SAND2023-03640C

Machine Learning Based Analytics Towards Automated Computing System Management

BY **AYSE K. COSKUN,[1] BEN SCHWALLER,[2]** BURAK AKSAR[1], EFE SENCAN[1], VITUS J. LEUNG[2], JIM BRANDT[2], MANUEL EGELE[1], BRIAN KULIS[1]

[1]Boston University; [2]Sandia National Laboratories

July 9-13 , 2023
Moscone West Center, San Francisco, CA, USA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solution of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525

Presenters

- Prof. Ayse Coskun - Boston University
  - Director, Center for Information and Systems Engineering
  - Research interests:
    - Energy-efficient computing
    - Cloud computing
    - High performance computing

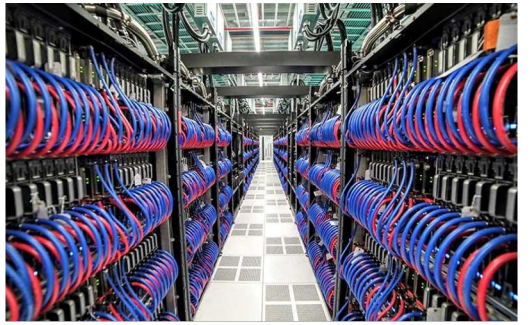
- Ben Schwaller - Sandia National Laboratories
  - Senior Computer Science R&D Staff
  - HPC development analysis and visualization lead
  - Lead for monitoring, metrics, analytics, and integration in Lawrence Livermore / Los Alamos / Sandia

Section 1:
The Nature of Computer System Telemetry Data

- What is High Performance Computing (HPC)?
  - What are common challenges in using large / distributed computing systems?
- Monitoring and analysis frameworks for large computing systems
  - Lightweight Distributed Metric Service (LDMS)
  - Overhead and resource usage tradeoffs
  - Data format / complexity
- Challenges in analyzing computing system telemetry data
  - What makes this a hard/interesting data science problem?

What is HPC?

SNL's Astra Supercomputer

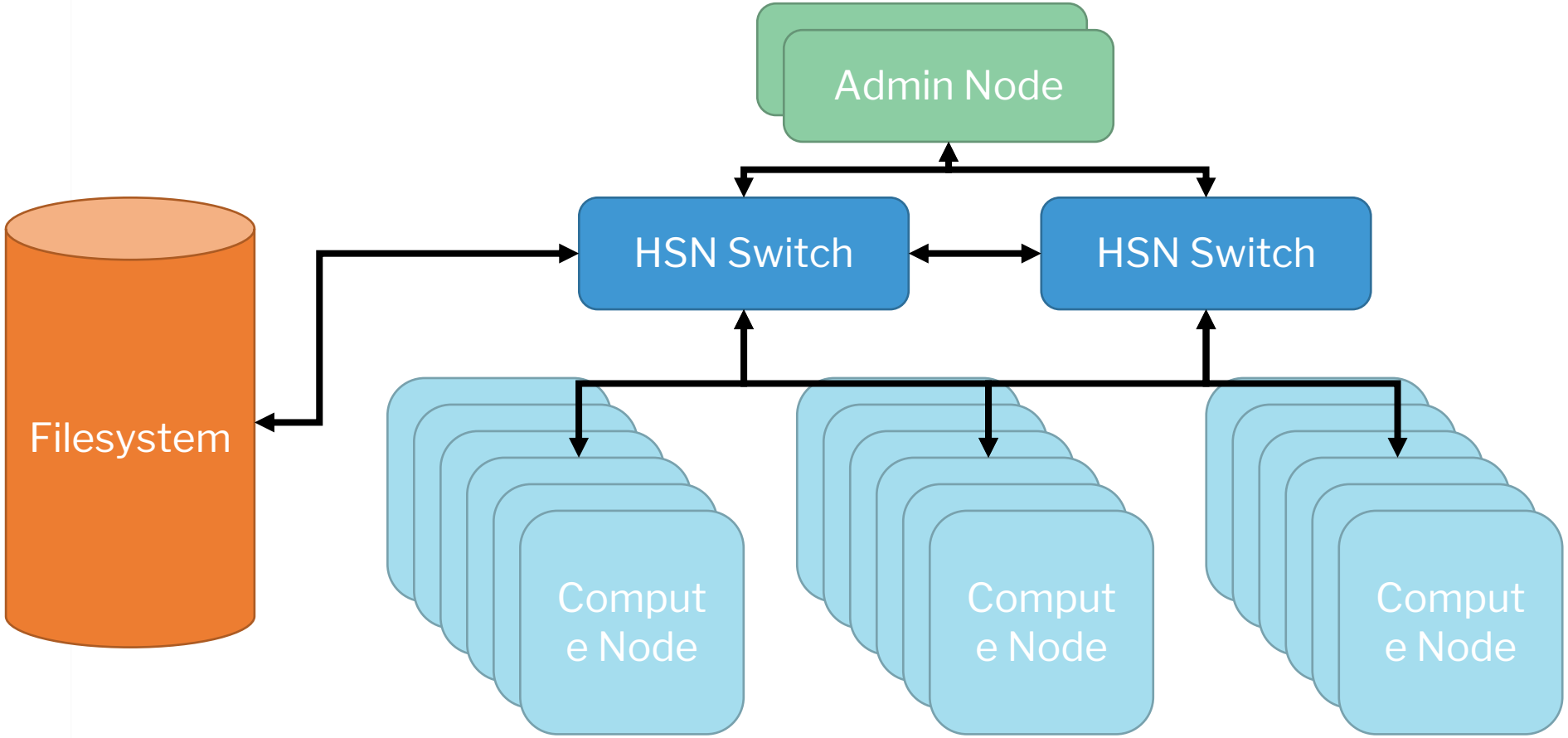

Frontier's racks with liquid cooling

- High-performance computing (HPC) systems are designed to enable greater performance than single commodity systems
  - Typically done by harnessing several single computers into a "cluster"
  - Current top supercomputer, Frontier, can perform ~1.7 quintillion operations per second
- HPC systems provide answers to some of the toughest science questions
  - Detailed weather simulations and forecasting
  - Drug development and disease forecasting (like simulating the spread of COVID)
  - Physics simulations ranging from material science to electromagnetic radiation to biological system modelling

How does HPC work?

- Applications run "in parallel" across multiple nodes of the cluster
  - A single application can run across multiple nodes (using MPI)
    - Pieces of the work to be done in the application are distributed across nodes
  - Multiple applications can run simultaneously on the system
- Nodes communicate with each other using high-speed networks
  - E.g. the popular InfiniBand boasts 400 gigabits/second throughput
  - Network bandwidth is shared amongst nodes / applications
- Applications typically pull input data from and save output data to large filesystems
  - Filesystem bandwidth is shared amongst nodes / applications

HPC System Diagram



How are HPC resources allocated?

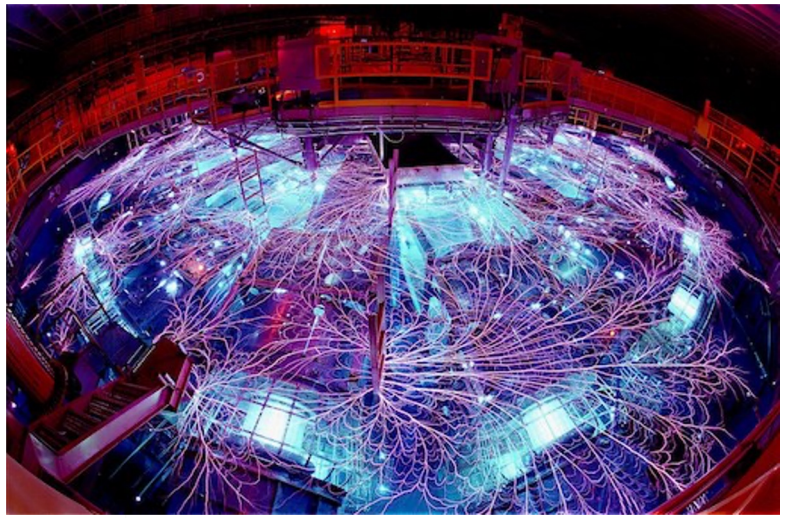
- Compute nodes are assigned to applications using a job scheduler
  - Common ones are Slurm, PBS, LSF, and Moab
- Users submit a job using the scheduler commands and then the jobs are put in a queue
  - The queue can be as simple as FIFO or as complex as a priority-based, system-aware scheduler
- The scheduler tries to allocate jobs to resources in an optimal way
  - Generally, having an application run on nodes that are close together in the network topology is preferred
  - "Optimal way" is the subject of much research
- The job scheduler only retains **some** information about the application
  - Tracks what the name of the binary or script run is
  - Assigns a unique job ID for all nodes in the job allocation
  - Typically does not track the input deck / app name unless given by the user

What are we trying to understand about HPC systems?

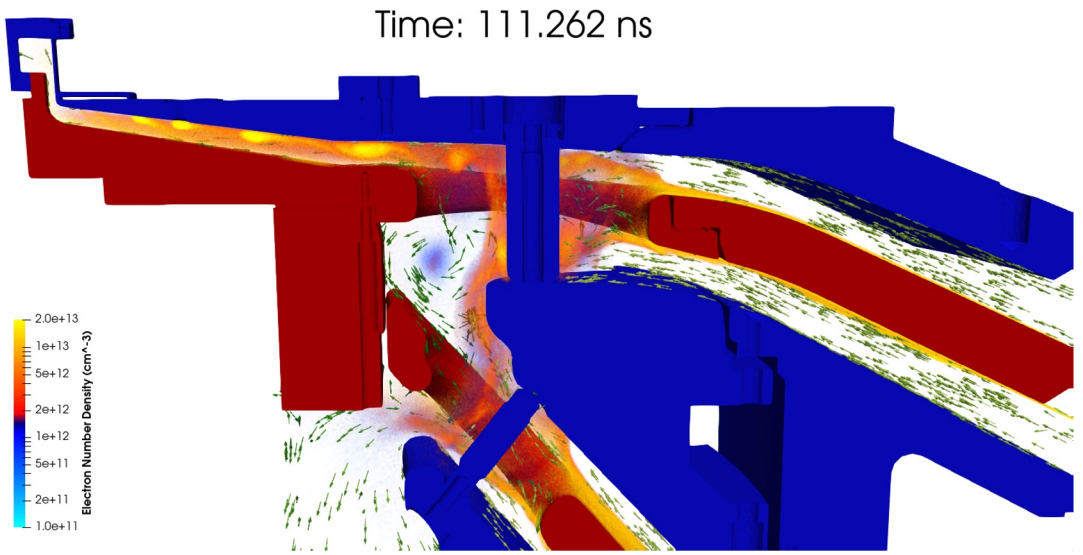
- The #1 HPC user complaint is "Why is my job running slow?"
  - Could be an issue with their code introduced by a new feature
  - Could be an issue with the system affecting their application
- HPC system administrators also want to know if something is wrong on the system to make proactive fixes i.e.
  - Is there a bad/slow node?
  - Is a user causing issues for others?
- HPC systems are becoming increasingly complex and understanding what is abnormal about an application/system state is challenging

Example Use Case: EMPIRE Performance Degradation

- EMPIRE is a computational physics application built for simulating electromagnetics and plasma physics. Due to the nature of the physics, EMPIRE runs need lots of computing power to resolve the quantities of interest analysts and designers need.
  - EMPIRE is also designed to make good use of next-generation exascale computing platforms. Especially at large scale, understanding actual real-world performance is of paramount importance



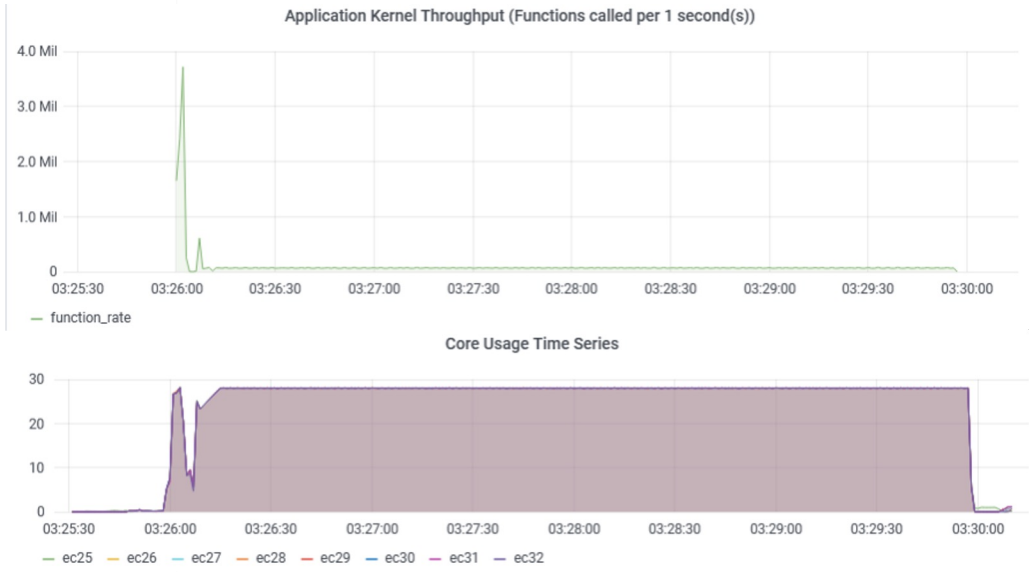
Sandia's Z-Machine Firing (~33m)

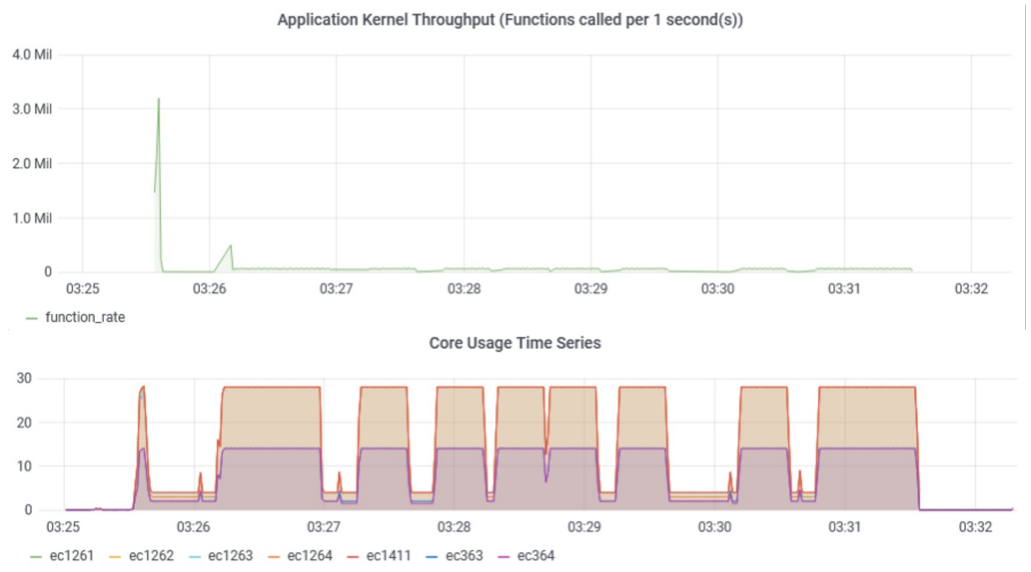


Z-Machine Convolute Simulation (~10cm)

Example Use Case: EMPIRE Performance Degradation

- Runs with EMPIRE were experiencing 1.2-1.5x performance degradation ~10% of the time for unknown reasons
- Initial inspection of HPC telemetry data showed that kernel throughput and CPU core usage profiles were notably different:





Example Use Case: EMPIRE Performance Degradation

- Further inspection into CPU usage showed high I/O wait in the degraded jobs
  - This extra I/O wait time correlated well with the extra runtime, more on that later
- EMPIRE occasionally writes out to a variety of files about state and science variables
  - Long write-out causes application to stall
    - Number of kernels called also drops during these write out stalls
- Application developers investigated and compared I/O performance of EMPIRE to other applications with similar file I/O mechanisms
  - EMPIRE was using a less efficient I/O version, a new upgrade produced more stability and less performance degradation

Overarching Goals of Computer Monitoring

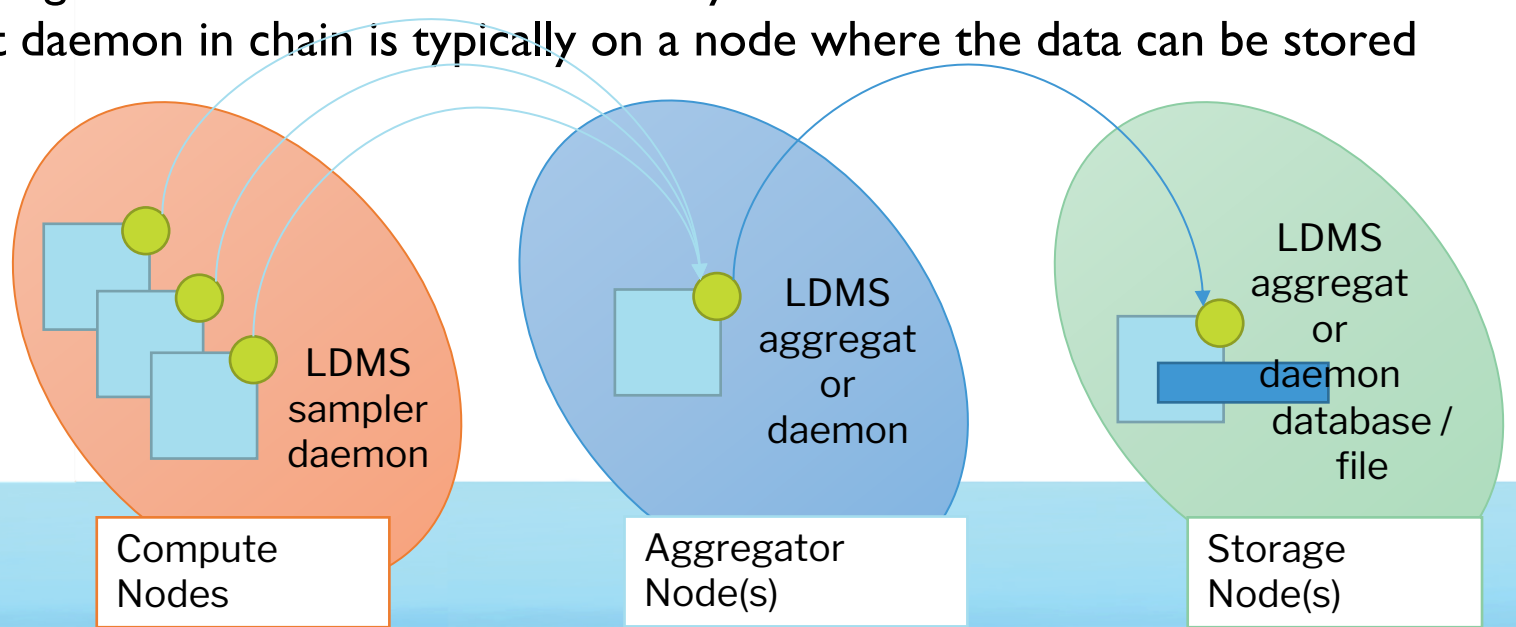
- Ideally, we could autonomously monitor the system and tune parameters to detect, mitigate, and/or resolve performance issues to make the most efficient system possible
  - Higher efficiency means more science / second!
- In the shorter term, we want to alert when anomalies like the EMPIRE degradation arise
  - Detecting anomalies early in the application's runtime could save wasted resources on a underperforming application
  - Finding ways to diagnose anomalies autonomously would save precious time root causing issues
- Knowing if anomalies exist and their cause can provide feedback to the application / resource manager
- Though discussed in the context of HPC, these goals are generalizable to other large distributed computing infrastructures such as cloud computing

How do we collect information about HPC systems?

