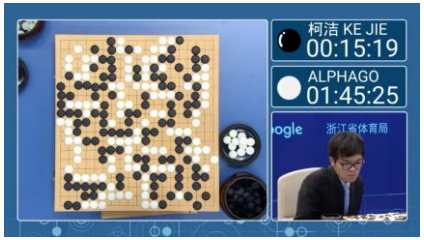
- Value-based approach (Q-learning, AlphaZero)
  - $Q^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}(s_{t+1}, a')$
- Policy-based Methods
  (Policy Gradient and PPO)

RL workflow

- Step 1: RL formulation
- Step 2: Training/Learning
- Step 3: Deploy/Testing/Inference

Case Study: DeepMind's Alpha-Series

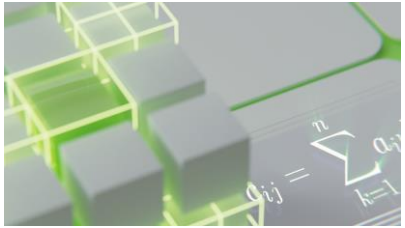
AlphaGo Lee (Mar. 2016)



AlphaGo Master (May. 2017)

AlphaGo Zero (Oct. 2017)

Without Human Knowledge

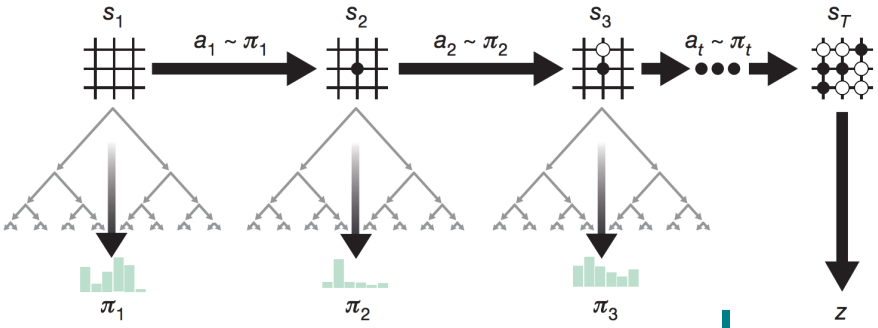


AlphaTensor (2022)

AlphaDev (2023)

Slides modified from Yuandong Tian, 2023.

AlphaGoZero / AlphaZero



Generate Training data

Without human knowledge

θ_i

Update Models

Self-Replays

[Silver et al, Mastering the game of Go without human knowledge, Nature 2017]

Slides modified from Yuandong Tian, 2023.

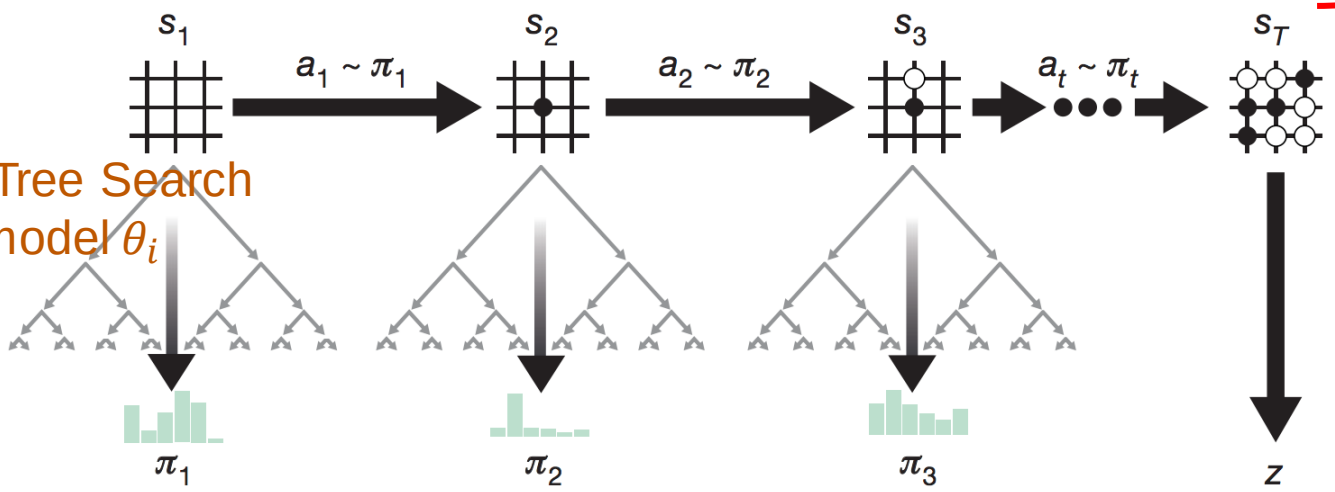
Generate Self-play Games

(s_1, π_1, z)

(s_2, π_2, z)

(s_3, π_3, z)

Training samples for θ_{i+1}



Monte Carlo Tree Search with current model θ_i

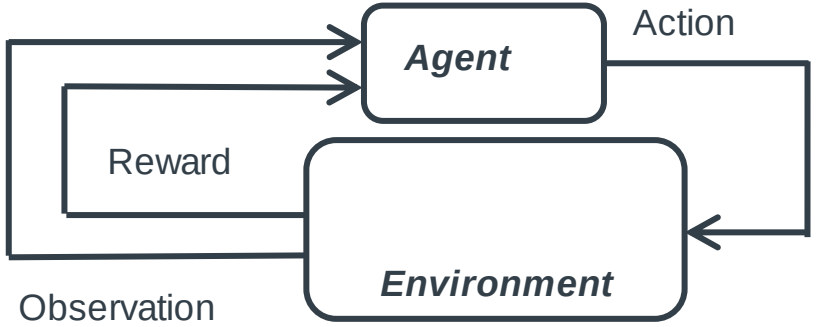
Slides modified from Yuandong Tian, 2023.

RL Application Scenarios

- RL comes with a policy (“multi-step strategy”) in a predefined environment (”game”)
  - Typical chess and video games
  - Trading strategy in the stock market
  - **Security attack and defense**

Cache-Timing Attack as RL

- Agent: Attacker
- Environment: Cache
  - architecture simulator
  - cache in the processor
- Actions
  - attacker makes an access
  - attacker waits for victim access
  - attacker guesses the secret
- Observation
  - latency of attacker accesses

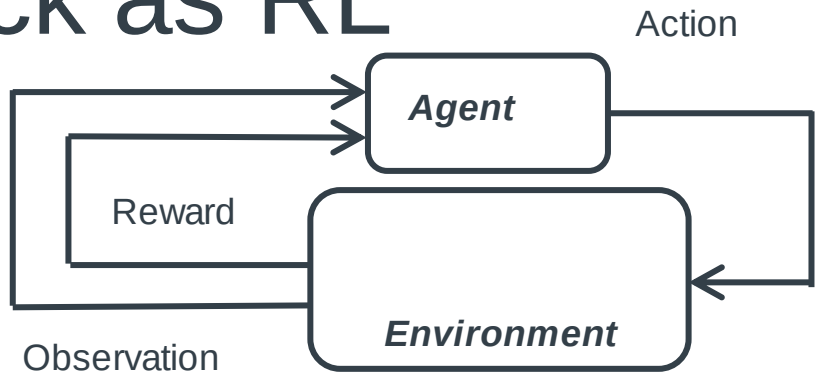


Summary of actions of existing attacks

Attacks	Attacker action	Victim action	Observations
prime+probe	access addrs	access an addr	attacker's latency
flush+reload	flush addrs	access an addr	attacker's latency
evict+reload	access addrs	access an addr	attacker's latency
evict+time	access addrs	access addrs	victim's latency

Cache-Timing Attack as RL

- Reward
  - guess correct: positive reward
  - guess wrong: negative reward
  - each step: small negative reward
- Maximizing long-term reward
  - more correct guesses
  - fewer wrong guesses
  - fewer number of steps

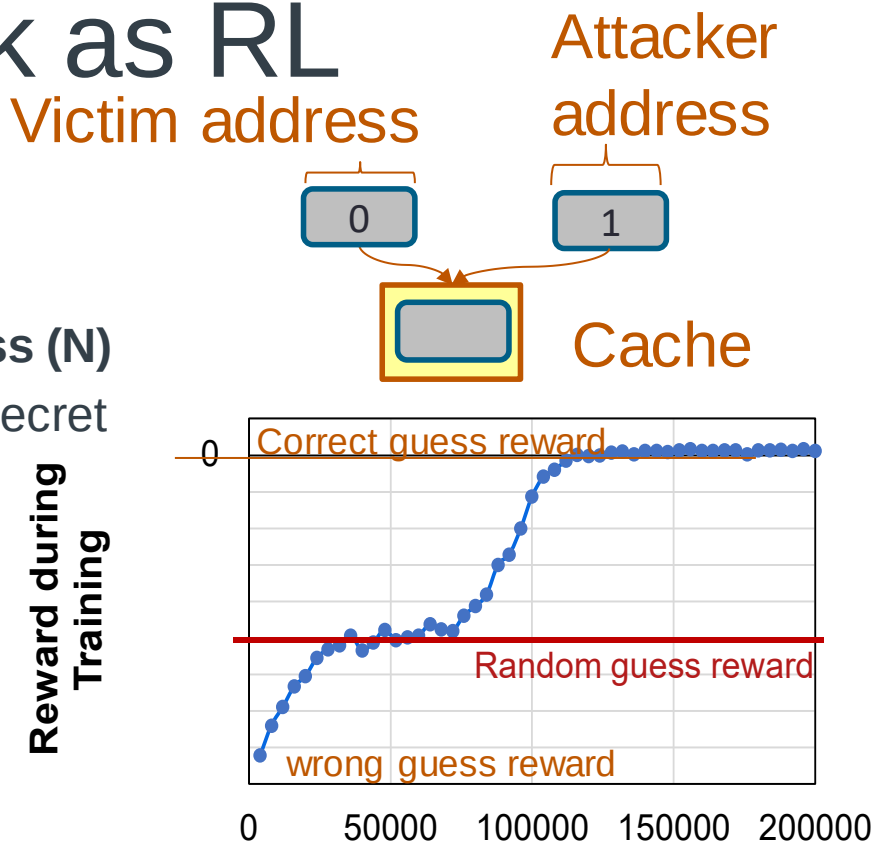


Summary of actions of existing attacks

Attacks	Attacker action	Victim action	Observations
prime+probe	access addrs	access an addr	attacker's latency
flush+reload	flush addrs	access an addr	attacker's latency
evict+reload	access addrs	access an addr	attacker's latency
evict+time	access addrs	access addrs	victim's latency

