Introduction to Parallel Computing (CMSC416 / CMSC818X)

Machine Learning and HPC

Alan Sussman, Department of Computer Science

UNIVERSITY OF MARYLAND

Announcements

- Quiz 3 will be posted on tomorrow, Wednesday, May 10, 11AM
  - In ELMS, for 24 hours
  - Mainly on topics since last quiz
- Course evaluation: https://www.courseevalum.umd.edu

Why machine learning for parallel computing/HPC?

- Proliferation of performance data
  - On-node hardware counters
  - Switch/network port counters
  - Power measurements
  - Traces and profiles
- Supercomputing facility and data center data
  - Job queue logs, performance
  - Sensors: temperature, humidity, power

Types of ML-related tasks in HPC

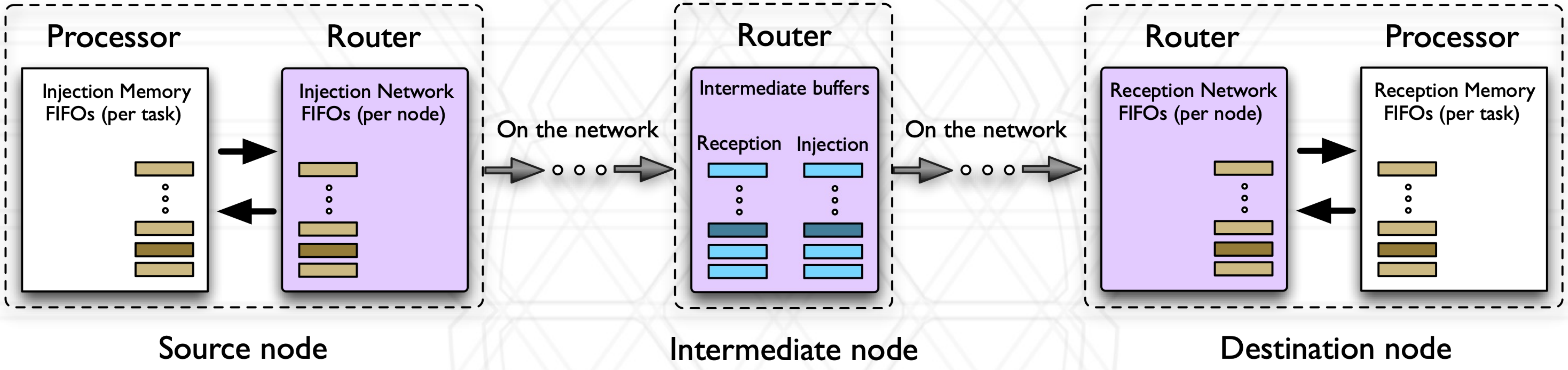
- Auto-tuning: parameter search
  - Find a well performing configuration
- Predictive models: time, energy, …
  - Predict system state in the future
  - Time-series analysis
- Identifying root causes/factors
  - For errors, failures (hardware/software), performance, …

Example 1 - Network congestion

- Responsible for performance degradation, variability and poor scaling
- Congestion and its root causes not well understood
- Study network hardware performance counters and their correlation with execution time
- Use supervised learning to identify hardware components that lead to congestion and performance degradation

https://www.osti.gov/servlets/purl/1184730

Life of a message packet



Hardware resource
Source node
Network link
Intermediate router
All

Experiment - Gathering data for machine learning

- Collect network hardware counters data on IBM Blue Gene/Q and use a functional simulator

	Hardware resource	Contention indicator
	Source node	Injection FIFO length
	Network link	Number of sent packets
	Intermediate router	Receive buffer length
	All	Number of hops (dilation)

- Use Rubik task mappings to get a range of execution times for the same application
  - Rubik is a tool from LLNL that maps tasks to nodes in torus or mesh-connected cluster

Gathering data for machine learning - features

	Feature name	Description
All Resources	avg dilation AO	Avg. dilation of average outliers (AO)
	max dilation	Maximum dilation
	sum dilation AO	Sum of dilation of AO
Network Link	avg bytes	Avg. bytes per link
	avg bytes AO	Avg. bytes per link for AO
	avg bytes TO	Avg. bytes per link for top outliers (TO)
	max bytes	Maximum bytes on a link
	#links AO bytes	No. of AO links w.r.t. bytes
Intermediate Router	avg stalls	Avg. receive buffer length
	avg stalls AO	Avg. receive buffer length for AO
	avg stalls TO	Avg. receive buffer length for TO
	max stalls	Maximum receive buffer length
	#links AO stalls	No. of AO links w.r.t. recv buffer length
	avg stallspp	Avg. number of stalls per rcv'd packet
	avg stallspp AO	Avg. no. of stalls per packet for AO
	avg stallspp TO	Avg. no. of stalls per packet for TO
	max stallspp	Maximum number of stalls per packet
	#links AO stallspp	No. of AO links w.r.t. stalls per packet
Source Node	max inj FIFO	Maximum injection FIFO length

Experimental Setup

- Three benchmarks: 5-point 2D Halo, 15-point 3D Halo, All-to-all over sub-communicators – MPI codes
- Two scientific applications: pF3D, MILC – number of task mappings (from Rubik)

	2D Halo		3D Halo		Sub A2A		MILC	pF3D	Total
#Nodes	16 KB	4 MB	16 KB	4 MB	16 KB	4 MB			
1024	84	84	84	84	84	84	208	94	806
4096	84	84	84	84	84	84	103	103	710
Total	168	168	168	168	168	168	311	197	1516

- Regression methods in scikit-learn (a Python ML library)
  - extremely randomized trees, gradient boosted regression trees

Predicting the execution time

- Scale the input features to values between 0 and 1
- Split samples into training and testing set (2/3 : 1/3)
- Generate all possible combinations (2^{19}) of the 19 input features
- Parallel runs to try all combinations and report prediction scores

Evaluation criteria

- Kendall rank correlation coefficient

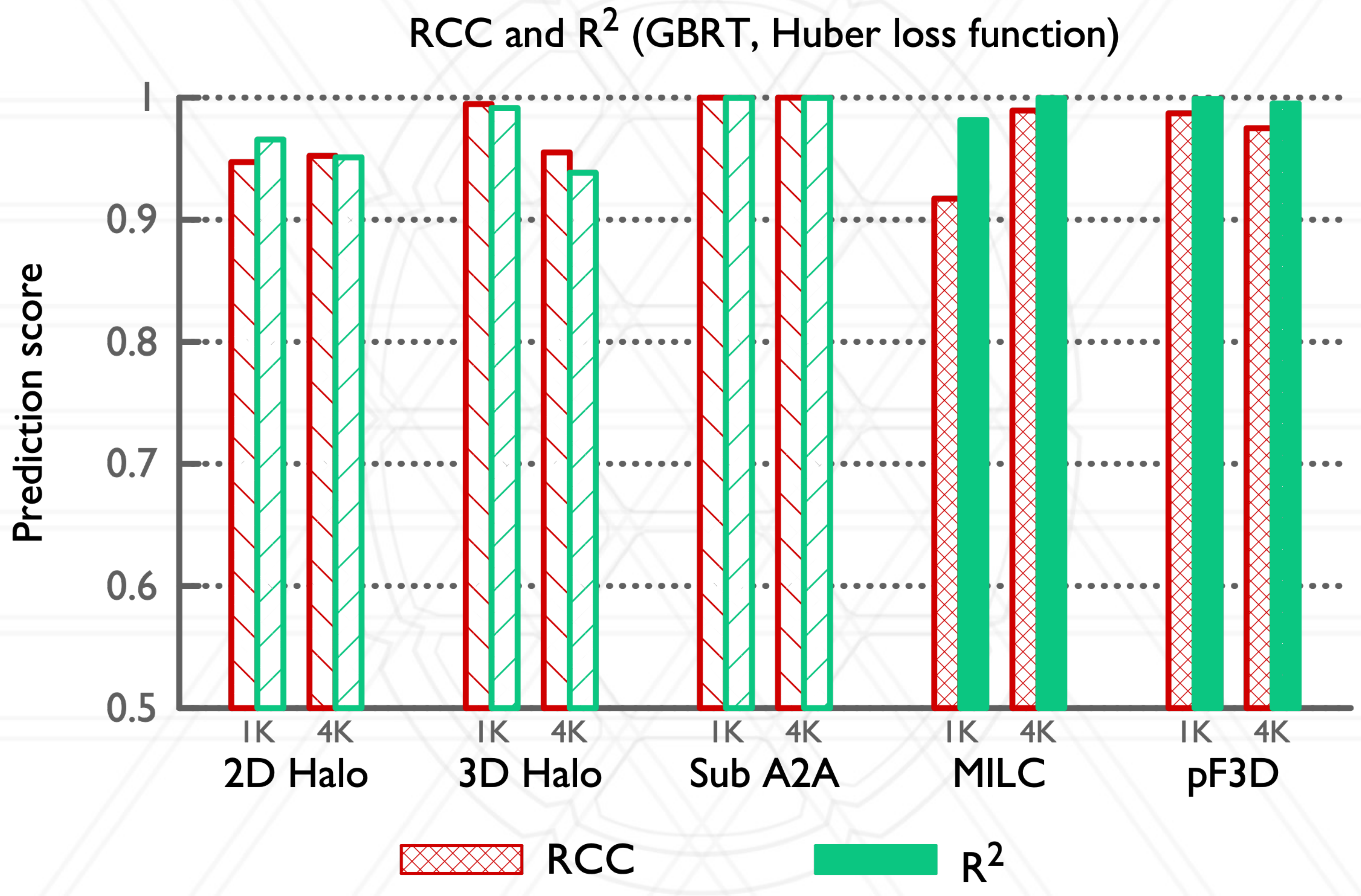
$$RCC = \Big(\sum_{0<=i<n}\sum_{0<=j<i} concord_{ij}\Big) / (\frac{n(n-1)}{2})$$

$$concord_{ij} = \begin{cases} 1, & \text{if } x_i >= x_j \ \&\ y_i >= y_j \\ 1, & \text{if } x_i < x_j \ \&\ y_i < y_j \\ 0, & \text{otherwise} \end{cases}$$

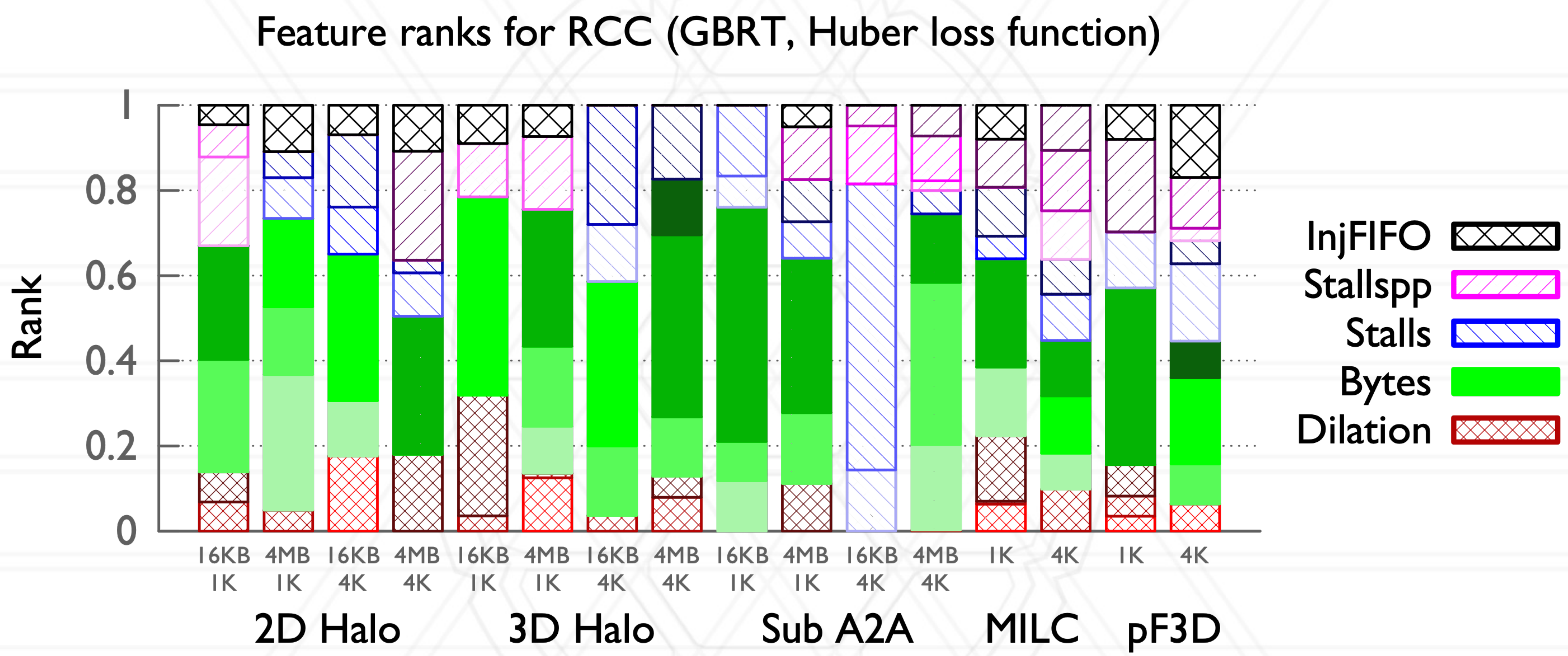
- Coefficient of determination, R^2

$$R^2(y, \hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Prediction on individual datasets