- Monitoring frameworks generally poll some kernel-level information from each node in the system at regular intervals
  - Most frameworks are modular and can "load" different samplers to collect data from different subsystems of a machine
  - Results in high-dimensional time-series data
- A wide variety of system monitoring frameworks exist
  - Ganglia and Nagios are great frameworks for high-level monitoring and alerting on select metrics across HPC systems
  - DCDB Wintermute can collect higher fidelity of metrics with some small analysis capabilities done on the node collected
  - Will discuss one framework in particular, but general methodologies and analysis challenges apply

Lightweight Distributed Metric Service (LDMS)

- The tri-labs, NERSC, and other vendors use Lightweight Distributed Metric Service (LDMS) to collect system data
  - LDMS can collect 1000s of system metrics at sub-second intervals, typically collect at 1 Hz
  - These metrics range from network performance counters to filesystem statistics to CPU and memory utilization
  - Typical collection results in ~10s of TB of data **each day**
- An LDMS sampler daemon on each compute node collects information and sends it synchronously to an LDMS aggregation daemon, typically on an admin node
  - Aggregator daemons can chain as many times as desired
  - Last daemon in chain is typically on a node where the data can be stored



https://github.com/ovis-hpc/ovis
https://github.com/ovis-hpc/ldms-containers

Push-pull System and Application Data Collection

- LDMS can collect data synchronously from a variety of subsystems at regular intervals
  - Typically used to get usage system data about the CPU, memory, filesystem, network etc.
- External data sources can also push data to LDMS asynchronously
  - Typically used to have applications periodically send function-timing and file operation information
  - Want to provide higher-order, lightweight application information rather than replicate application performance profilers
- Provides time-aligned application and system view to investigate how system conditions affect applications and vice versa

Monitoring Framework Tradeoffs

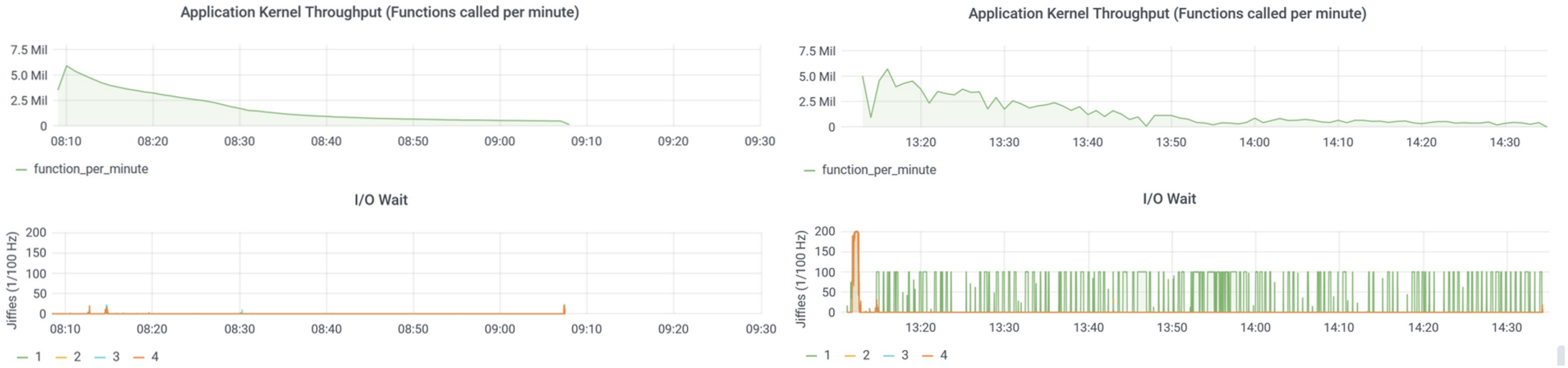
- Monitoring frameworks must capture enough information at a high enough fidelity to resolve features of interest
  - Anecdotal evidence has shown that 1 second resolution is needed to debug application-level issues
- Frameworks must have a small resource footprint to not impede applications
  - Typical frameworks aim to have ~10 MB footprint and < 1% CPU usage
- These restrictions limit the detail of data we can collect
  - Frameworks typically support higher fidelity and a wider field of collection but that introduces higher overhead

Achieving Low Overhead Monitoring

- All monitoring frameworks aim to be as low overhead as possible to mitigate effects on the system / applications
  - Most frameworks are also modular and can add / subtract data source samplers to meet overhead requirements
- Data source samplers often just do simple reads of files of the /proc filesystem (i.e memory information from /proc/meminfo)
  - /proc filesystem acts as an interface to internal data structures in the kernel
  - Samplers also often interface with special hardware, such as HSN switches, through C APIs to poll usage statistics
- Frameworks often piggyback off HSN or available message brokers to reduce communication latency / overhead
  - Frameworks often try to reduce data sent by tracking if source data has changed or by only sending metadata as needed

Analysis and Visualization Framework

- SNL has deployed "AppSysFusion" to combine kernel timing data and system metrics in an analysis and visualization framework to enable real-time insights
  - Uses Grafana as a front-end with a Django application to transform Grafana requests into database queries and analyses



Application performance (top) and **system** conditions/characteristics (bottom)
Normal application profile (left) and **degraded** application profile (right)

AppSysFusion

- Created a system for in-query analyses so only user-requested analyses happen rather than always analyzing HPC data and storing results to a separate database
  - Saves significant compute and storage resources at the cost of increased query times
  - The “always analyze” use case is still available but is not the default
  - Allows for analyses to be easily changed without needing to recompute and store results
  - Users can easily add new analyses to be used in-query
- Created our own database, Distributed Scalable Object Store (DSOS), to handle the continuous ingest rate and scalability needs presented by HPC monitoring

Analysis and Visualization Pipeline

- User queries from Grafana dashboards are sent through a backend python application which can call python analyses to derive metrics from raw data
  - In-query analyses save significant computation time/resources for creating analysis results
    - Only data of interest is analyzed and new analyses can be created without recreation of analysis results across the database
    - Analyses can easily be changed / added to meet new challenges and decrease
- Python modules can query the database and return pandas DataFrames for analysis
- The backend application then takes DataFrames and formats them as JSON objects which Grafana can interpret

Challenges in HPC analysis

Why can’t ChatGPT tell me what’s wrong?

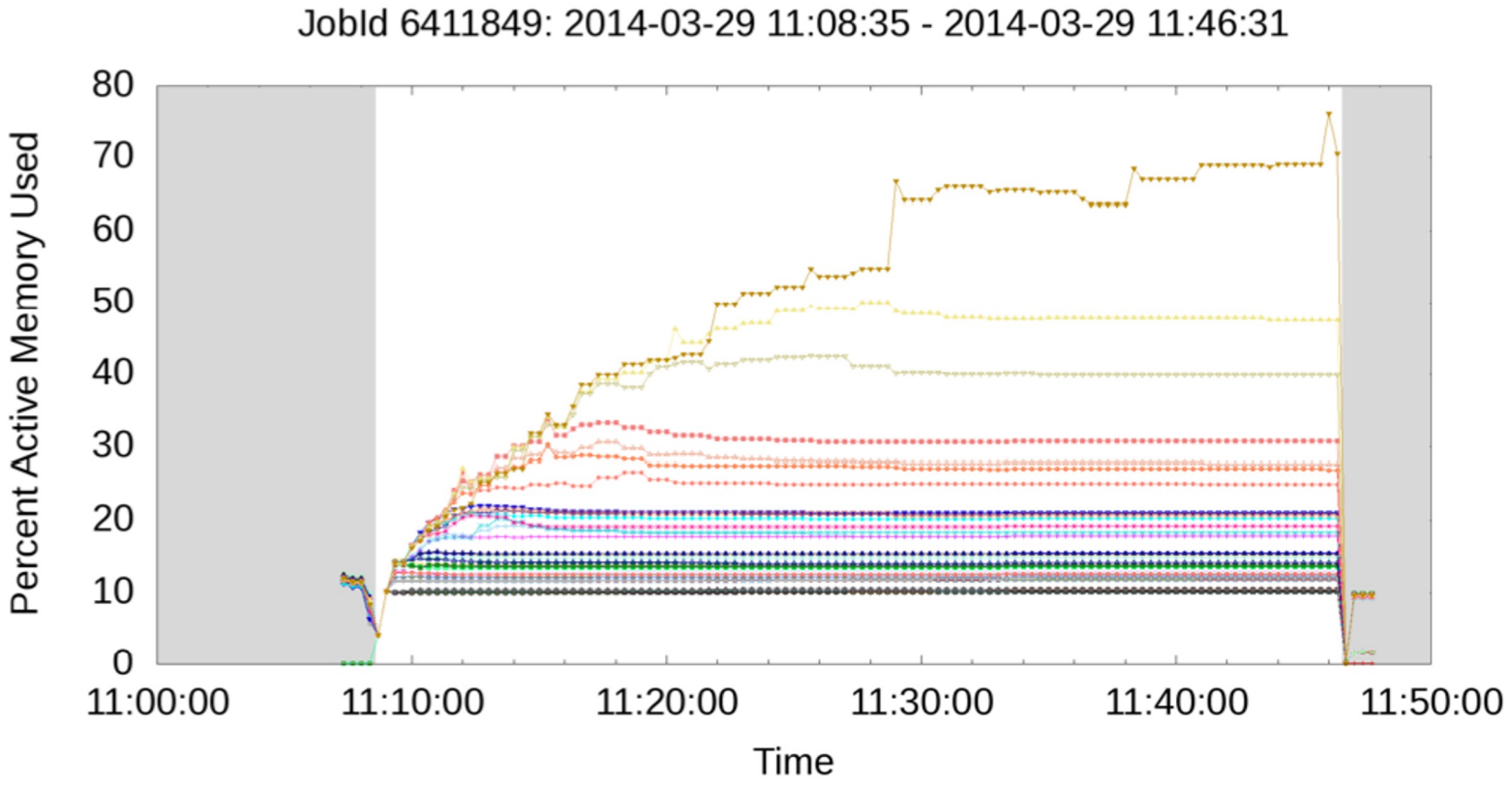
Why are HPC systems difficult to understand?

- System generates terabytes of high-dimensional, often non-linear, time-series data for each node
  - Variance thresholding and resampling / time-aggregation can be useful for some HPC data analysis, but there is no perfect way (yet) to simplify the dataset for ML
- The application mix on the system can change throughout the day and all system states may be perfectly "normal"
  - Often HPC clusters have 100s to 1000s of jobs running jointly on them each day
- Job schedulers only collect some information about an application
  - The exact parameters of the job are often opaque making application "labels" scarce
  - In research settings, this is often circumnavigated by working with users or submitting our own applications

Why are HPC applications difficult to understand?

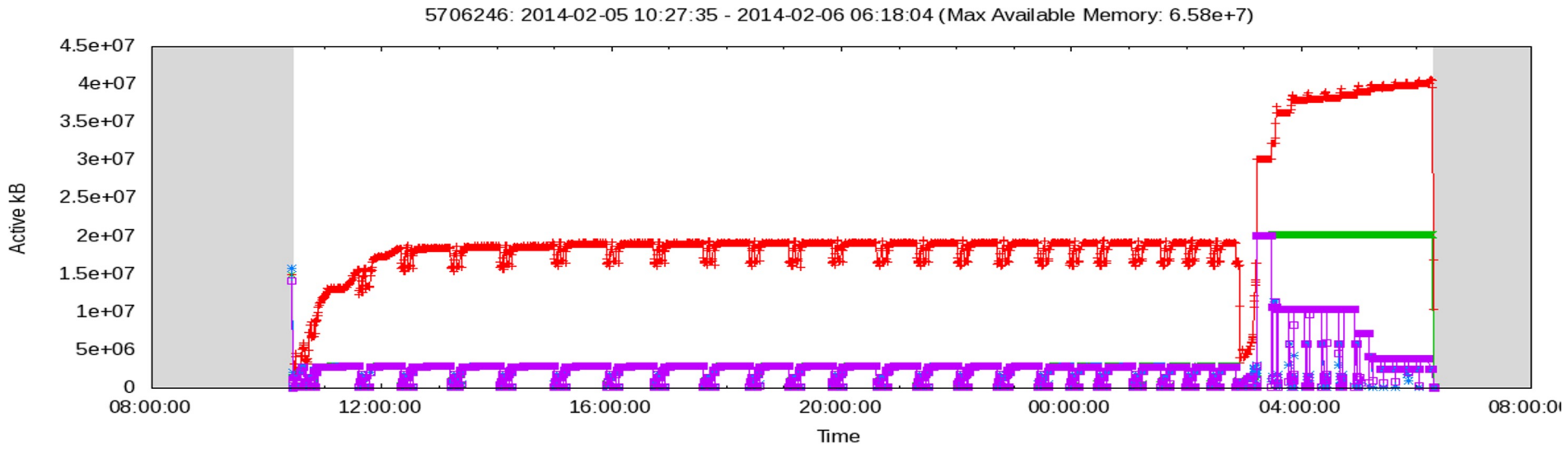
- Difficult to characterize normal and abnormal behavior for HPC applications
  - Applications can have different resource utilization characteristics throughout execution
  - The same application can use a different input deck and exhibit different resource utilization characteristics
- Many system conditions can affect an application's performance, and all must be considered
- Sources of application performance bottlenecks include:
  - CPU issues (slow node)
    - Usually caused by a degradation or failure of some component
  - Memory
    - Ranks within an application contend for memory usage on the node
  - Network
    - Separate applications can use the same routers and interfere with each other
  - I/O
    - Separate applications can r/w the same filesystem over a shared network

