AMERICAN UNIVERSITY OF BEIRUT

FACULTY OF ARTS & SCIENCES

Hewlett Packard Enterprise

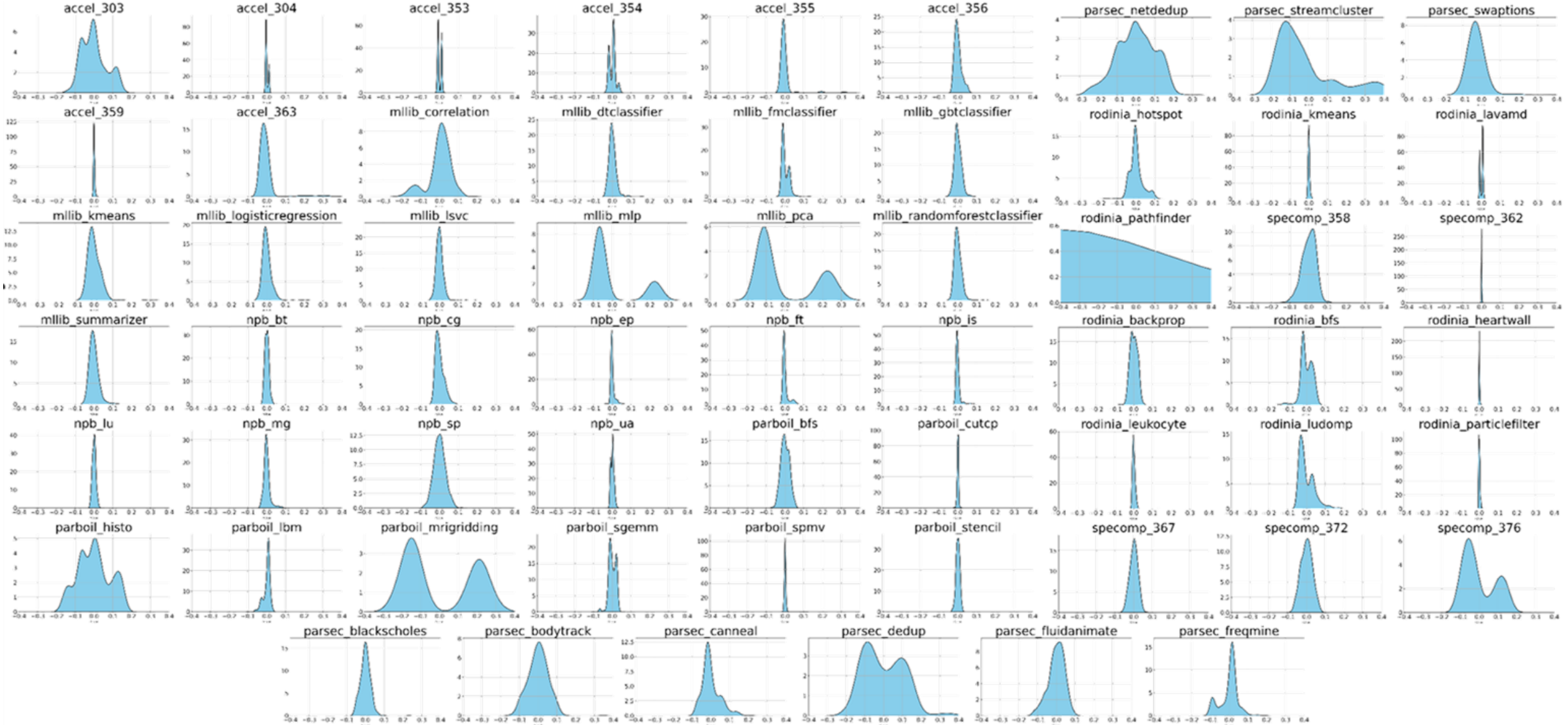
PREDICTING PERFORMANCE VARIABILITY

Mohammed Baydoun (AUB), Mohammad Sonji (AUB), Pedro Bruel (HPE), Dejan Milojicic (HPE), Eitan Frachtenberg (HPE), Izzat El Hajj (AUB)

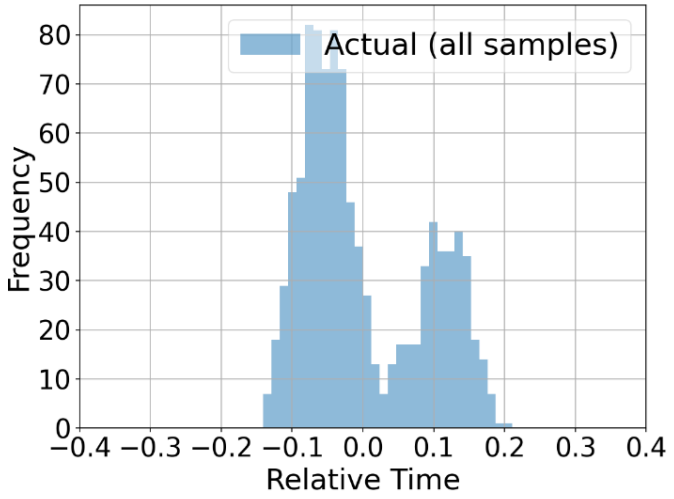
The International Workshop on Automatic Performance Tuning (iWAPT'25)

June 3, 2025

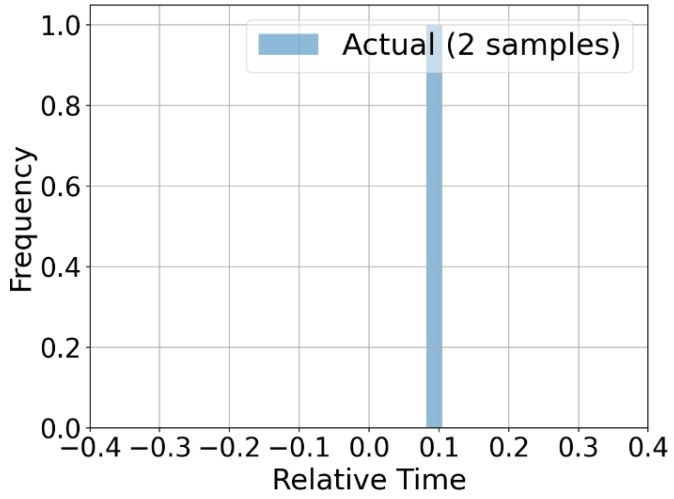
Motivation: Computer Applications have Performance Variability



