Unlocking CPU telemetry for software identification

Work in Progress

Brendan Moran, Michael Bartling, Casey Battaglino, Archie Licudi
September 4, 2024

Side channels: a curse of modern computing?

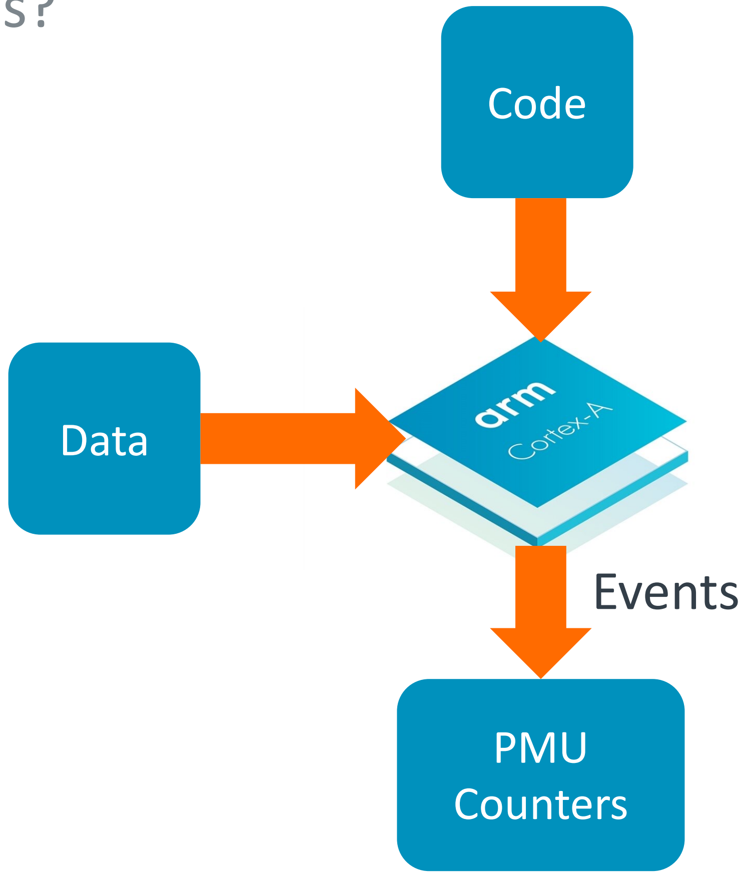
- Improving CPU performance means deviating from the programmer's model
- Deviations with state create side channels
  - Unintentional paths for information
  - Side channels are the inescapable reality of modern processors
- Attackers can use side channels to extract data
- Telemetry: outside the programmer's model
- What if we could turn the tables and use side channels to identify attacks?
  - One very promising side channel: hardware performance counters (HPC) (e.g. the Arm Performance Monitoring Unit (PMU) or the Intel Performance Counter Monitor (PCM))
  - Existing research in this area, but subject to limitations of extant hardware

What is the PMU?

Are Performance Monitoring Units side-channels?

- Collects events from the CPU core, e.g.:
  - Loads/stores
  - Branches taken
  - Instructions retired
  - Cache misses
- Aggregates events in a few counters
- Allows profiling of code

- Profiled code shows behavior
- Enables timing attacks
- Enables other profile-based attacks



Pitfalls of PMUs for software analysis

- External sources
- Non-determinism
- Overcounting
- Variations in tool implementation
- Loss of temporal ordering

- Needs very high accuracy to be deployable
  - Accuracy limits operational relevance:
    - 5% FPR means 1 in 20 samples are misclassified as malware
    - 80% TPR means 1 in 5 malware attacks are missed

SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security

Das et al, Oakland 2019

PMU, but for security?

Which constraints/pitfalls could we avoid?

- What if we redesigned the event processing hardware, to be used for security?

- External sources
  - Cache behavior necessarily multi-process: system-level behavior monitoring
- Non-determinism / overcounting
  - Architectural events are deterministic => use these
- Variations in tool implementation
  - Correctible, but arguably not what we're looking for.
- Loss of temporal ordering
  - Fundamental to counter architecture => do not use counters

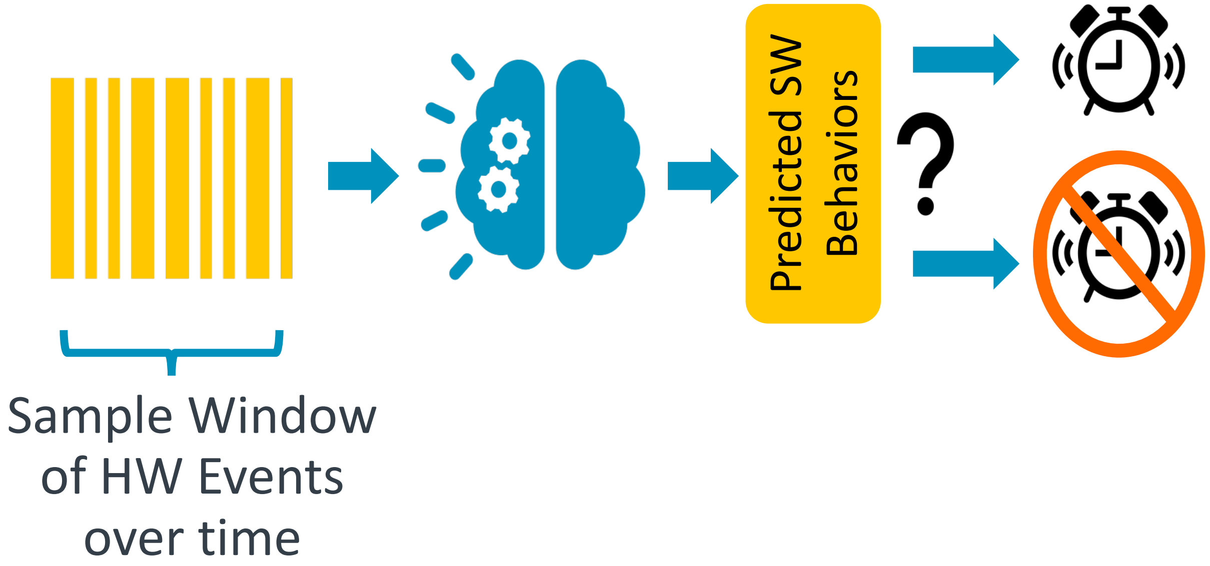
Program behavior as indicator of identity

- Programs contain substantial subdivisions in complexity
  - Basic blocks and sequences of basic blocks make up functions
  - Functions make up modules
  - Modules make up programs
  - Program execution often takes a circuitous route through all of these
- More detail presents further challenges
  - Preserving detail at context switches
  - Noise is not filtered out as effectively
  - Data, bandwidth, memory, compute load increase
- Options for extracting sufficient detail
  - Long windows might provide enough detail for direct classification
  - Short windows could be classified and used as the input to a second-tier classifier