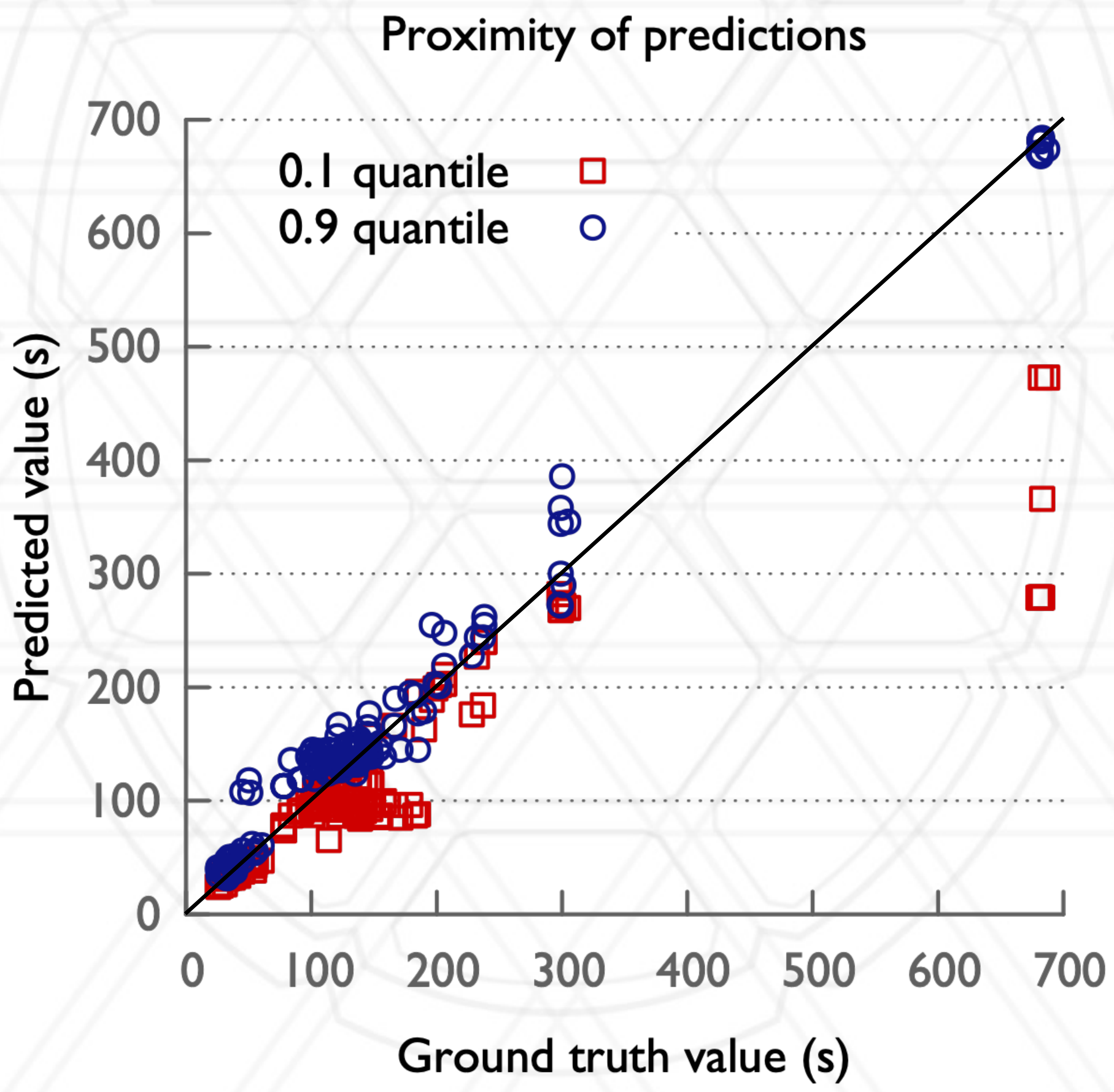


Feature importance (individual datasets)

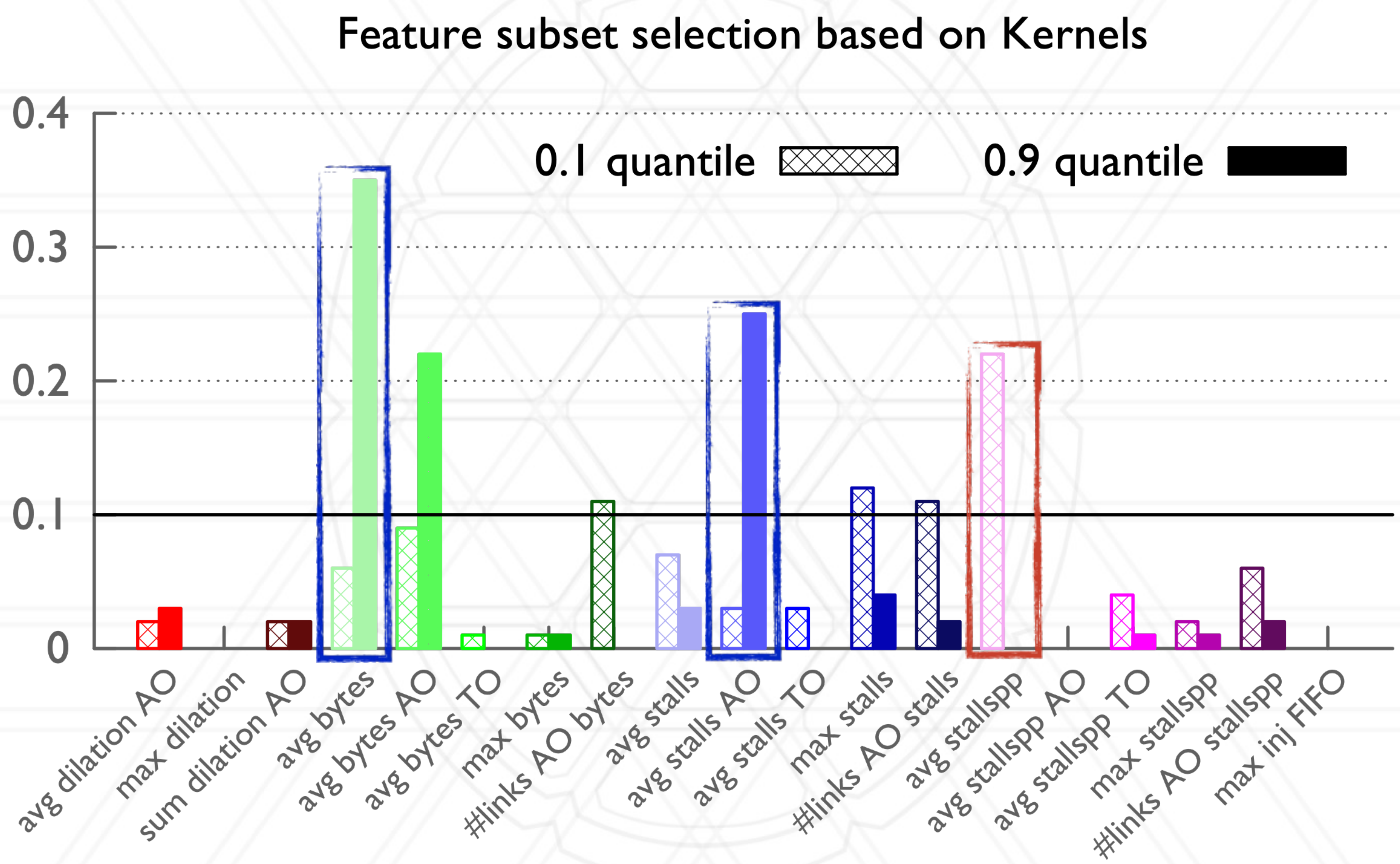


Identifying important features

- Use quantile loss function in the GBRT regressor



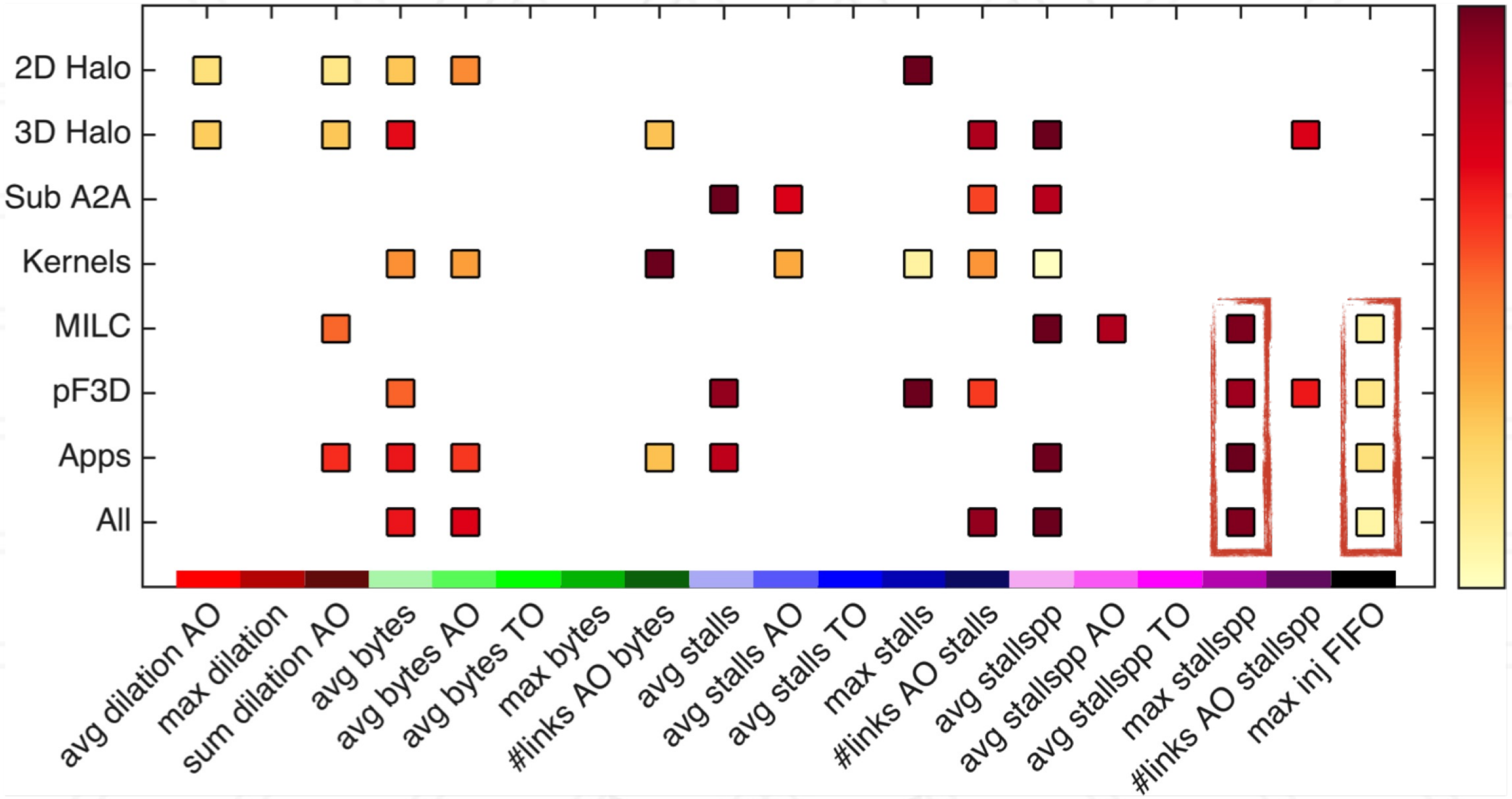
Identifying important features



Technique for feature selection

- Create split of dataset into training and testing set
- Learn GBRT regressor with quantile loss function at 0.1 quantile and 0.9 quantile
- Identify feature subsets that are important at different quantiles
- Use the subsets to identify new feature importances