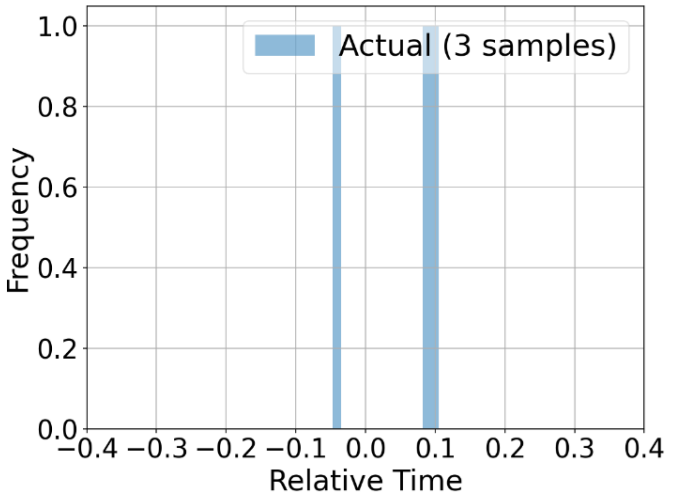
Motivation: Many Samples are Needed to Measure an Accurate Distribution



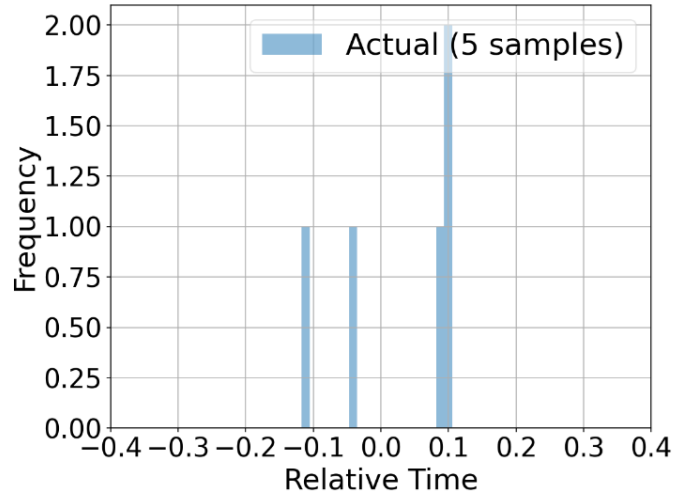
Measured distribution from 1,000 samples



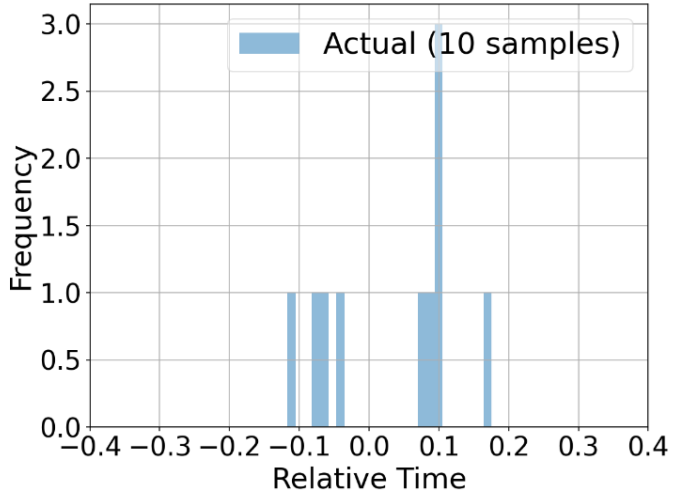
Measured distribution from 2 samples



Measured distribution from 3 samples



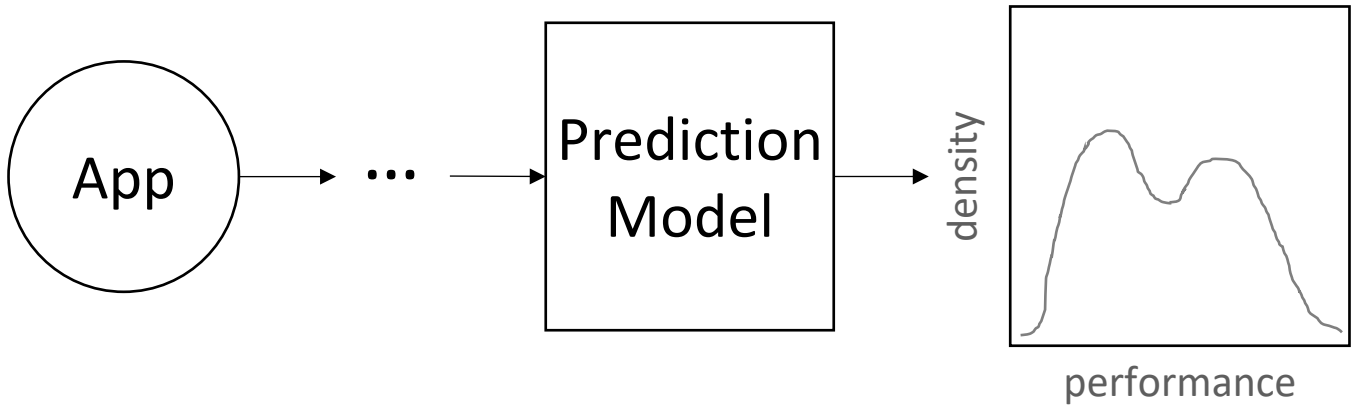
Measured distribution from 5 samples



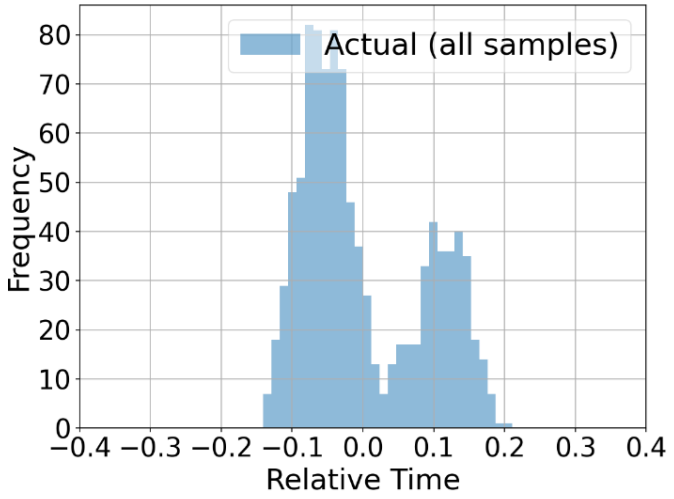
Measured distribution from 10 samples

Research Question

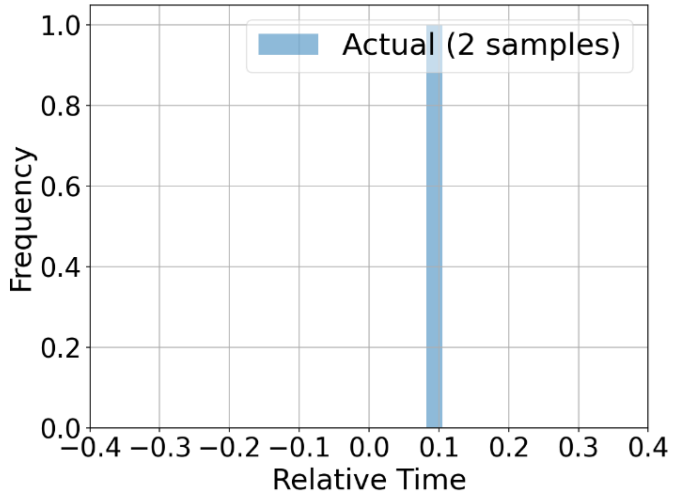
- What if we do not have the resources to run the application many times on the system?
  - Can we predict the performance distribution from just a few samples?
  - Can we predict the performance distribution from running on another system?
- Train model by learning from other applications