Akira Architecture

Akira's approach

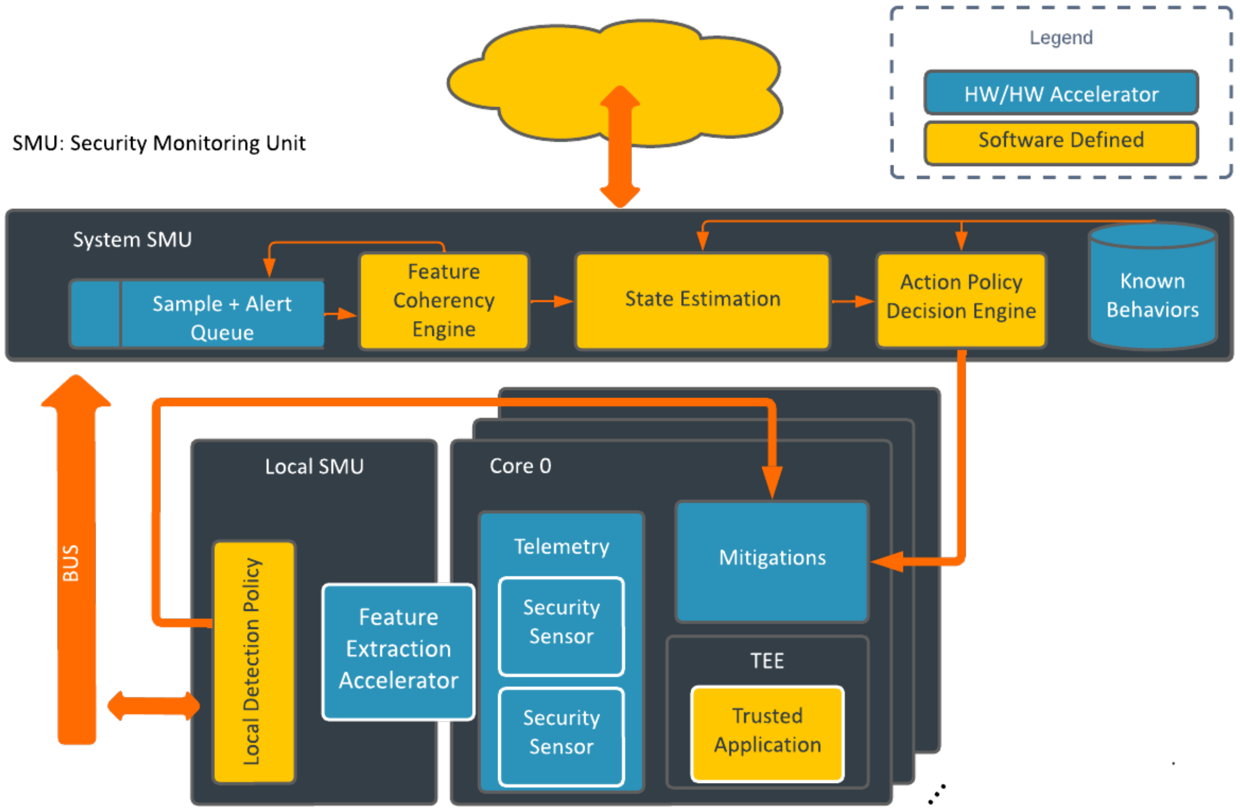
- Local low-power always-on TinyML
  - "Listen" for software behaviors
  - Enable mitigations / throw alerts when needed
  - "Respond" to emerging threats for years
- State of the Art uses standard or augmented PMU
  - **Events** are **aggregated** by PMU prior to analysis
  - E.x. 34 cache misses and 10 branch mispredicts since last time sample
- Akira processes Raw Telemetry Events
  - E.x. 1 [new] cache miss @ cycle: 5678



Akira Architecture: Tiered approach

On-SoC: The blocks

- Telemetry and Mitigations highly privileged blocks!
  - Careful to protect a high-resolution side channel
- Local Security Monitoring Unit (SMU)
  - On or near core
  - Custom feature extraction accelerator HW
  - Need access to a secure ML Accelerator
    - Updateable with lifecycle
  - Triggers basic mitigations if confidence high enough
  - Emits alerts + feature summarizations
    - Benign features sampled at random
- System SMU
  - Aggregates across local SMUs, while maintaining temporal consistency
  - Correlates samples w/ system context and known patterns to estimate
    - 1) what samples are “linked” or related
    - 2) and to what behaviors
  - Proposes course of action (or not)



Temporal Ordering

- Raw telemetry events
  - More detail for analysis
  - Departure from established practice
  - CPU designer needs to implement
- Impact on non-determinism
  - Multi-issue, superscalar reordering, speculative execution produce noise
- Impact on bandwidth
  - Counters: $m \log_2 n$ bits/window
  - Event trace: mn bits/window

- A transform may help
  - Path Signatures [Chen 1958, Chevyrev 2016]
    - Good properties. Resistant to:
      - Offsets
      - Timing variation
      - Warping
      - reparameterization
  - Bandwidth: $\left(\sum_{i=1}^{L} im^i\right) \log_2 n$ bits/window
    - More bandwidth than counters, less than trace
    - dependent on
      - event sources m
      - window size n
      - signature depth L
- $S(X)_{a,t}^{i_1,\ldots,i_k} = \int_{a<t_k<t} \cdots \int_{a<t_1<t_2} dX_{t_1}^{i_1} \cdots dX_{t_k}^{i_k}$
- $S(X)_{a,b} = \left(1, S(X)_{a,b}^1, \ldots, S(X)_{a,b}^d, S(X)_{a,b}^{1,1}, S(X)_{a,b}^{1,2}, \ldots\right)$

Akira's approach: Compress instead of aggregate

- A limited set of events routed into a feature extraction accelerator
- Accelerator for 5 event sources has negligible area cost
- 5 event sources generates a 55-element vector (log signature)

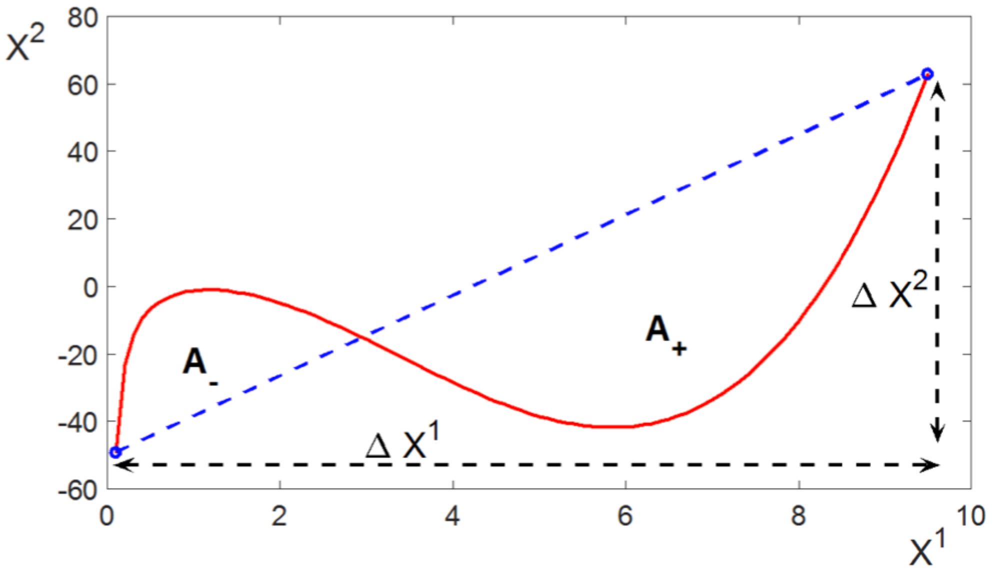
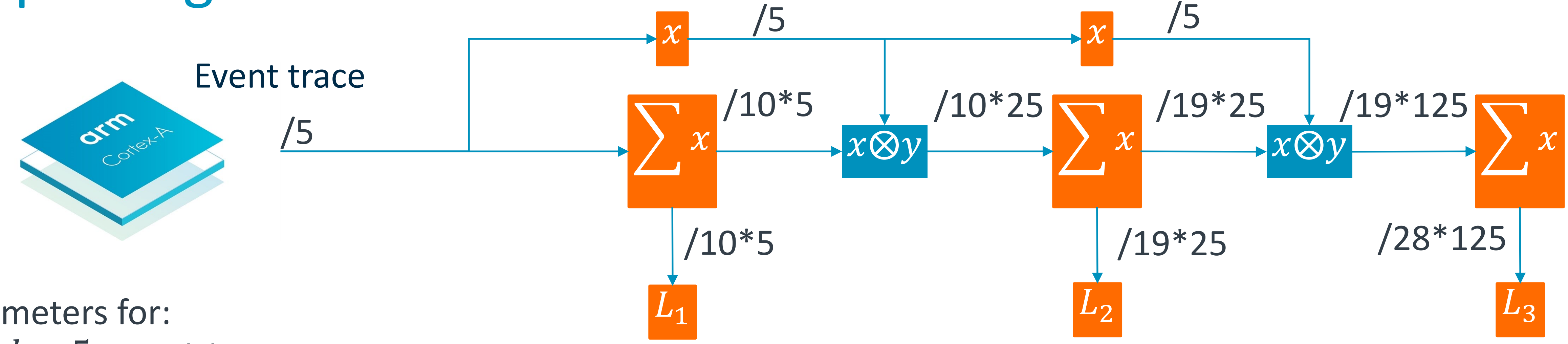


Figure 5: Example of signed Lévy area of a curve. Areas above and under the chord connecting two endpoints are negative and positive respectively.