The causes of network congestion



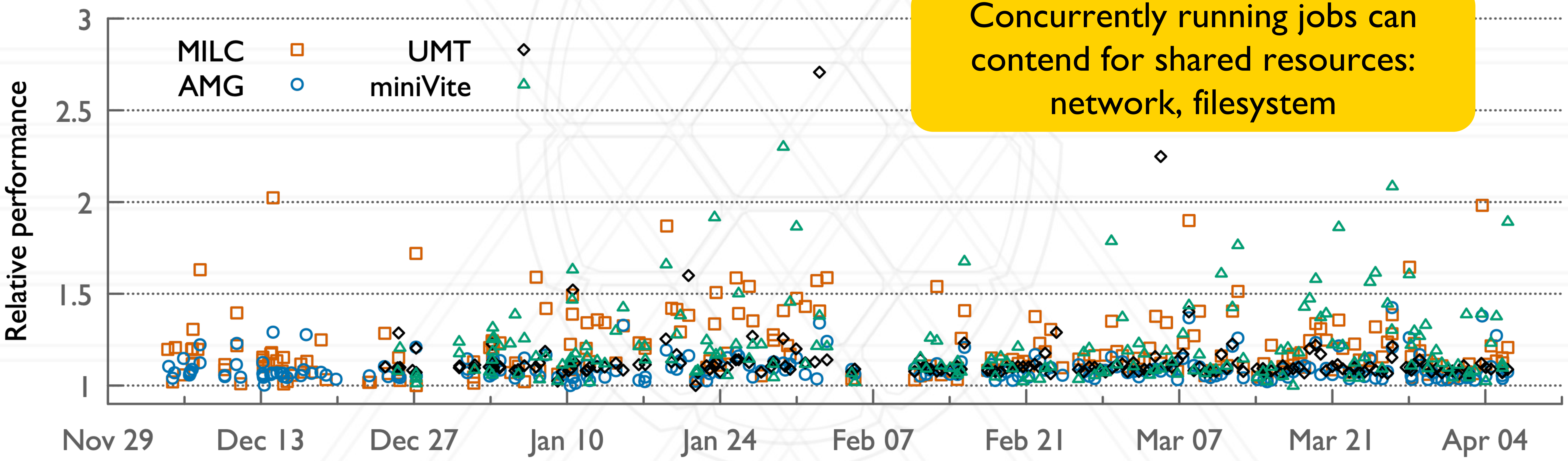
Feature ranks (maroon/red is high and yellow is low)

The causes of network congestion

- Average and maximum length of receive buffers
- Average load on network links
- Maximum length of injection FIFOs

Example 2 - Interference from other jobs

Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019



Concurrently running jobs can contend for shared resources: network, filesystem

Bhatele et al. The case of performance variability on dragonfly-based systems, IPDPS 2020

Data analytics study to understand variability

- Primarily focus on variability arising from sub-optimal communication on the network
- Set up controlled experiments on a dragonfly-based Cray system:
  - Submit jobs of the same applications periodically in the batch queue for ~4 months
- Collect network hardware counters per iteration for each job and other data described later
- Use machine learning to analyze the gathered performance data

Run four applications in control jobs

- Gather network hardware counters on Aries routers connected to my jobs' nodes
- Hardware counters and execution time recorded per iteration

Six datasets

Application	No. of nodes	Input Parameters
AMG 1.1	128	-P 32 16 16 -n 32 32 32 -problem 2
AMG 1.1	512	-P 32 32 32 -n 32 32 32 -problem 2
MILC 7.8.0	128	n128_large.in
MILC 7.8.0	512	n512_large.in
miniVite 1.0	128	-f nlpkkt240.bin -t 1E-02 -i 6
UMT 2.0	128	custom_8k.cmg 4 2 4 4 4 0.04

Other sources of data for analytics

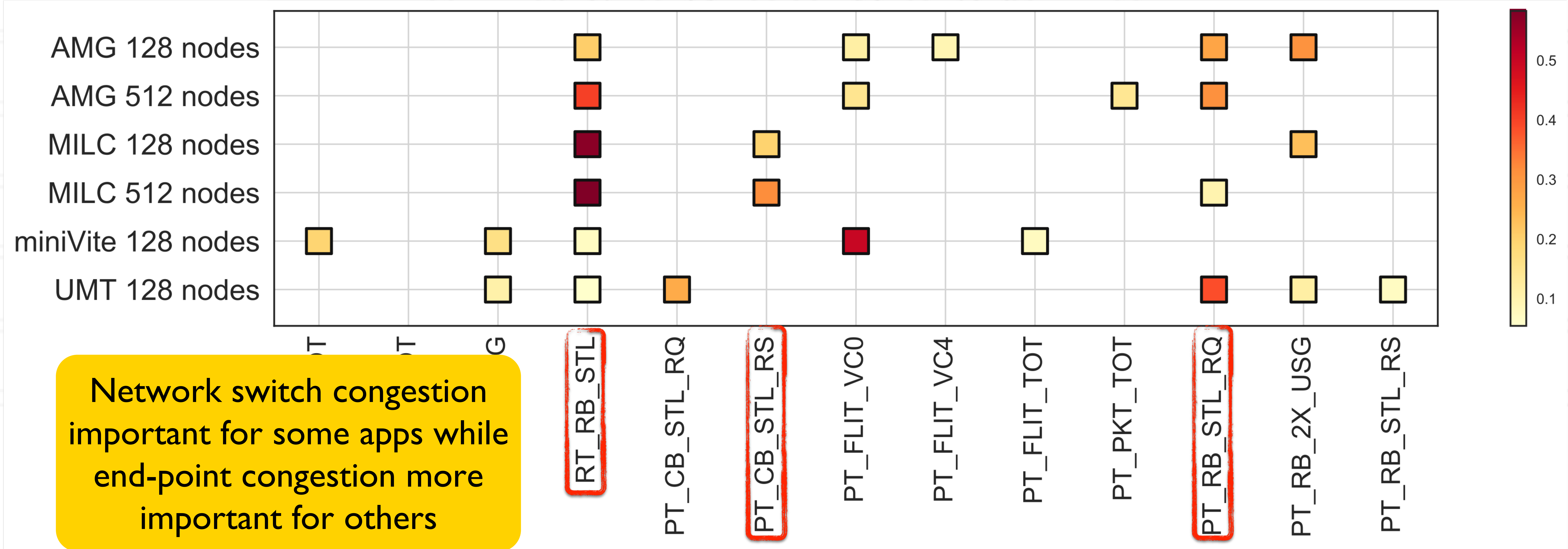
- Job queue logs
  - Information about jobs running concurrently with a specific control job
- Job placement
  - Number of unique groups and routers to which a control job is assigned
- System-wide counters for all Aries routers gathered using LDMS (Lightweight Distributed Metric Service)
  - All routers: all routers connected to compute or I/O nodes
  - I/O routers: only routers connected to I/O servers

Analysis I: Identifying predictors of deviation

- Execution times and network counters data are available for each iteration of the application
    - Each iteration is treated as an independent sample
- Create models to predict the deviation of the execution time instead of the absolute time
- Use gradient boosted regression to generate a predictive model and recursive feature elimination (RFE) to study feature importances

Results: Identifying predictors of deviation

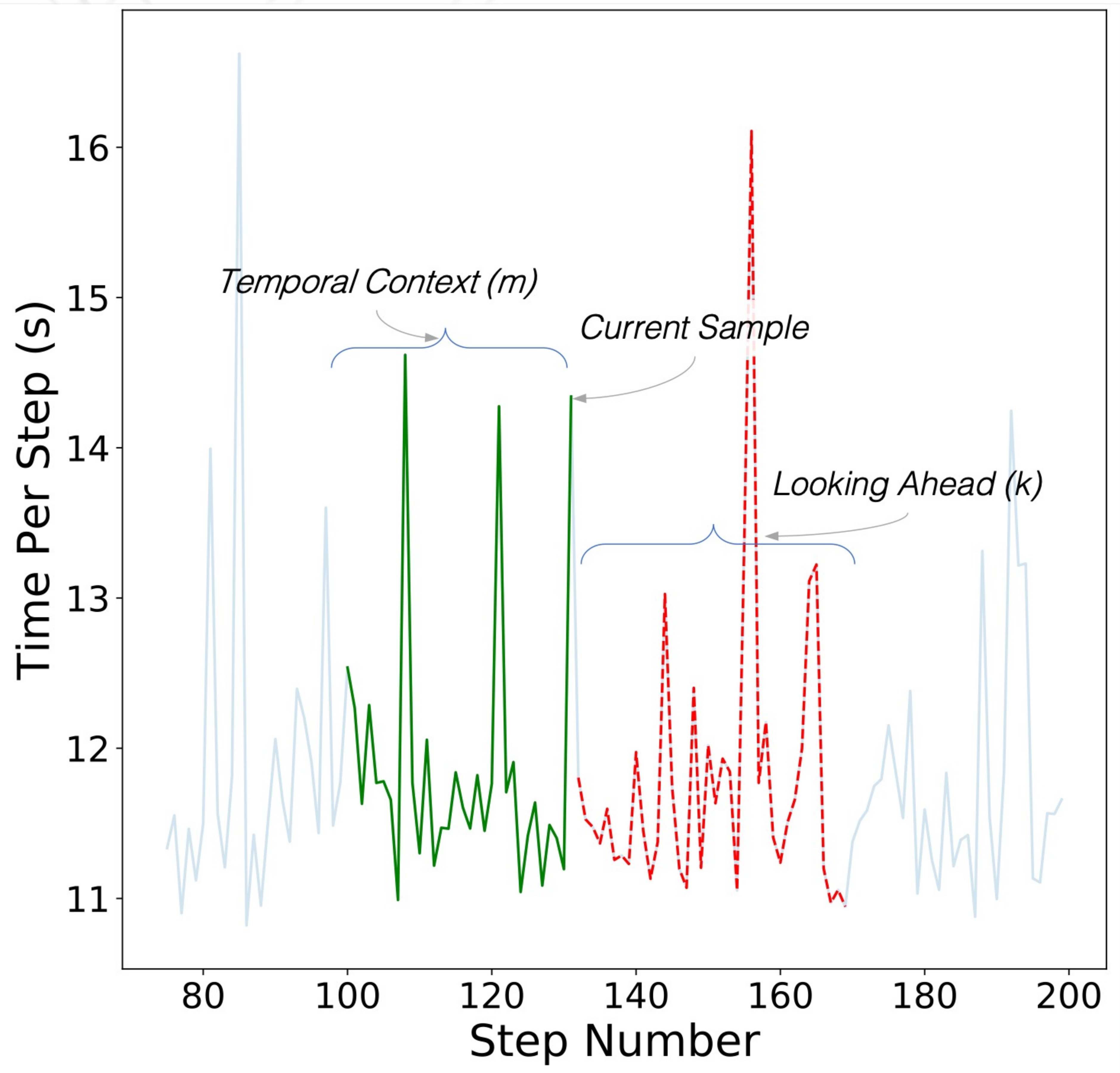
Relevance scores of each counter in predicting the deviation from mean behavior for the different datasets.