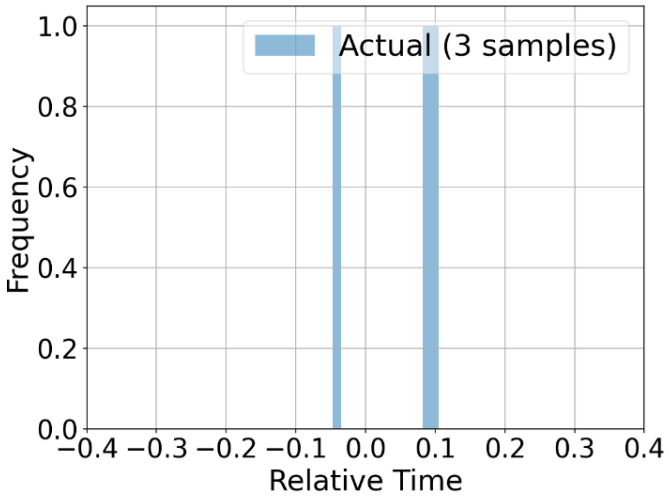
Sneak Peak at Results



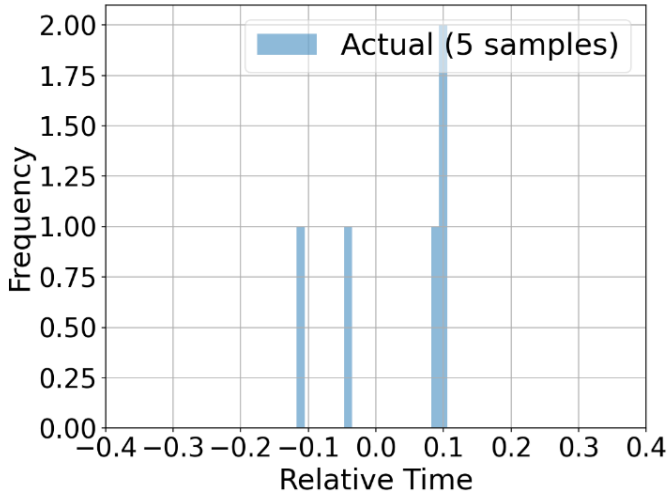
Measured distribution from 1,000 samples