Cache-Timing Attack as RL

- Settings
  - 1 set 1-way cache
  - attacker can access address 1
  - victim secret: **access 0 (0) / no access (N)**
  - attacker want to infer whether victim secret **is 0/N**
- Attack found
  - step 1: attacker accesses 1
  - step 2: then wait for a while
  - step 3: attacker accesses 1 again
  - step 4: guess the secret **0/N**



Summary

- Supervised learning
- Unsupervised learning
- Reinforcement learning

APRIL 2024

TEXAS
The University of Texas at Austin

MACHINE LEARNING-BASED DETECTION MICROARCHITECTURE

ASPLOS 2024 tutorial

https://ut-ldma.github.io

MULONG LUO
Postdoctoral researcher, The University of Texas at Austin

Network access for lab

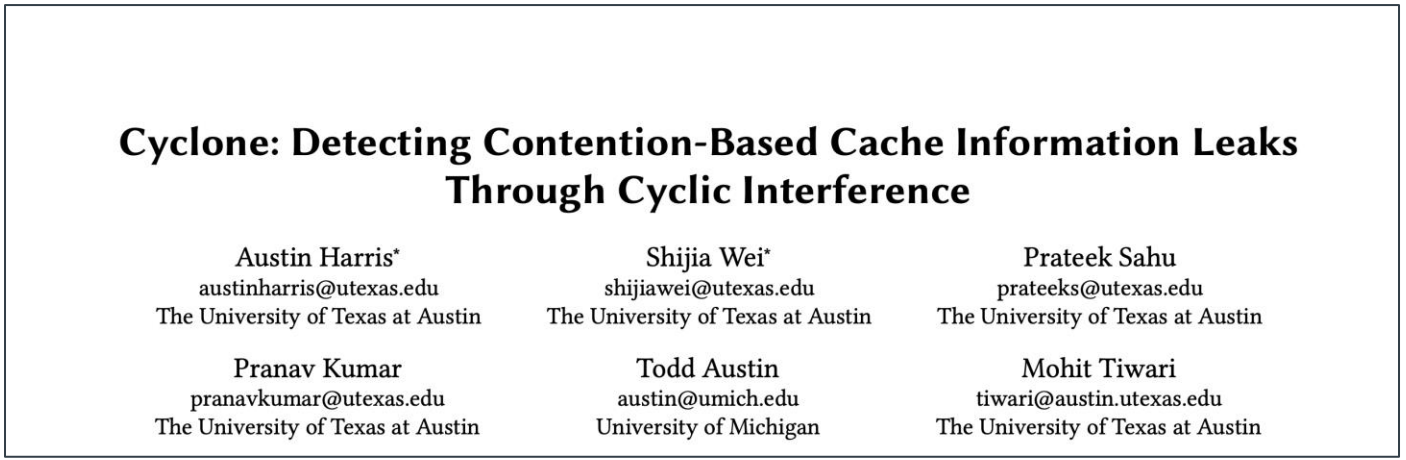
- Wifi: Hilton Honors Meeting
- Password: ACM2024
- AWS credentials will be handed seperately

Outline

- HW architectures
  - Cyclone: Global/Local detector architecture
  - PerSpectron: Peceptron-based detector architecture
- Advanced Model Trainings
  - MACTA: multiagent reinforcement learning
  - EVAX: Generative adversarial network for data augmentation

Runtime architecture for ML Detection

- Learning-based architecture
  - Feature extraction
  - Learning algorithm



Cyclone: Detecting Contention-Based Cache Information Leaks Through Cyclic Interference

Austin Harris*
austinharris@utexas.edu
The University of Texas at Austin

Shijia Wei*
shijiawei@utexas.edu
The University of Texas at Austin

Prateek Sahu
prateeks@utexas.edu
The University of Texas at Austin

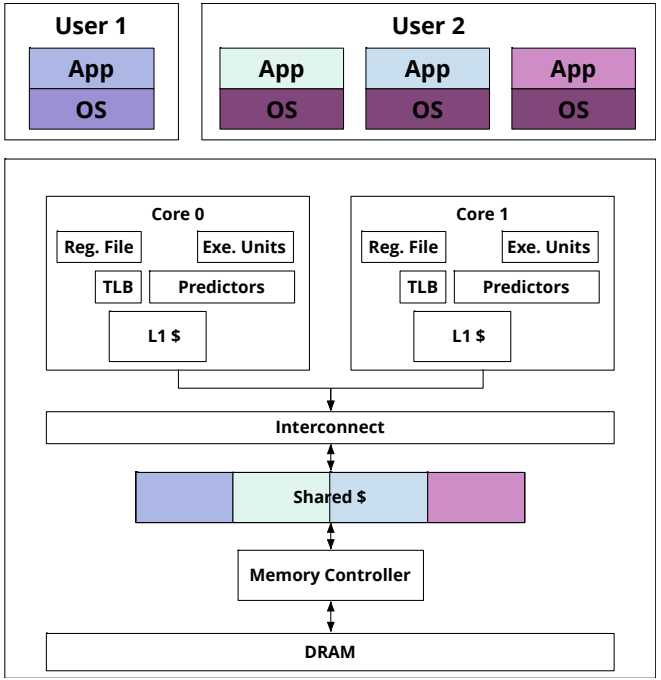
Pranav Kumar
pranavkumar@utexas.edu
The University of Texas at Austin

Todd Austin
austin@umich.edu
University of Michigan

Mohit Tiwari
tiwari@austin.utexas.edu
The University of Texas at Austin

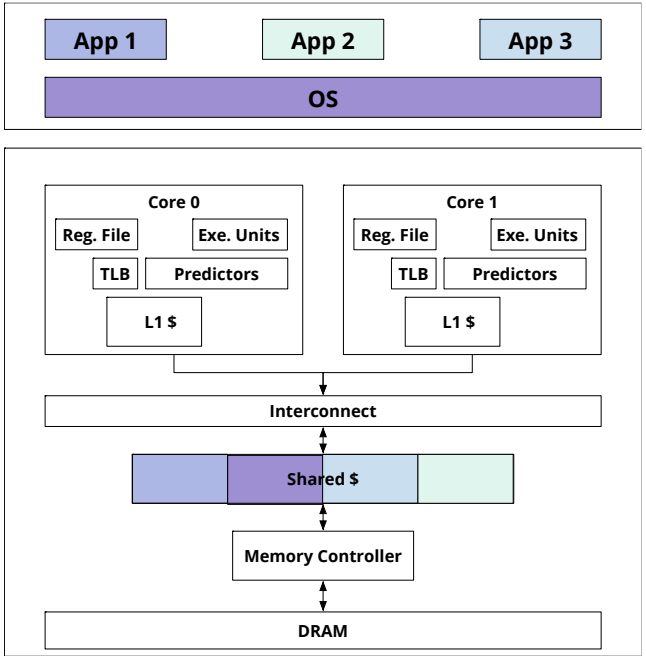
Cyclone: Sharing Resource Across Security Domains

- Security domains
  - Virtual machines on the public cloud



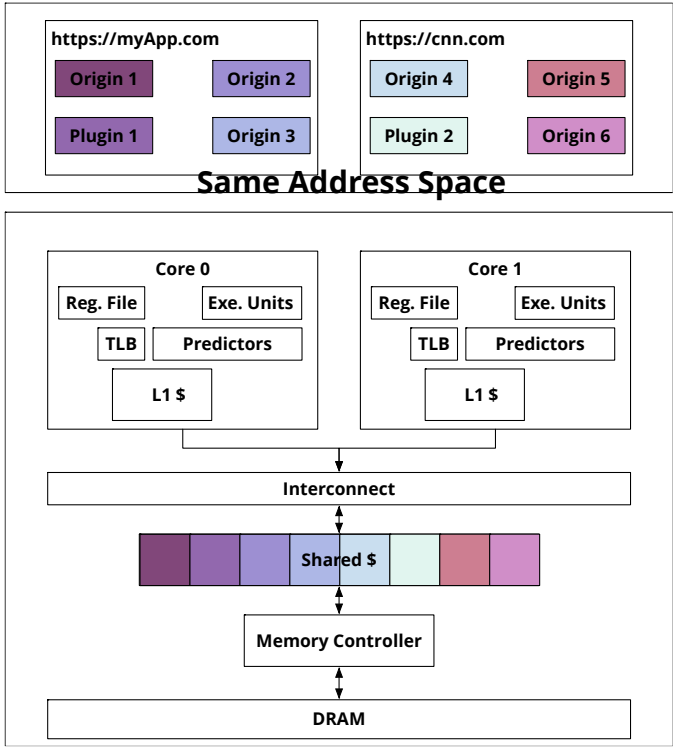
Cyclone: Sharing Resource Across Security Domains