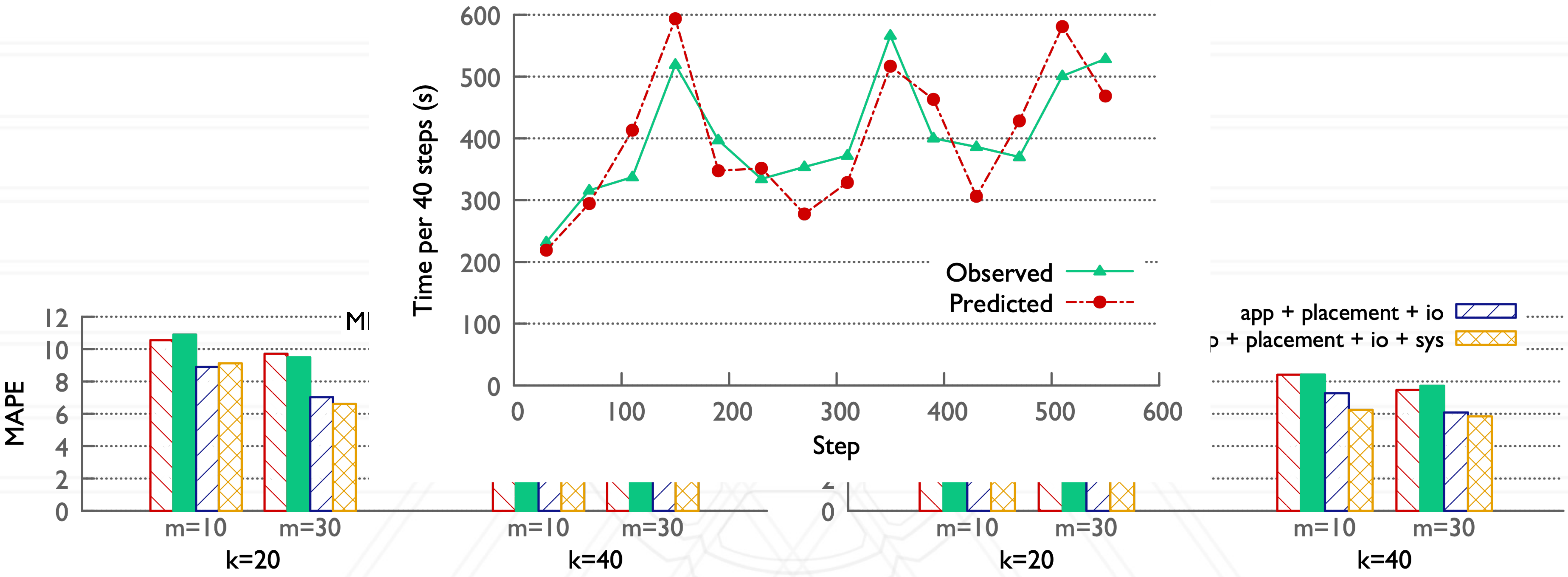
Network switch congestion important for some apps while end-point congestion more important for others

Analysis II: Forecasting within-run variation

- Idea is to predict next *k* time steps based on knowledge of *m* previous time steps
- Use a sliding window approach to create the training set
- Use the popular *scalar dot-product* attention model along with a fully connected neural network
- Explore using different groups of features to understand the impact on model accuracy



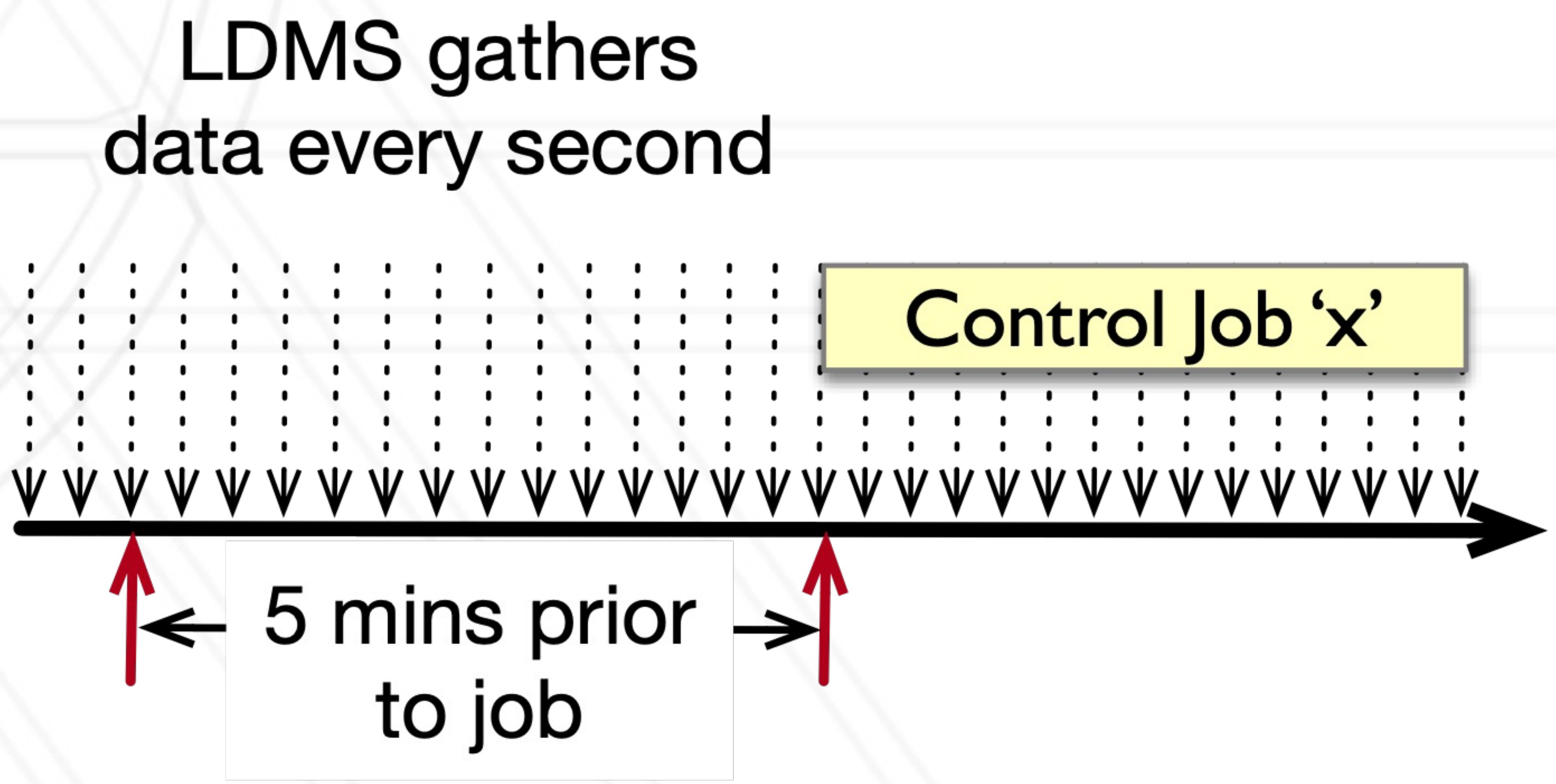
Results: Forecasting within-run variation



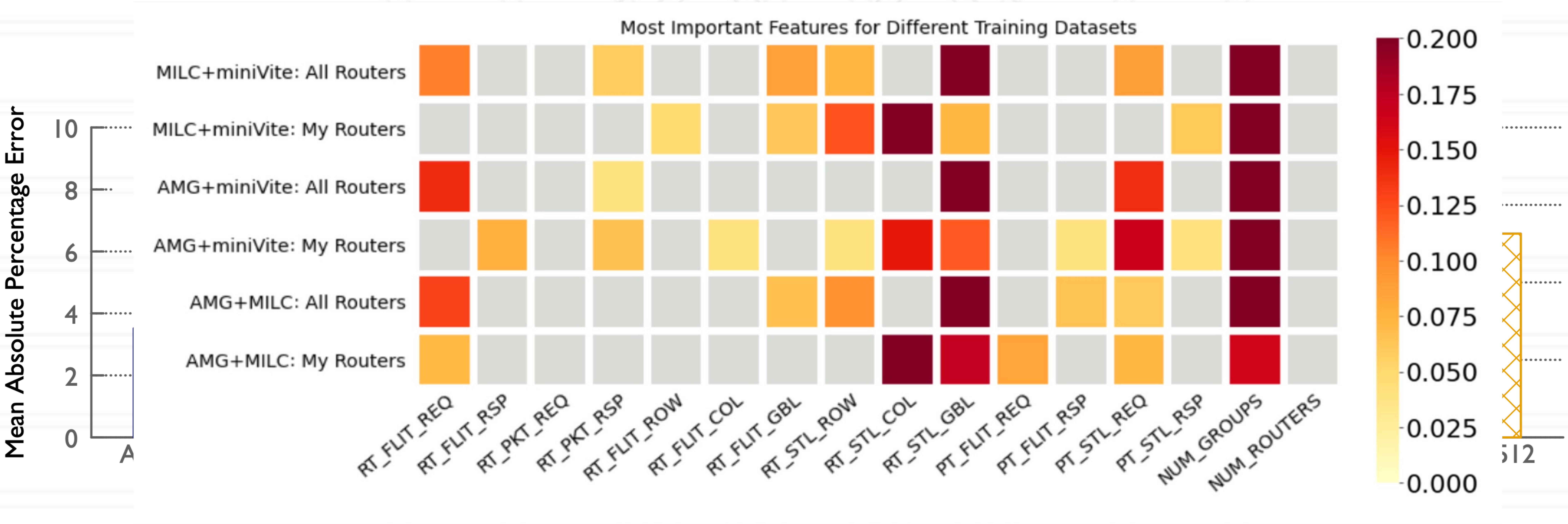
MAPE = Mean Absolute Percentage Error, m = temporal context, k = predicting future time steps

Analysis III: Using only system data

- Use system state before a job starts running to predict performance
- No application-specific features are used
- Train a 2-layer neural network that combines multiple datasets
- Goal: develop application-agnostic models