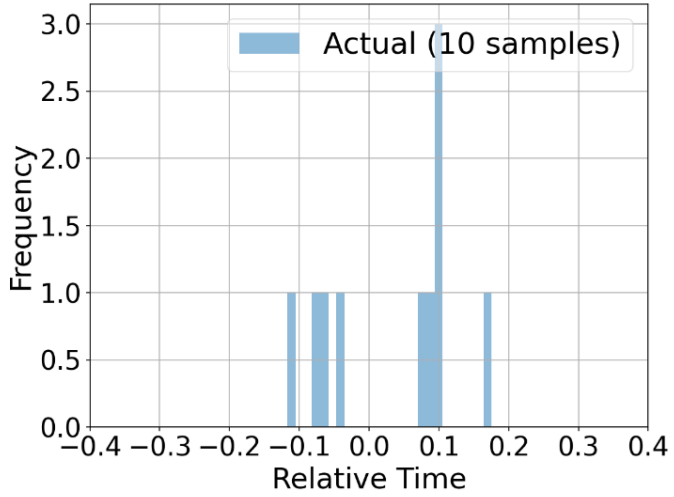
Measured distribution from 2 samples



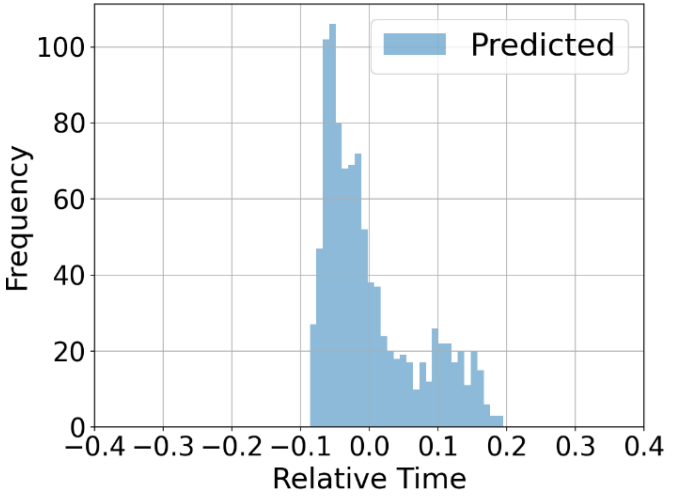
Measured distribution from 3 samples



Measured distribution from 5 samples

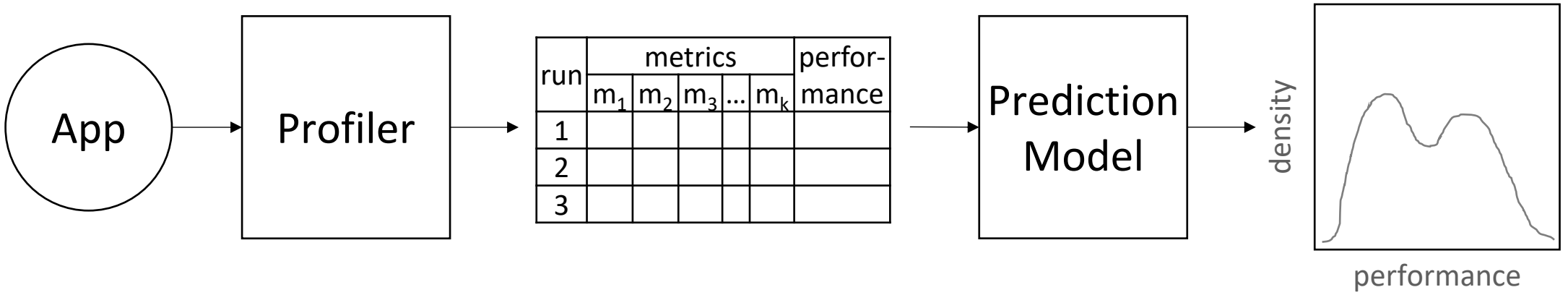


Measured distribution from 10 samples



Predicted distribution from 10 samples

Predicting Performance Distribution from Few Samples



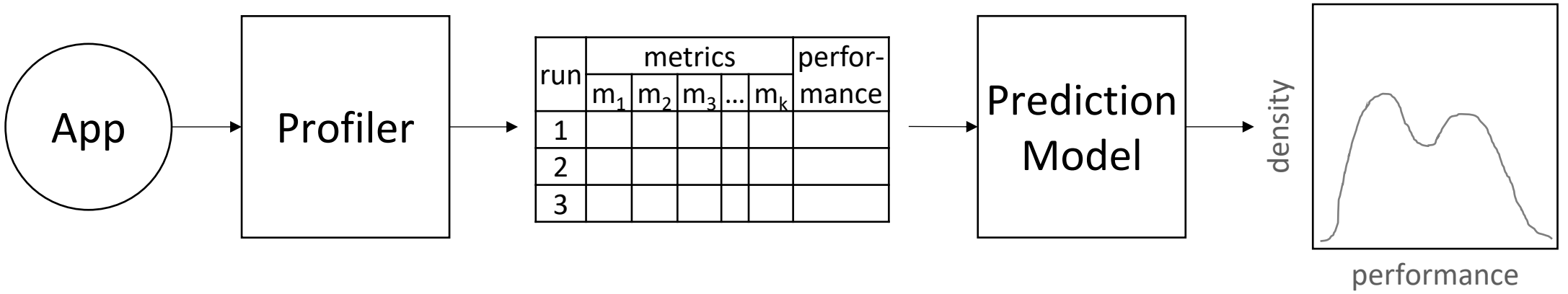
How to represent an application's profile as a feature vector?

Which model to use?

What data to train it with?

How to represent a performance distribution as a feature vector?

Predicting Performance Distribution from Few Samples



How to represent an application's profile as a feature vector?

Which model to use?

What data to train it with?

How to represent a performance distribution as a feature vector?

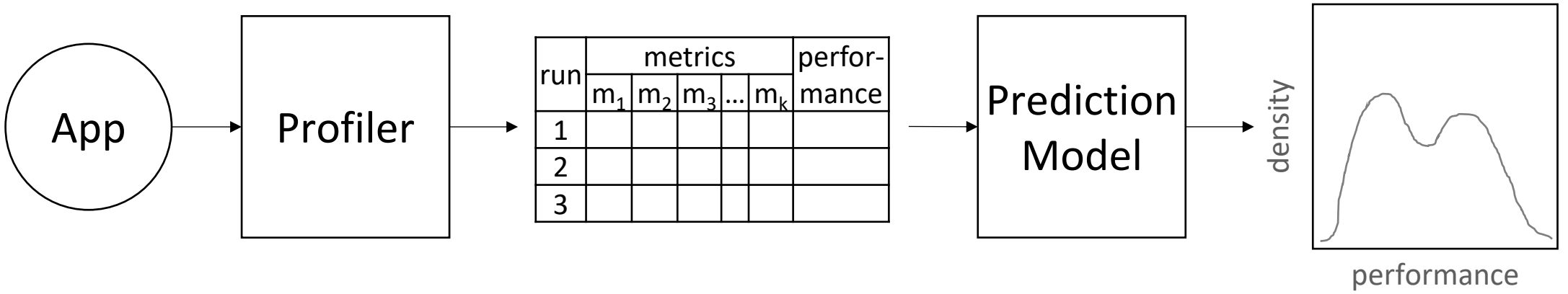
Representing an Application's Profile

ID	Metric
0	branch-instructions
1	branch-misses
2	bus-cycles
3	cache-misses
4	cache-references
5	cpu-cycles
6	instructions
7	ref-cycles
8	alignment-faults
9	bpf-output
10	cgroup-switches
11	context-switches
12	cpu-clock
13	cpu-migrations
14	emulation-faults
15	major-faults
16	minor-faults
17	page-faults
18	task-clock
19	duration_time
20	L1-dcache-load-misses
21	L1-dcache-loads
22	L1-dcache-stores
23	l1d.replacement
24	L1-icache-load-misses
25	l2_lines_in.all
26	l2_rqsts.all_demand_miss
27	l2_rqsts.all_rfo
28	l2_trans.l2_wb
29	LLC-load-misses
30	LLC-loads
31	LLC-store-misses
32	LLC-stores
33	longest_lat_cache.miss

ID	Metric
34	mem_inst_retired.all_loads
35	mem_inst_retired.all_stores
36	mem_inst_retired.lock_loads
37	branch-load-misses
38	branch-loads
39	dTLB-load-misses
40	dTLB-loads
41	dTLB-store-misses
42	dTLB-stores
43	iTLB-load-misses
44	node-load-misses
45	node-loads
46	node-store-misses
47	node-stores
48	mem-loads
49	mem-stores
50	slots
51	assists.fp
52	cycle_activity.stalls_l3_miss
53	assists.any
54	topdown.backend_bound_slots
55	br_inst_retired.all_branches
56	br_misp_retired.all_branches
57	cpu_clk_unhalted.distributed
58	cycle_activity.stalls_total
59	inst_retired.any
60	lsd.uops
61	resource_stalls.sb
62	resource_stalls.scoreboard
63	dtlb_load_misses.stlb_hit
64	dtlb_store_misses.stlb_hit
65	itlb_misses.stlb_hit
66	unc_cha_tor_inserts.io_hit
67	unc_cha_tor_inserts.io_miss

- **Application-independent** hardware and software metrics (68 metrics from from Linux perf)
  - Instruction counts (branches, loads, stores, floating point, total)
  - Cache metrics (references, hits, misses)
  - Main memory metrics
  - TLB metrics
- **Relative** metrics normalized per second
  - Unify model across applications with different durations
- If multiple samples, **mean, standard deviation, skewness, kurtosis**
  - Higher order moments did not improve results

Predicting Performance Distribution from Few Samples



How to represent an application's profile as a feature vector?

Which model to use?

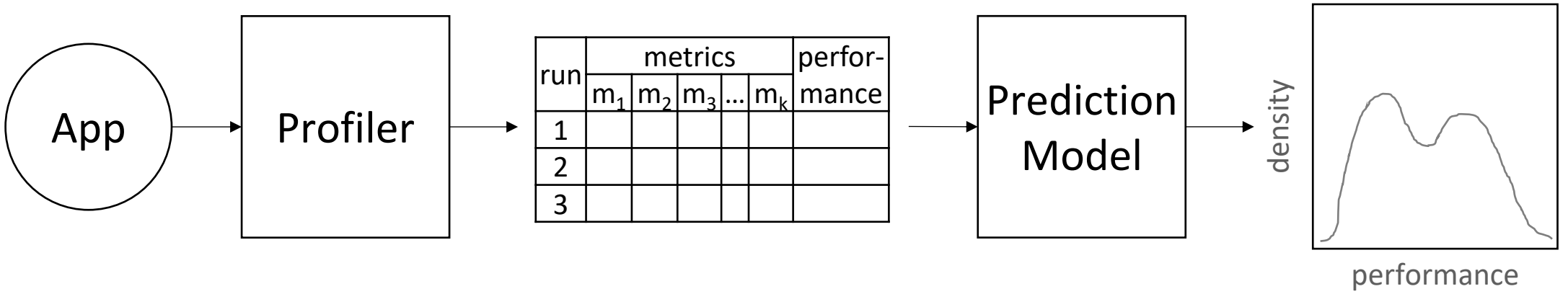
What data to train it with?

How to represent a performance distribution as a feature vector?