A path signature accelerator



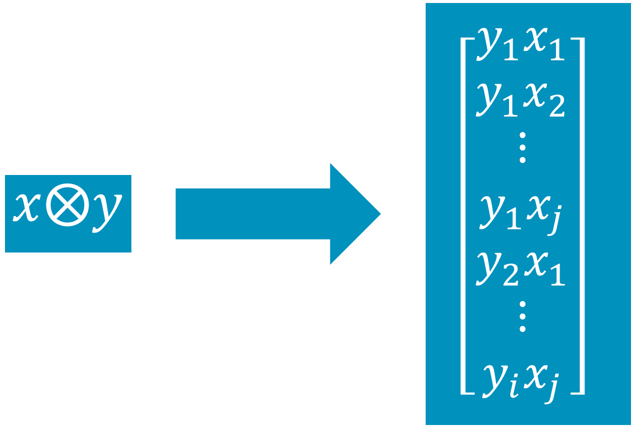
Parameters for:

$d = 5 \; event \; types$

$n = 1024 \; events/window$

Combinatorial elements

Sequential elements



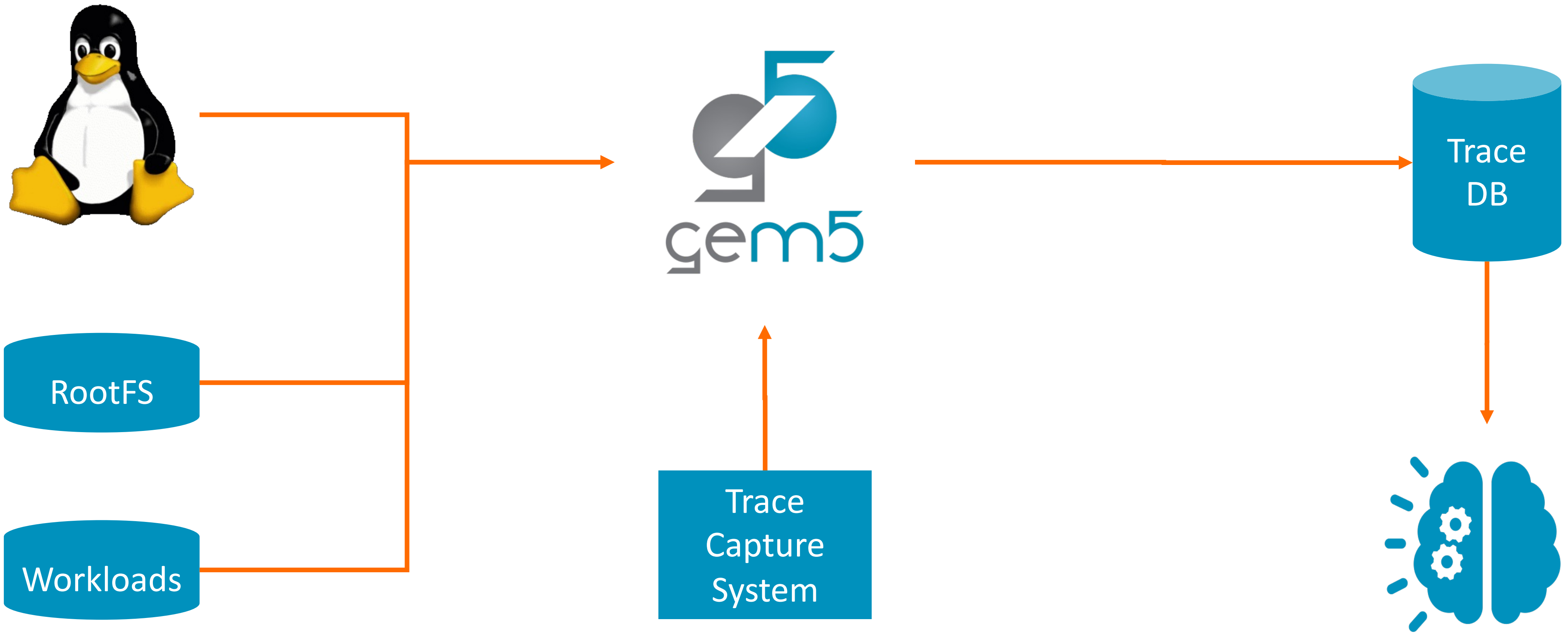
$x \otimes y$ can be implemented by gating each element of y on each element of x, forming an outer product

Experimental Methodology & Early Results

Early answers about early warnings

Data collection setup

Akira Simulation Infrastructure



Akira: The Data

Trace Capture

Goal – Know exactly when and where things run

- Modified Gem5 instances
  - Direct PMU event capture in CPU
  - OS PIDs pushed to CPU regs
  - Magic NOPs for marking windows
  - Running bare metal + booting Linux
  - Capturing PC trace for labelling

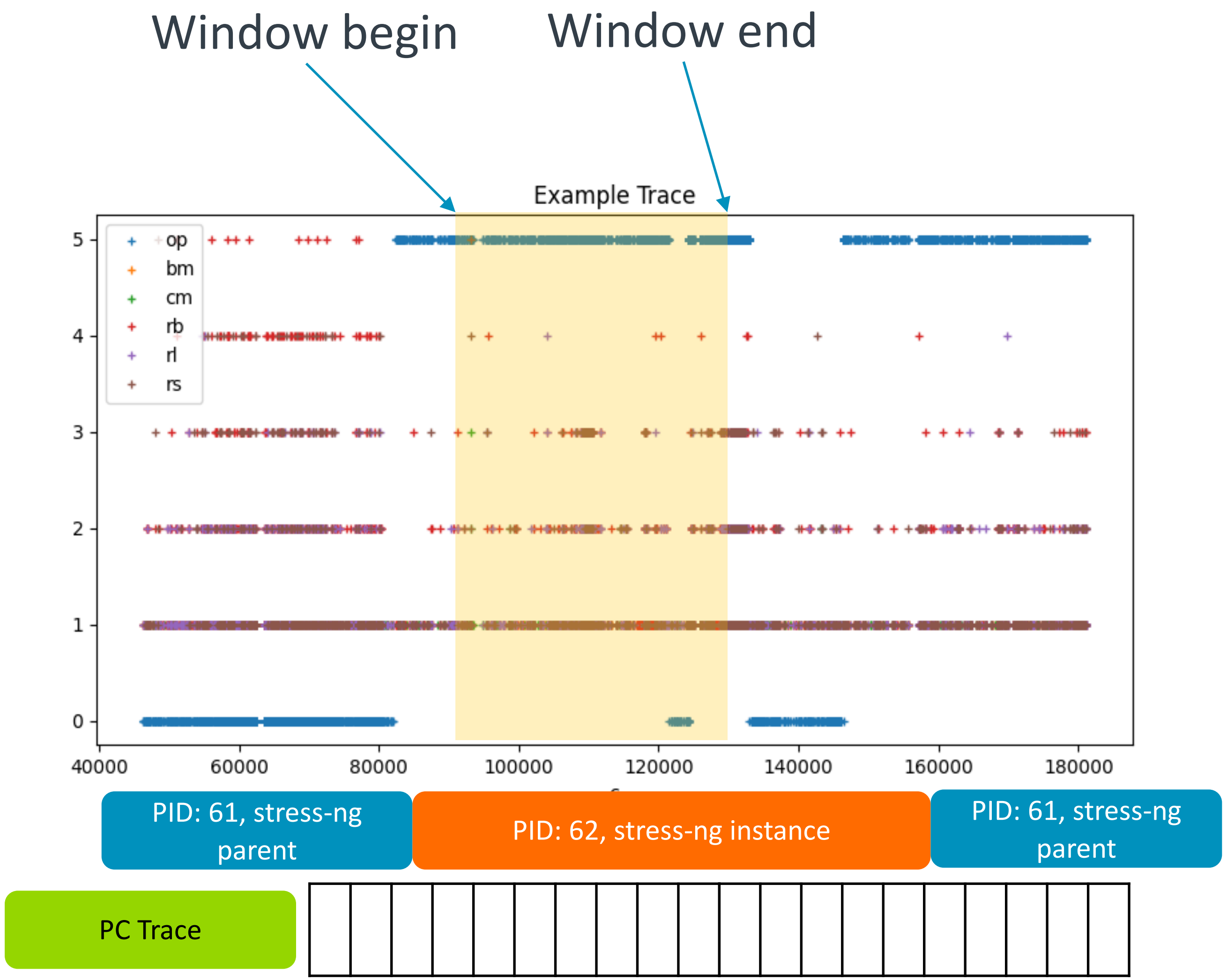
Workloads (More on this later)