- Security domains
  - Virtual machines on the public cloud
  - Processes in a shared machine



Cyclone: Sharing Resource Across Security Domains

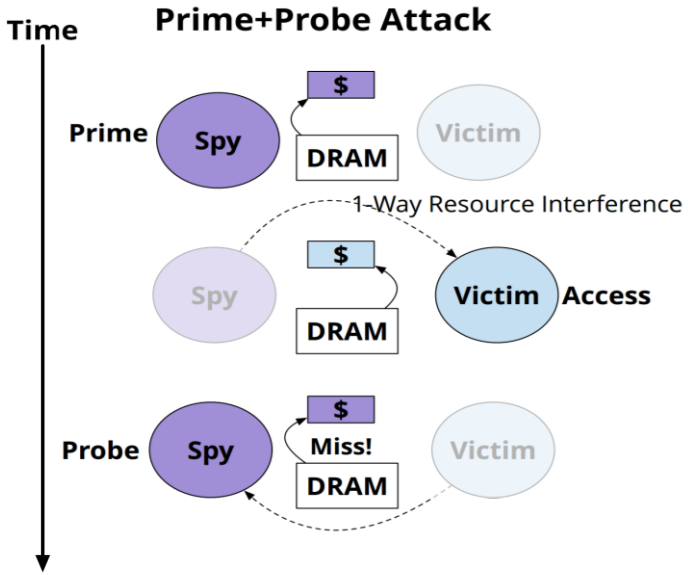
- Security domains
  - Virtual machines on the public cloud
  - Processes in a shared machine
  - Sandboxes in a browser
- Micro-arch side-channels break isolation!
- Partitioning/flushing expensive!



Cyclone: Interference

- Interference is root of many micro-arch attacks

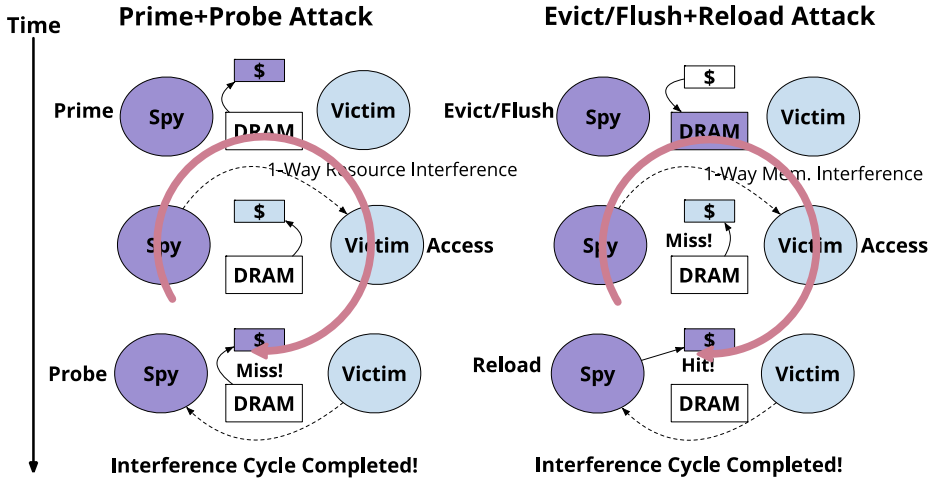
  Attack Types
  - Prime+Probe
  - Flush+Reload
  - Evict+Reload
- Resources
  - Caches
  - TLBs
  - Predictors
  - Queues
  - Memory



Cyclic Resource Interference (CRI)

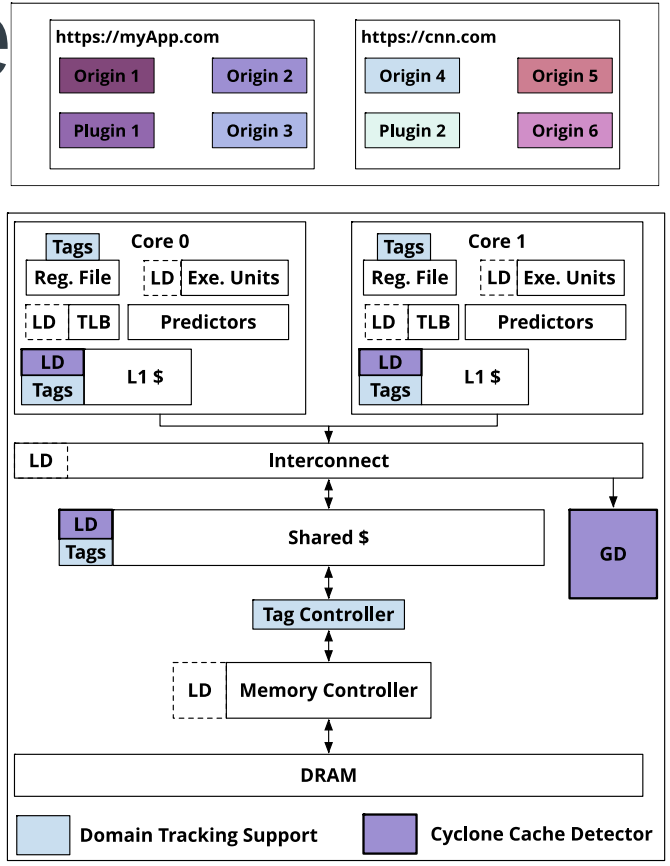
Cyclone: Cyclic Interference

- Interference is root of many micro-arch attacks
- Cyclic Interference is a robust feature
  - Opportunity: detect attacks as anomalous cyclic interference



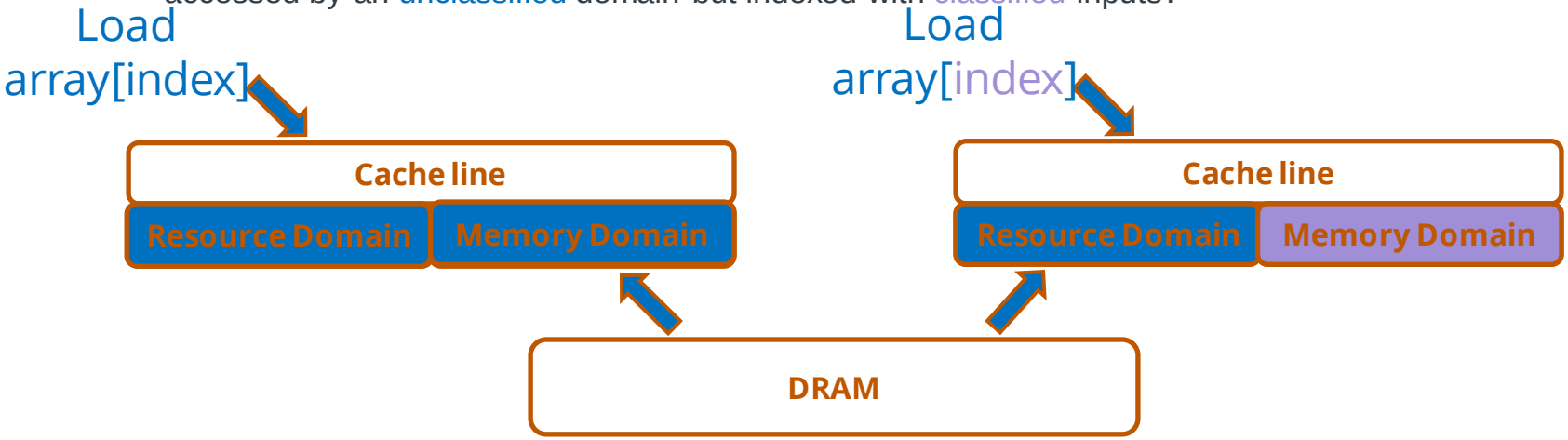
Cyclone Architecture

- Software defined security domains
  - OS kernel, Sandbox engine (e.g. browser)
  - Micro-arch Info. Flow Tracking
- Configurable Local Detectors (LDs) for each µarch component
  - Non-Invasive, snooping
  - Track cyclic interference
  - Simple logic (e.g. thresholding)
- Programmable Global Detector (GD)
  - Off the critical path, programmable
  - Correlates across components
- Domain propagation
  - Memory locations, hardware resource
  - Speculative