Representing the Performance Distribution

- **Histogram**: The feature vector is the bins of a histogram of the relative time (similar to a discretized PDF)
- **Moments**: The feature vector is the moments of the distribution (we consider the first four moments: mean, standard deviation, skewness, and kurtosis)
  - **PyMaxEnt**: Reconstruct the distribution from the moments using the principle of maximum entropy [30]
  - **PearsonRnd**: Reconstruct the distribution by drawing random numbers from the distribution in the Pearson system with these moments

Predicting Performance Distribution from Few Samples



How to represent an application's profile as a feature vector?

Which model to use?

What data to train it with?

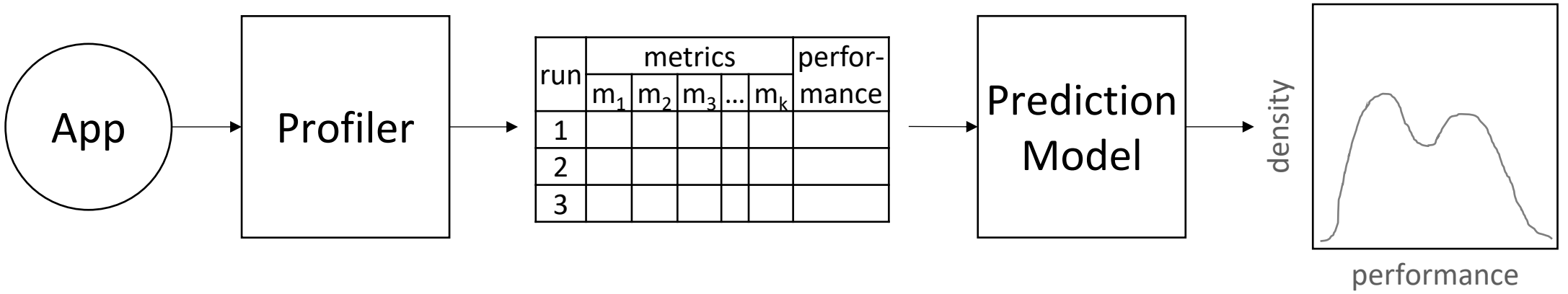
How to represent a performance distribution as a feature vector?

Choice of Prediction Model

- ***k*-nearest neighbors (kNN)**
  - Can deal with noisy data, which is the case for the metrics we collect
  - $k = 15$
  - Distance metric: cosine similarity

- **Random forests (RF)** and **extreme gradient boosting (XGBoost)**
  - Can deal with a large and diverse set of input features

Predicting Performance Distribution from Few Samples



How to represent an application's profile as a feature vector?

Which model to use?

What data to train it with?

How to represent a performance distribution as a feature vector?

Training the Model

Suite	Benchmarks
NPB [38]	bt, cg, ep, ft, is, lu, mg, sp, ua
PARSEC3.0 [39]	blackscholes, bodytrack, canneal, dedup, fluidanimate, freqmine, netdedup, streamcluster, swaptions
SPEC OMP [2]	358, 362, 367, 372, 376
SPEC Accel [40]	303, 304, 353, 354, 355, 356, 359, 363
Parboil [41]	bfs, cutcp, histo, lbm, mrigridding, sgemm, spmv, stencil
Rodinia [42]	backprop, bfs, heartwall, hotspot, kmeans, lavaMD, leukocyte, ludomp, particle_filter, pathfinder
MLlib [43]	correlation, dtclassifier, fmclassifier, gbtclassifier, kmeans, logisticregression, lsvc, mlp, pca, randomforestclassifier, summarizer

- Use **60 benchmarks** from popular HPC and data analytics applications and libraries
- Run application for **1,000 repetitions** (samples) to measure the distribution

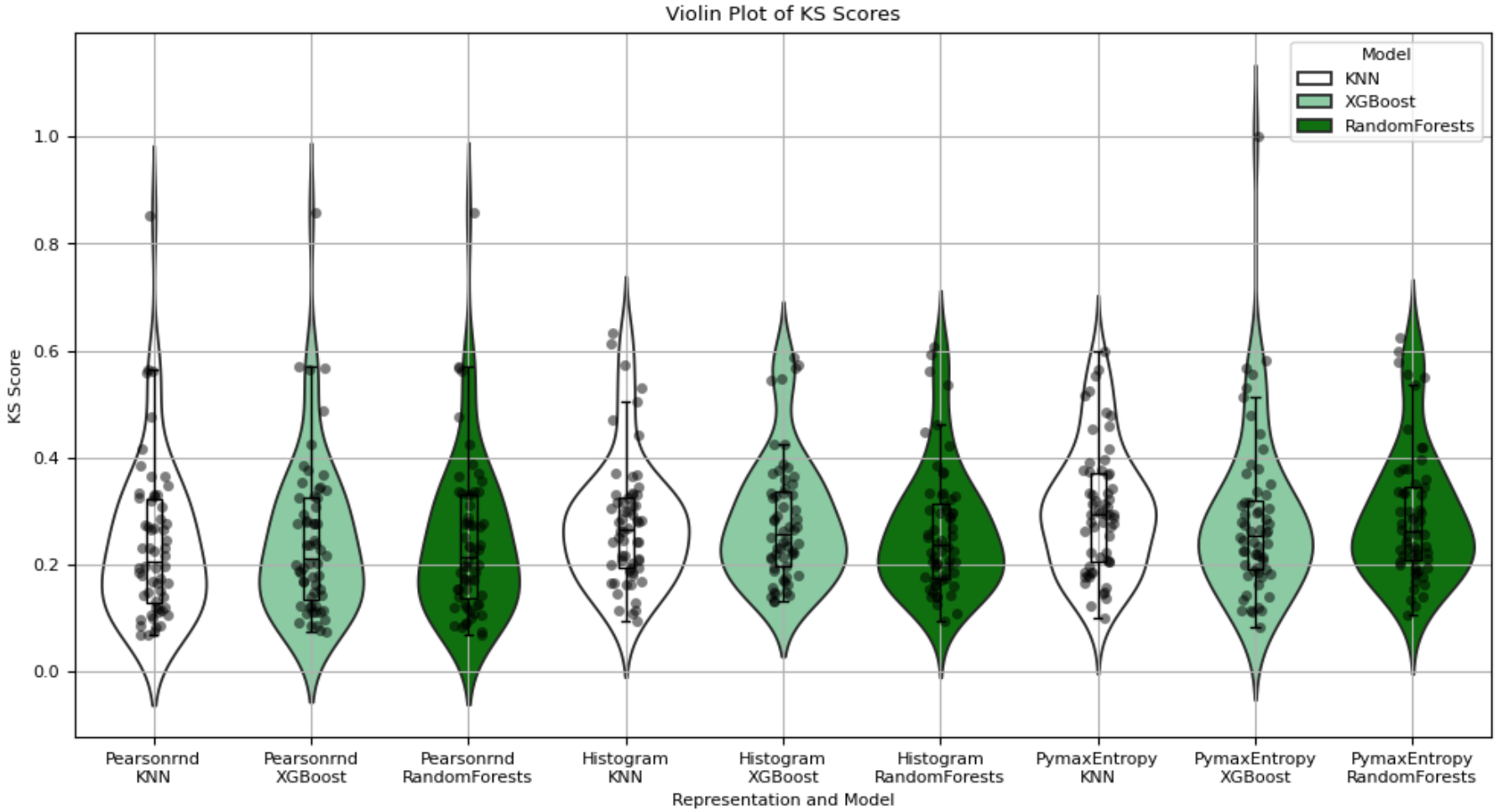
Evaluation Methodology

- **Hardware specifications**
  - Intel Xeon Platinum 8358 CPU
  - 512GB of DDR4 RAM
  - 64 cores total (2 sockets, 32 cores per socket)