- Stress-NG
- Juliet
- SpectreV1

Linux Testing Project

IARPA StoneSoup (SSL)

SPEC 2017

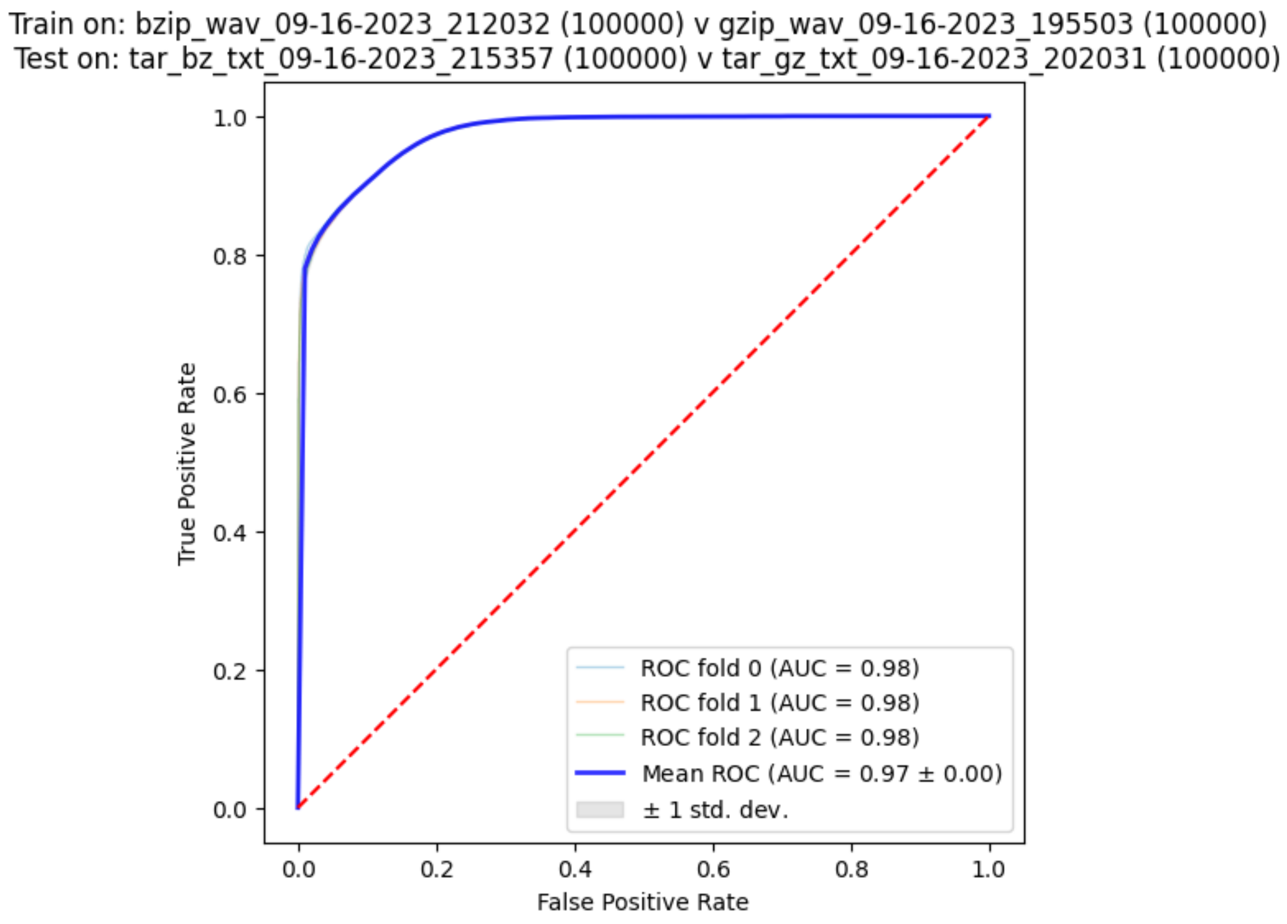


Identifying programs

Interim Results – Program Identification

gzip vs bzip2

- Trained on gzip, bzip2 of wav file
- Tested on tar -z, tar -j of text file
- Model:
  - XGBoost
  - Path Signature feature extraction
  - Long windows (512)

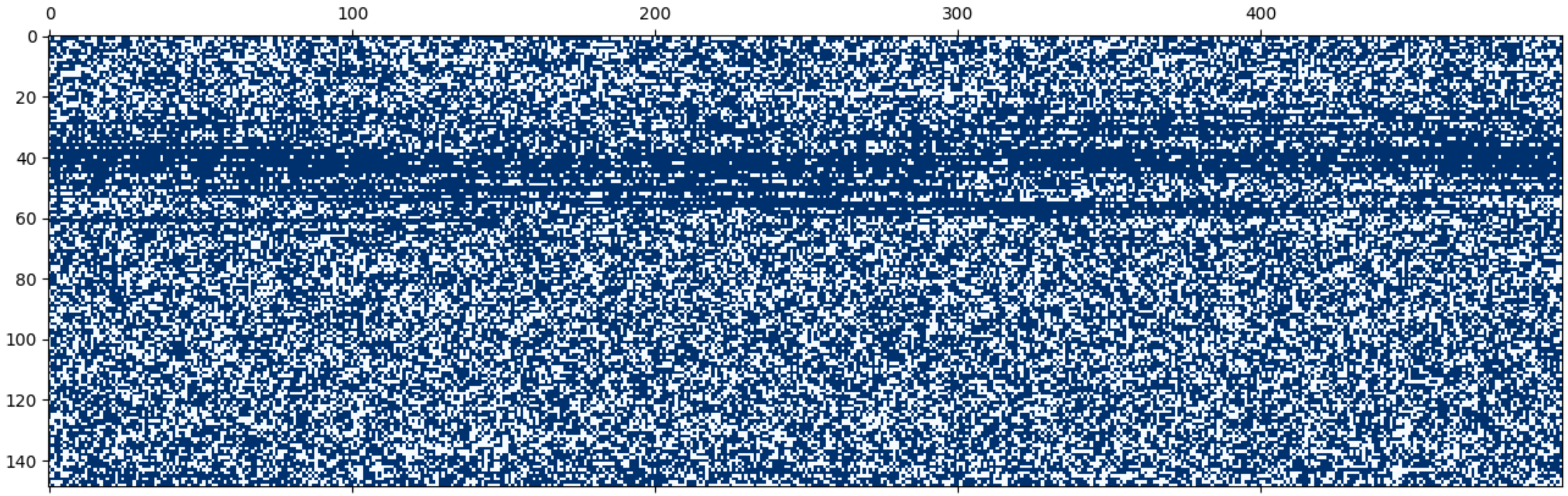


Linux kernel behavior identification

Interim Results – Kernel Function Identification

Classifying kernel behavior by PMU events

- Ran long-running Linux Testing Project test cases
- Filtered via Exception Level
- Matched PC Trace to kernel functions
  - 84 behaviors used often enough to be relevant
- Ranked possible classes of test sample (25-event windows, XGBoost, path signatures)
- Accuracy
  - 97.5% top-1
  - 99% top-4



'LTP_fcntl27_06-07-2023_204644'