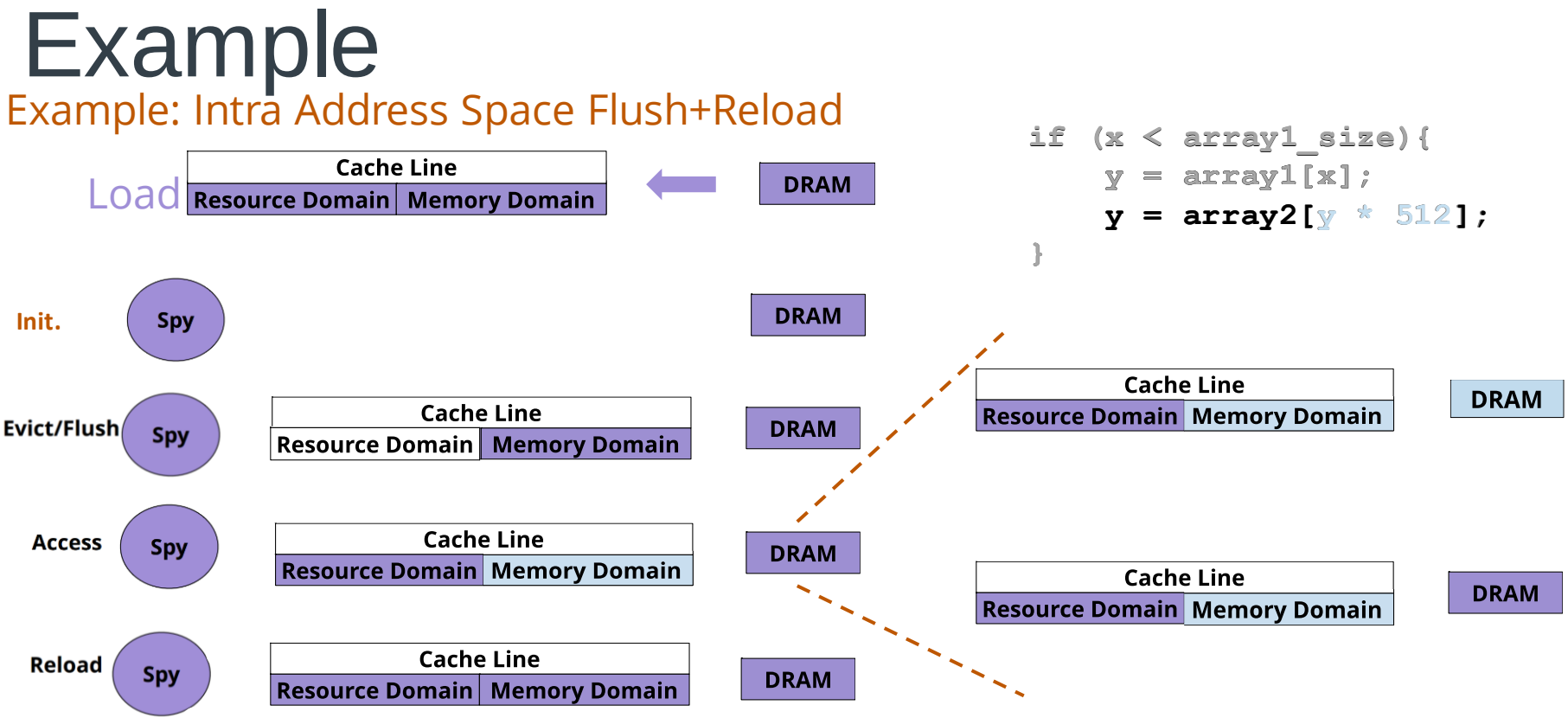
Cyclone: Domain Propagation

- Micro-architectural information flow tracking
  - Memory ↔ Registers
  - Memory/Register ↔ Micro-arch state
- Associate resource and memory domains with each cache line
- Propagate domains in the microarchitecture based on access initiator and addressor
  - accessed by an unclassified domain?
  - accessed by an unclassified domain but indexed with classified inputs?



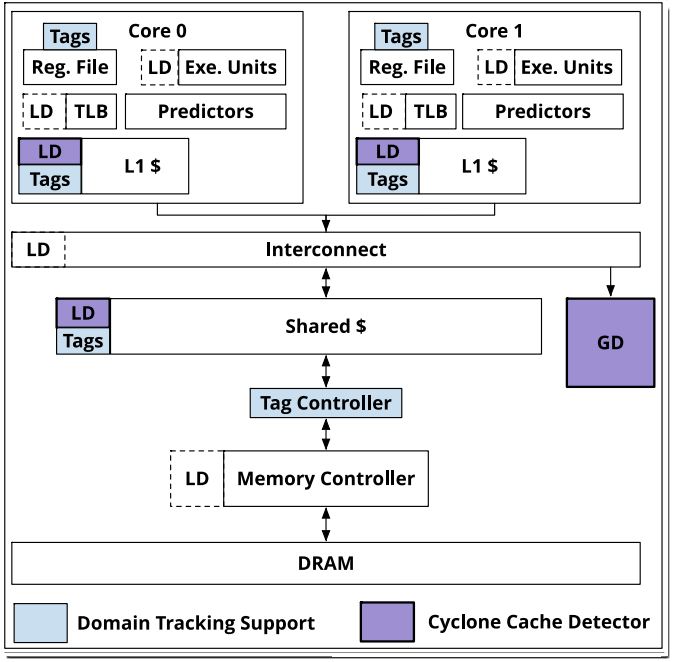
Cyclone Domain Propagation: Attack Example

Example: Intra Address Space Flush+Reload



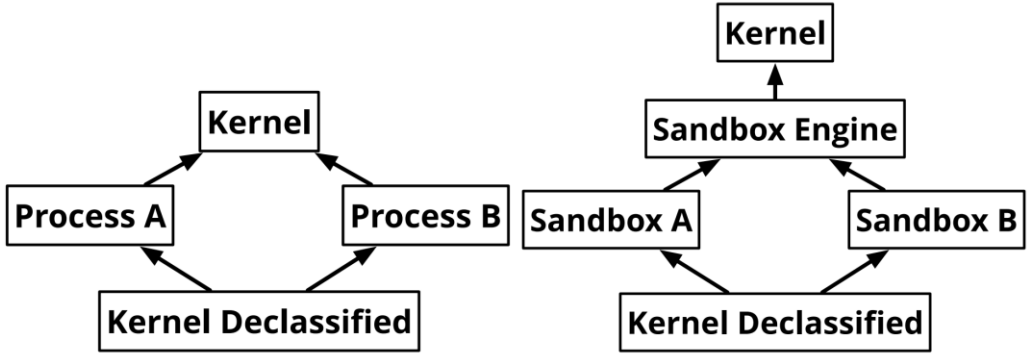
Cyclone Architecture

- Software defined security domains
  - OS kernel, Sandbox engine (e.g. browser)
  - Micro-arch Info. Flow Tracking
- Configurable Local Detectors (LDs) for each µarch component
  - Non-Invasive, snooping
  - Track cyclic interference
  - Simple logic (e.g. thresholding)
- Programmable Global Detector (GD)
  - Off the critical path, programmable
  - Correlates across components
- Domain propagation
  - Memory locations, hardware resource
  - Speculative
- Declassification
  - OS
  - Sandbox engine



Cyclone: Declassification

- Lattice defines the “secrets”
- But higher privileged software operates on lower domains’ memory
  - Process creation (fork)
  - Memory management (Copy-On-Write, zero-init)
- Proper declassification is needed



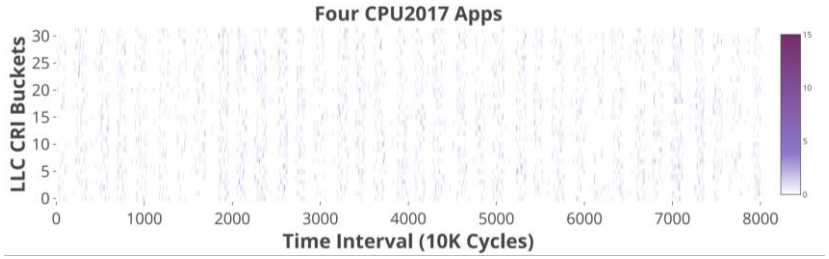
(a) Coarse-Grained isolation *(b) Fine-Grained isolation*

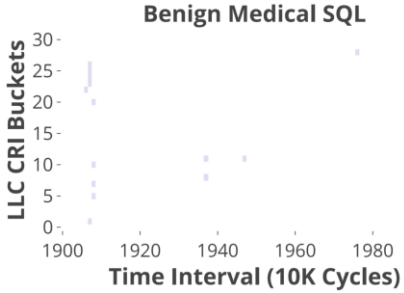
Cyclone: Evaluation Challenges

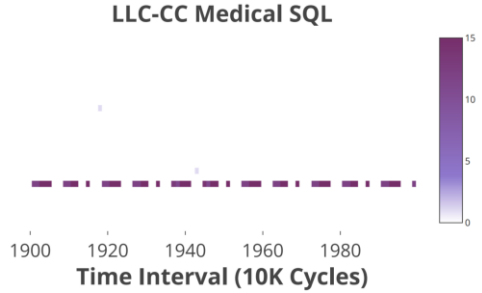
- Detection false positives
  - Stress test with concurrent SPEC2017
- Attacks on privacy-sensitive applications
  - Medical database, face recognition, ECG classification
    - Cross-process cache covert-channels
- Fine-grained domains in same address space
  - Web browser
    - Spectre-V1 password-stealing attack in JavaScript
- OS/JavaScript Engine required to demonstrate attacks
  - Gem5 ARMv8 OoO running
  - Modified Linux, PhantomJS (WebKit)
- Baseline
  - State-of-the-Art Contention Tracking
  - (Improved) Bucketed Contention Tracking