- **Benchmarking conditions**
  - The benchmarks ran on an entire node and without any interference

- **Error evaluation**
  - We use cross-validation (leave-one-group-out) to assess the accuracy of the model
  - We use the Kolmogorov-Smirnov (KS) divergence test to assess quantify the agreement between the observed and predicted distributions (0 is a perfect match, 1 is the worst)

Evaluation: Model and Output Representation

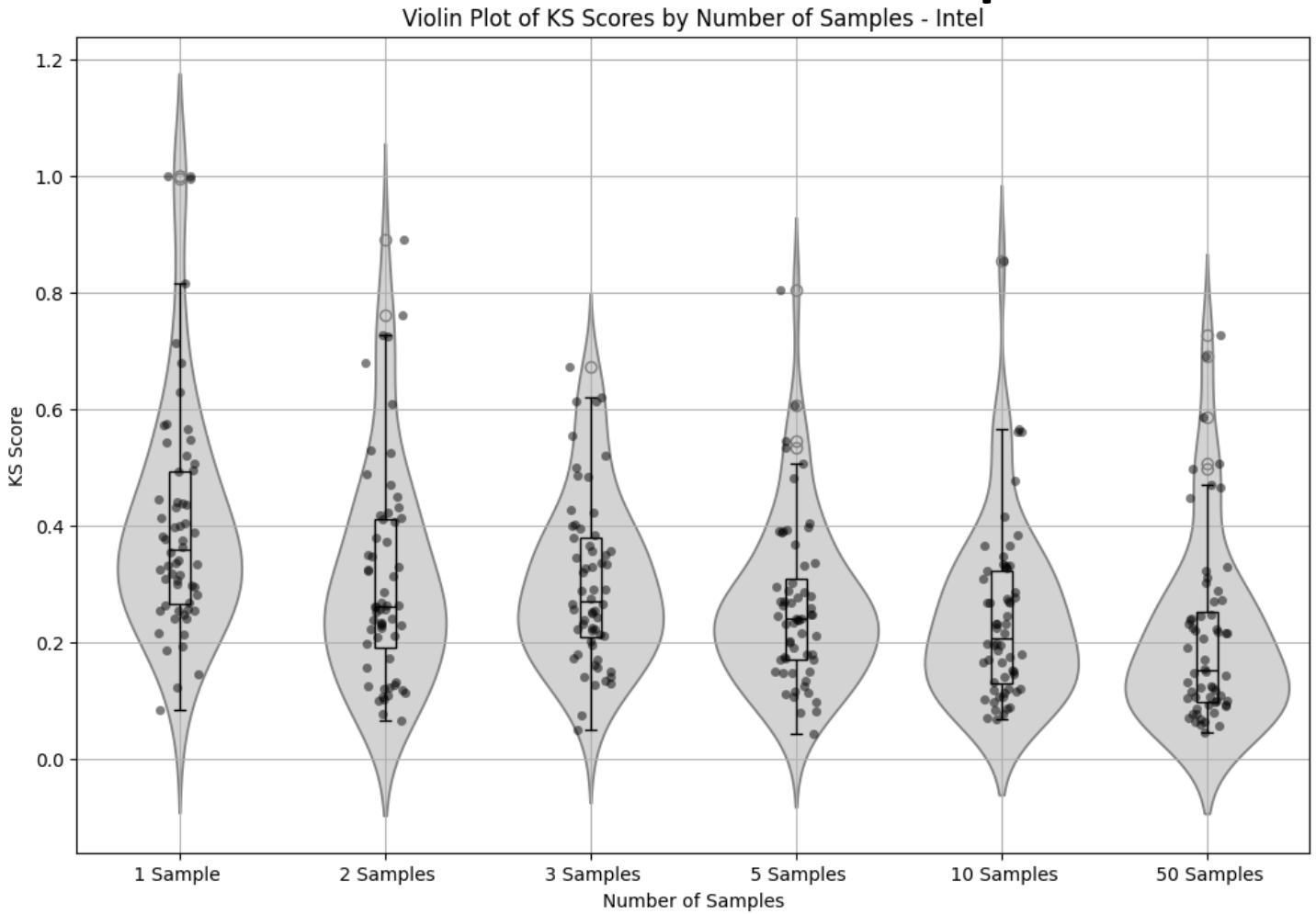


Observation #1: kNN has the lowest median KS score

Observation #2: PearsonRnd has the lowest median KS score

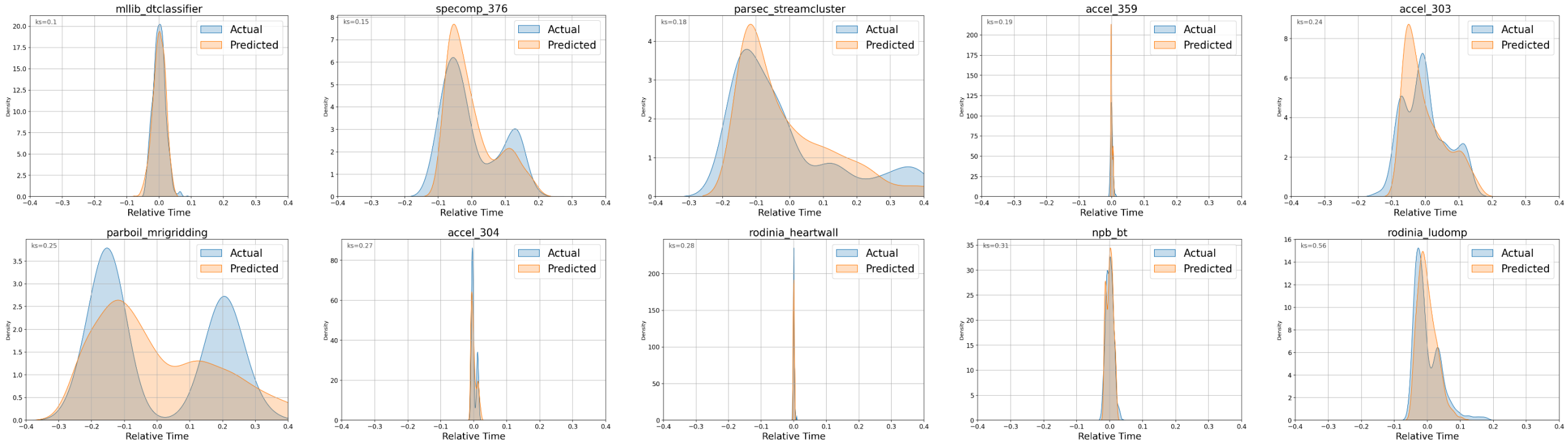
Observation #3: Most benchmarks have KS score lower than 0.4