Results: Predicting perf. of unseen jobs

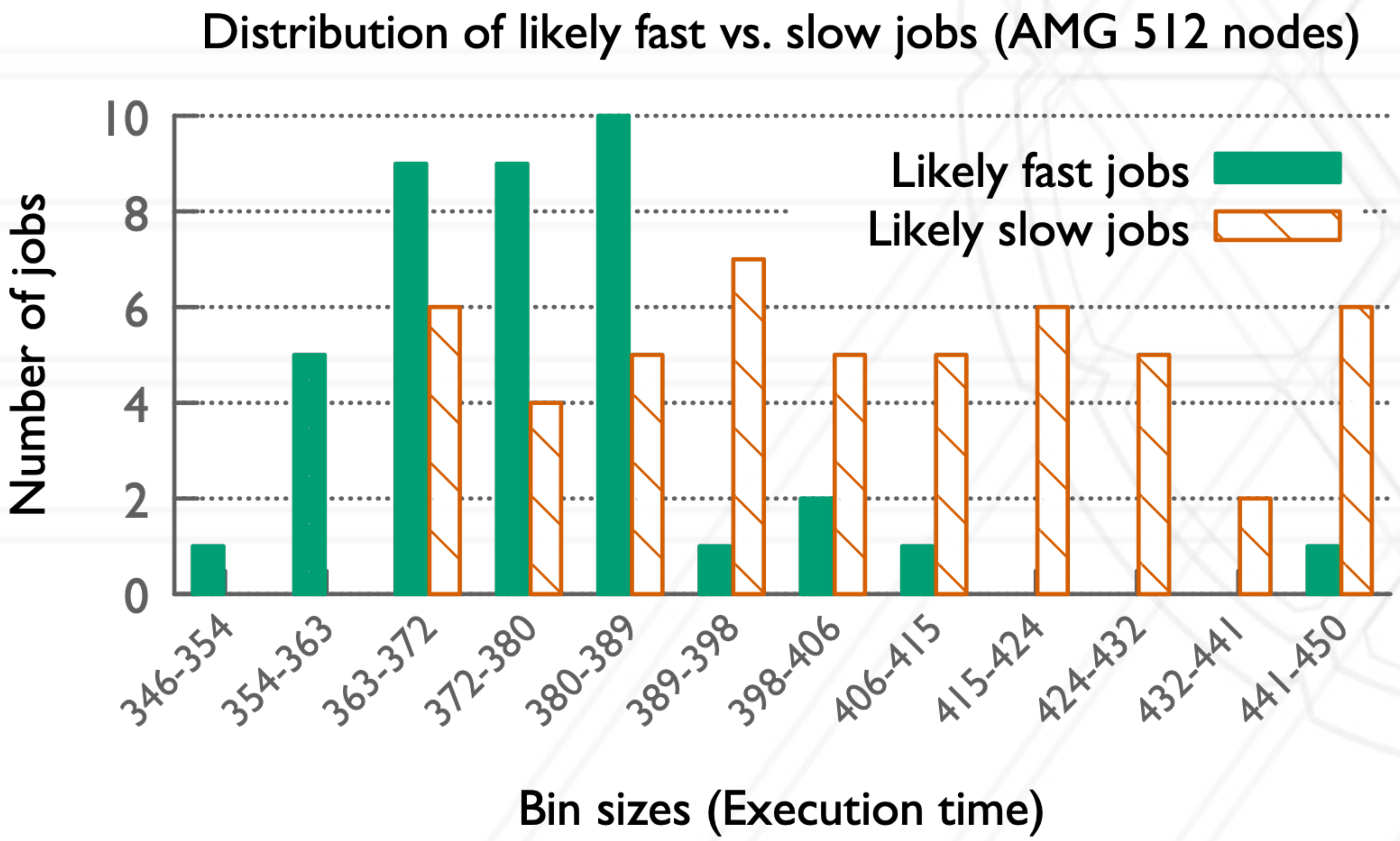


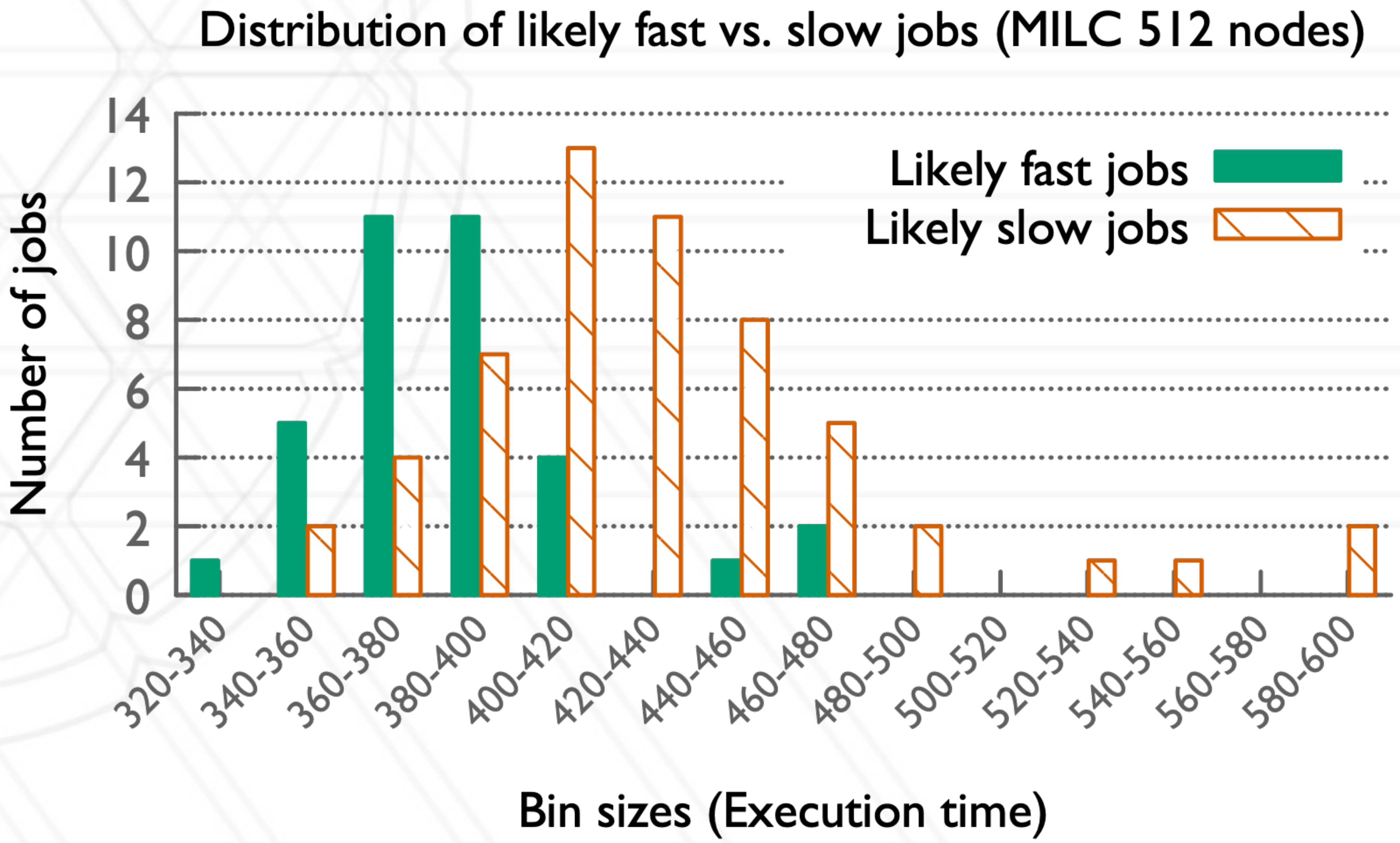
Based on global routers

Ian Costello et al. Analytics of Longitudinal System Monitoring Data for Performance Prediction. https://arxiv.org/abs/2007.03451

Results: Potential impact on job schedulers

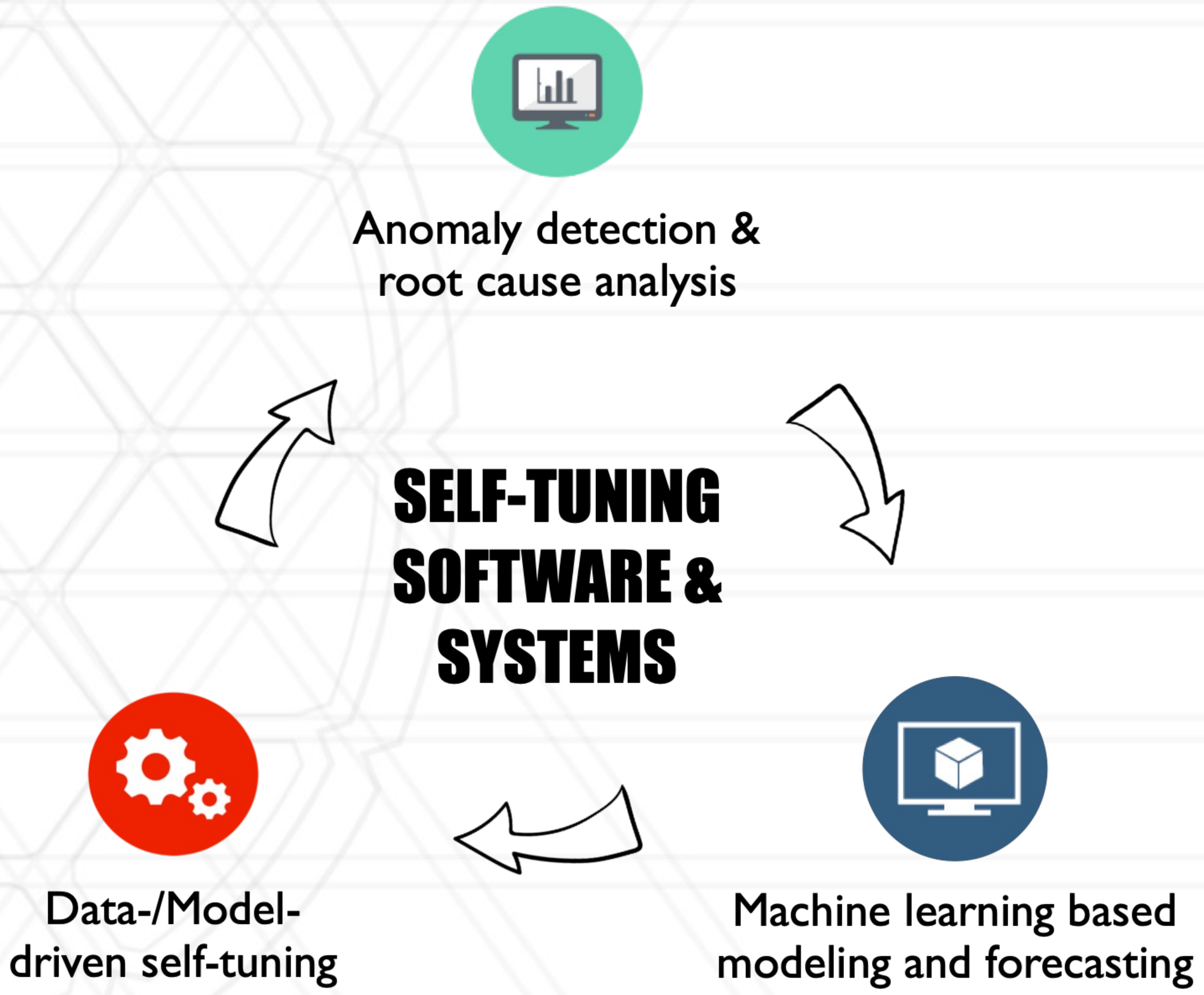
- Classify jobs into likely fast or likely slow based on values of three most important features
- Based on whether values of these features are above or below the median





How to minimize performance variability?

- Topology-aware job scheduling
- Self-tuning systems
  - Adaptive congestion-aware routing
  - Adaptive scheduling of jobs



Availability of large-scale monitoring data

- Several Department of Energy laboratories are using LDMS to record monitoring data: LLNL/LC, LBNL/NERSC, ANL/ALCF
- Vast quantities of rich but noisy data: on-node (flops, memory, caches), network, filesystem, power, cooling