LRAP: Label Ranking Average Precision

Evaluating Akira’s multi-label ranking behavior

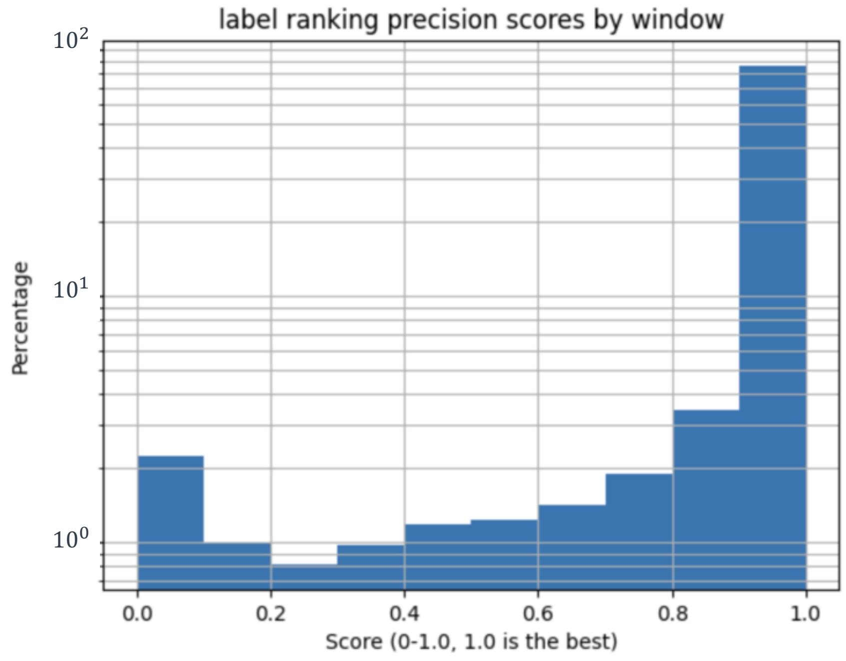
- Rank samples by most likely behavior “class”
- Identify how many false classes are ranked below ground truth for each sample
- Average these results across all samples for a given test
- May be multiple ground truths for some samples
  - i.e. One sample window == mix of software "behaviors"
  - For better score all the ground truths in a sample should be nearer the top rank

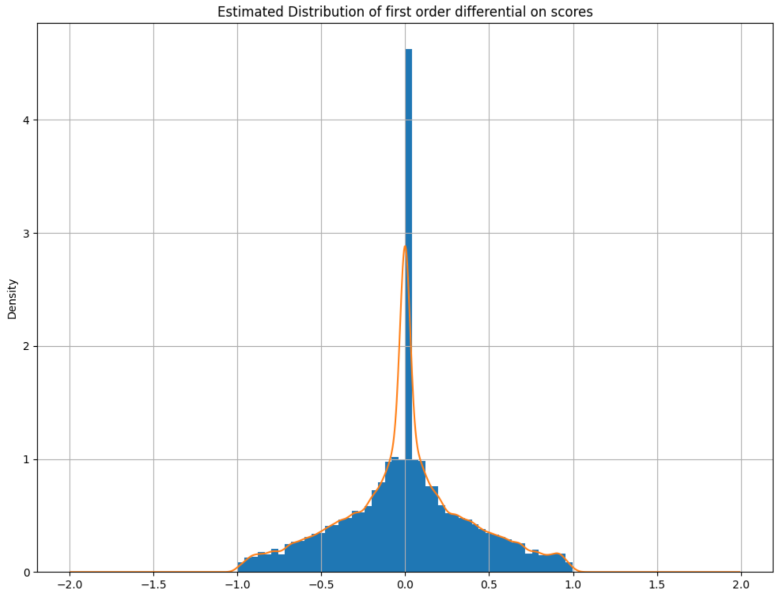
- Practical implications:
  - Real-world classifier will consume top-N ranked classes for each sample and perform disambiguation.

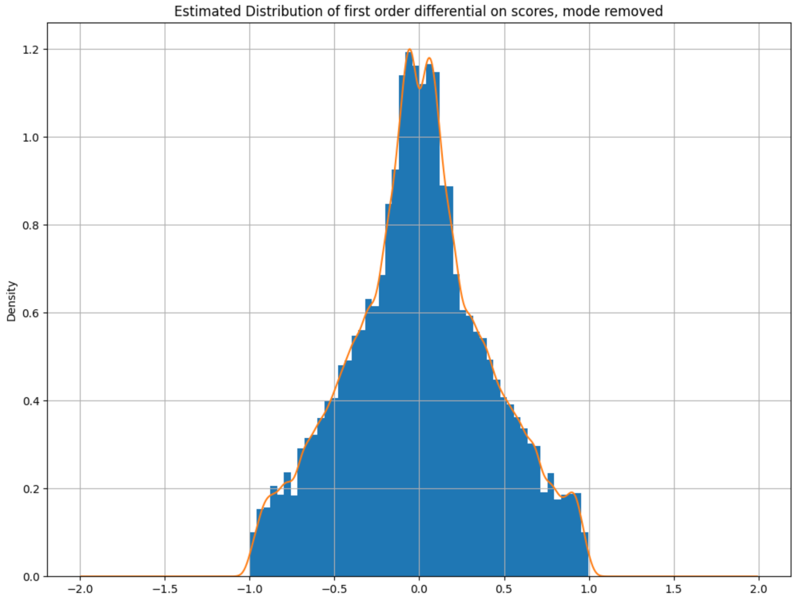
Interim Results – Kernel Function ID

Can we go deeper?

- Q: Can we identify specific code paths in software using telemetry?
  - We can distinguish between 1372 kernel functions with >85% "accuracy" using samples **only 25 events long**



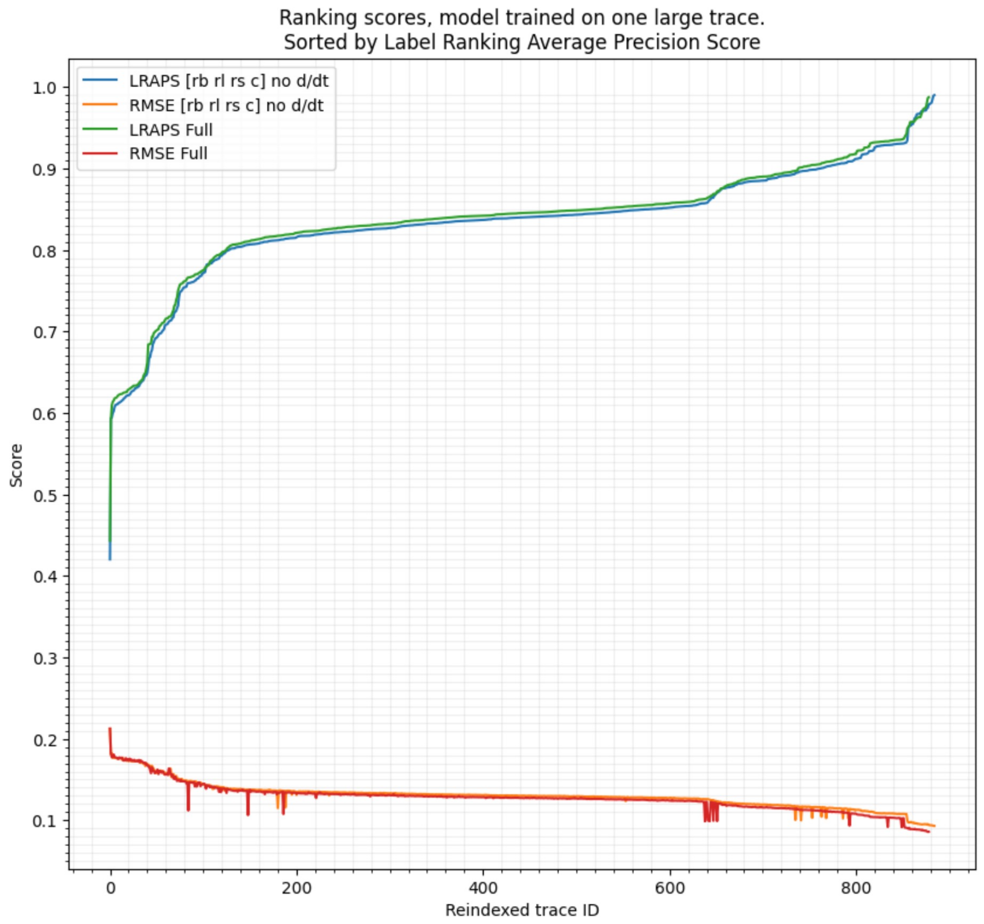




Interim Results – Generalizing Kernel Function ID

How well does training on one trace generalize to other traces?