Cyclone: Evaluation Challenges

- Detection false positives
  - Stress test with concurrent SPEC2017

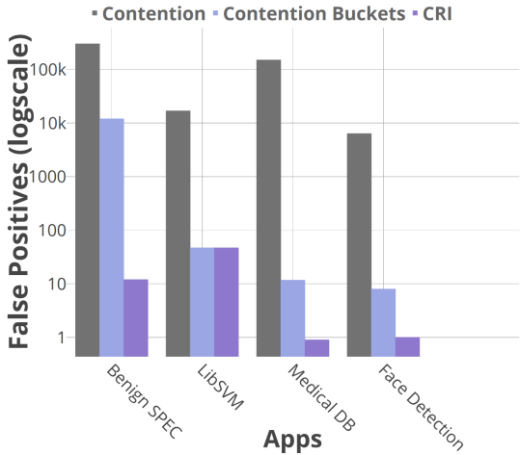


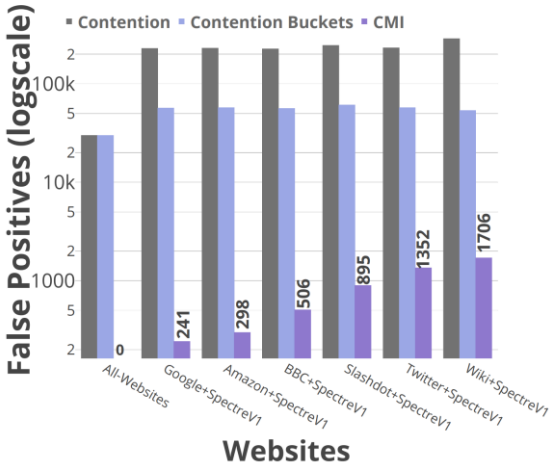




Results: low false positives *close to 100% detection rates*

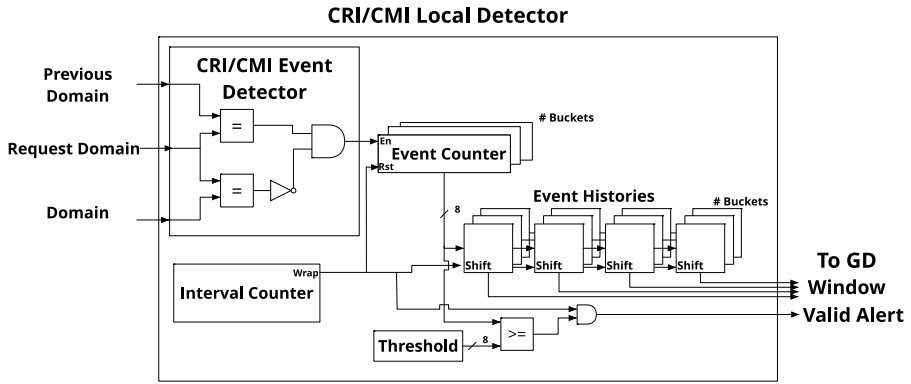
- Detecting resource-based covert channel
  - Attack using set-targeted LLC Prime+Probe
  - Cyclone detects attacks through Cyclic Resource Interference (CRI)
- Detecting memory-based leak (Spectre V1)
  - Spectre.js in PhantomJS using Evict+Reload
  - Cyclone detects attacks through Cyclic Memory Interference (CMI)



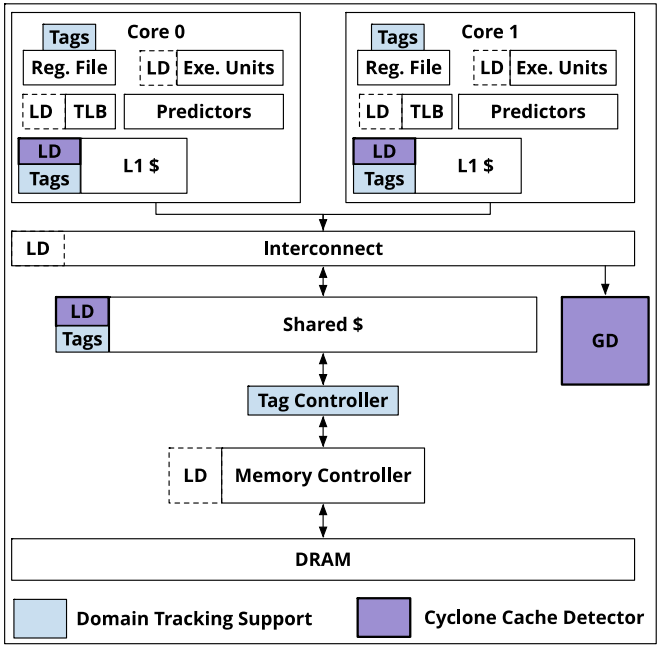


Cyclone Overheads

- Memory overhead and extra DRAM traffic
  - 8-bit domain ID tags for every 64B cache line (1.5%)
  - Unoptimized tag controller (2.4% IPC decrease in SPEC)
- Cache overhead
  - Four 8-bit domain IDs per cache line (6.25%)
- Local detector overheads
  - Snooping, off critical path
- Global detector
  - Programmable small core
  - Fixed-function SVM, etc.
  - Model size < 512B



Cyclone: Q&A



- Distributed anomaly detector
- Micro-arch information flow tracking
- Integration with software-level detection (e.g., OSQuery)
- RISC-V Prototype (BOOM)

PerSpectron: Perceptron Detector

2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)

PerSpectron: Detecting Invariant Footprints of Microarchitectural Attacks with Perceptron

Samira Mirbagher-Ajorpaz
Computer Science and Engineering
Texas A&M University
College Station, USA
samiramir@tamu.edu

Gilles Pokam
Intel Labs
Santa Clara, USA
gilles.a.pokam@intel.com

Esmaeil Mohammadian-Koruyeh
Computer Science and Engineering
University of California, Riverside
Riverside, USA
emoha004@ucr.edu

Elba Garza
Computer Science and Engineering
Texas A&M University
College Station, USA
elba@tamu.edu

Nael Abu-Ghazaleh
Computer Science and Engineering
University of California, Riverside
Riverside, USA
naelag@ucr.edu

Daniel A. Jiménez
Computer Science and Engineering
Texas A&M University
College Station, USA
djimenez@acm.org

PerSpectron

- Key Contributions
  - Using Perceptron (Neural Net)
  - Feature Selections
  - Hardware Architecture

PerSpectron: using Perceptron

- Single Layer Perceptron
  - Can be implemented efficiently in HW
  - Good prediction performance

Model	**DT-CART***	**DT-CART**	**Logistic Regression***	**Perceptron**	**KNN**	**NN***	**NN**	**PerSpectron**
Feature	**MAP**	**PerSpectron**	**MAP**	**1159 features**	**PerSpectron**	**MAP**	**PerSpectron**	**PerSpectron**
Mean Accuracy	0.8718	0.9058	0.7594	0.8974	0.9487	0.8026	0.9822	0.9979
95.00% Confidence	0.8718 ± 0.1005	0.9058 ± 0.0120	0.7594 ± 0.0018	0.8974 ± 0.2030	0.9487 ± 0.1435	0.8026 ± 0.0026	0.9822 ± 0.0089	0.9979 ± 0.0065
False Positives $> 20 samples$	dealII,gcc, gobmk,bzip2	dealII, gcc	h264ref,povray,gcc sjeng,gobmk	gobmk,dealII	sjeng,gobmk	dealII,povray bzip2,gobmk	gobmk,dealII	gobmk
False Negatives *post/preleakage*	prime+probe, spectre$< 0.75x$ post leakage	spectre$< 0.75x$, polymorphic post leakage	prime+probe, spectre$< 0.75x$, polymorphic post leakage	prime+probe, spectre$< 0.75x$ pre leakage	calibration-ff/pp* post leakage	prime+probe spectre$< 0.75x$ pre leakage	not post leakage	not post leakage
Hardware Complexity	low	low	low	low	high	high	high	low

**pp: prime+probe*

TABLE IV: ML Model and Feature Set comparison

PerSpectron: Feature Selections

- 1159 available features

group 1	group 2	group 3	group 4
commit.SquashedInsts	commit.SquashedInsts	commit.NonSpecStalls	branchPred.condIncorrect
lsq.squashedStores	iew.lsq.forwLoads	l2.ReadSharedReq_misses	commit.op_class_0::No_OpClass
iew.memOrderViolationEvents	dtlb.rdMisses	rename.serializingInsts	iew.iewExecSquashedInsts
fetch.SquashCycles	iew.SquashCycles	membus.trans_dist::ReadSharedReq	iew.lsq.thread0.ignoredResponses
iew.lsq.forwLoads	fetch.SquashCycles	l2.ReadSharedReq_mshr_miss_latency	iq.iqSquashedInstsIssued
decode.SquashCycles	iq.SquashedNonSpecRemoved	dcache.ReadReq_mshr_miss_latency	iew.iewDispSquashedInsts
iq.SquashedInstsExamined	decode.SquashCycles	l2.ReadSharedReq_accesses	branchPred.RASInCorrect
lsq.squashedLoads	iew.memOrderViolationEvents	l2.ReadSharedReq_miss_latency	iq.fu_full::FloatMemWrite
iew.SquashCycles	fetch.IcacheSquashes	dcache.ReadReq_mshr_misses	commit.op_class_0::FloatAdd
iew.BlockCycles	lsq.squashedLoads	tol2bus.trans_dist::ReadSharedReq	fetch.PendingQuiesceStallCycles
memDep.conflictingStores	iq.SquashedInstsExamined	commit.membars	iew.lsq.thread0.rescheduledLoads
dtb.rdMisses	iew.BlockCycles	dcache.ReadReq_misses	commit.branchMispredicts
iq.SquashedNonSpecRemoved	memDep.conflictingLoads	l2.ReadSharedReq_mshr_misses	branchPredindirectMispredicted
rename.SquashCycles	iq.fu_full::MemRead	membus.trans_dist::ReadResp	commit.op_class_0::SimdCvt
memDep.conflictingLoads	lsq.squashedStores	rename.serializeStallCycles	iq.fu_full::IntAlu
rename.UndoneMaps	rename.UndoneMaps	mem_ctrls.selfRefreshEnergy	iew.branchMispredicts
fetch.IcacheSquashes	memDep.conflictingStores	iq.NonSpecInstsAdded	iew.predictedNotTakenIncorrect
iq.SquashedOperandsExamined	rename.SquashCycles	tol2bus.trans_dist::ReadResp	tol2bus.snoop_filter.tot_requests

TABLE I: Subset of highly correlated features. Features are ranked by correlations between the true class labels and the feature value from top to bottom and left to right. Features in each group have high mutual correlation but belong to different pipeline components and are used to construct *Replicated Perceptron detectors.*