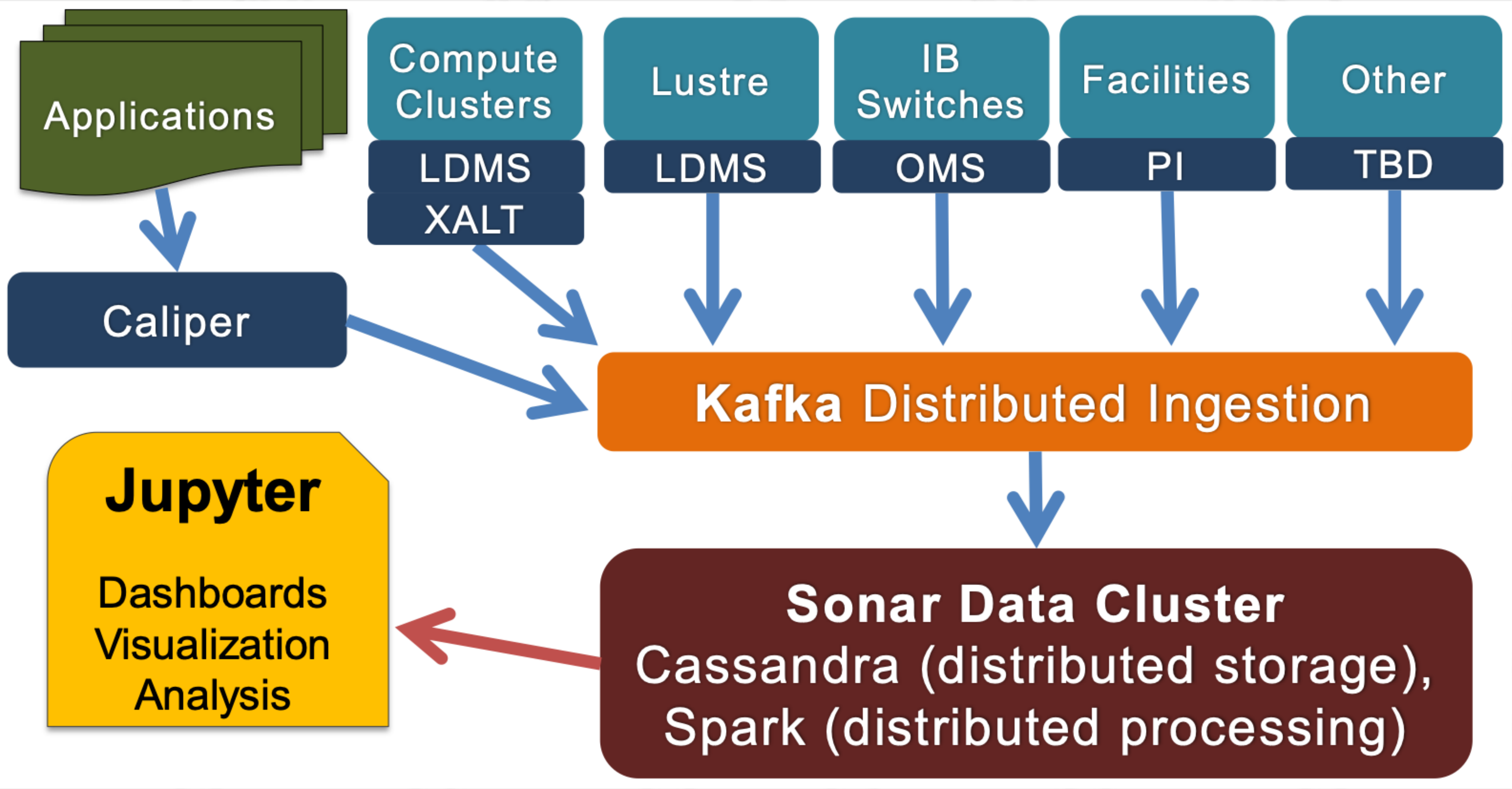
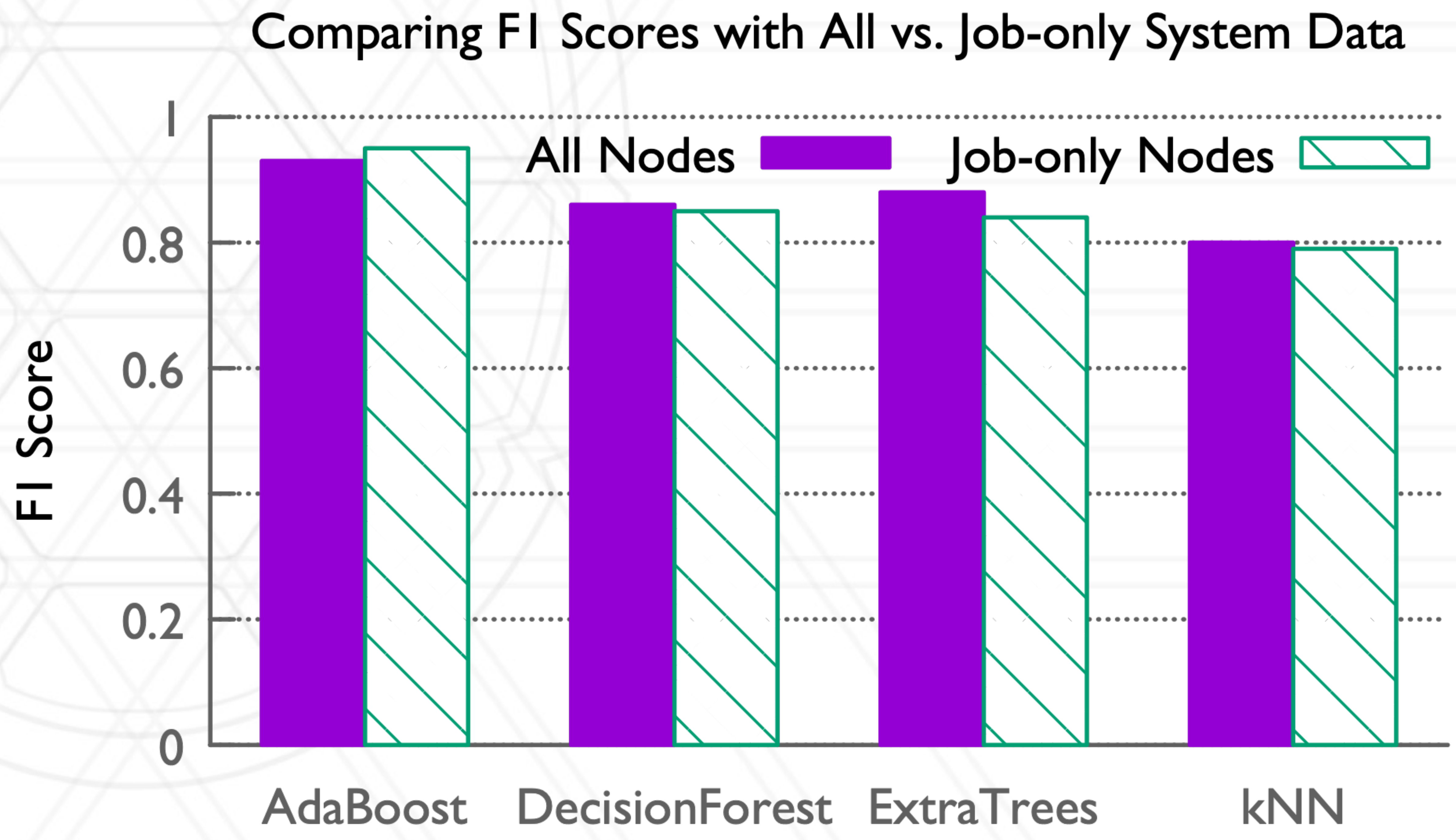


Image from Kathleen Shoga's slides at LLNL

Variability prediction

- Ran a large number of control jobs (hundreds per application): 7 different applications
- Train a classifier (AdaBoost) to predict if an app will experience variation

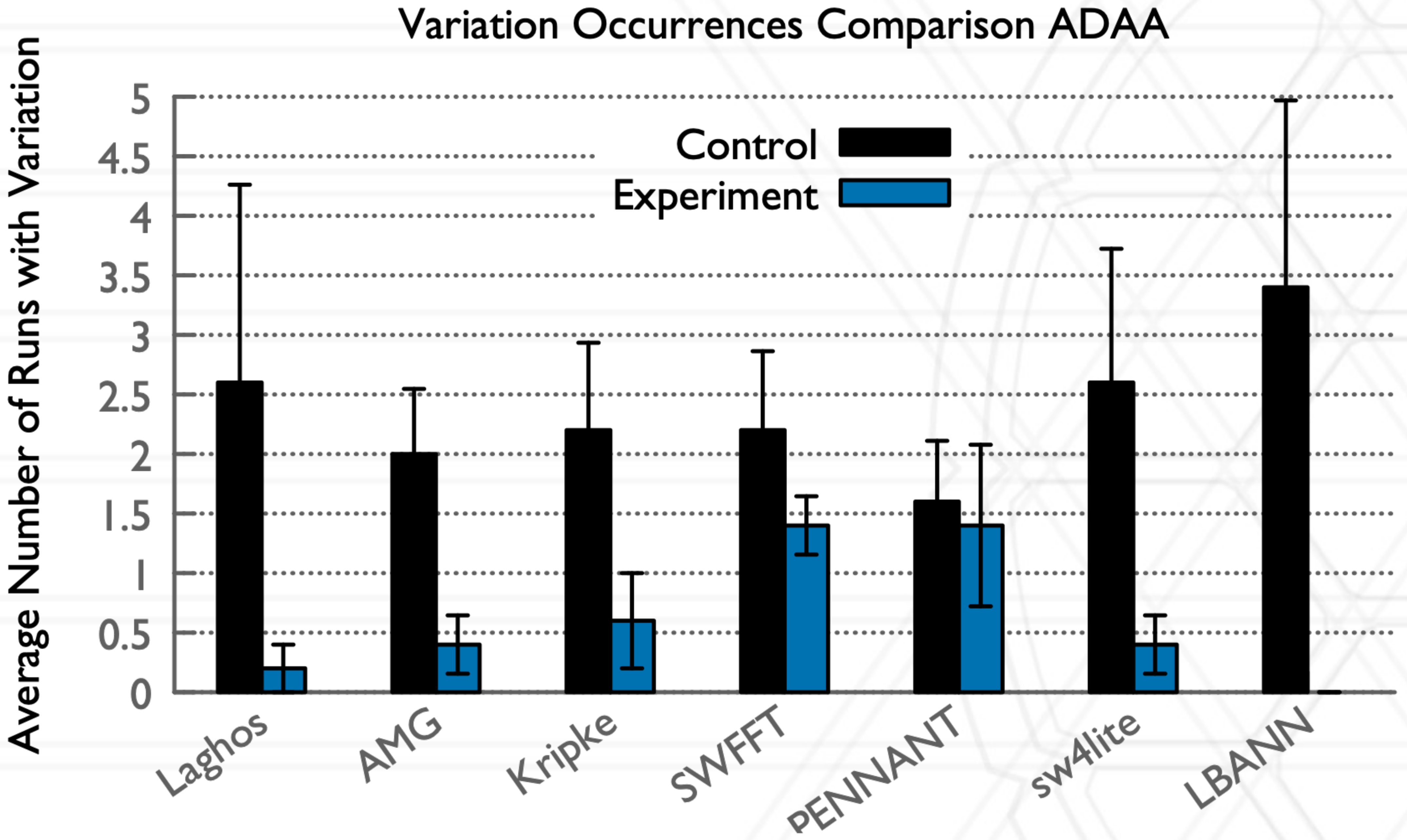
Input source	Counters	Features	Description
sysclassib	62	186	infiniband counters
opa_info	62	186	Omni-Path switch counters
lustre2_client	44	132	Lustre client metrics
MPI benchmarks	3	9	Execution time
Proxy applications	-	1	Compute Intensive
	-	1	Network Intensive
	-	1	I/O Intensive