- **Training:**
  - Ingest sampled form of one large trace (v4.7.1)
    - Treat as a lower bound on model performance
  - Target extremely small models and learn 2 rank
    - 25-event windows, path signatures, XGBoost
  - See if we can correctly identify top ~60 kernel functions
- **Testing:**
  - How well does this model work on all other traces (v5.12.2)?
    - Hint: it's very good

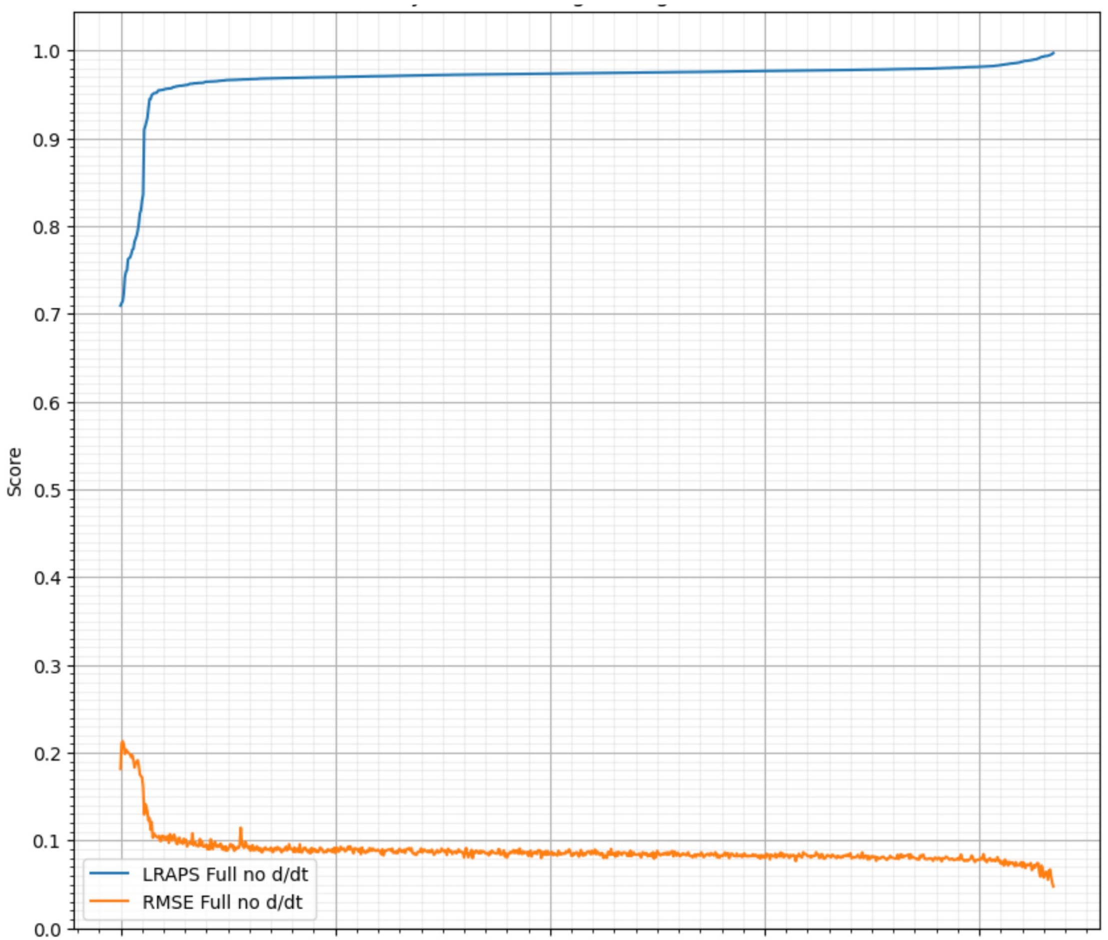


Userspace code identification

Results – User space code classification

Does the kernel-space classification system work on user-space code?

- Top 20 user-space labels in the 969 LTP traces
- Training:
  - Windows containing target set labels for at least 25% of duration
  - No “unknown” labels
  - 25-event windows, path signatures, XGBoost
- Testing:
  - holdout set
  - LRAPS is 97.2%, per window
  - RMSE is 8.7%, per window