PerSpectron: HW Design

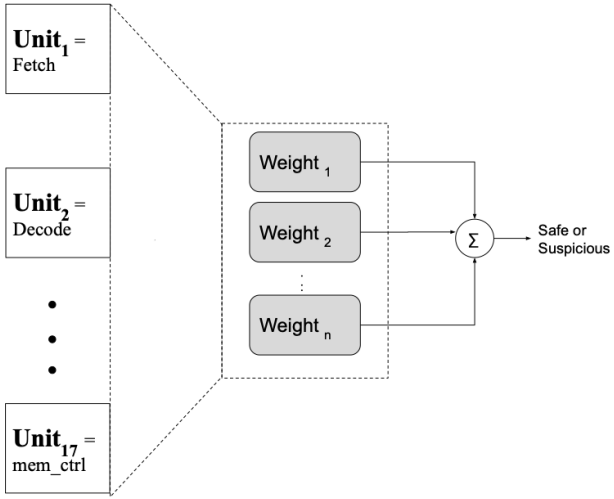
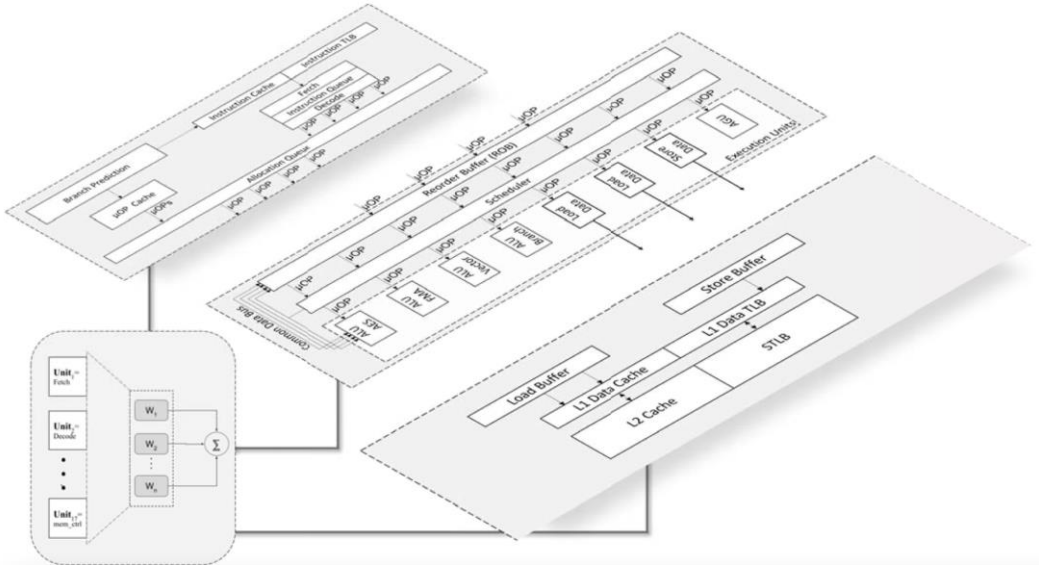


Fig. 2: Architecture of PerSpectron



Summary: HW Architecture of ML Detections

- Everything needs to be implemented in HW
  - Collecting ML features in HW
  - Implementing ML models in HW
  - Trigger defense/response in HW

	feature collection	ML models	Model implementation
Cyclone	Cyclic features	One class SVM	Flexible/SW model
PerSpectron	Hardware Performance counters	Perceptron	HW Perceptron

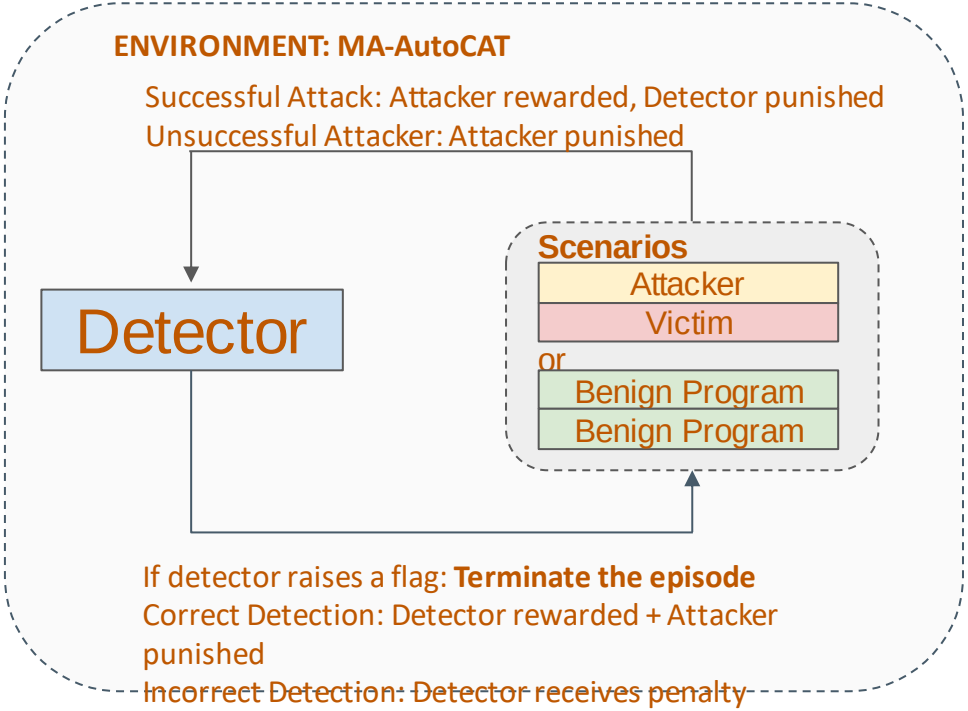
Detector Training Methods

MACTA: A Multi-agent Reinforcement Learning Approach for Cache Timing Attacks and Detection

Jiaxun Cui[1] **Xiaomeng Yang**[*2] **Mulong Luo**[*3] **Geunbae Lee**[*4] **Peter Stone**[1,5]
Hsien-Hsin S. Lee[6] **Benjamin Lee**[2,7] **G. Edward Suh**[2,3] **Wenjie Xiong**[†4] **Yuandong Tian**[†2]

[1]The University of Texas at Austin [2]Meta AI [3]Cornell University [4]Virginia Tech
[5]Sony AI [6]Intel Corporation [7]University of Pennsylvania
cuijiaxun@utexas.edu, yangxm@meta.com, ml2558@cornell.com, geunbae@vt.edu, pstone@cs.utexas.edu,
linear@acm.org, leebcc@seas.upenn.edu, edsuh@meta.com, wenjiex@vt.edu, yuandong@meta.com

MACTA: Environment



- Fixed episode length: Max Step=64
- **Detector (D)**

To find out whether there is an attacker as soon as possible

- **Attacker (A)**

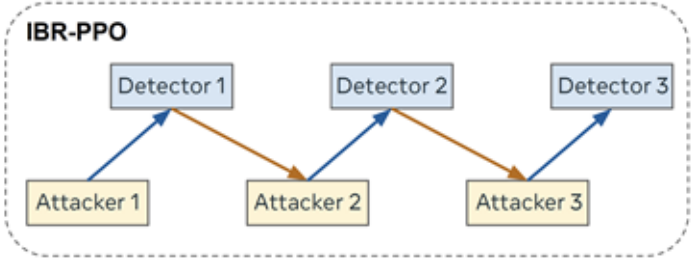
To attack (guess victim's secret) as many times as it can before the detector finds out

- **Victim (V)**

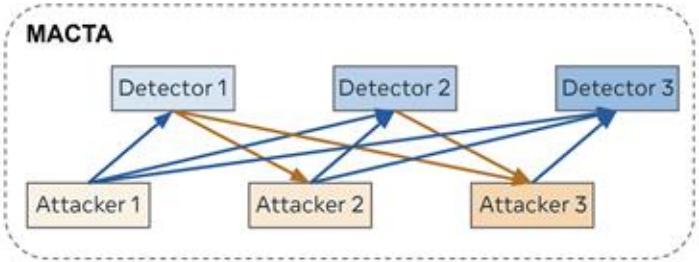
It has a multi-bit secret address, which is the target of the attacker

MACTA Key Concepts

- Transformer observation encoder
- Maintain a policy pool for each agent and increase the pool size with policy checkpoints during training
- Approximate Best Responses to a uniform mixture of opponents using (Dual-Clip) Proximal Policy Optimization (PPO)



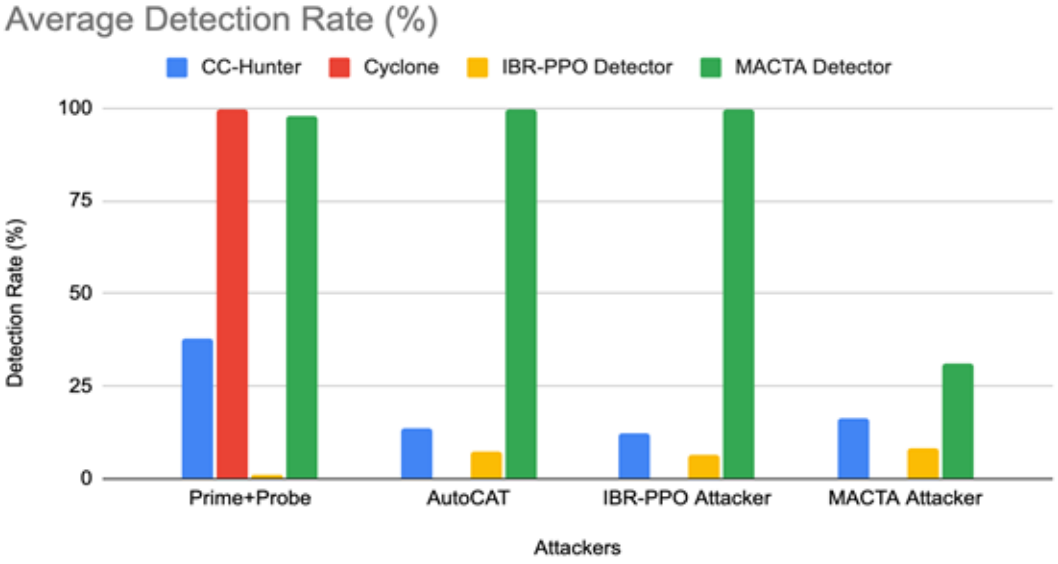
Iterated Best Response training could result in Cyclic Policies