Example: Different memory usage amongst nodes in job



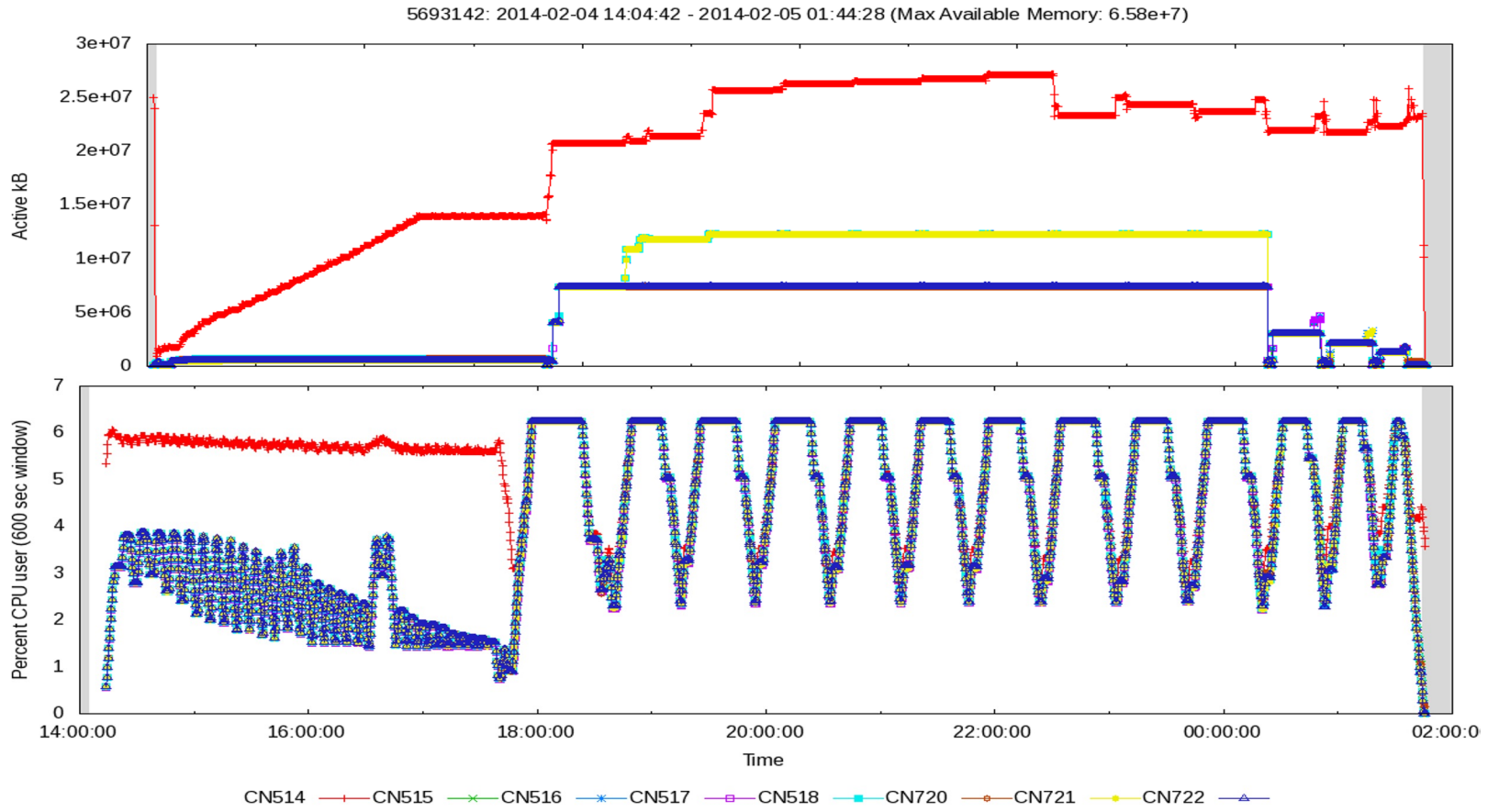
- A single job can exhibit a wide variety of behaviors on different nodes

Example: Different node behavior / different phases



- The red node exhibits different memory usage behavior than the rest of the nodes
  - Often this is the case for rank 0 in a parallel MPI application
- Behavior of the application changes around 4:00:00 for all nodes
  - Many HPC applications have multiple “phases”

Example: Similar behavior in one metric, different in another



- Just because a node behaves similarly to all others in one system metric, does not mean it will be the same across all metrics

Example Takeaways

- "Normal" or "healthy" behavior on HPC systems is incredibly hard to define
  - Applications can have different resource utilization characteristics throughout execution
  - The same application can use a different input deck and exhibit different resource utilization characteristics
    - Which input parameters are chosen by users is often opaque to administrators / analysts making labelling what is running equally difficult
- Applications can indirectly interact with each by contending for shared resources
  - Application profiles like in the examples can vary under different system conditions
  - Applications vie for filesystem and network bandwidth and can block each other
  - The application mix on the system can change throughout the day, and all system states may be perfectly normal, leading to many valid application profiles

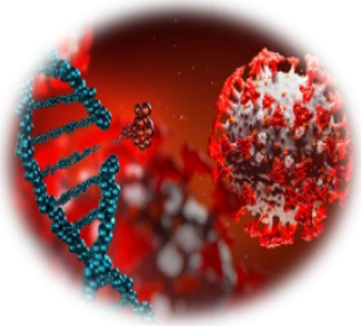
Wrap Up

- High-performance computing systems are critical to many sectors
  - Detecting and diagnosing performance degradation is necessary to maintain scientific throughput
- HPC monitoring, analysis, and visualization provides real-time insights to combat performance issues
  - Monitoring frameworks must balance high data capture with low overhead
  - Analysis frameworks must handle the data volume and be able to perform
- Analyzing HPC monitoring data presents unique data science challenges
  - High-dimensional time-series data with many non-intuitive features

Section II:
ML-based Analytics in Large-scale Computing Systems

- Performance variations in large scale systems
- ML-based analytics
  - Anomaly detection and diagnosis
  - Application detection
- Towards real-world deployments

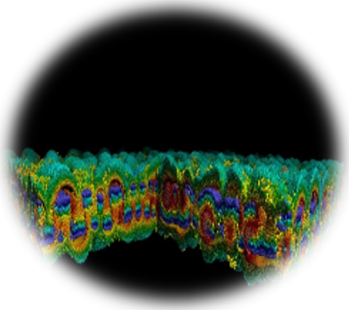
Large Scale Computing Systems Today



Medicine

Security

Climate



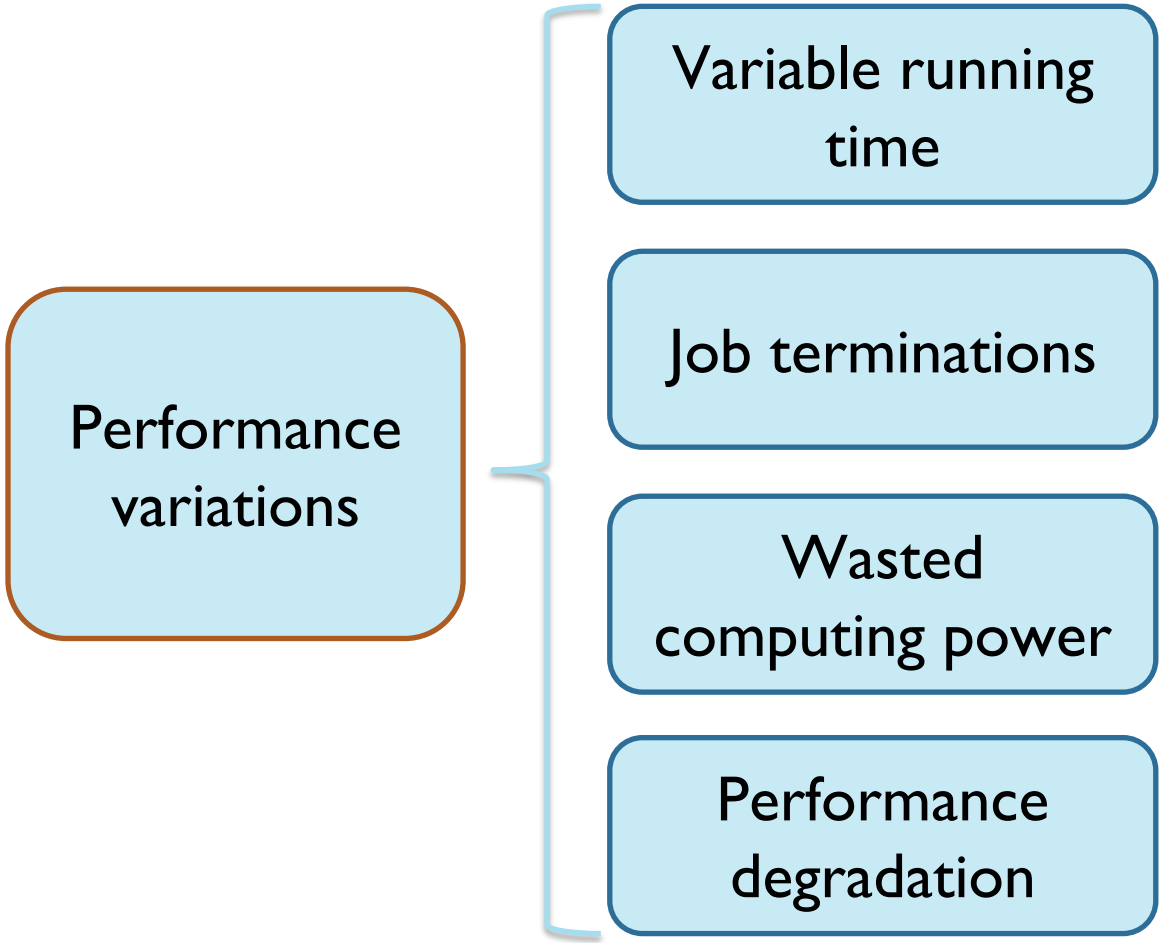
Simulations

- Over 1 million cores
- Parallel applications

→ Shared Resources → Performance Variation

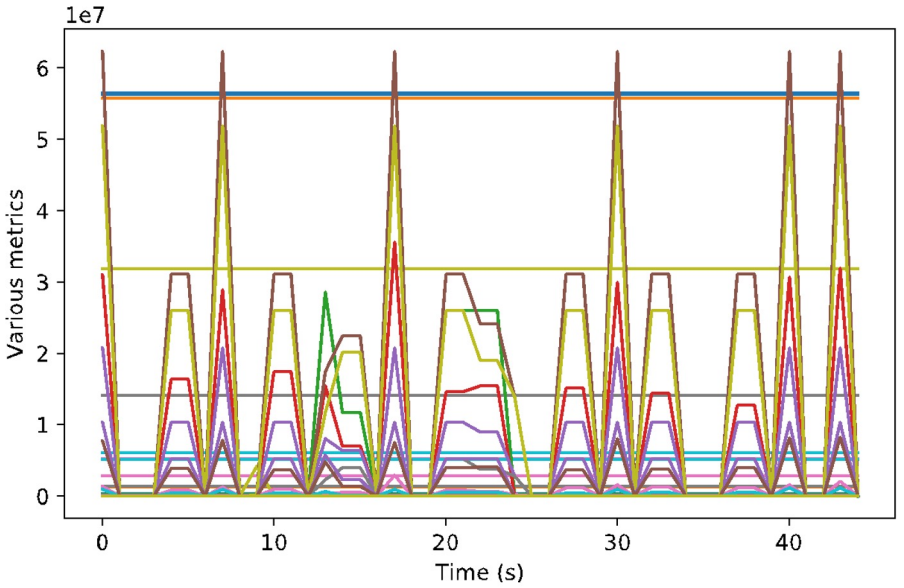
Performance Unpredictability at Large Scale

- **8x** delay in in job execution time *[Zhang et al., Cluster'20]*
- Most performance variations are caused by **system anomalies**:
  - Shared resource contention
  - Orphan processes from previous jobs
  - Software/Hardware bugs



Numeric Monitoring Data

- Anomalies manifest themselves in performance metrics
  - CPU: system time, user time, …
  - Memory: usage, page count, cache metrics, …
  - Network: sent packages, blocked packages, …
- Diagnosing performance anomalies is necessary for:
  - Understanding the root cause of the problem
  - Increased energy efficiency
  - Better resource utilization



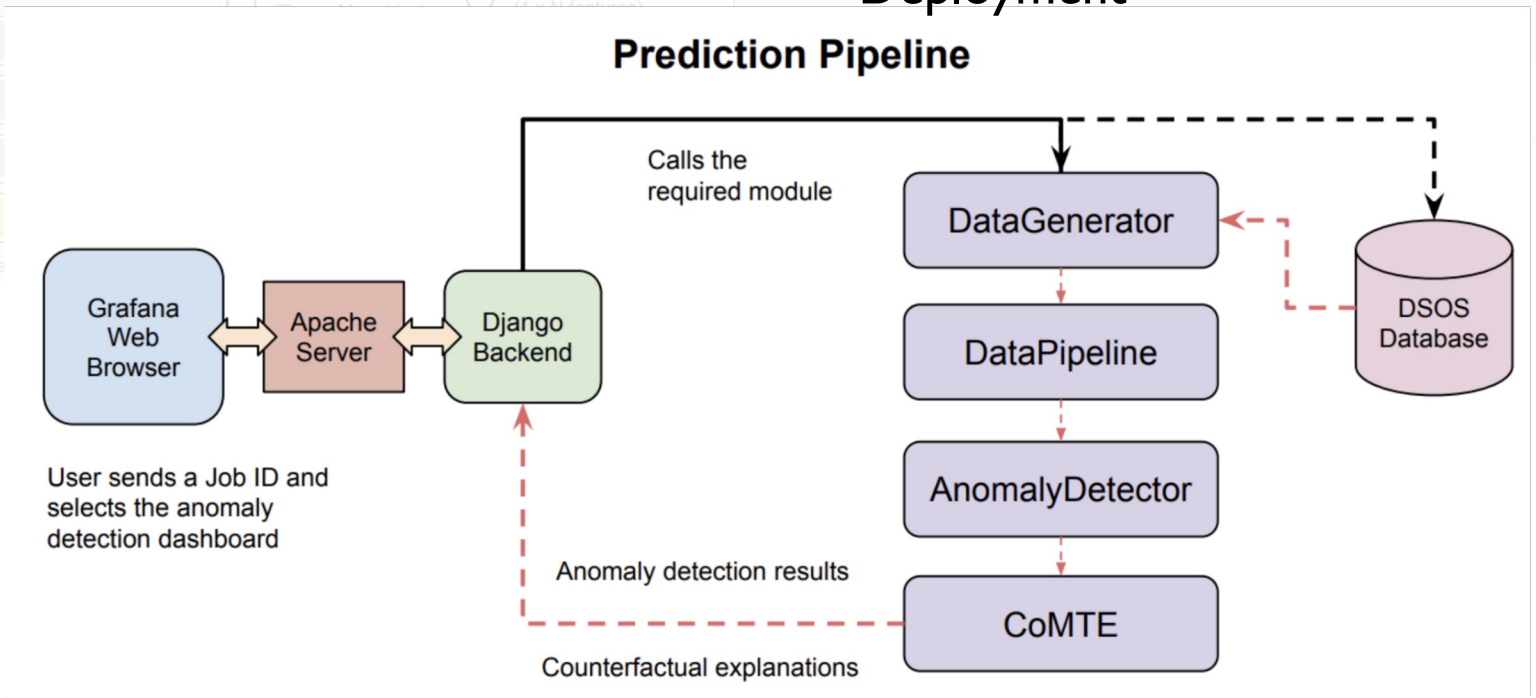
Various performance metrics collected from a compute node

ML is a Good Fit for This Problem

- TBs of numeric data in the form of logs, traces, performance metrics
  - Data is complex and huge in size
- Similar problems often repeat
- ML helps us identify previously encountered performance problems rapidly
  - Adapt from historical data to diagnose anomalies without heavily relying on human expertise

Research Landscape

Deployment



Related Work - ML for Anomaly Detection

Supervised Frameworks

- Data Mining-Based Analysis of HPC Center Operations *[Klinkenberg et al., Cluster'17]*
- Interpretable Anomaly Detection for Monitoring of HPC Systems *[Baseman et al., KDD'16]*
- Diagnosing Performance Variations in HPC Applications Using Machine Learning *[Tuncer et al., ISC'17]*

Semi-Supervised Frameworks

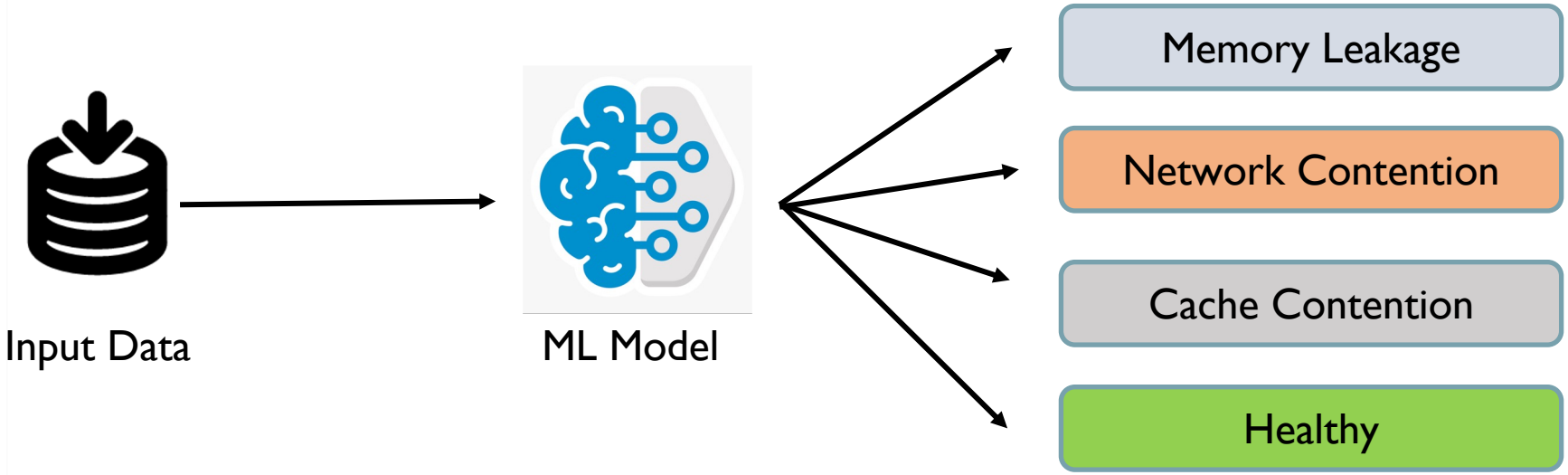
- A semisupervised autoencoder-based approach for anomaly detection in HPC systems *[Borghesi et al., EAAI'19]*
- Proctor: A semi-supervised performance anomaly diagnosis framework for production hpc systems [Aksar et al., ISC'21]
- ALBADross: Active Learning Based Anomaly Diagnosis for Production HPC Systems [Aksar et al., Cluster'22]