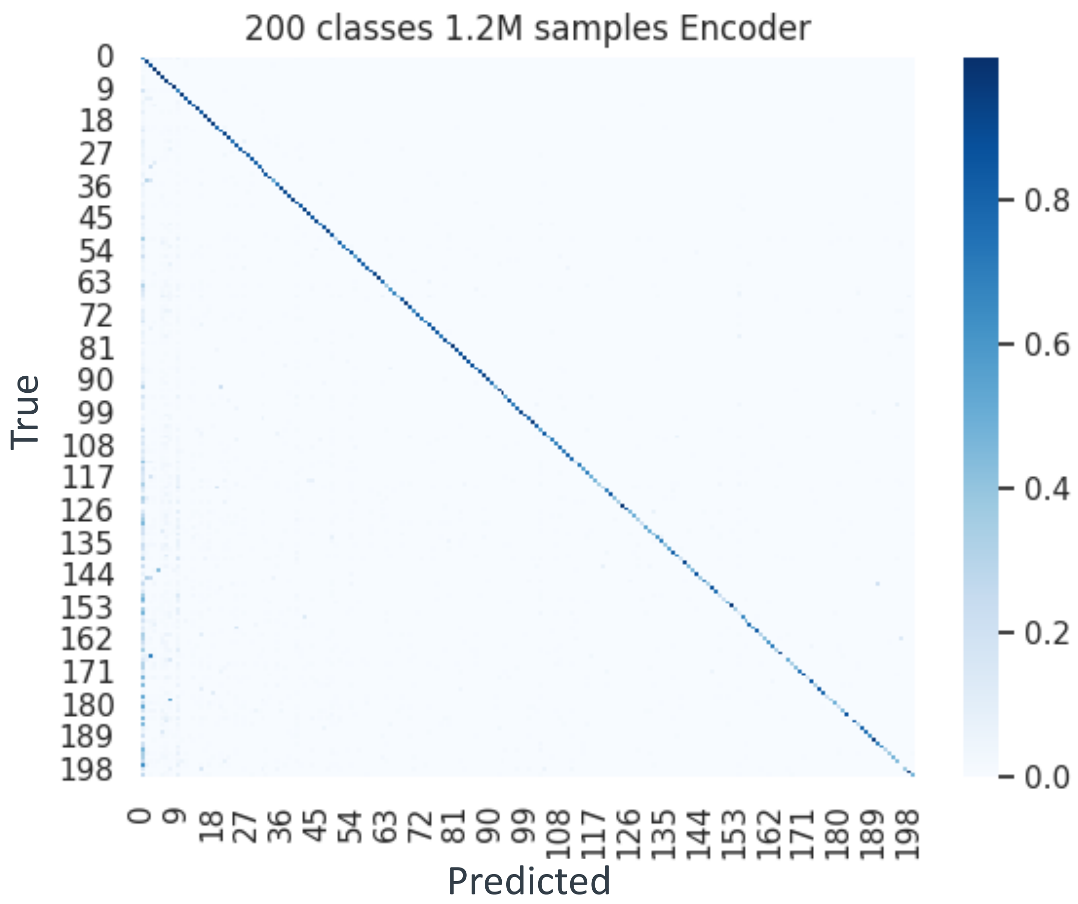


General behavior identification

Distinguishing between code segments

Using a transformer to identify code based on behavior

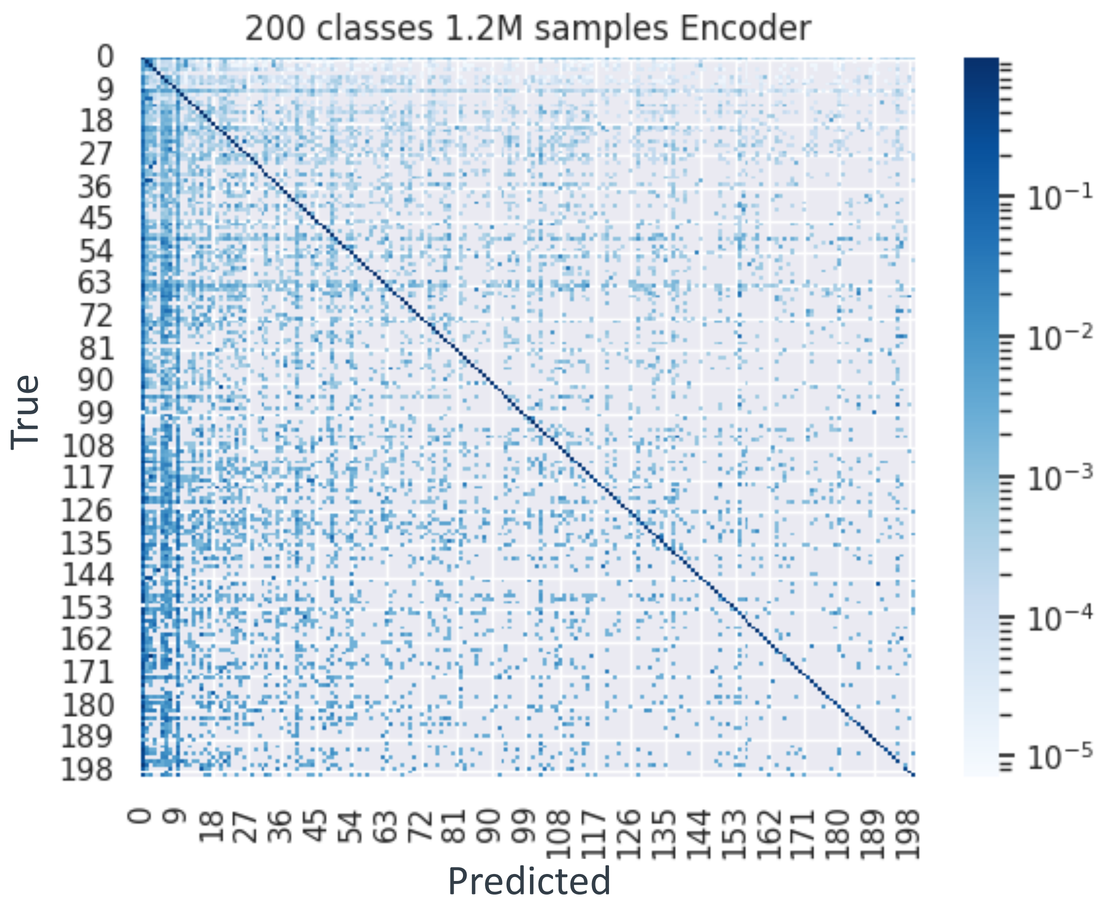
- Transformer is trained on 1.2M samples with the top 200 classes highlighted
  - 969 traces are sampled, with a training set and a holdout set
- Sources of misclassification:
  - Each sample labelled with top-1
  - All classes not in the top 200 are binned into class 0
  - Most confusion occurs when the model predicts class 0 (the catchall class)
- Avenues for improvement:
  - Top-n labelling
  - More classes (fewer in catchall)



Distinguishing between code segments

Confusion matrix in log-scale

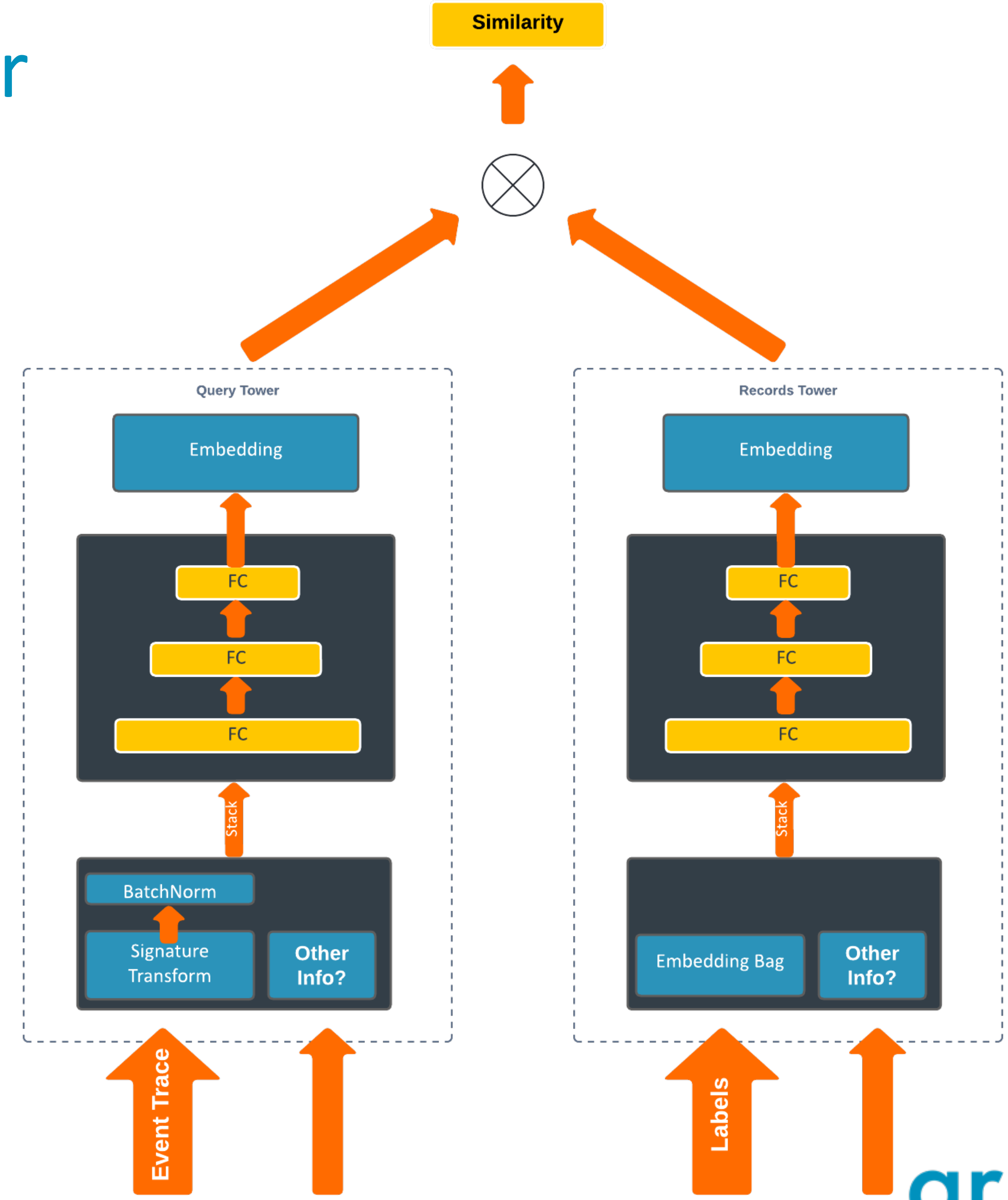
- Most incorrect predictions < 1:10
- Clear bias towards more common classes
- Misprediction rate improves substantially below class 10
- Avenues for improvement:
  - Improve sampling balance



Handling imbalance

Two Tower Candidate Generator

- One Tower encodes the measurements
- Other Tower encodes our knowledge of this record
- Model pairwise trained
  - "Sample related to Record"
- True validation is maximum inner product search
  - Expensive, but for science!
- In practice, inference leverages vector DBs
- Can add info to input vectors for cheap