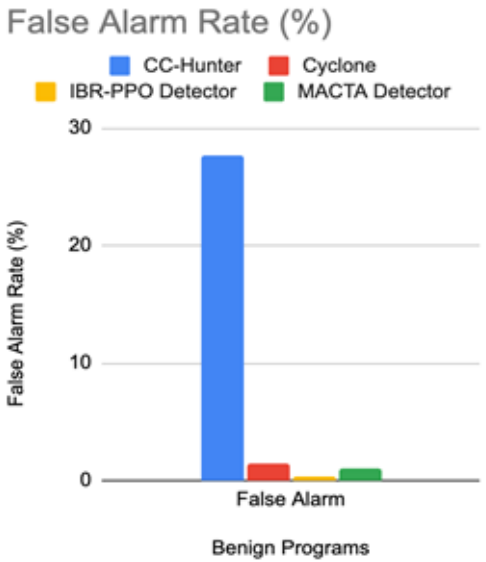


P → Q Q learns policy against P using PPO

$$\Pi^i_{\tau+1} \leftarrow \Pi^i_\tau \cup \{\pi^i_*(\mathbb{U}(\Pi^{-i}_\tau))\}$$

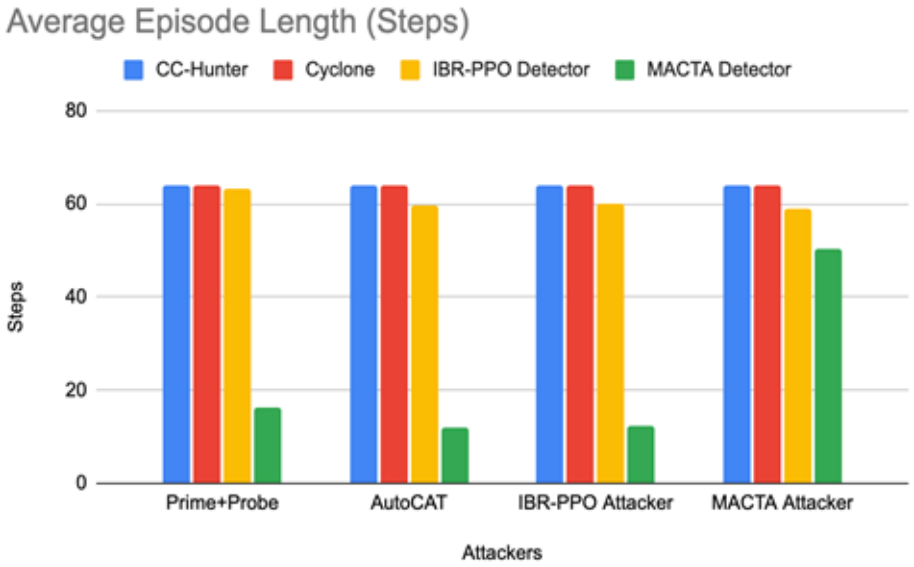
MACTA: Detector Evaluation





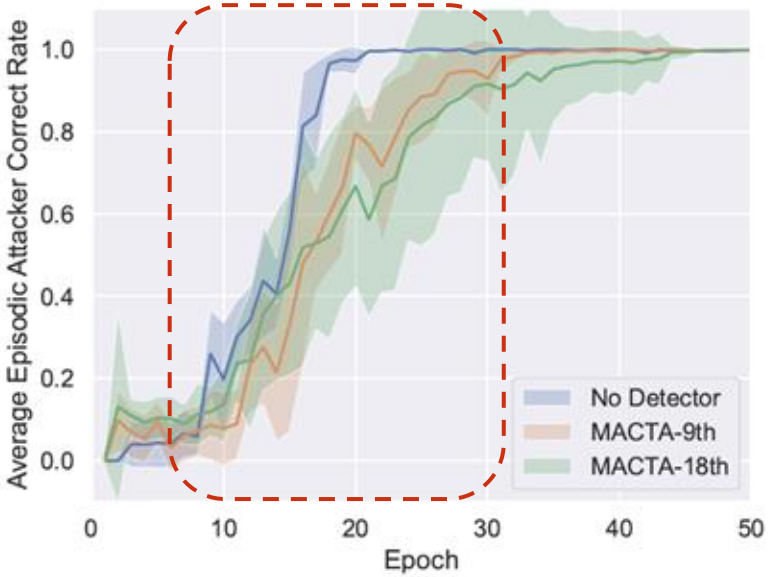
MACTA detector is able to outperform baseline detectors and **generalize** to **unseen attackers** while maintaining **low false alarm rate** for benign programs.

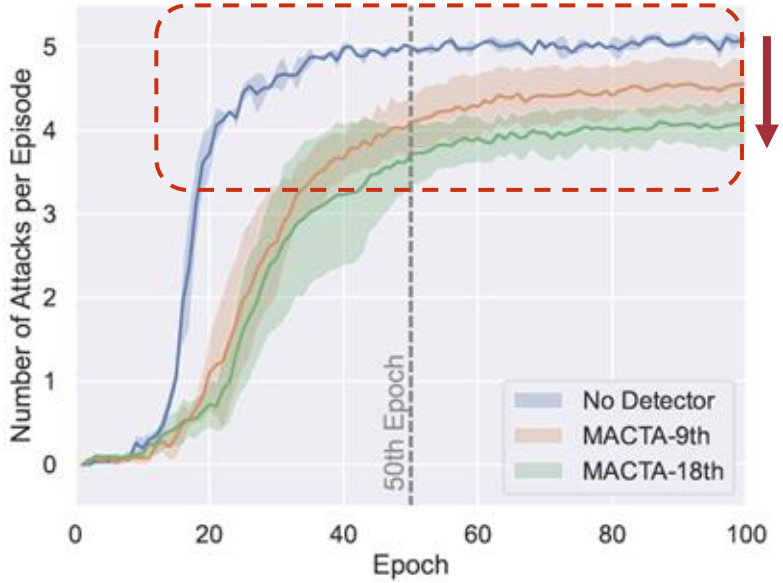
MACTA: Detector Evaluation



MACTA terminates attackers early to prevent further information leakage

MACTA: Detector "Exploitability"

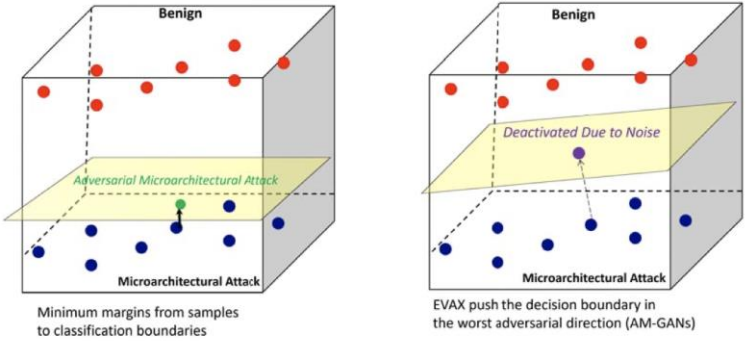


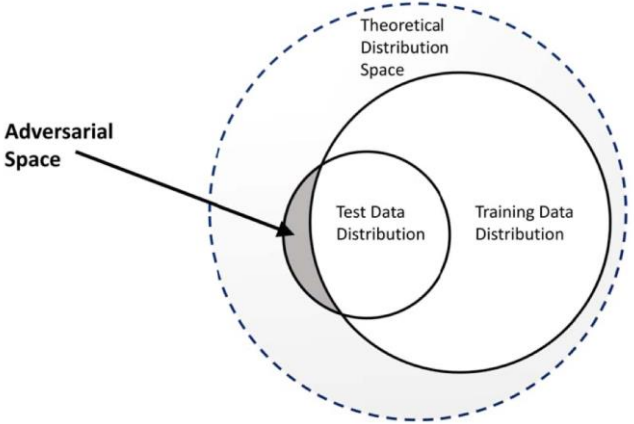


- Slow down future adaptive attackers' learning speed
- Reduces the information leakage against adaptive attackers