Unsupervised Frameworks

- Characterizing HPC Performance Variation with Monitoring and Unsupervised Learning [Ozer et al., ISC'20]
- Failure prediction in datacenters using unsupervised multimodal anomaly detection [Zhao et al., ICBG'20]
- RUAD: Unsupervised Anomaly Detection in HPC Systems [Molan et al., FGCS'23]

Requirements for Supervised ML Frameworks

- To diagnose performance anomalies using supervised ML models:
  - We need the labels corresponding to different anomalous application runs
  - The ML model learns the relationship between the input features and the target prediction output

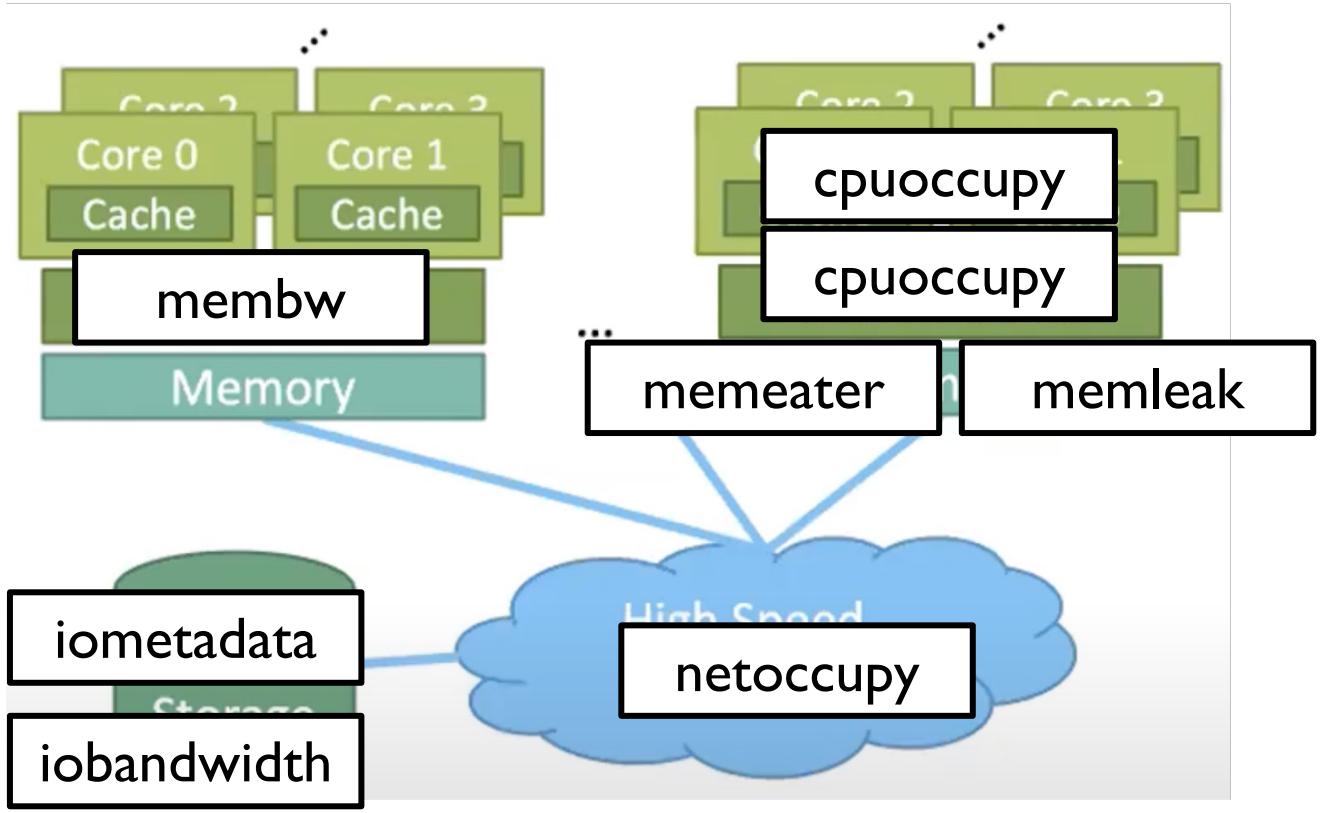


Supervised Frameworks - Challenges of Obtaining a Labeled Dataset

- Supervised frameworks require a **large** labeled dataset consisting of anomalous and healthy samples
- Labeled data is often scarce in real-world scenarios, particularly anomalies, which are usually rare events
- No standardized way of reproducing performance variability with synthetic anomalies
- One way to solve this problem is to generate labeled data using **synthetic** performance anomalies

HPAS: An HPC Performance Anomaly Suite

- **Goal**: Mimic real-world performance anomalies that reproduce performance variations
- Each synthetic anomaly targets specific subsystem (CPU, memory, network, storage) and has adjustable anomaly intensity
- Implemented in C language



[Ates et al., ICPP'19 - github.com/peaclab/HPAS]

Synthetic Anomalies in HPAS

- Each anomaly has multiple intensity settings and targets a specific subsystem (CPU, memory, etc.)
- Anomaly runs on *one of the nodes* of the application

Anomaly Name	Subsystem	Operation	Real-Life Event Modeled
Dial	CPU	Floating point operations	CPU intensive orphan process
Dcopy	Cache	Read and write	Cache interference
Linkclog	Network	Inject sleep before MPI calls	Network interference
Memeater	Memory	Allocate, write and realloc()	Memory intensive orphan process
Leak	Memory	Allocate	Memory leak

HPC Systems

Eclipse

Benchmark	Application	Description
Real Applications	LAMMPS	Molecular dynamics
	HACC	Cosmological simulation
	SW4	Seismic modeling
ECP Proxy Suite	EXAMINIMD	Molecular dynamics
	SWFFT	3D Fast Fourier Transform
	SW4LITE	Numerical kernel optimizations

- Production HPC system
- 1488 compute nodes
- Running on 4 nodes for 20 – 45 minutes

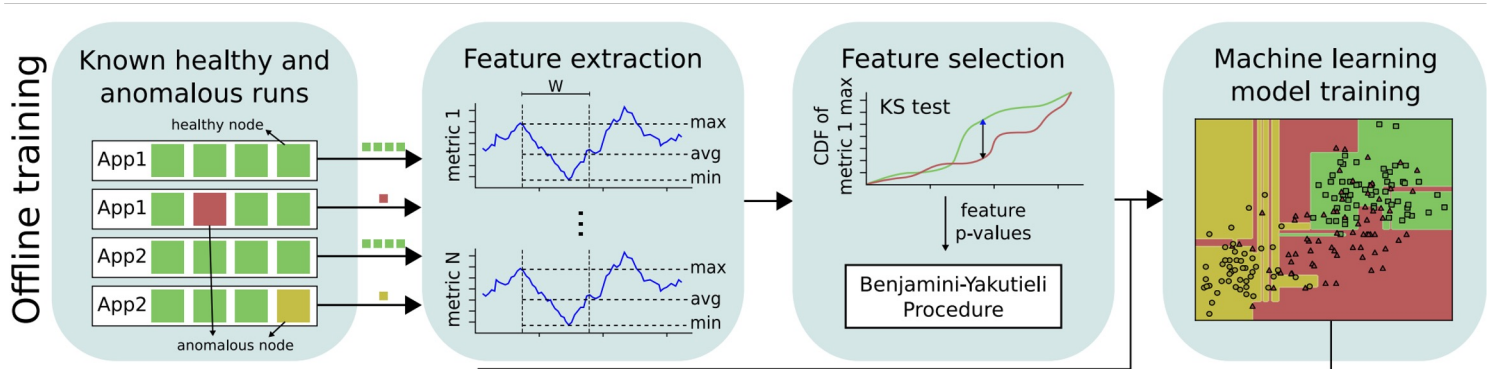
Volta

Benchmark	Application	Description
NAS	BT	Block tri-diagonal solver
	CG	Conjugate gradient
	FT	3D Fast Fourier Transform
	LU	Gauss-Seidel solver
	MG	Multi-grid on meshes
	SP	Scalar penta-diagonal solver
Mantevo	MINIMD	Molecular dynamics
	COMD	Molecular dynamics
	MINIGHOST	Partial differential equations
	MINIAMR	Stencil calculation
Other	KRIPKE	Particle transport

- Testbed HPC system
- 52 compute nodes
- Running on 4, 8, 16 nodes for 10 – 15 minutes

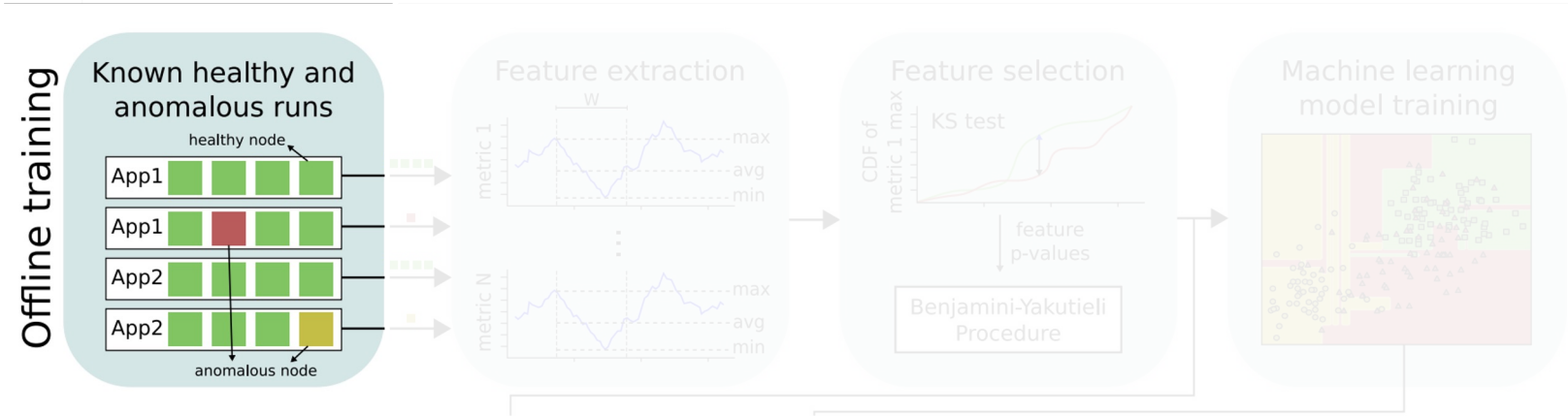
Diagnosing Performance Variations using ML

- Automated analytics framework
- Learns anomaly characteristics using labeled historical data
- Sources of labeled data:
  - Controlled experiments
  - Run applications with and without known anomalies



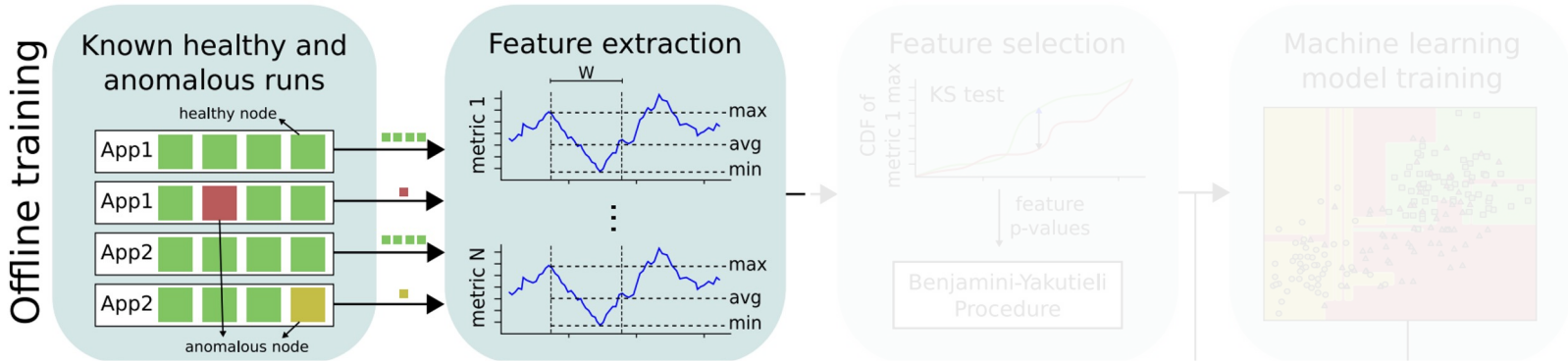
[Tuncer et al., ISC-HPC'17, Tuncer et al., TPDS'18]

Training Phase: Collecting the Dataset



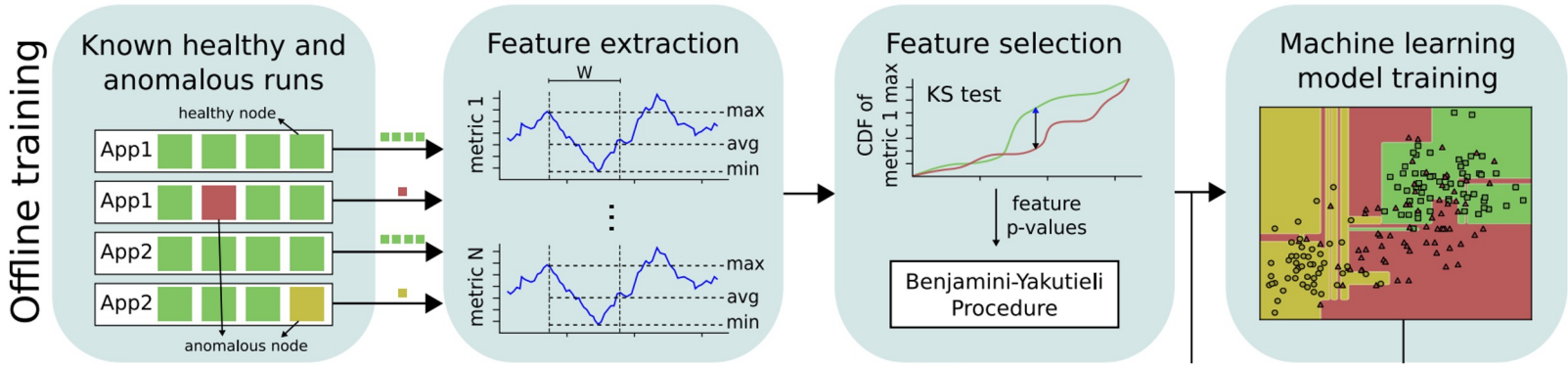
- Inject synthetic anomalies:
  - Anomaly runs on one of the nodes of the application run
  - Each anomaly stresses specific subsystem
    - Memory, network, CPU, …

Training Phase: Monitoring



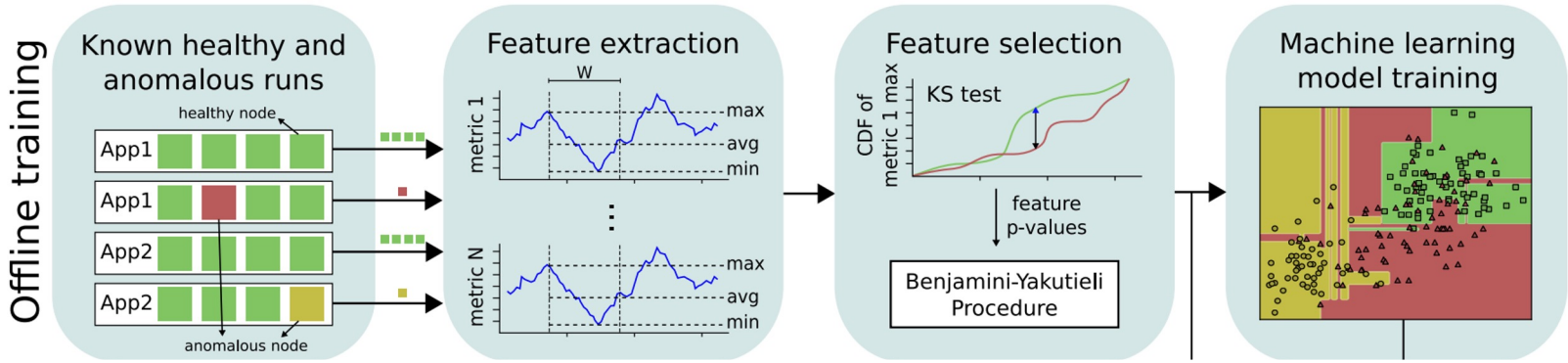
- Collect numeric monitoring data:
  - Lightweight Distributed Metric System *[Agelastos et al., SC'14]*
    - Collect the data from software and hardware counters such as *meminfo*, *procstat*, *vmstat*, …

Training Phase: Data Preprocessing



- Feature Extraction
  - Generate features per rolling window
  - Min, max, mean, average, skewness, …
- Feature Selection
  - Reduces computational overhead
  - Increases model's accuracy

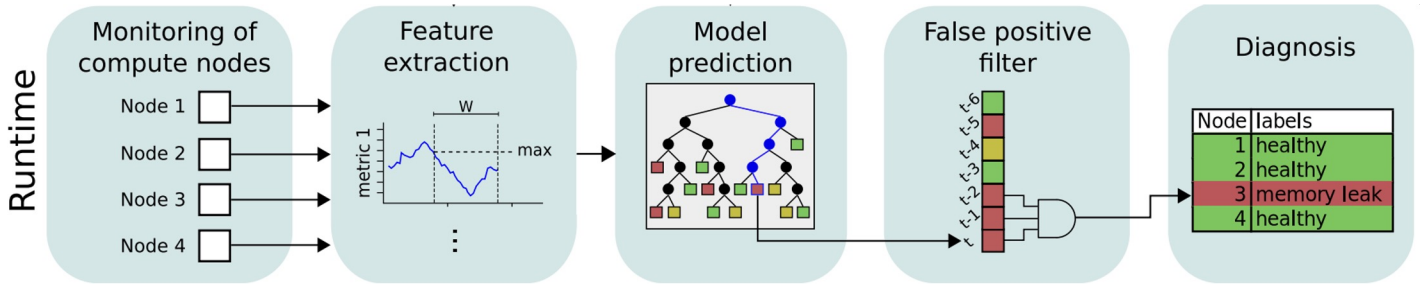
Training Phase: Model Training



- Train the supervised ML model using selected features
  - Tree-based models:
    - Random Forest, Decision Tree, AdaBoost