Evaluation: Number of Samples



Observation: Improved accuracy as number of samples increases (users can trade off number of samples for accuracy)

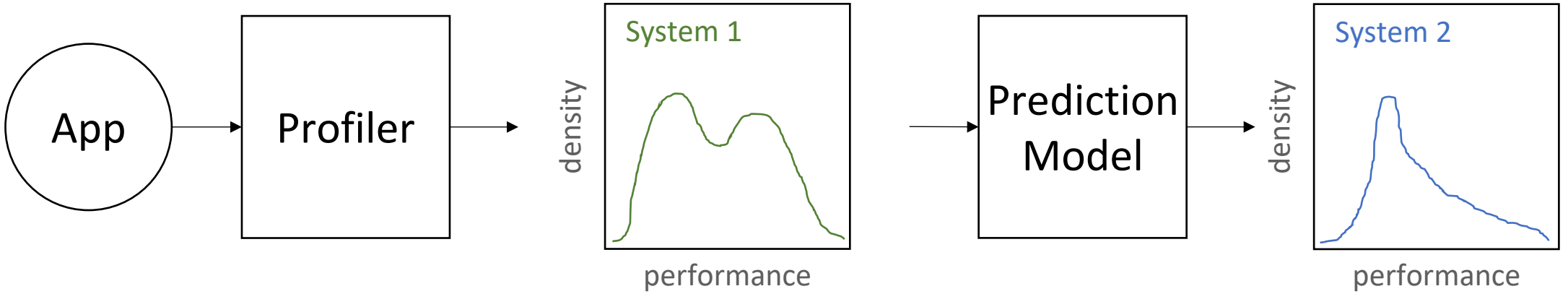
Evaluation: Examples



Observation #1: Width of the distribution predicted accurately

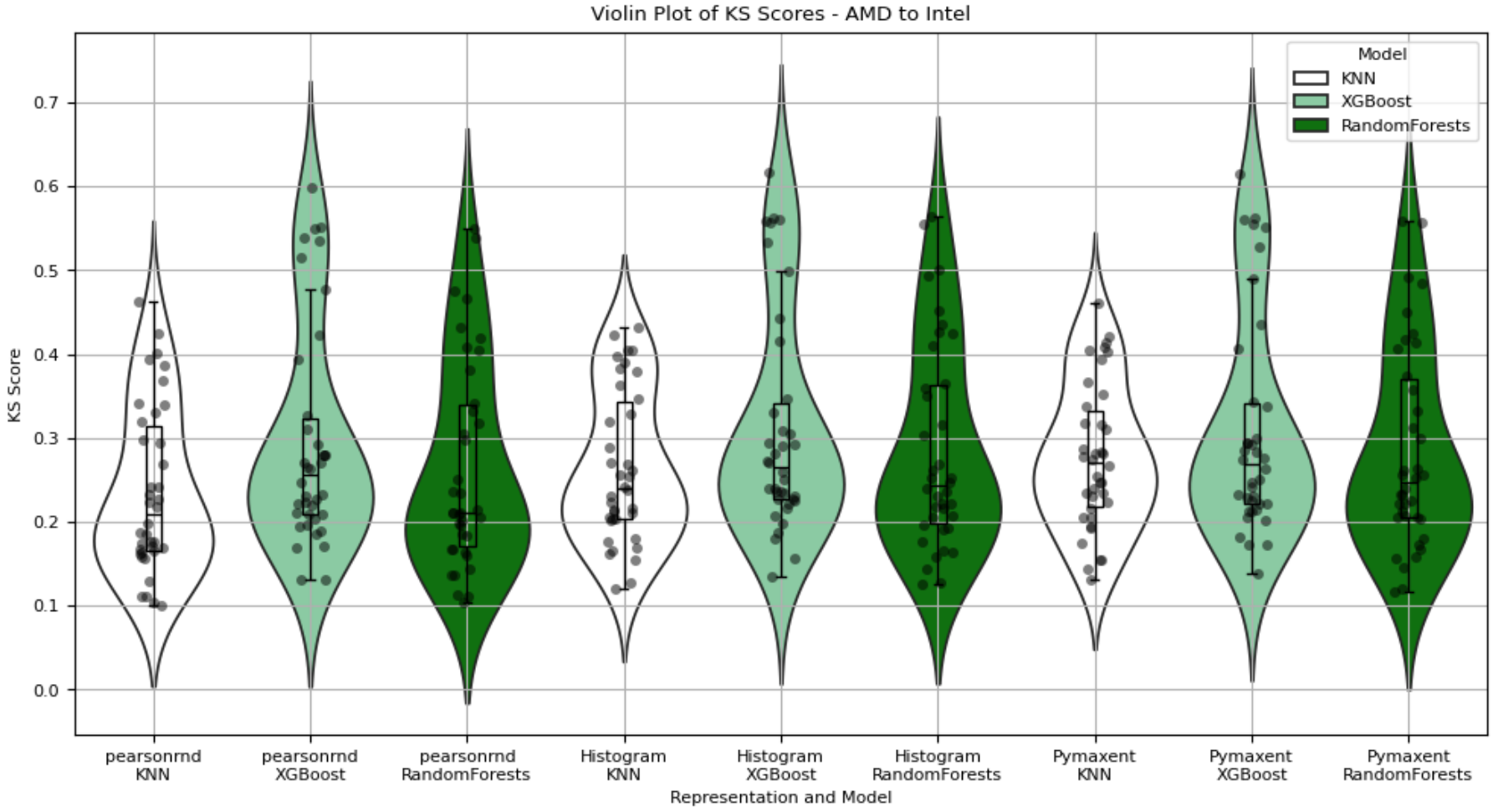
Observation #2: Number of modes and relative positions and sizes predicted accurately

Predicting Performance Distributions from Another Distribution on Another System



- **Same design considerations** as the first use case
  - Input distribution includes distribution of profiling metrics, not just perfomance
- **Hardware specifications** of the second system used
  - AMD EPYC 7543 CPU
  - 512GB of DDR4 RAM
  - 64 cores total (2 sockets, 32 cores per socket)
- **75 profiling metrics** different from the ones on the Intel system