EVAX: Generative Adversarial Network for Detection

- Variant of attacks may not be presented in the dataset



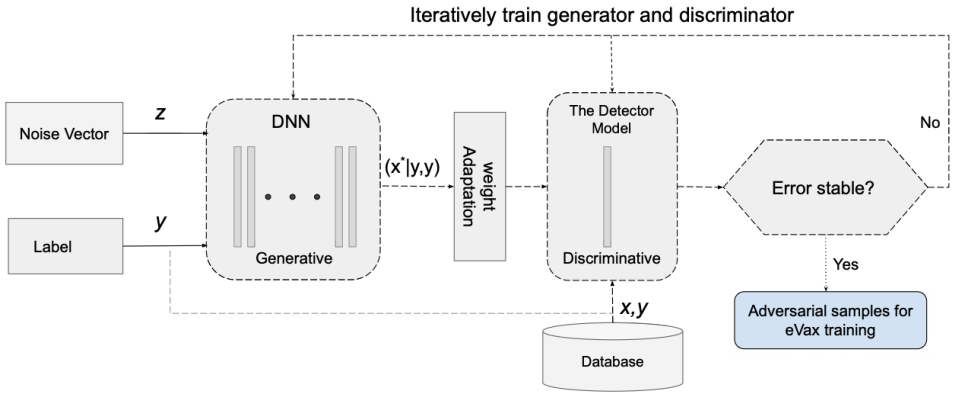


Large classification margin

ML vulnerability root cause

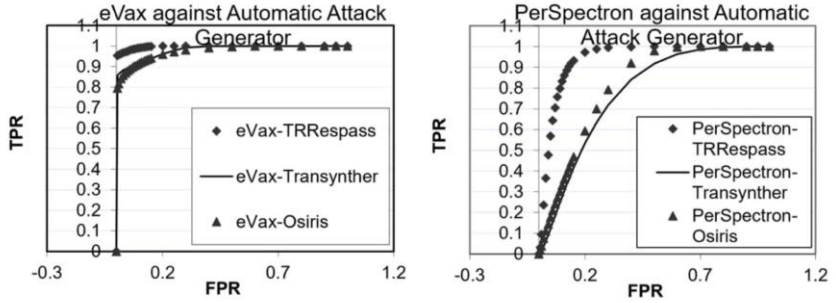
EVAX: AM-GAN

- generative adversarial network
  - Generator: generate attacks not in the database
  - Discriminator: detecto

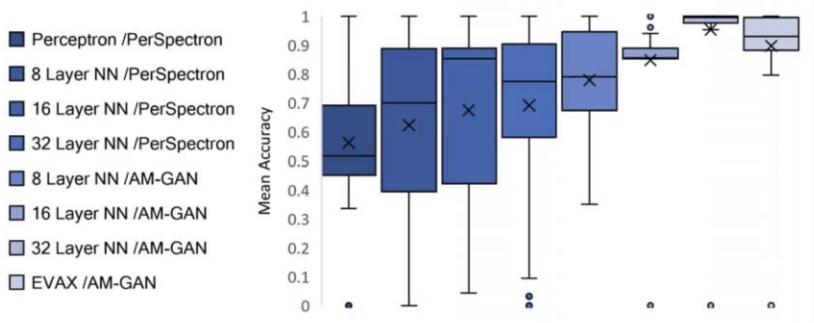


EVAX AM-GAN Training Diagram

EVAX: Evaluation of AM-GAN



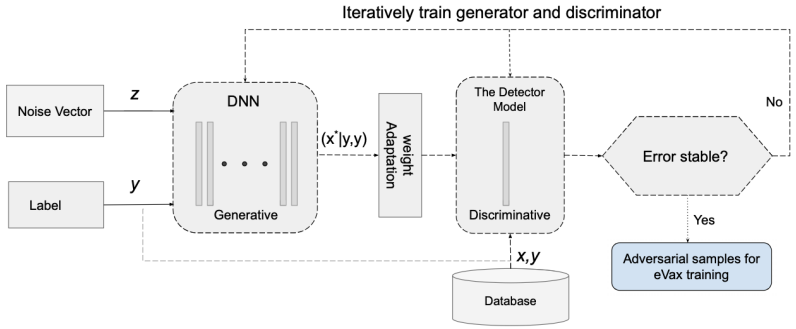
ROC curve against 1.2M Evasive attacks



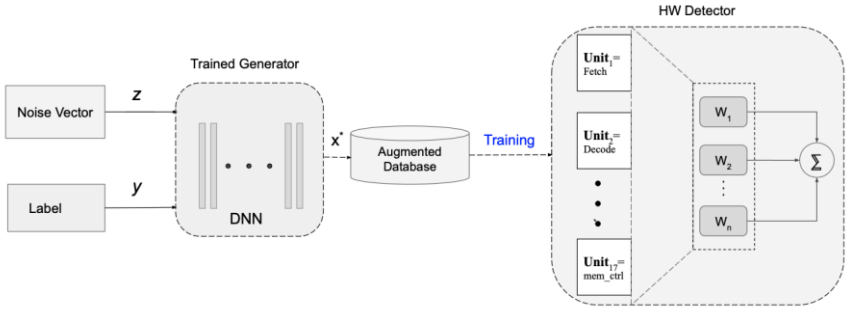
Improving other ML models with EVAX

EVAX: HW architecture

- Trained generator is used for augmenting the database



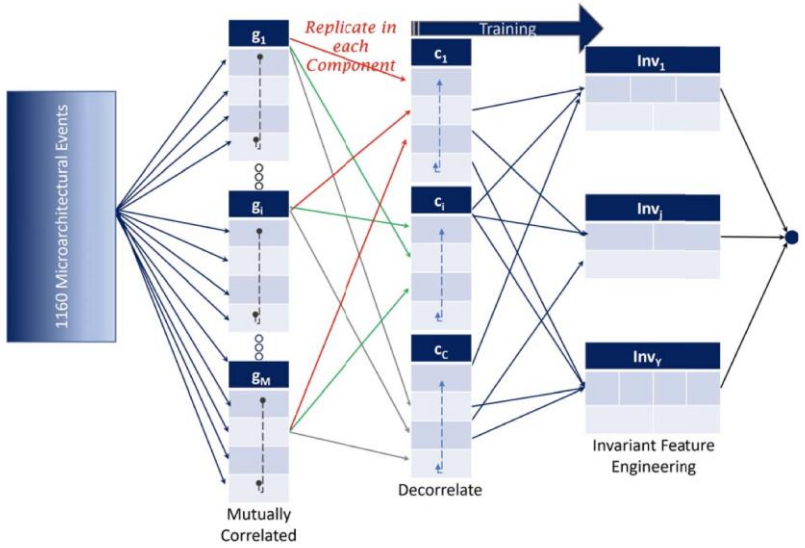
EVAX AM-GAN training diagram



EVAX HW detector training diagram

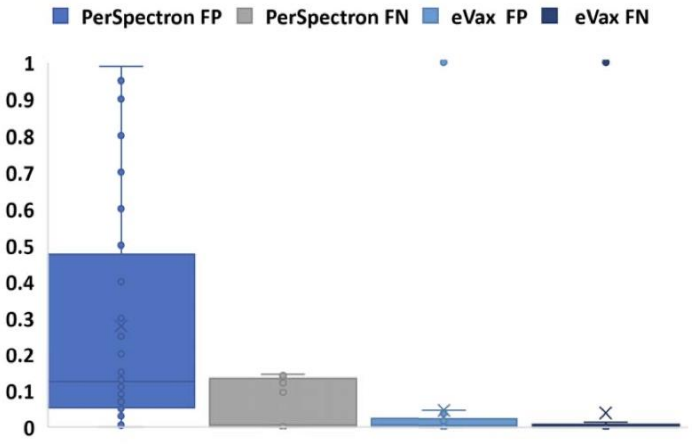
EVAX: Feature Selection

- Start with 1160 features
- Correlate features
- Find

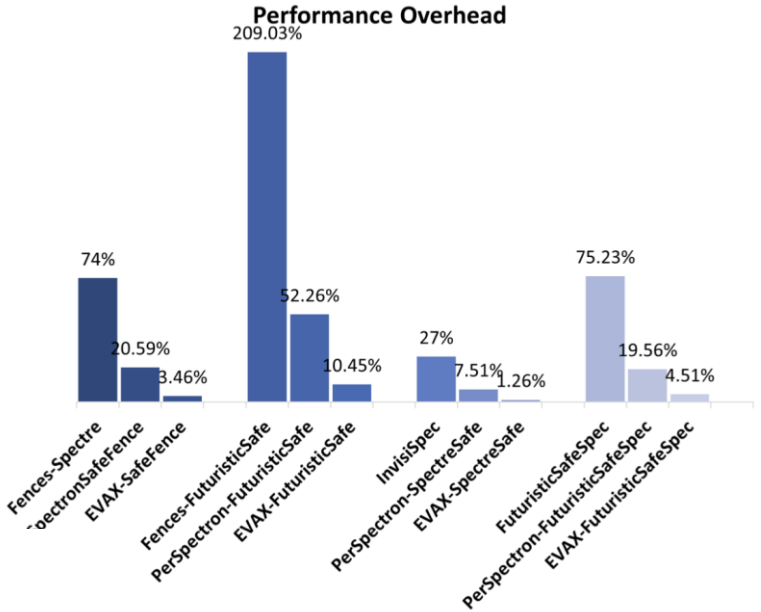


EVAX feature selection

EVAX: Benefit of Precise Detection



False positive and false negative dist



End to end performance comparison

AM-GAN vs MACTA

- Similarity
  - Both aim at unseen attacks
  - Both involve two parties during learning
- Difference
  - AM-GAN need to start with a known attack, while MACTA does not
  - AM-GAN first augment the data using GAN, then train the detector offline, MACTA train the detector online

Summary

- HW architecture for detection

Work	feature collection	ML models	Model implementation
Cyclone	Cyclic features	One class SVM	Flexible/SW model
PerSpectron	Hardware Performance counters	Perceptron	HW Perceptron

- Learning methods

Work	ML learning method	Known attack needed
MACTA	Multiagent RL	no
EVAX	GAN	yes

APRIL 2024

TEXAS
The University of Texas at Austin

LAB: REINFORCEMENT LEARNING FOR ATTACK PREVENTION

ASPLOS 2024 tutorial

https://ut-ldma.github.io

MULONG LUO
Postdoctoral researcher
SPARK Lab, The University of Texas at Austin
https://spark.ece.utexas.edu