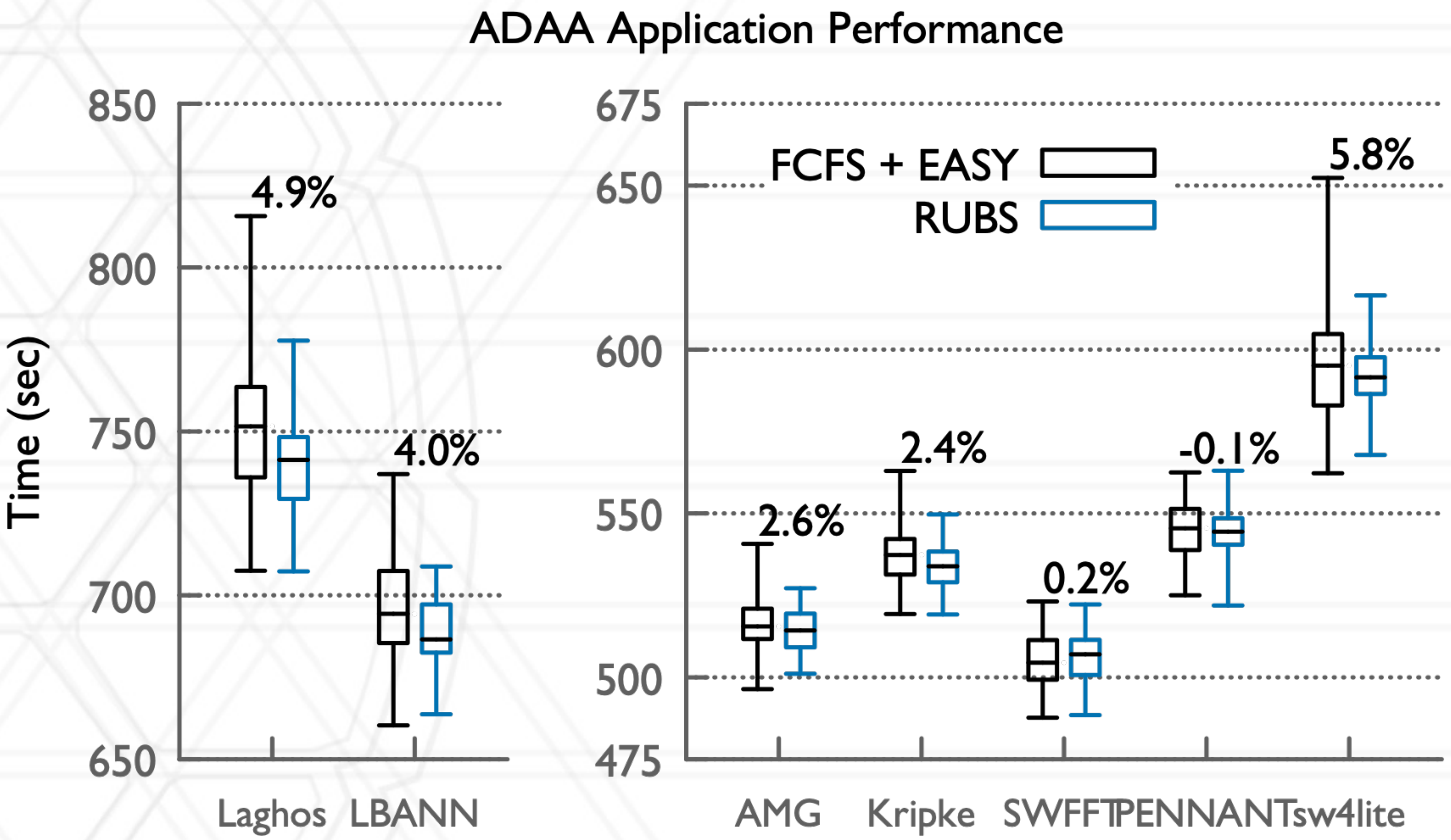


Example 3 - Self-tuning job scheduler

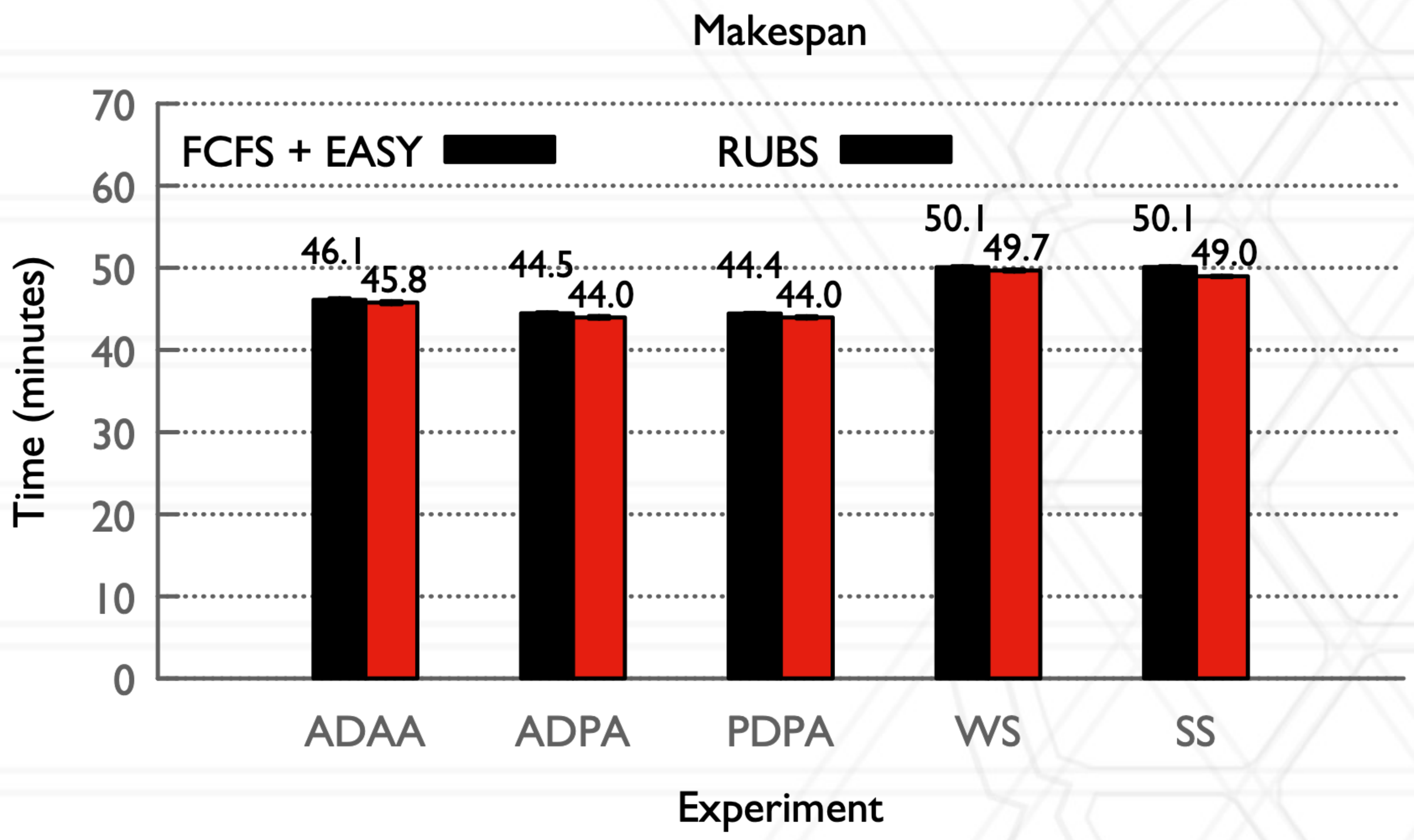
- Modify the job scheduler to:
  - Obtain recent values of system counters
  - Predict if the next job in the queue will experience variability
  - If yes, put it back in the queue and try scheduling the next job
- Leverage the Flux scheduler framework developed at LLNL
- Enables running a scheduler within a job partition allocated by the system scheduler (slurm)