Evaluation: Model and Output Representation

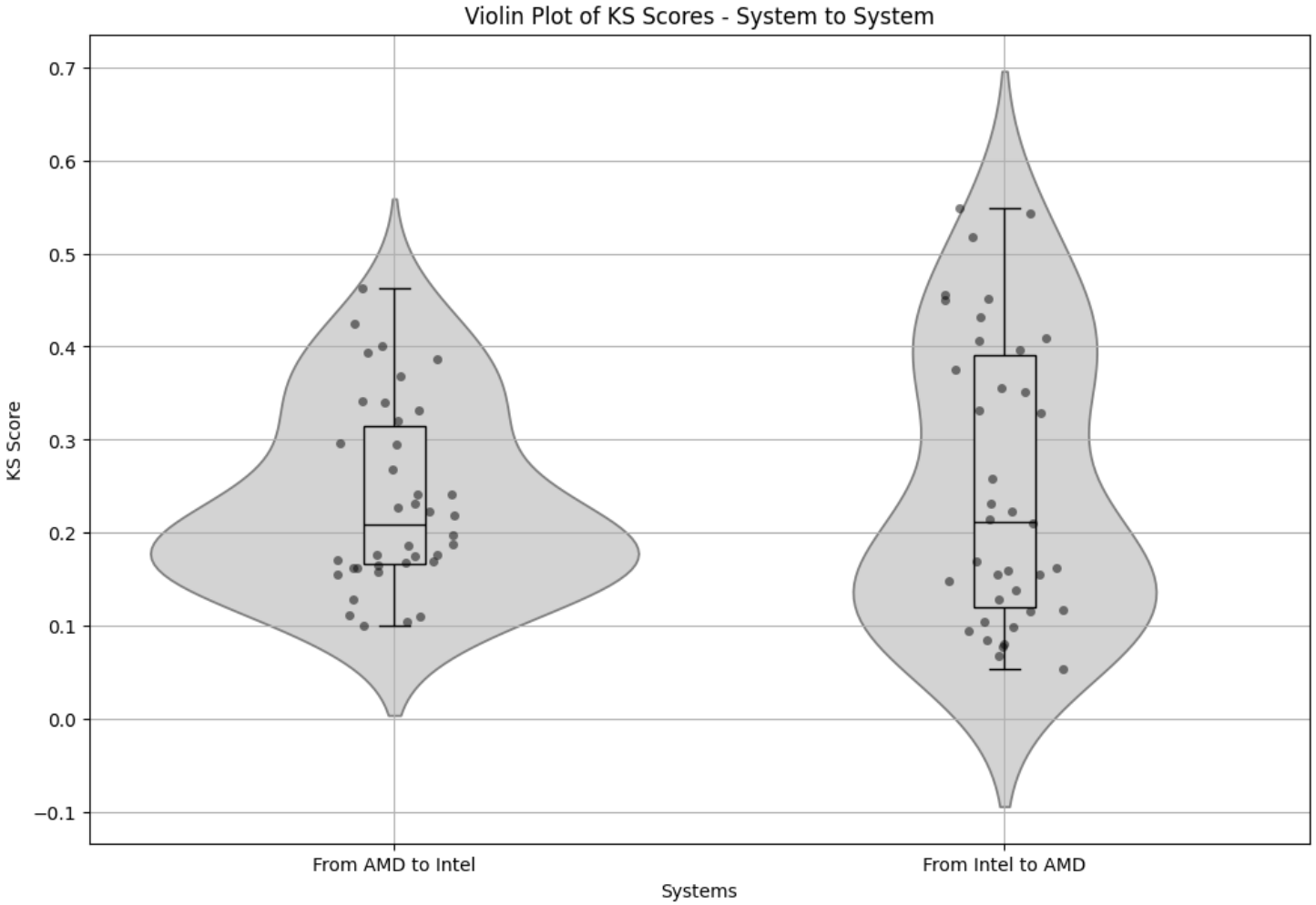


Observation #1: kNN has the lowest median KS score

Observation #2: PearsonRnd has the lowest median KS score

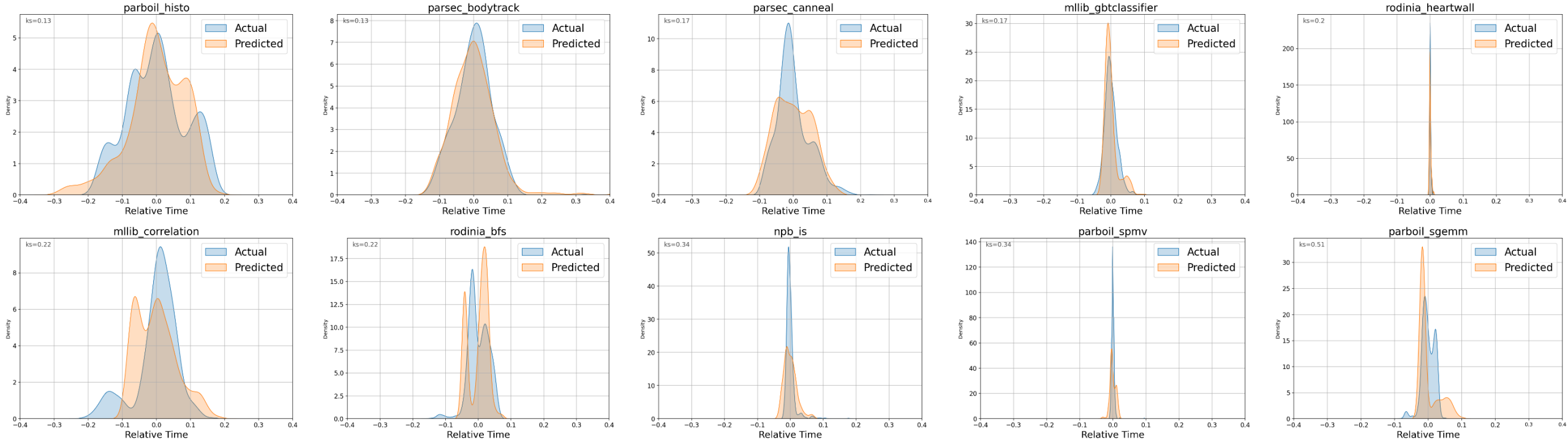
Observation #3: Most benchmarks have KS score lower than 0.4

Evaluation: Direction of Prediction



Observation: Prediction accuracy differs based on direction of prediction but only slightly

Evaluation: Examples



Observation #1: Width of the distribution predicted accurately

Observation #2: Number of modes and relative positions predicted accurately, but mixed success on their relative sizes

Summary and Future Work

- We show that application **performance variability can be predicted** by learning from other applications
- We use **application-independent profiling** and **relative profiling metrics** to unify model across diverse applications with different durations
- We show that **kNN** is the best performing model and **PearsonRnd** is the best method for representing the distribution for the purpose of prediction
- Future work:
  - Increase training data by covering **more applications and systems**
  - Use explainable models and leverage explanations to **optimize for variability**

AMERICAN UNIVERSITY OF BEIRUT

FACULTY OF ARTS & SCIENCES

Hewlett Packard Enterprise

PREDICTING PERFORMANCE VARIABILITY

Mohammed Baydoun (AUB), Mohammad Sonji (AUB), Pedro Bruel (HPE), Dejan Milojicic (HPE), Eitan Frachtenberg (HPE), Izzat El Hajj (AUB)

The International Workshop on Automatic Performance Tuning (iWAPT'25)

June 3, 2025