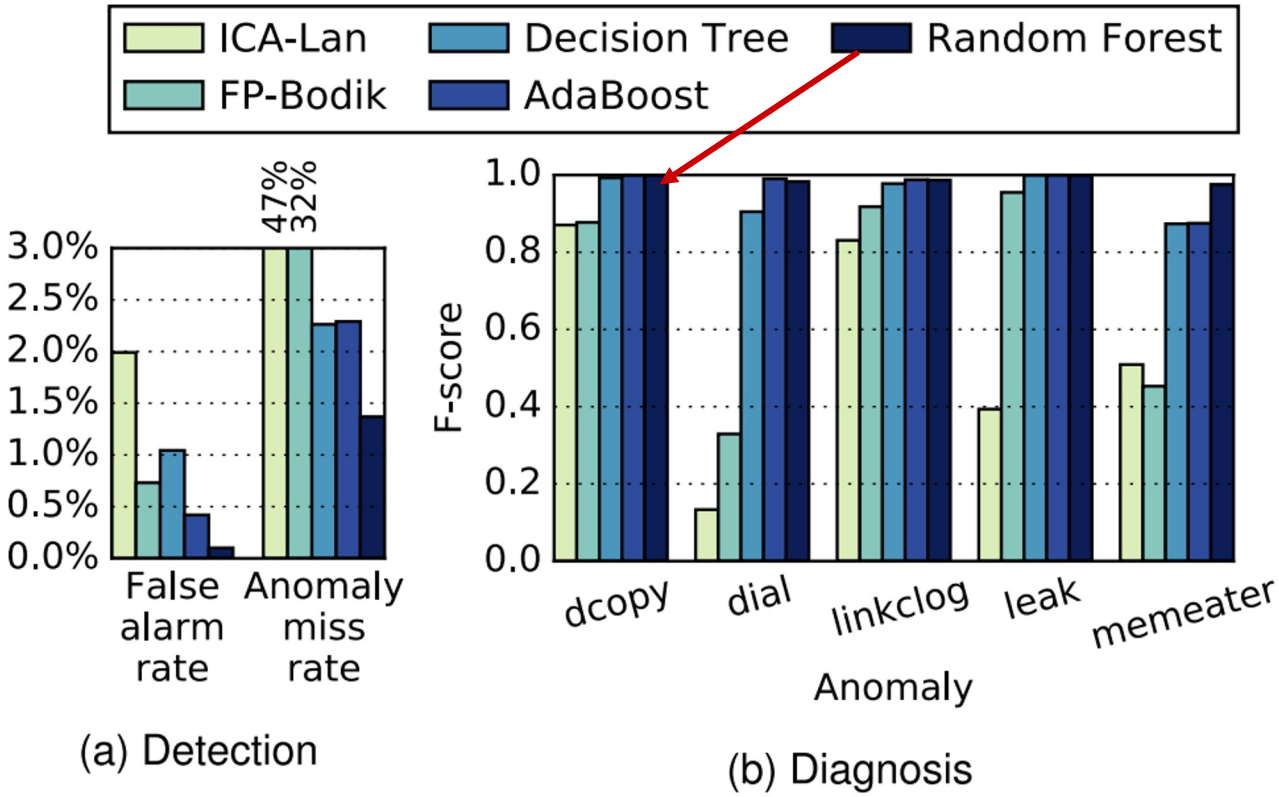
Online Anomaly Diagnosis Framework

- At runtime:
  - Collect monitoring data per compute node
  - Extract only selected features
  - Use previously trained ML model to predict anomalies
  - Apply false positive filter to reduce false alarms



Evaluation

- Random Forest identifies **98**% of injected performance anomalies
- **0.08**% false alarm rate



Limitations of Supervised Frameworks

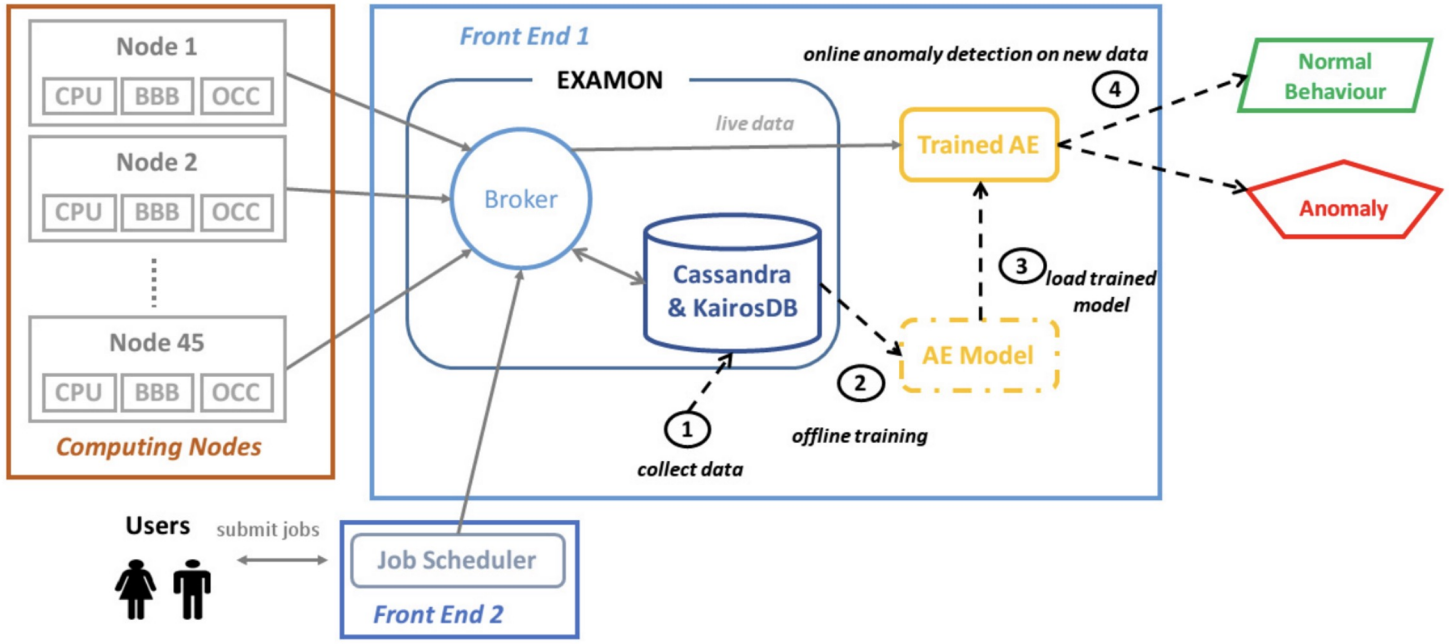
- Require ***a large set of labeled data*** corresponds to the normal & anomalous state of a compute node
- It is not possible to mimic all types of real-world performance anomalies using synthetic anomaly suite

Relaxing the Label Requirement

- What happens when there are not enough amount of labeled samples?
- Goal: Achieve target anomaly detection/diagnosis performance using fewer labeled samples
- Semi-supervised training settings:
  - First setting:
    - Large amount of healthy samples and few/no labeled samples anomalous samples
  - Second setting:
    - Few labeled samples and large amount of unlabeled samples
    - Human annotator is available to provide the label of selected sample upon request

When Training Dataset Contains Only Healthy Labeled Samples

- Method:
  - Train an autoencoder using the **normal** state of compute nodes
  - **Detect** anomalous compute nodes based on the reconstruction error
- No need for labels of anomalous samples during the training stage



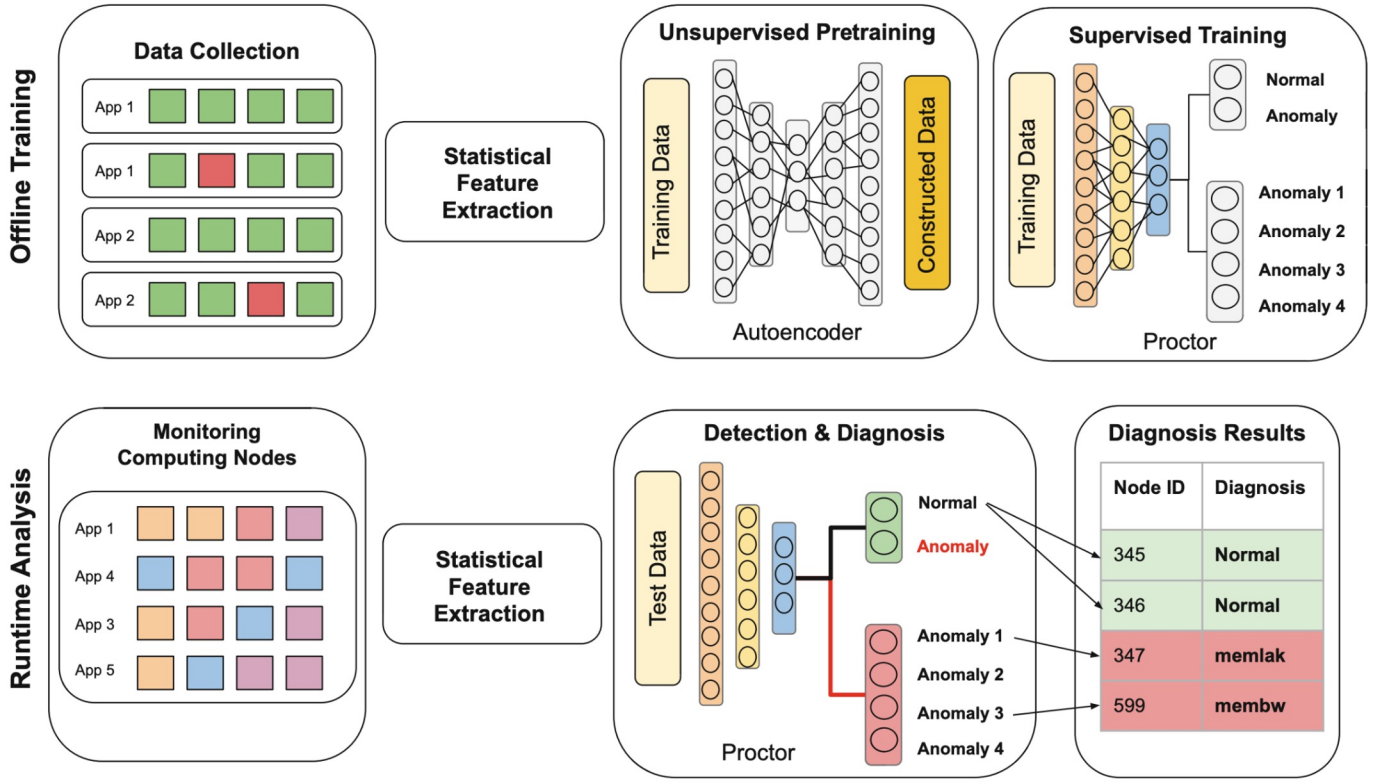
Architecture of data collection infrastructure and anomaly detection scheme

[Borghesi et al., EAAI'19]

When Training Dataset Contains Few Healthy and Anomalous Labeled Samples

Proctor: A Semi-Supervised Performance Anomaly Diagnosis Framework

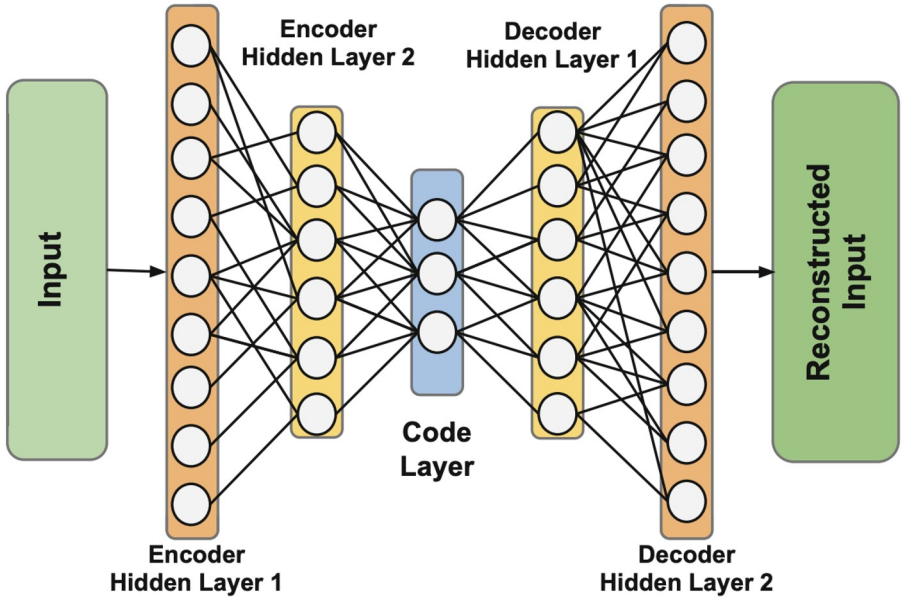
- Operates with significantly **less** labeled data compared to supervised baselines
- Not just detect but **diagnose** anomalies
- Evaluation on a production HPC system and a testbed HPC cluster
  - **11**% performance improvement in F-score on average
  - **0.06**% anomaly miss rate on average



Overall framework

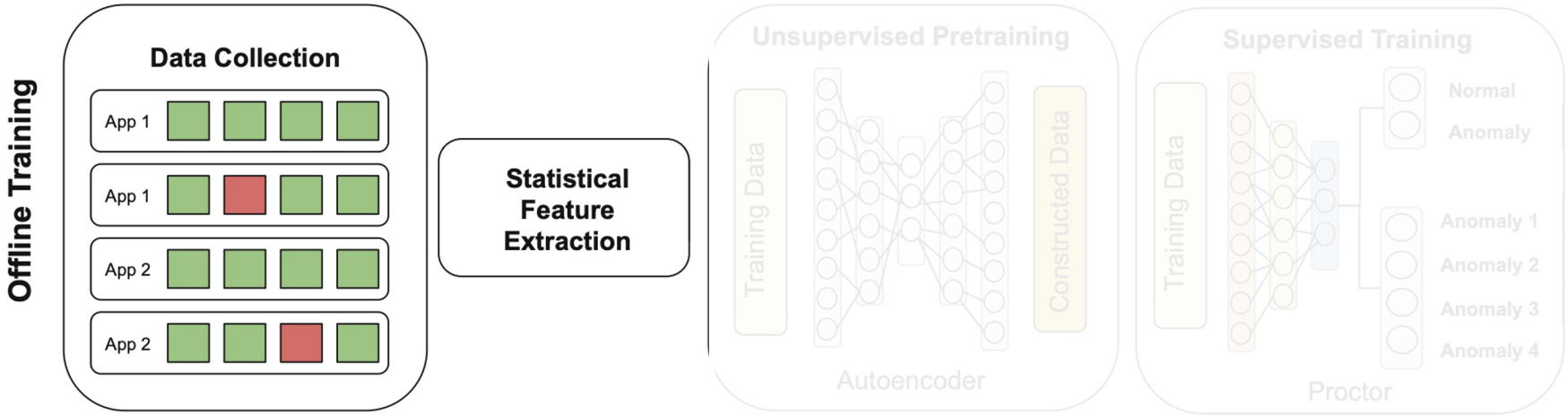
[Aksar et al., ISC'21 – github.com/peaclab/Proctor]

How to Train an Autoencoder?



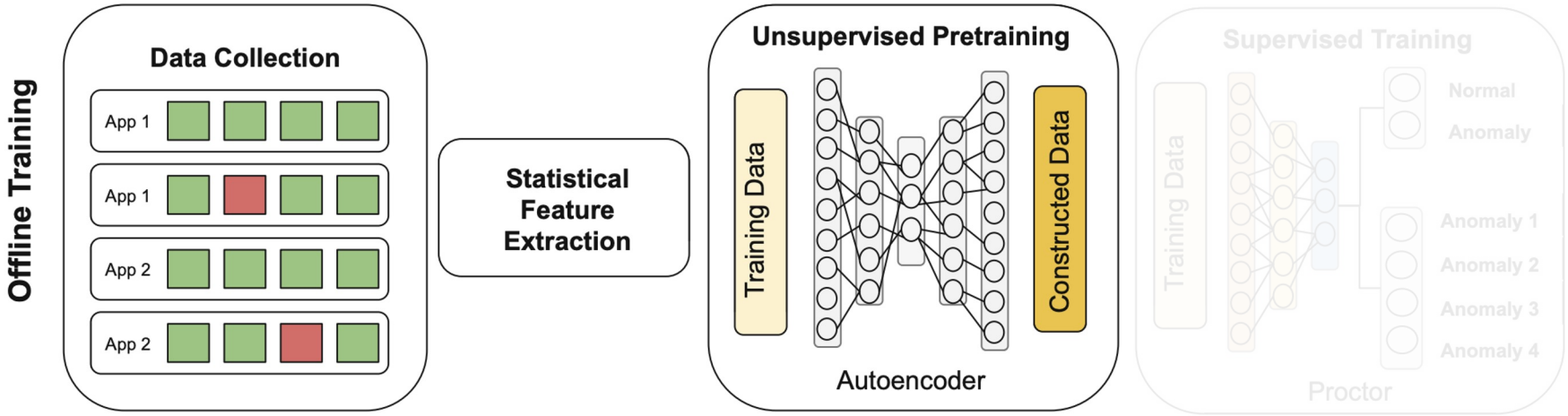
- Training objective is to learn the weights for the encoding and decoding layers so that the reconstructed input is as close to the original input as possible
- Compressed knowledge exists in the ***code*** layer

Training Phase: Autoencoder Training



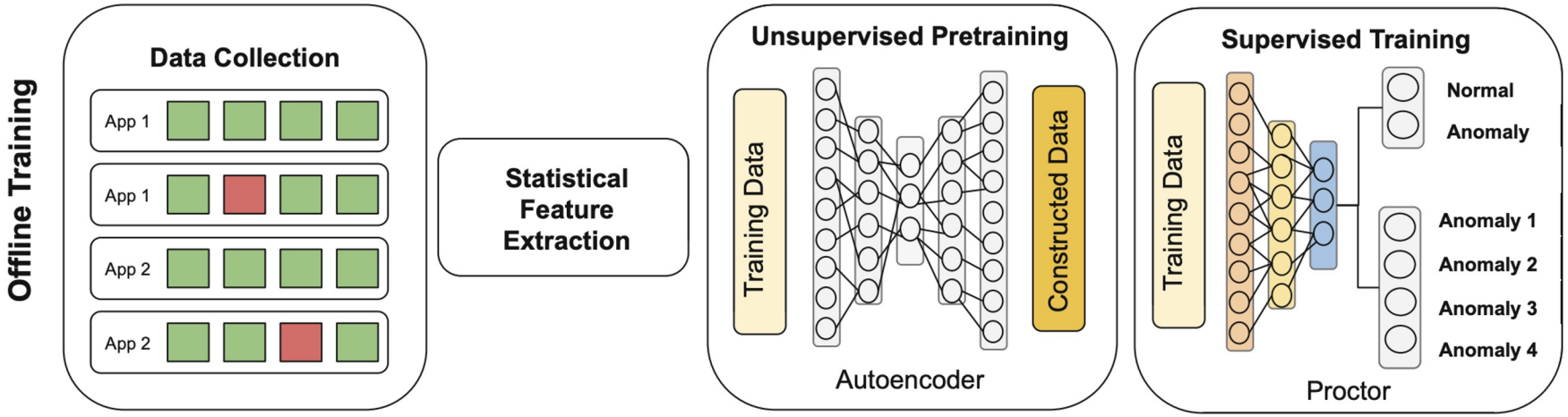
- Extract statistical features that retain the raw time series characteristics
  - Remove application initialization and finalization periods
  - Transform cumulative counters into events/sec

Training Phase: Autoencoder Training



- Extract statistical features that retain the raw time series' characteristics
- Train an autoencoder to learn a representation (encoding) of normal and anomalous runs in an unsupervised manner

Testing Phase: Diagnosis

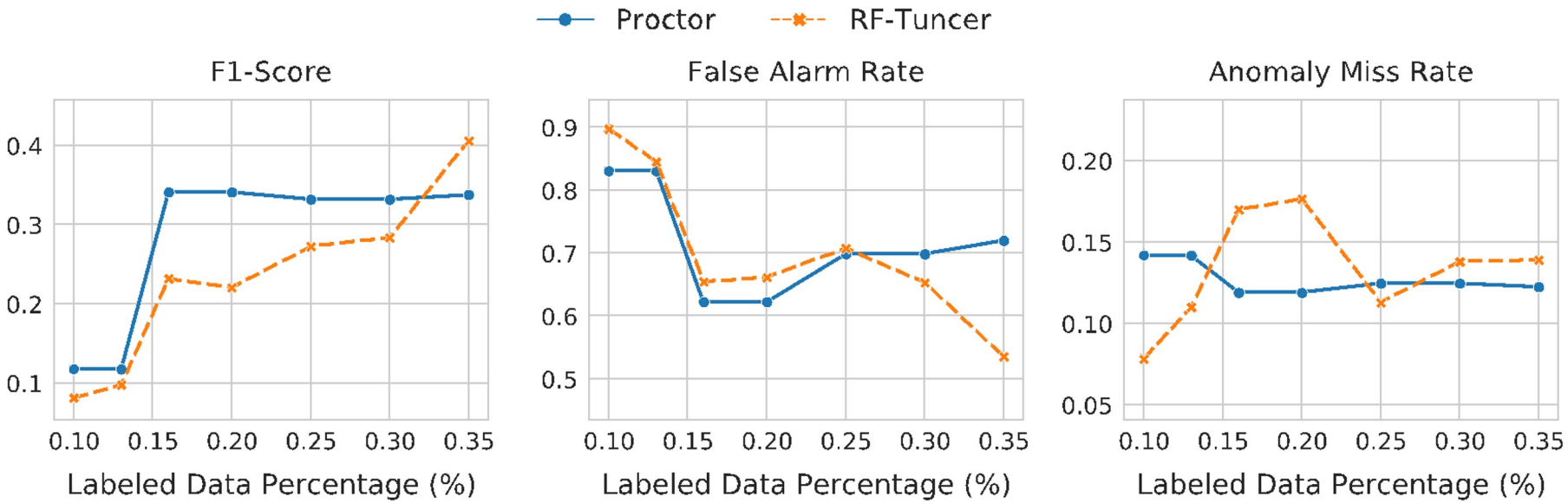


- Use the trained autoencoder's encoder and perform *two-level* classification
  - First classifier learns to classify anomalous vs normal
  - Second classifier learns to classify the type of the anomalies using a few labeled samples

Proctor – Baseline Methods

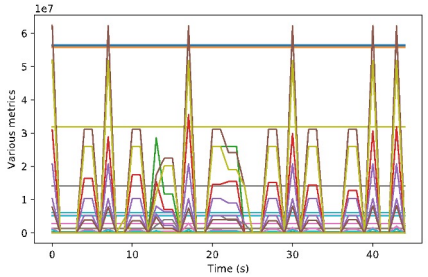
- RF-Tuncer *[Tuncer et al., TPDS'18]*
  - Uses statistical feature extraction and feature selection to train supervised machine learning models
  - It can diagnose anomalies
- AE-Borghesi *[Borghesi et al., EAAI'19]*
  - Uses an autoencoder trained with only *normal* samples and selects a threshold
  - Only high-level anomaly detection no anomaly diagnosis

Evaluation – Anomaly Diagnosis (Volta)



- Proctor outperforms RF-Tuncer by 25% on average (up to 50%) in F1-score and maintains a similar false alarm and miss rate

Can We Minimize Labeling Cost?



Add the sample to the labeled dataset

Query the label of a sample

Unlabeled Dataset

No

Initially labeled Dataset

Build a model

Evaluate the model on constant test set

Is my model accurate enough?

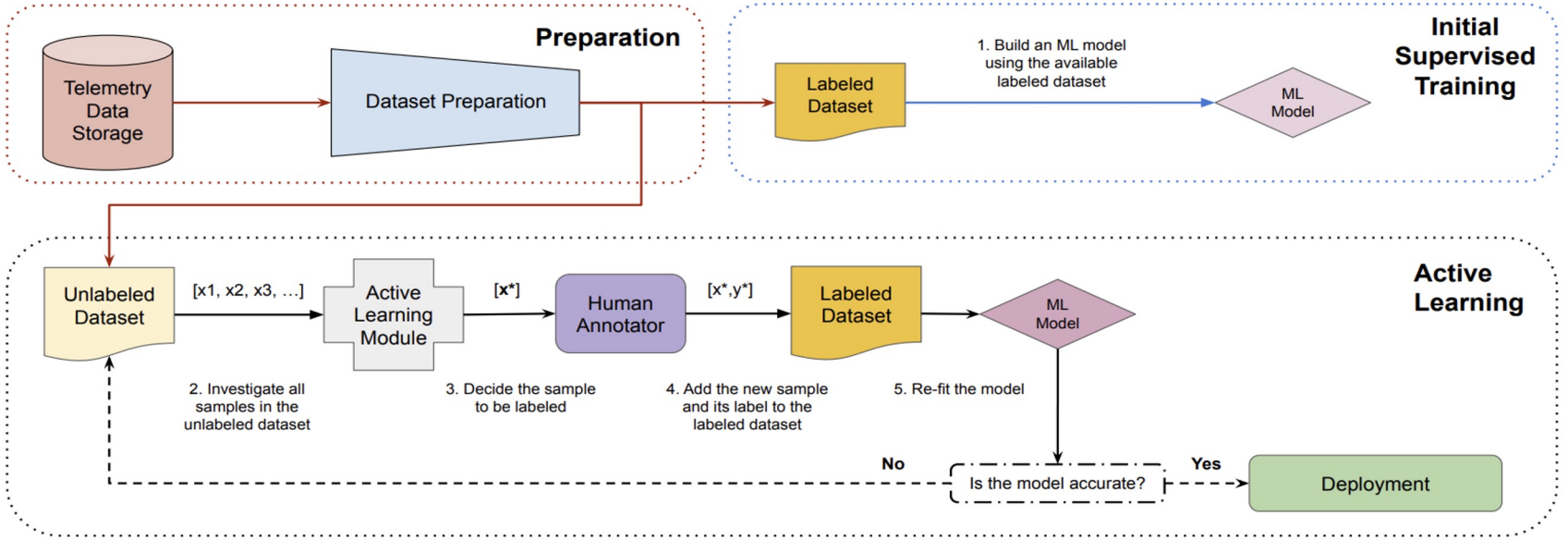
Yes

Terminate

Active Learning Based Anomaly Diagnosis

ALBADross: Active Learning Based Anomaly Diagnosis for Production HPC Systems

- Goal: Minimize the number of labeled samples during the training phase while achieving target F1-score
- Achieves the same F1-score as fully supervised baseline using **28x** fewer labeled samples even with previously unseen applications



Active Learning Based Anomaly Diagnosis Framework

[Aksar et al., Cluster'22 - github.com/peaclab/ALBADross]

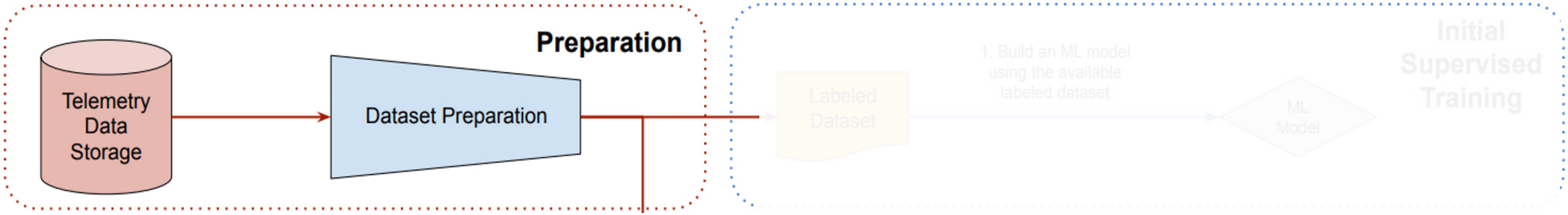
ALBADross: Research Problem

- Scenario:
  - A few labeled and a large set of unlabeled data samples
  - A subject matter expert (SME) that provides the label of the selected sample

- Research Problem:
  - How to design a robust anomaly diagnosis framework using the least amount of labeled data samples?

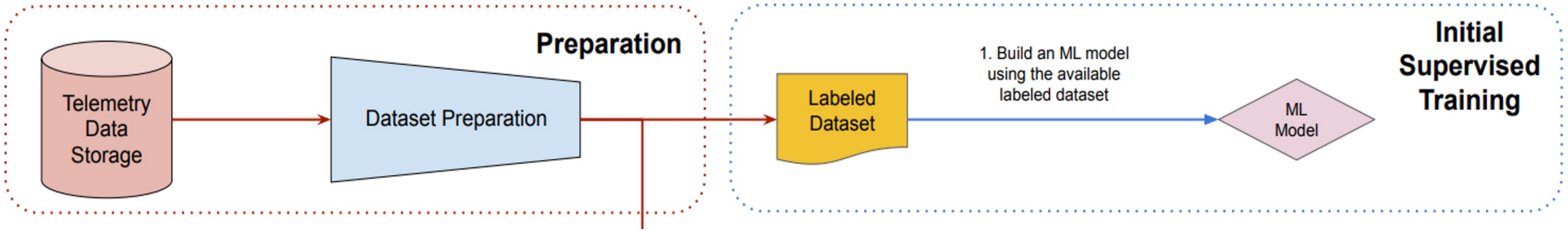
Sample*: *Feature extracted version of the telemetry data collected from one compute node throughout an application run*

ALBADross: Dataset Preparation



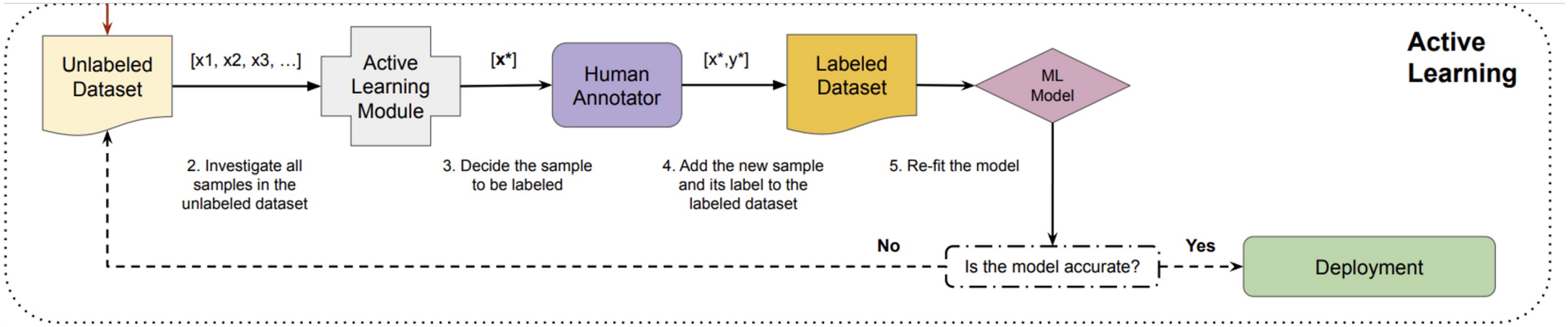
- Run synthetic anomalies with different real and proxy HPC applications
  - Each anomaly stresses a specific subsystem (CPU, memory, cache, network)
- Extract statistical features
  - TSFRESH (743 metrics)
  - MVTS (48 metrics)
- Feature selection
  - Chi-Square feature selection

ALBADross: Initial Supervised Training



- Few labeled and a large number of unlabeled samples
- Methodology:
  - Start with 1 sample from all application-anomaly pairs in the Eclipse and Volta datasets
  - 30 samples for Eclipse
  - 55 samples for Volta
  - Train a supervised ML model using labeled dataset & make predictions on test dataset

ALBADross: Active Learning



- Determines the sample to be labeled from the unlabeled data pool
- Adds the new sample to the labeled dataset and retrains the model

Active Learning – Query Strategies

- How does active learning determine the samples to be labeled?
  - *x*: Sample to be predicted
  - y: Most likely prediction (class label)
  - $[p_1, p_2, p_3, \ldots, p_k]$: Class probabilities
- **Uncertainty Sampling:** $U(x) = 1 - P_{max}(y|x)$
  - P_1 = [0.1, **0.85**, 0.05] => 0.85
  - P_2 = [**0.6**, 0.3, 0.1] => 0.6
  - P_3 = [0.3, **0.7**, 0.0] => 0.61

U_{list} = [0.15, **0.4**, 0.3]

Experimental Methodology – Anomaly Diagnosis

- **Goal**:
  - Determine how many samples should be labeled to reach target F1-scores, anomaly miss rates, and false alarm rates
- **Methodology**:
  - Start with 1 sample from all application-anomaly pairs in the Eclipse and Volta dataset
  - Compare the performance of active learning and baselines for 250 queries:
    - Active Learning:
      - Entropy, Margin, and Uncertainty Sampling
    - Baselines:
      - Random sampling

Anomaly Diagnosis with Active Learning

Dataset	Feature Extraction	The Best Sampling Strategy	Starting F1-score	F1-score: 0.95	All Active Learning Training Data F1-score	Max Score (5 fold CV)
Volta	TSFRESH	Uncertainty	0.86	76 Samples	**0.95 (6329 Samples)**	0.99 (16732 Samples)
Eclipse	MVTS	Margin	0.72	230 Samples	**0.95 (5619 Samples)**	0.99 (19652 Samples)

- Active learning achieves the same F1-score as a fully supervised approach using significantly **fewer** data samples

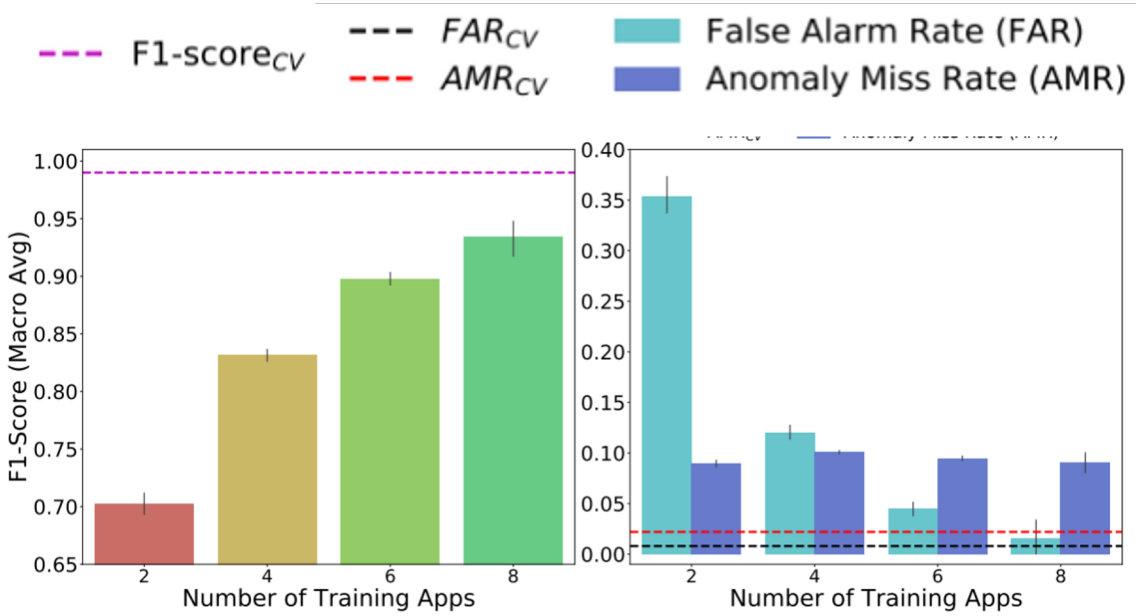
Anomaly Diagnosis with Active Learning

Dataset	Feature Extraction	The Best Sampling Strategy	Starting F1-score	F1-score: 0.95	All Active Learning Training Data F1-score	Max Score (5 fold CV)
Volta	TSFRESH	Uncertainty	0.86	76 Samples	**0.95 (6329 Samples)**	0.99 (16732 Samples)
Eclipse	MVTS	Margin	0.72	230 Samples	**0.95 (5619 Samples)**	0.99 (19652 Samples)

- Active learning can achieve a **0.95** F1-score with **48**x and **22**x fewer samples in the Volta and Eclipse datasets respectively

Evaluating Robustness

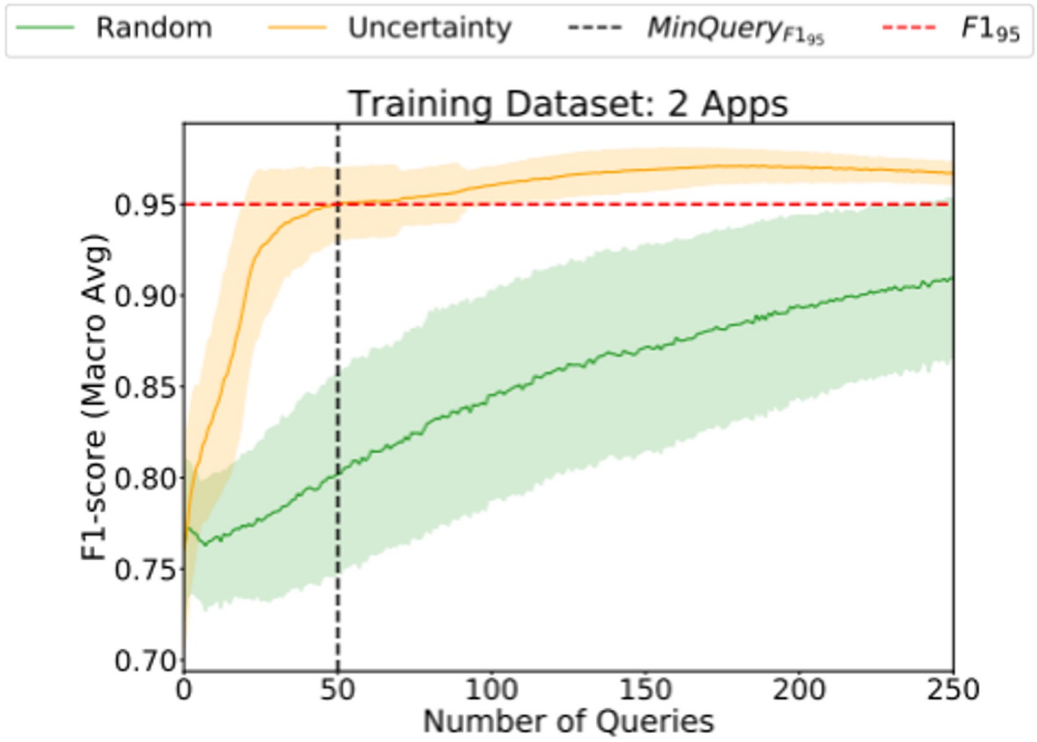
- Robustness:
  - Being able to diagnose anomalies with a small error margin with previously unseen applications and application inputs
- Compared to 5-fold CV results:
  - **30% drop in F1-score**
  - **35x increase in FAR**



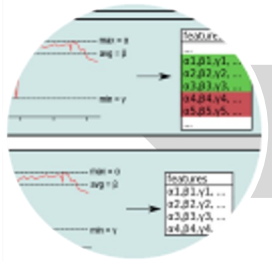
Error bars: 95% confidence interval
Dashed lines: 5-fold CV results

Active Learning With Previously Unseen Apps

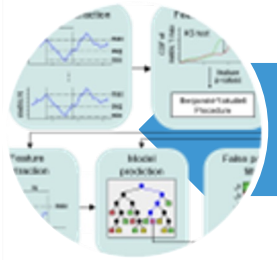
- Start with 2 applications in the training dataset, and place the remaining 9 applications to the test dataset
- Uncertainty achieves 0.95 F1-score using **50** additional samples
- Random achieves 0.95 F1-score using **~550** samples



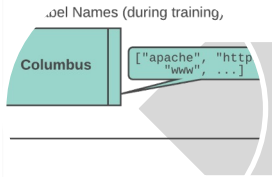
Outline



Diagnosing performance variations



Identifying applications

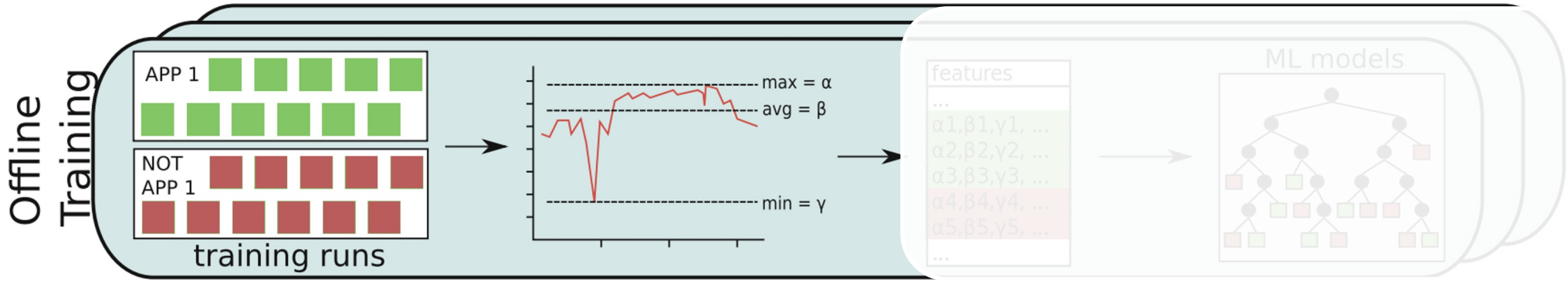


Towards real-world deployments

Can We Go Beyond Anomaly Diagnosis?

- Identifying Applications
  - Why is it important?
    - **Detect illegal/unwanted applications**
      - Known malicious or fraudulent apps
      - Wasteful apps
      - Vulnerable or buggy apps
    - **Detect (types of) applications to improve system**
      - Performance
      - Energy efficiency
      - System design or software optimization

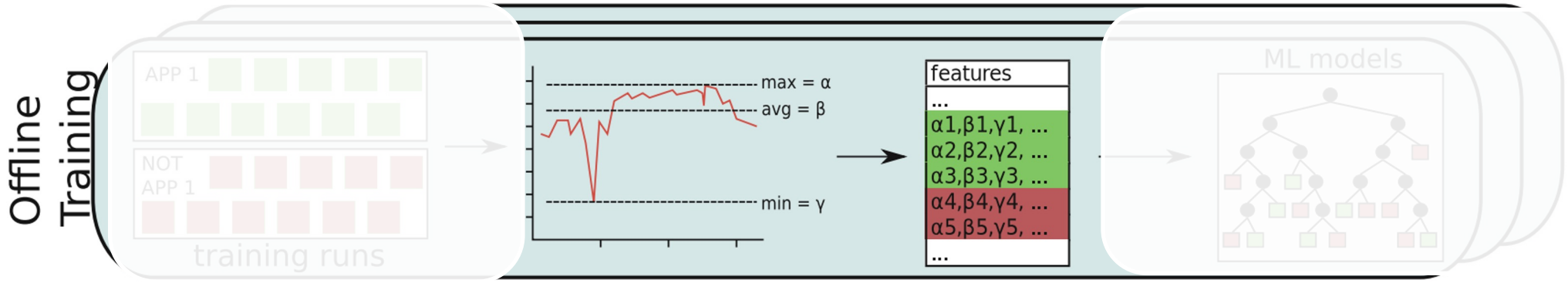
Taxonomist: An Application Detection Framework



- Data is collected from applications of interest
- 100s of time series per node

[*Ates et al., EuroPar'18 (Best Artifact Award)* - github.com/peaclab/taxonomist]

Taxonomist: An Application Detection Framework

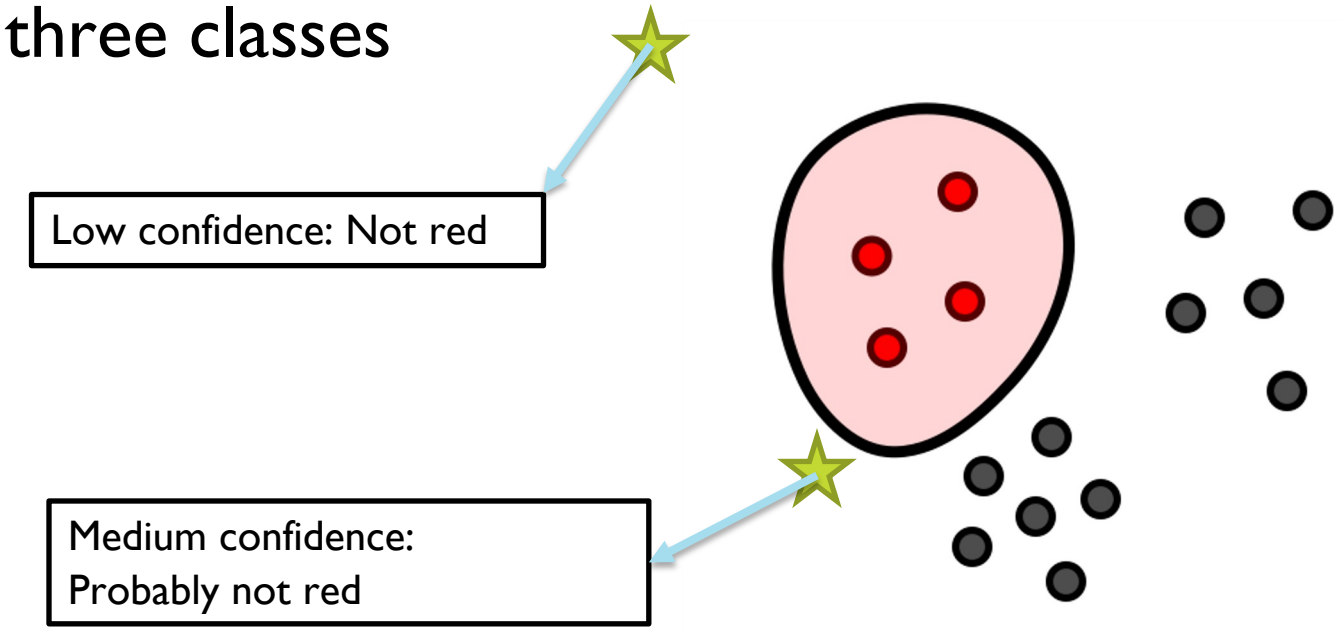


- Statistical feature extraction for summarizing time series
  - Min, max, mean, standard deviation, skewness, kurtosis, percentile

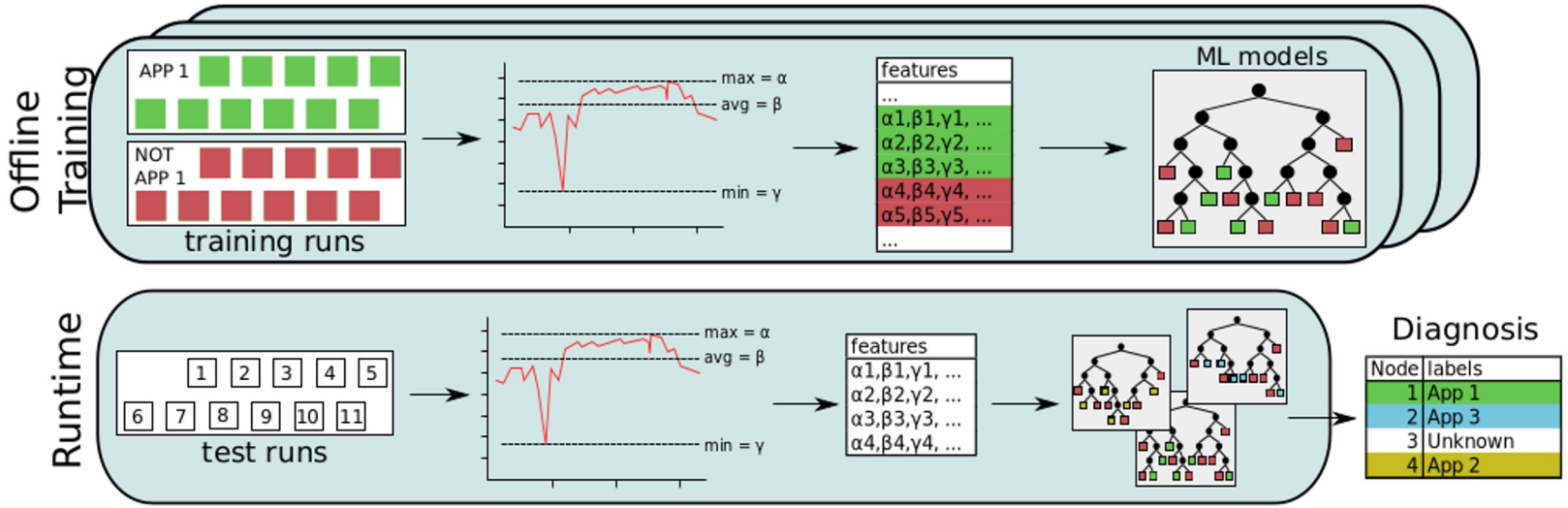
[Ates et al., EuroPar'18 (Best Artifact Award) - github.com/peaclab/taxonomist]

One vs. Rest Classifier

- Observations from three classes



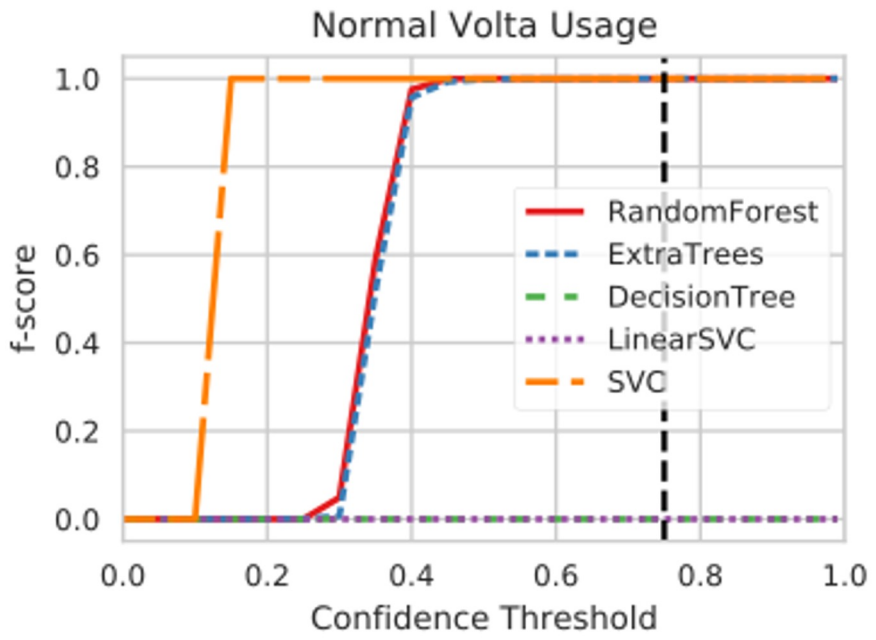
Taxonomist: An Application Detection Framework



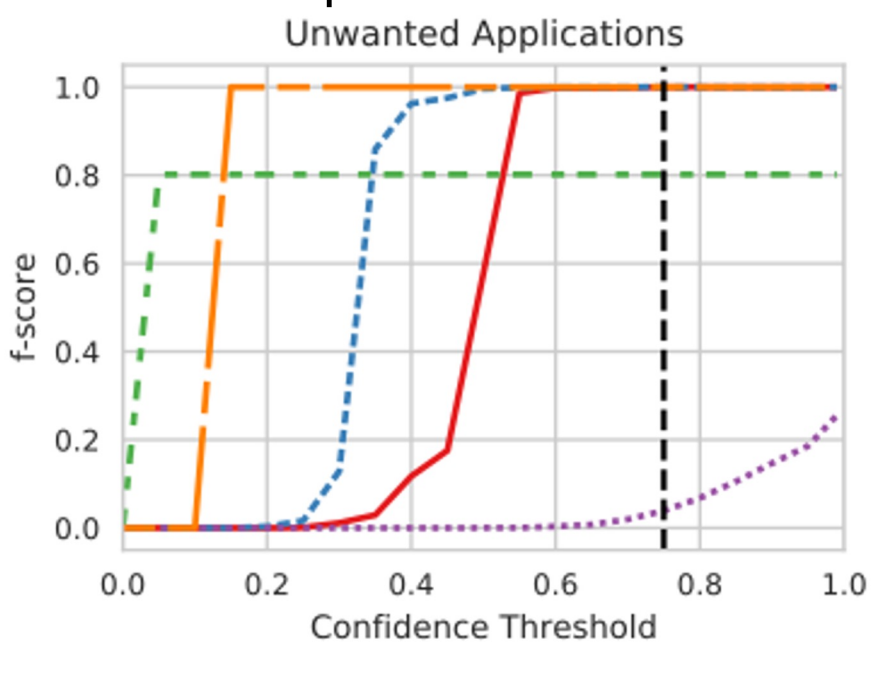
- At runtime, take predictions from every classifier
  - If confidence is under a threshold, mark as unknown

Evaluation

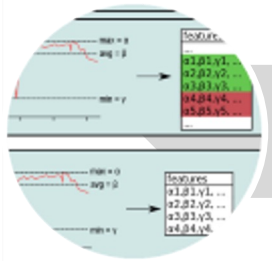
Volta usage over 6 months



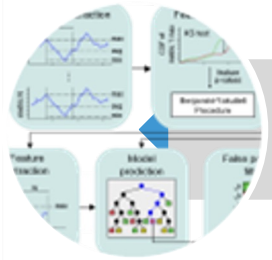
6 Cryptocurrency miners and password crackers



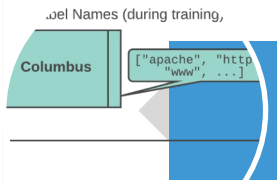
Outline



Diagnosing performance variations



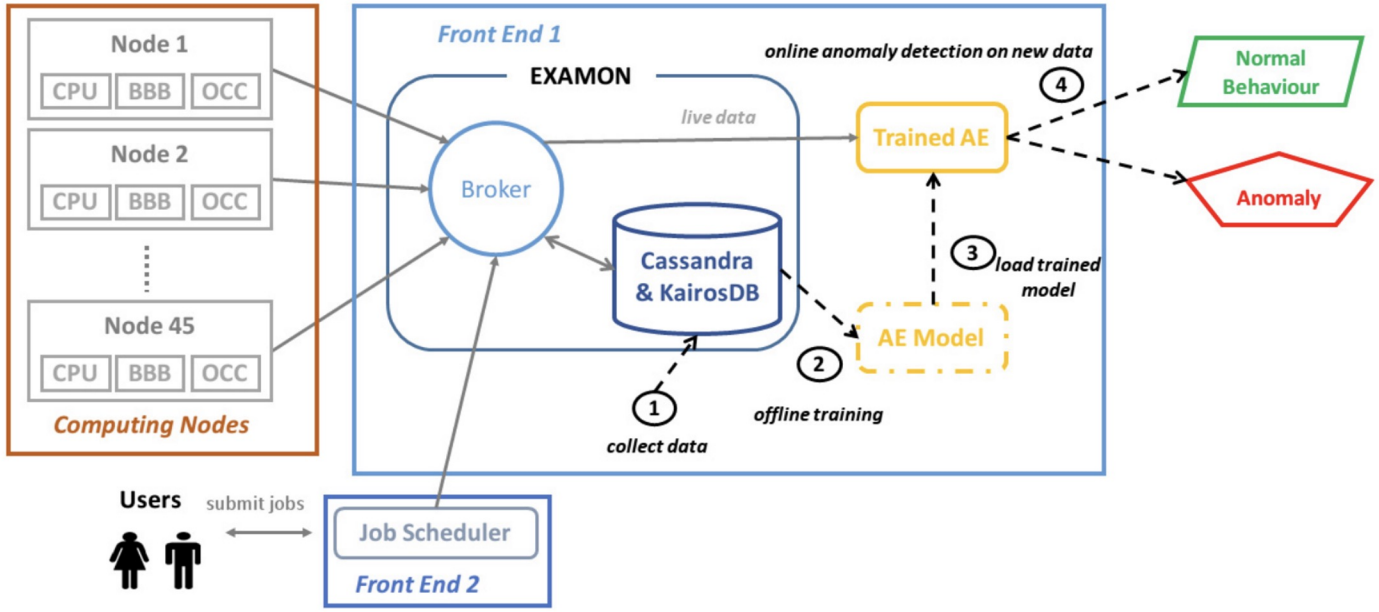
Identifying applications



Towards real-world deployments

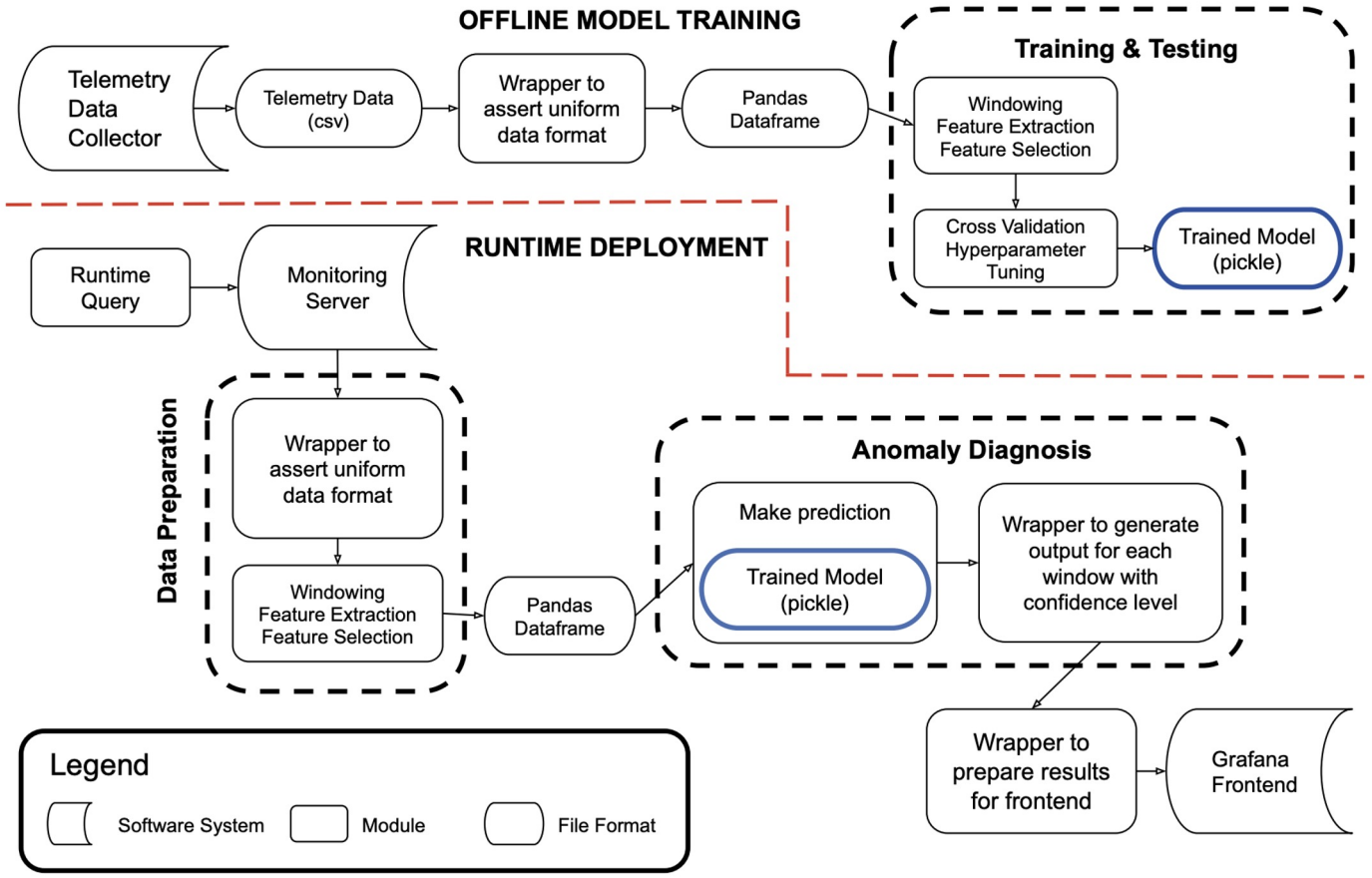
Deployment - What Has Been Achieved So Far?

- Generic model deployment *[Borghesi et al., EAAI'19]*
  - Deploy a single model that predicts anomalies in different compute nodes
- Compute node specific model deployment *[Molan et al., FGCS'23]*
  - Deploy autoencoder based anomaly detection model on a production HPC system with 980 nodes



Deployment - Our Key Achievements

- We deploy the state-of-the-art fully supervised anomaly diagnosis framework to a 1500-node production HPC system



The high-level architecture of E2EWatch

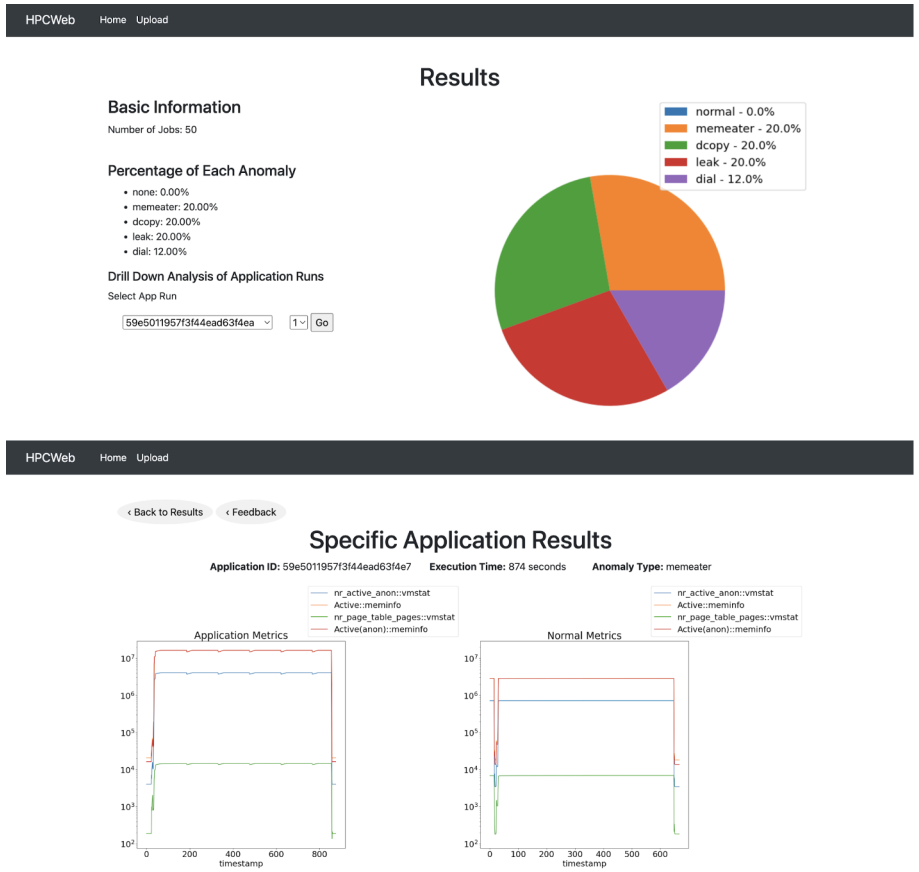
[Aksar et al., EuroPar'21 https://github.com/peaclab/Proctor]

Deployment - Challenges

- Compiling applications properly with different input decks requires domain expertise
- Verifying that synthetic anomalies are creating the desired impact on application runs is not trivial
- Compute node-specific deployment:
  - High training and maintenance cost
    - Hyperparameter tuning
    - Selecting new anomaly threshold for each model
- Generic deployment:
  - Developing a generic ML model that accurately detects/diagnose anomalies for thousands of compute nodes is hard

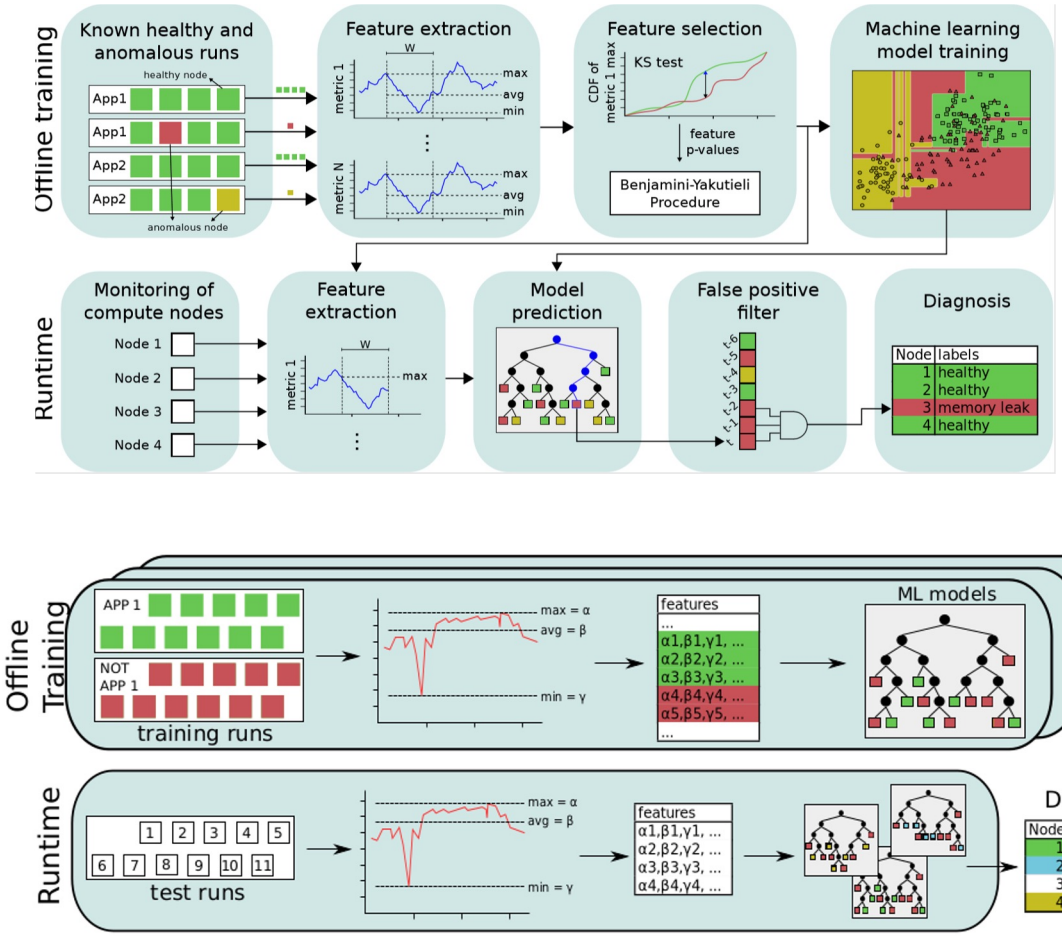
Web Based Anomaly Diagnosis Framework

- To make our supervised ML-based anomaly diagnosis framework widely accessible, we developed a website
- Determines anomalies in user uploaded system telemetry data
- Website link: http://ai4hpc.bu.edu/



Takeaways

- Diagnosing performance variation/problems
  - Supervised training setup:
    - Works well if there are sufficient amount of labeled samples
    - Unfortunately, that's not case in the real-world production system scenario
  - Semi-supervised training setup:
    - Achieves better than fully supervised framework when there are **limited** labeled samples
    - Active learning for **minimizing** the labeling cost
- Application discovery
  - At development
  - At installation
  - During execution



References

- Diagnosing Performance Variations in HPC Applications Using Machine Learning *[Tuncer et al., ISC'17]*
- Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning *[Tuncer et al., TPDS'18]*
- Taxonomist: Application Detection Through Rich Monitoring Data *[Ates et al., EuroPar'18 - github.com/peaclab/taxonomist]*
- HPAS: An HPC Performance Anomaly Suite for Reproducing Performance Variations *[ICPP'19 - github.com/peaclab/HPAS]*
- Proctor: A Semi-Supervised Performance Anomaly Diagnosis Framework for Production HPC Systems [Aksar et al., ISC'21 github.com/peaclab/Proctor]
- E2EWatch: An End-to-End Anomaly Diagnosis Framework for Production HPC Systems *[Aksar et al., EuroPar'18 github.com/peaclab/E2EWatch]*
- ALBADross: Active Learning Based Anomaly Diagnosis for Production HPC Systems [Aksar et al., Cluster'22 github.com/peaclab/ALBADross]

Backup Slides

Vision: ML to Improve and Automate HPC Management

- Significant outages, major losses of profit, problems due to widespread vulnerabilities, … are all very common in computing systems
- Automated analytics can tremendously help solve or pinpoint many important problems
- Many open research problems exist in the design of data-driven management solutions