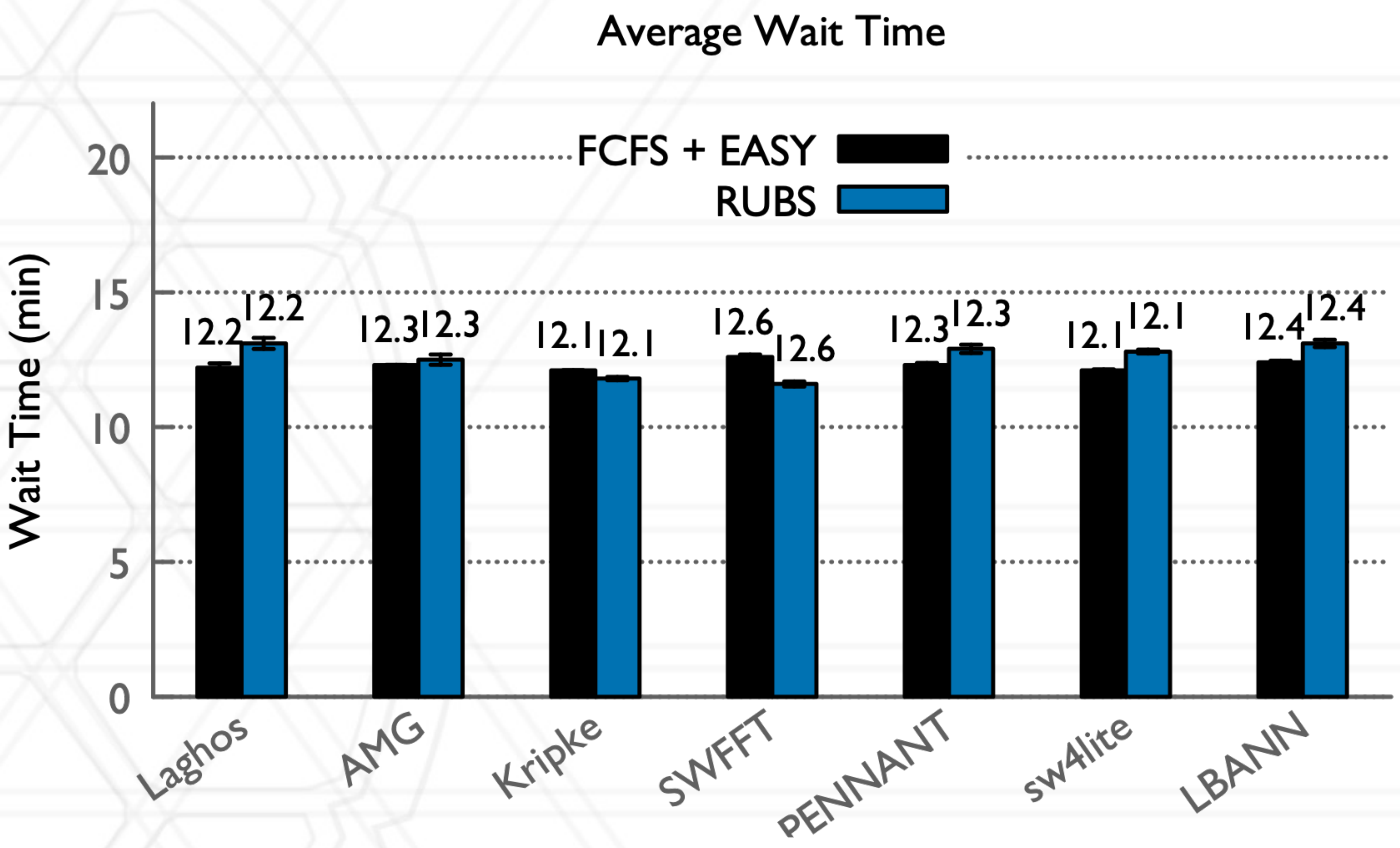
Application performance





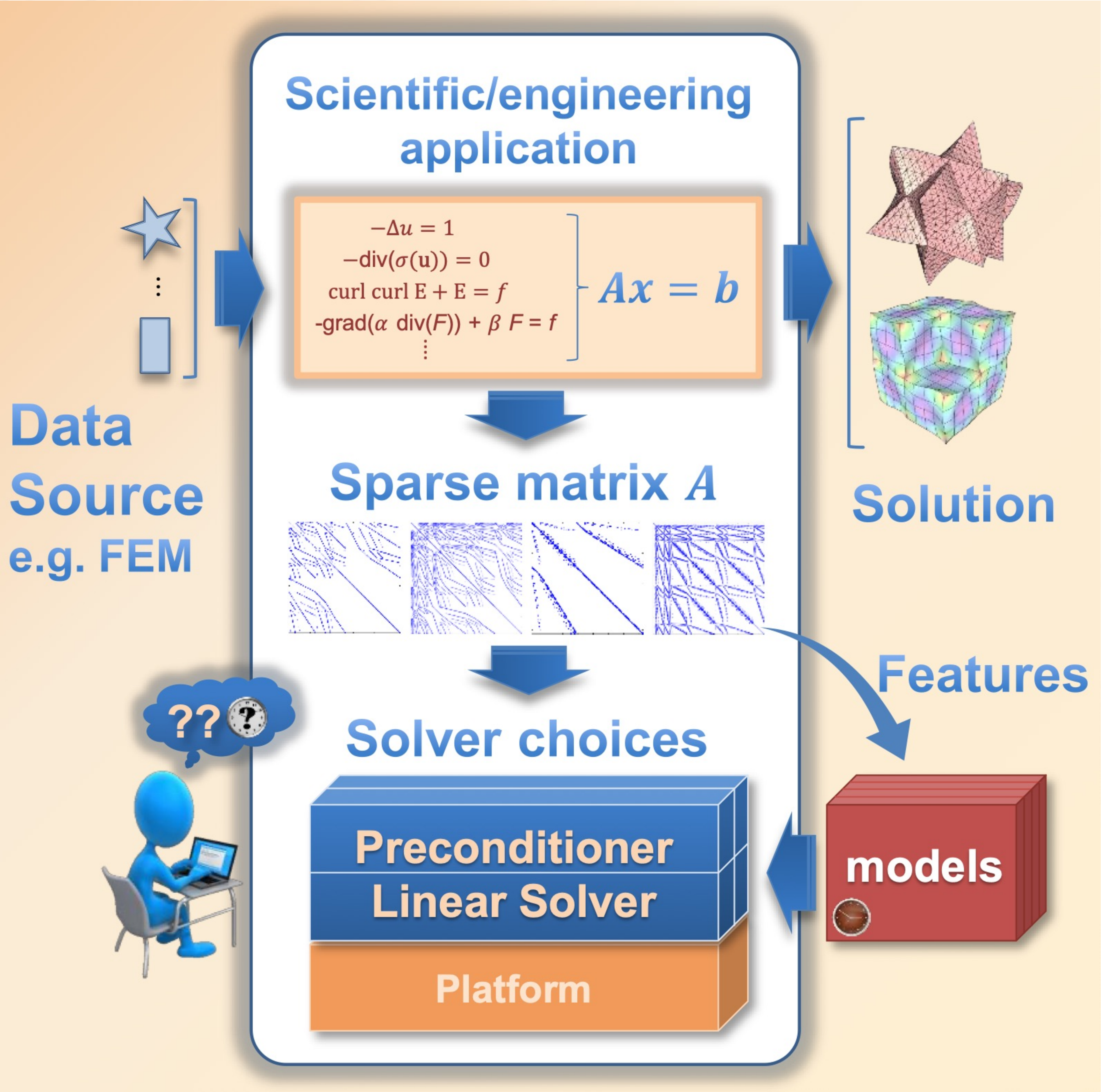
Scheduler throughput



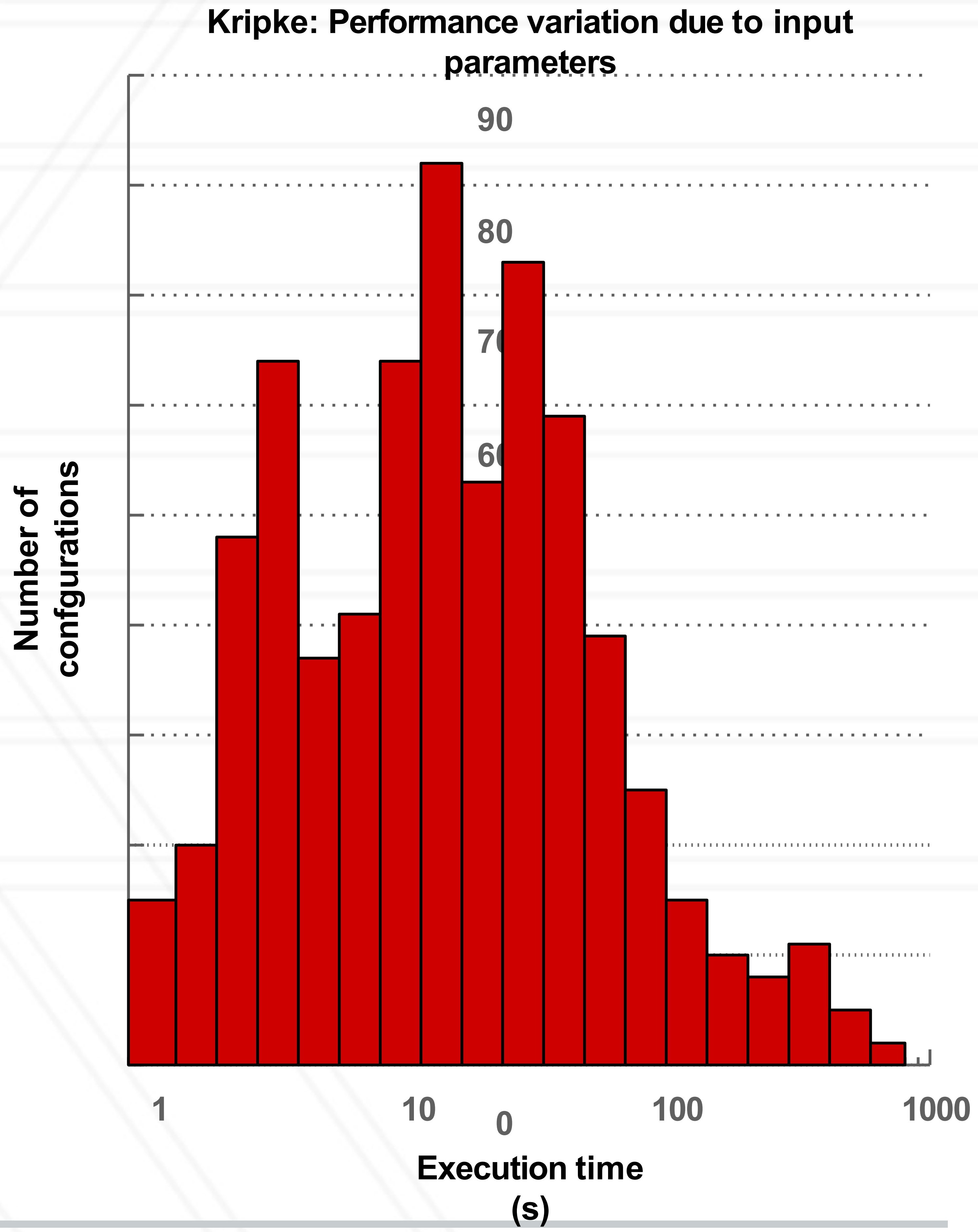


Identifying best performing code variants

- Many computational science and engineering (CSE) codes rely on solving sparse linear systems
- Many choices of numerical methods
- Optimal choice w.r.t. performance depends on several things:
  - Input data and its representation, algorithm and its implementation, hardware architecture

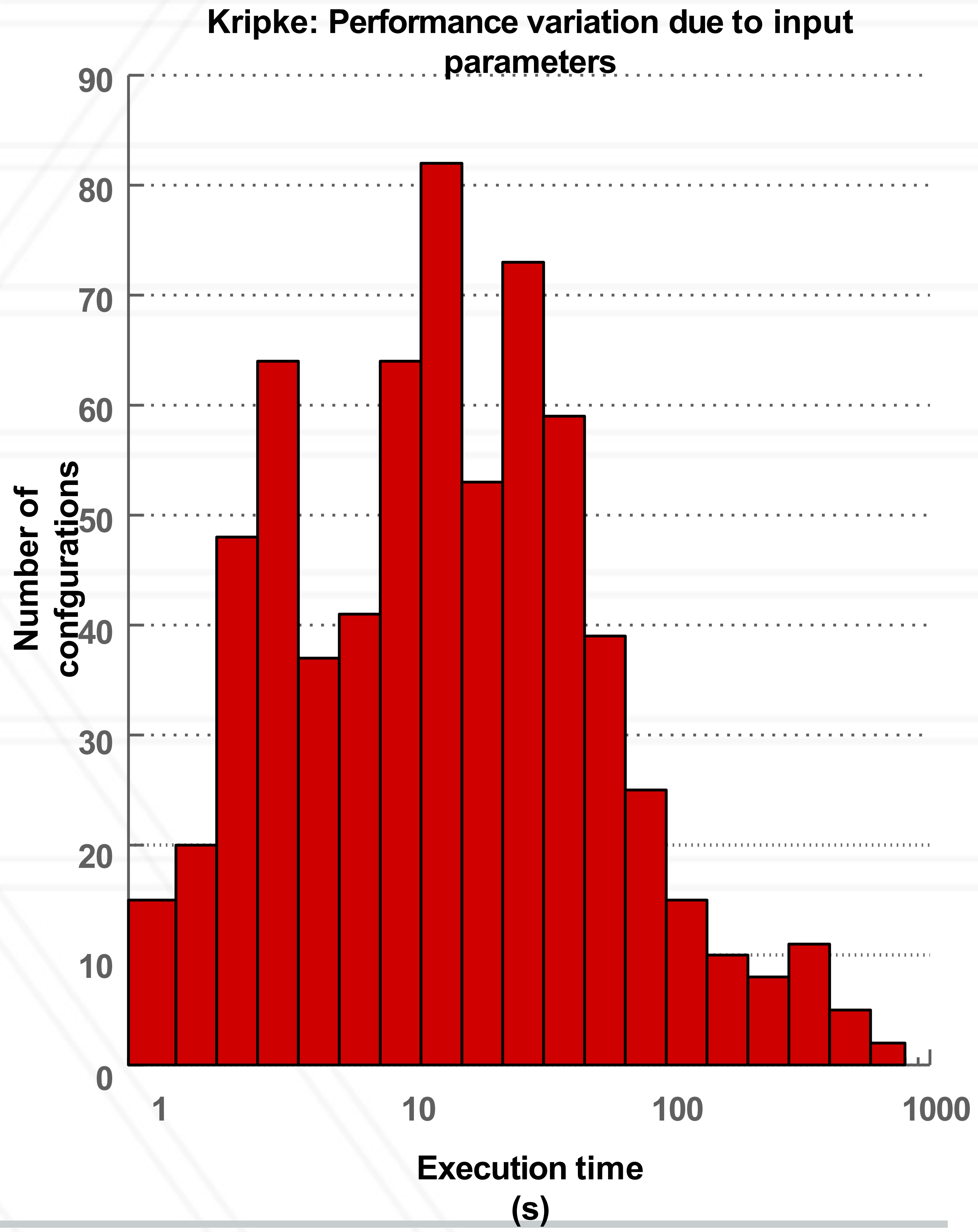


Auto-tuning with limited training data



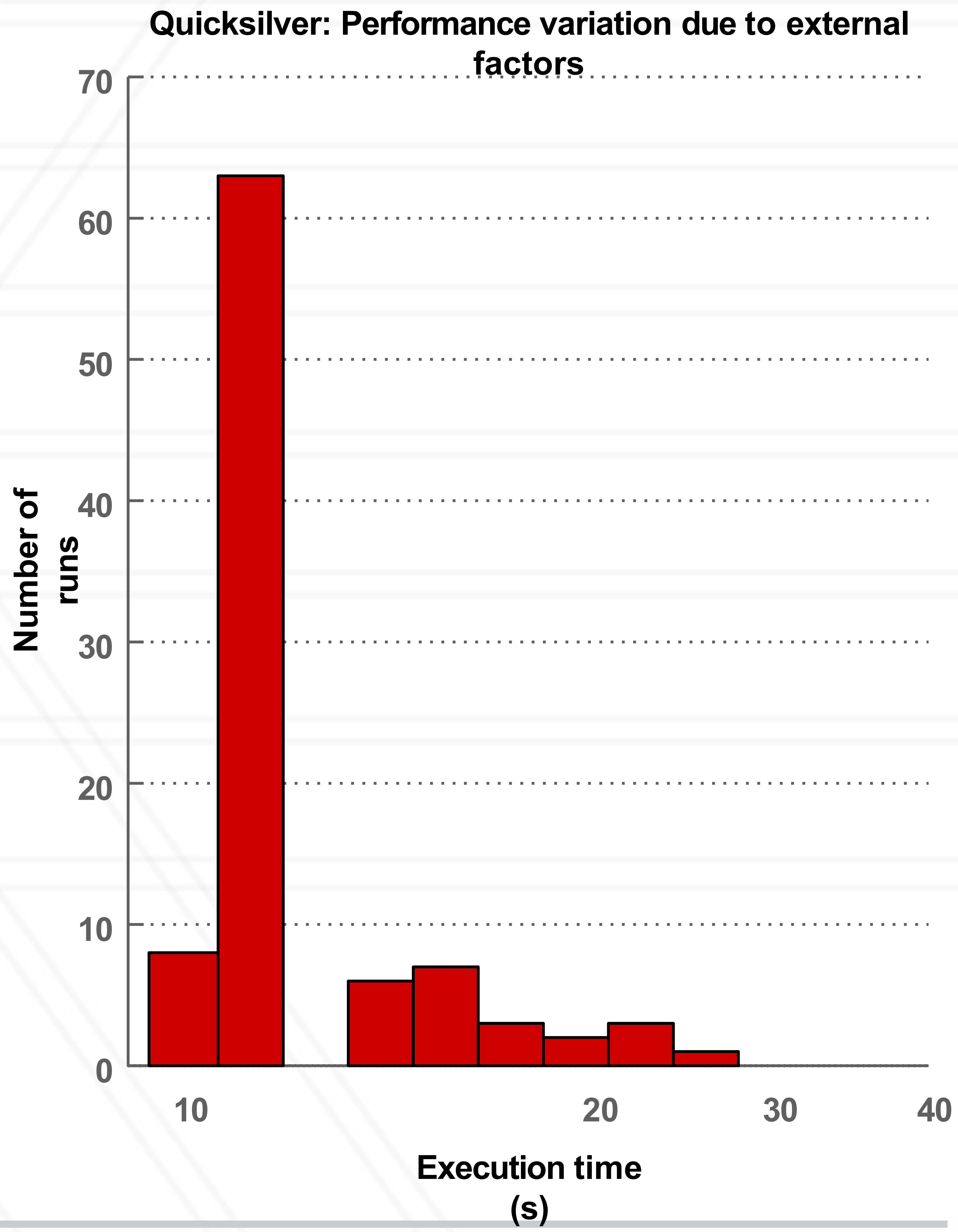
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters



Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters
- Performance also depends on:
  - Code changes, linked libraries
  - Compilers, architecture



Questions?

UNIVERSITY OF
MARYLAND