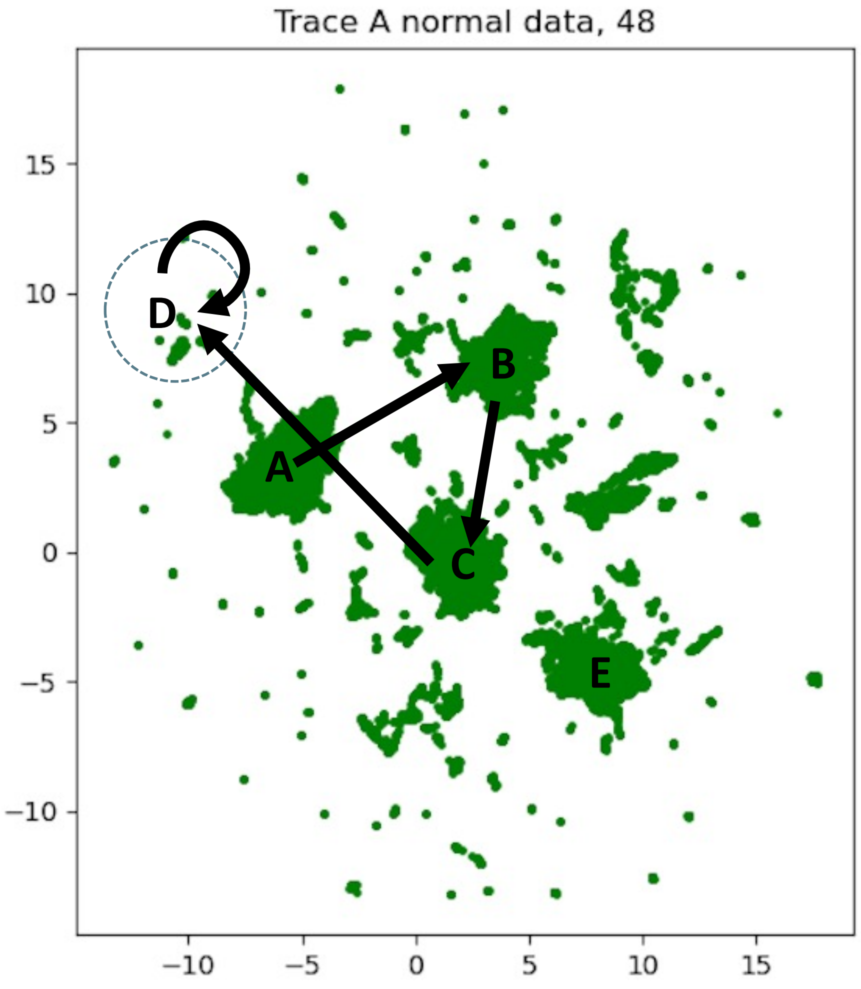
Two Towers (cont.)

- Training scales well to massive datasets
- Models are extremely flexible
  - Depending on data concatenated to input vectors we can handle different tasks
  - Missing sections can just be zeroed out
  - e.x. embedding encoded basic blocks -> identify/group data dependent behaviors
  - e.x. embedding local graph of basic blocks -> guide model towards ontologies
    - Similar to playlist learning
  - e.x. embedding doc strings for known benign code bases
  - e.x. embedding description metadata for known malicious samples (think VirusTotal)
  - e.x. embedding syscall trace
- Well suited for both cloud and system-level applications
- Currently in-progress
  - Sorting out details in the training

Second tier classification

Goal: use cluster transitions to identify software

1. Mini-batch k-means: Unsupervised labeling of all important clusters across LTP (and beyond)
2. A Trace is then a path (a 'string') between these clusters
3. We have efficiently applied this to the entire LTP data set (takes ~1 minute)
4. The resulting output has allowed us to classify program types in the LTP data set with good accuracy (75% top-1 accuracy on 12 classes, using fully unsupervised features)



String representation

Trace *i*: "ABCDD"

Freq representation

A	B	C	D	E
1	1	1	2	0

Creates a Term-Frequency structure:
Can analyze with Topic Models

Transition freq representation

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	1	0
D	0	0	0	2	0
E	0	0	0	0	0

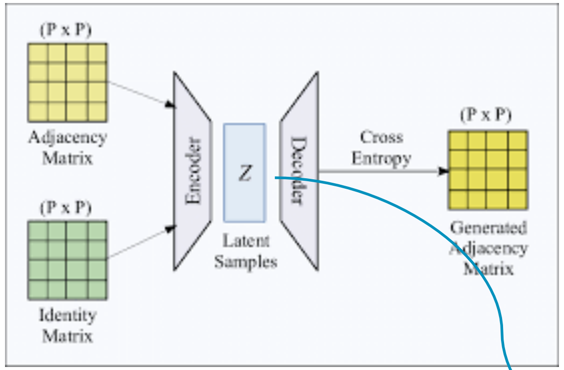
Creates a graph structure:
Can analyze with GNN, TDA

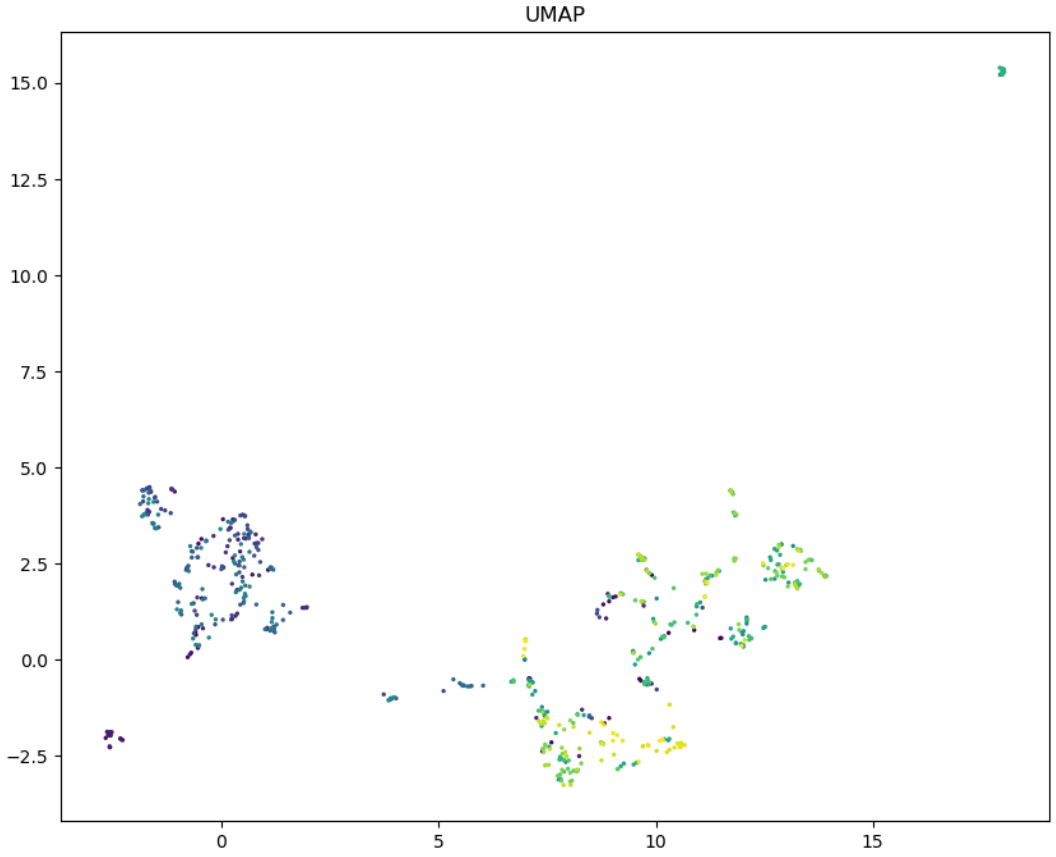
Graph Autoencoder on Soft-Label Adjacency Matrix

Transition freq representation

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	1	0
D	0	0	0	2	0
E	0	0	0	0	0

Variational Graph Autoencoder





Points are colored temporally (window ID)

We can identify coarser phases within a trace

Conclusions & Future work

Challenges

- Deployment of a high resolution performance monitor requires action by CPU designers
  - CPU designers need strong evidence for adding a feature like this
  - Providing evidence for this feature requires extensive data collection
- Data collection without dedicated hardware can only be done in simulation
  - Collecting data in simulation is very slow
  - Running malicious software in simulation presents unique challenges

Conclusions

- The PMU provides high resolution data about software
  - As long as we keep ordering information
- Event information can be used to identify software
- Path signatures retain ordering information, but reduce data size
- Path signatures are low overhead in hardware
- Event traces can identify short behaviors with high confidence
- Sequences of short behaviors can provide identification “strings”

References

References

- Chevyrev, Ilya & Kormilitzin, Andrey. (2016). A Primer on the Signature Method in Machine Learning.
- Chen, K. T. (1958). Integration of paths--A faithful representation of paths by noncommutative formal power series. Transactions of the American Mathematical Society, 89(2), 395-407.
- Das, S., Werner, J., Antonakakis, M., Polychronakis, M., & Monrose, F. (2019, May). Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 20-38). IEEE.

Acknowledgements

- This material is based on work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) under agreement number W911NF20C0045.

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు