FLASH: Fast Model Adaptation in ML-Centric Cloud Platforms

Haoran Qiu[1], Weichao Mao[1], Archit Patke[1], Shengkun Cui[1], Chen Wang[2]
Hubertus Franke[2], Zbigniew Kalbarczyk[1], Tamer Basar[1], Ravishankar Iyer[1]
[1]UIUC [2]IBM Research

MLSys 2024

How do we make ML for systems useful?

Data center ma

- Power mana
- Job schedul

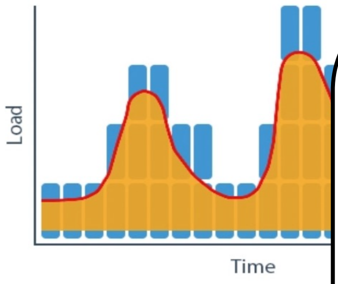
amazon web services

Azure

Goog

Cluster orchest

- Container p



Resource manage

- Resource con
- Autoscaling
- Load balancin

Networking tasks

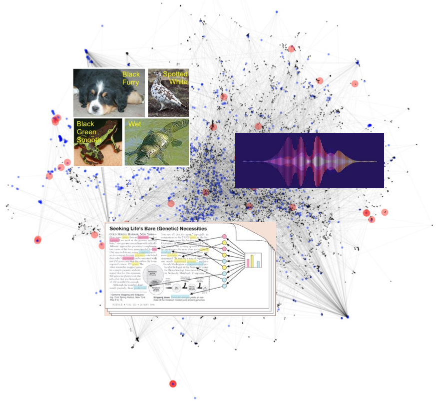
- Congestion control
- Adaptive video

General recipe:

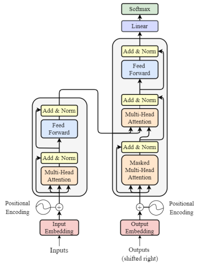
- 1 Agent
- 1 Task
- 1 Environment
- 1 Policy/Model

Lack of Generalizability: Great for research and local setup; not for actually **usable, deployable** models!

How does the rest of the world build **usable**, **deployable** models?



Pretraining



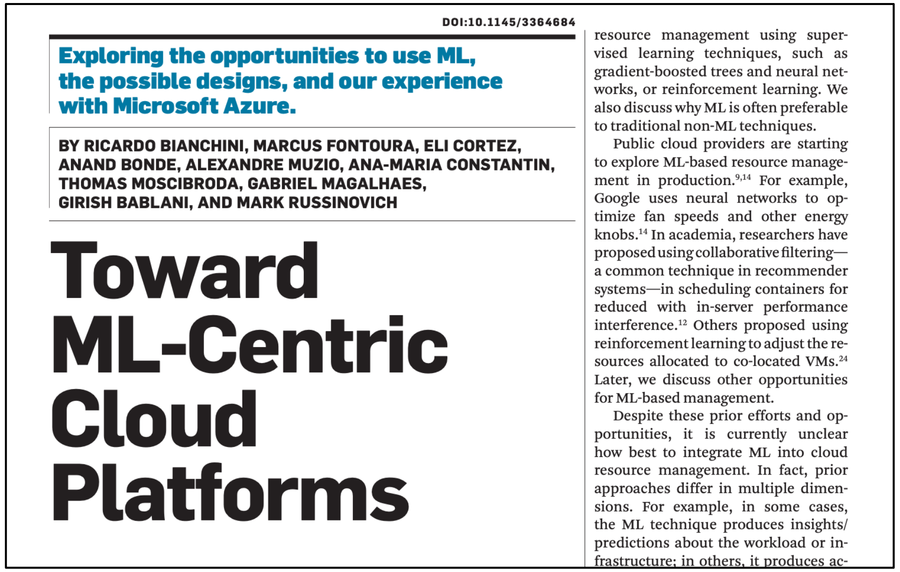
"Pretrained Models"

Finetuning

Exacerbated Challenges in ML-Centric Cloud Platforms

Four major components in ML for Systems:

- ***Tasks***: e.g., resource management, load balancing, etc.
- ***Environments***: Infra/platform (e.g., a 5-node Kubernetes cluster)
- ***Applications***: Workloads (e.g., Kubernetes Deployment)
- ***Agents***: e.g., reinforcement learning (RL) agent


DOI:10.1145/3364684

Exploring the opportunities to use ML, the possible designs, and our experience with Microsoft Azure.

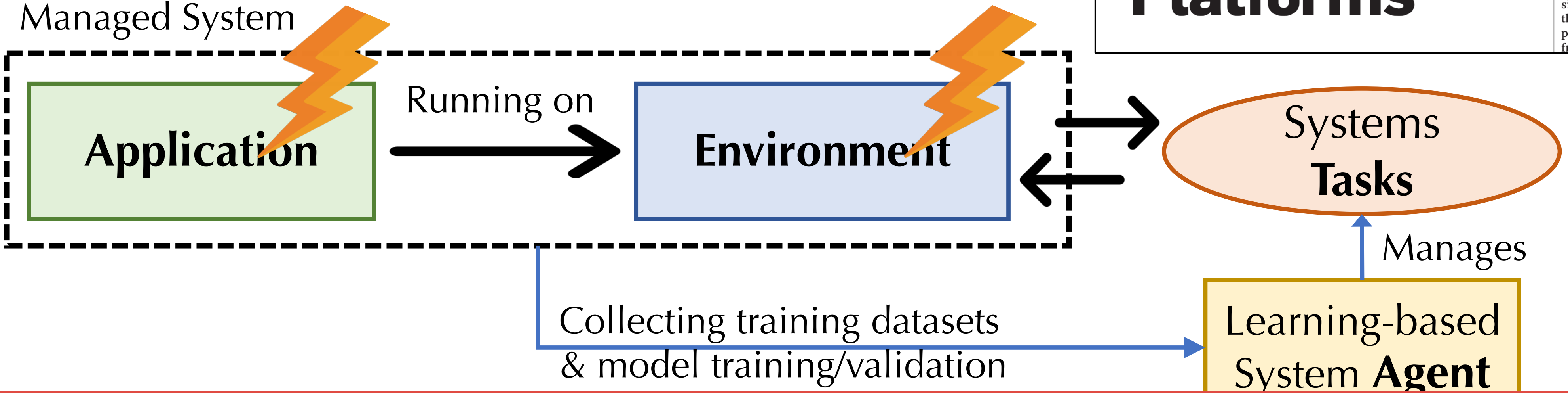
BY RICARDO BIANCHINI, MARCUS FONTOURA, ELI CORTEZ, ANAND BONDE, ALEXANDRE MUZIO, ANA-MARIA CONSTANTIN, THOMAS MOSCIBRODA, GABRIEL MAGALHAES, GIRISH BABLANI, AND MARK RUSSINOVICH

Toward ML-Centric Cloud Platforms

resource management using supervised learning techniques, such as gradient-boosted trees and neural networks, or reinforcement learning. We also discuss why ML is often preferable to traditional non-ML techniques.

Public cloud providers are starting to explore ML-based resource management in production.[9,14] For example, Google uses neural networks to optimize fan speeds and other energy knobs.[14] In academia, researchers have proposed using collaborative filtering—a common technique in recommender systems—in scheduling containers for reduced with in-server performance interference.[12] Others proposed using reinforcement learning to adjust the resources allocated to co-located VMs.[24] Later, we discuss other opportunities for ML-based management.

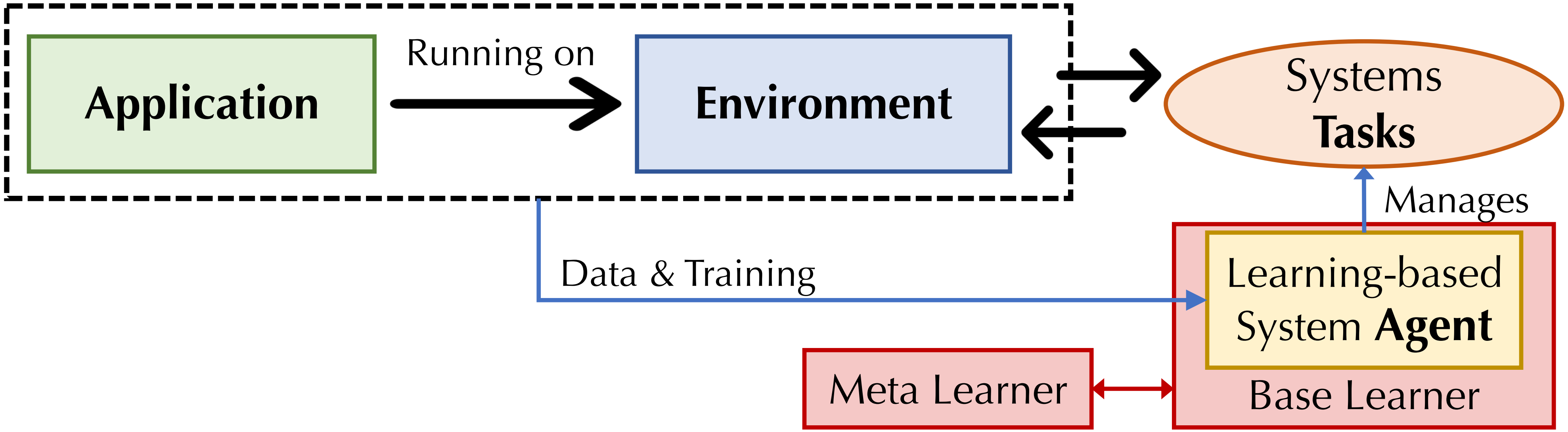
Despite these prior efforts and opportunities, it is currently unclear how best to integrate ML into cloud resource management. In fact, prior approaches differ in multiple dimensions. For example, in some cases, the ML technique produces insights/predictions about the workload or infrastructure; in others, it produces ac-



Adaptation to diverse, novel **applications** and **environments** require significant data collection and retraining

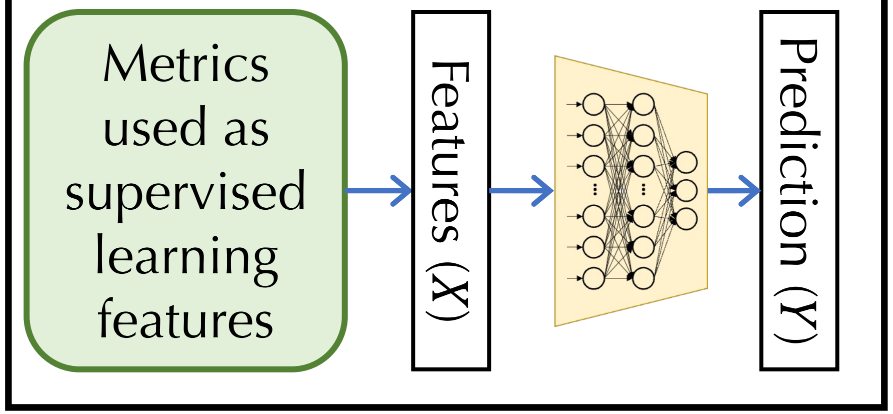
FLASH: Fast Model Adaptation in ML for Systems

- **Goal:** To achieve **fast ML-for-Systems model adaption** to new, evolving cloud applications and/or infrastructures within each task
  - Focus on supervised learning and reinforcement learning (RL)
- Extends a unified API for ML/RL agent developers to automatically integrate their developed agents without any changes
  - Key enabler: A *pretrain-finetune* paradigm with meta-learning for fast model adaptation across ***applications*** and ***environments***



Example Use Cases That FLASH Supports

Supervised Learning



Resource Config Search:

- E.g., Ernest (NSDI16), PARIS (SoCC17), Selecta (ATC18), Sinan (ASPLOS21)

Power Capping:

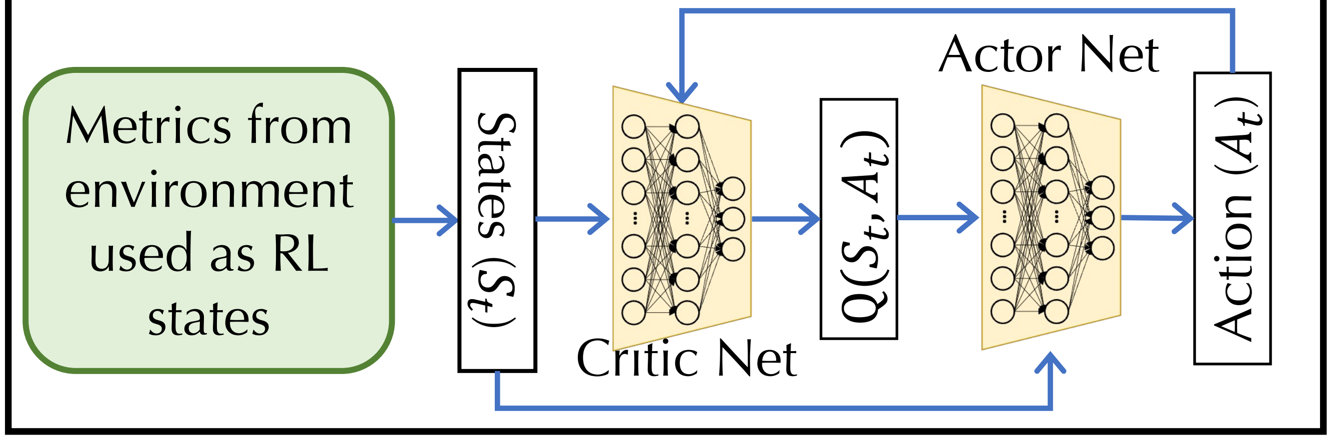
- E.g., NeurIPS23

Capacity Planning:

- E.g., IEEE CLOUD16, IEEE ICPADS17

… …

Reinforcement Learning



Workload Autoscaling / Resource Management:

- E.g., MIRAS (ICDCS19), FIRM (OSDI20), ADRL (TPDS21), SIMPPO (SoCC22), DeepScaling (SoCC22)

Job/Data Scheduling / Placement:

- E.g., ICLR18, SIGCOMM19, NeurIPS19, ICML20, ISCA22, MLSys23

Congestion Control:

- E.g., ICML19, SIGCOMM22, AAAI22

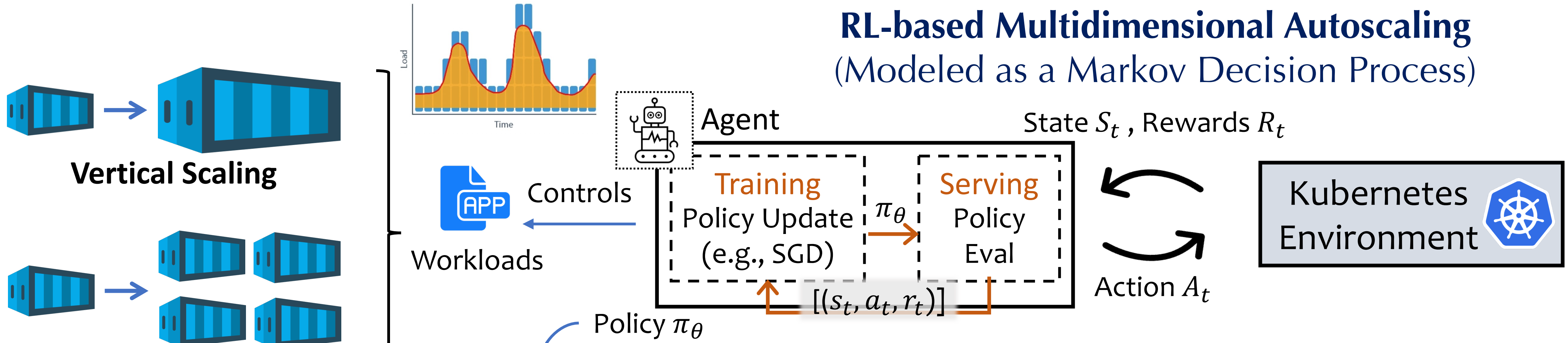
Power Management:

- E.g., SOL (ASPLOS22), IEEE SJ17, DATE15

… …

Background: Cloud Workload Autoscaling with RL

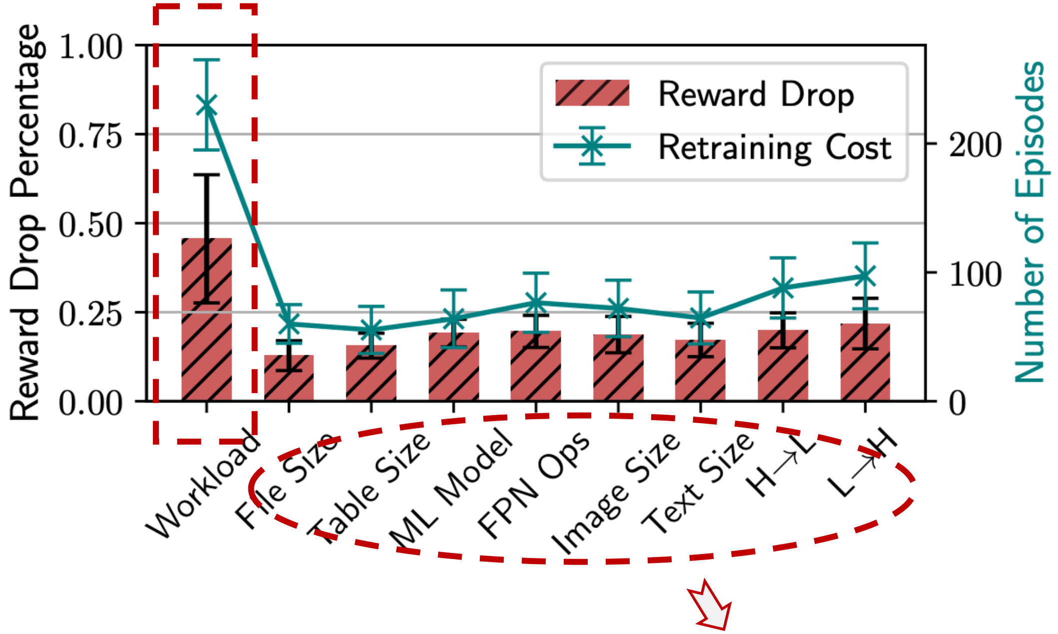
- RL agent interacts with an environment, step by step taking observations (s_t), making actions (a_t), receiving rewards (r_t)
- Reward functions (i.e., agent performance) are directly aligned with objectives: Meeting SLOs & High resource utilizations
- Specialize for specific workloads (e.g., periodicity or high scaling factor) by **reward maximization**



Goal: Maximize the expected cumulative reward
$\mathbb{E}[\sum_{t=0}^{T} \gamma^t \cdot r_t]$ *(in any T-step trajectory)*
--> *Resource util (%), SLO attainment*

Reference: FIRM (OSDI20), AWARE (ATC23)

Challenge of Heterogeneous Cloud Applications



Workload changes leads to 21.8% reward drops

Challenge #1

Trained policies are application-specific, **costly to adapt to new applications**

- 45.6% reward degradation (~230 eps retraining)

Challenge #2

During policy-serving stage, RL agent performance **degrades** when dealing with **updated workloads**

How to **automatically identify** these heterogeneous cases and handle them (adaptation) smoothly?

Conceptual Idea of Embedding-based Meta-learning

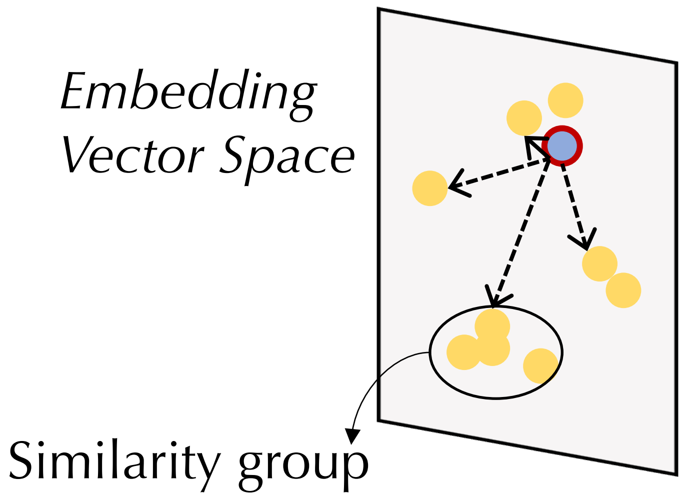
Goal: To reduce RL model retraining time (cost) and adapt quickly to new application workloads (unseen during training)

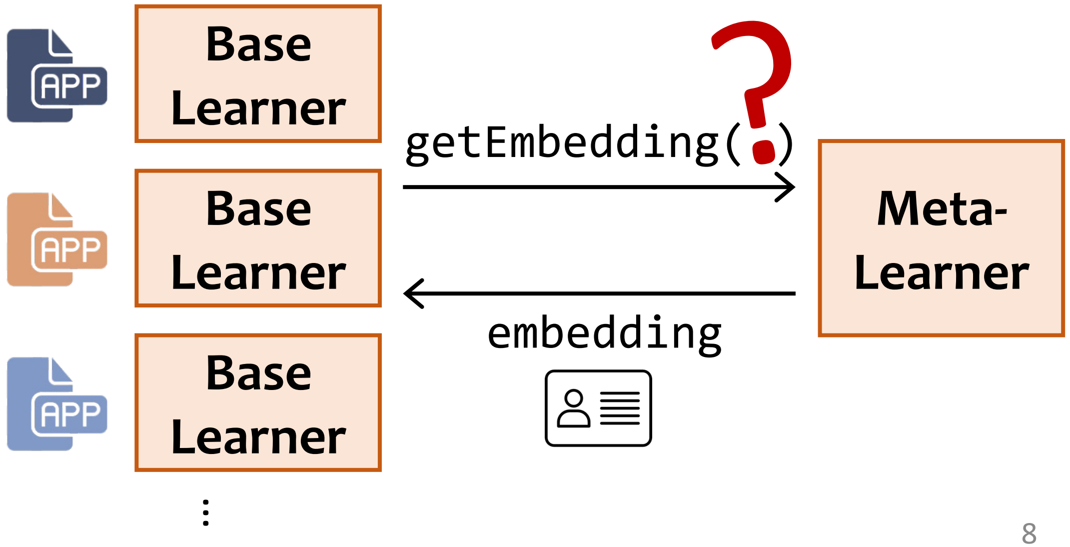
Key Idea: Models the RL agent as a **base-learner** and creates a **meta-learner** to learn to generate ***embeddings**** that can precisely differentiate and represent applications

*A fixed-sized low-dimensional vector

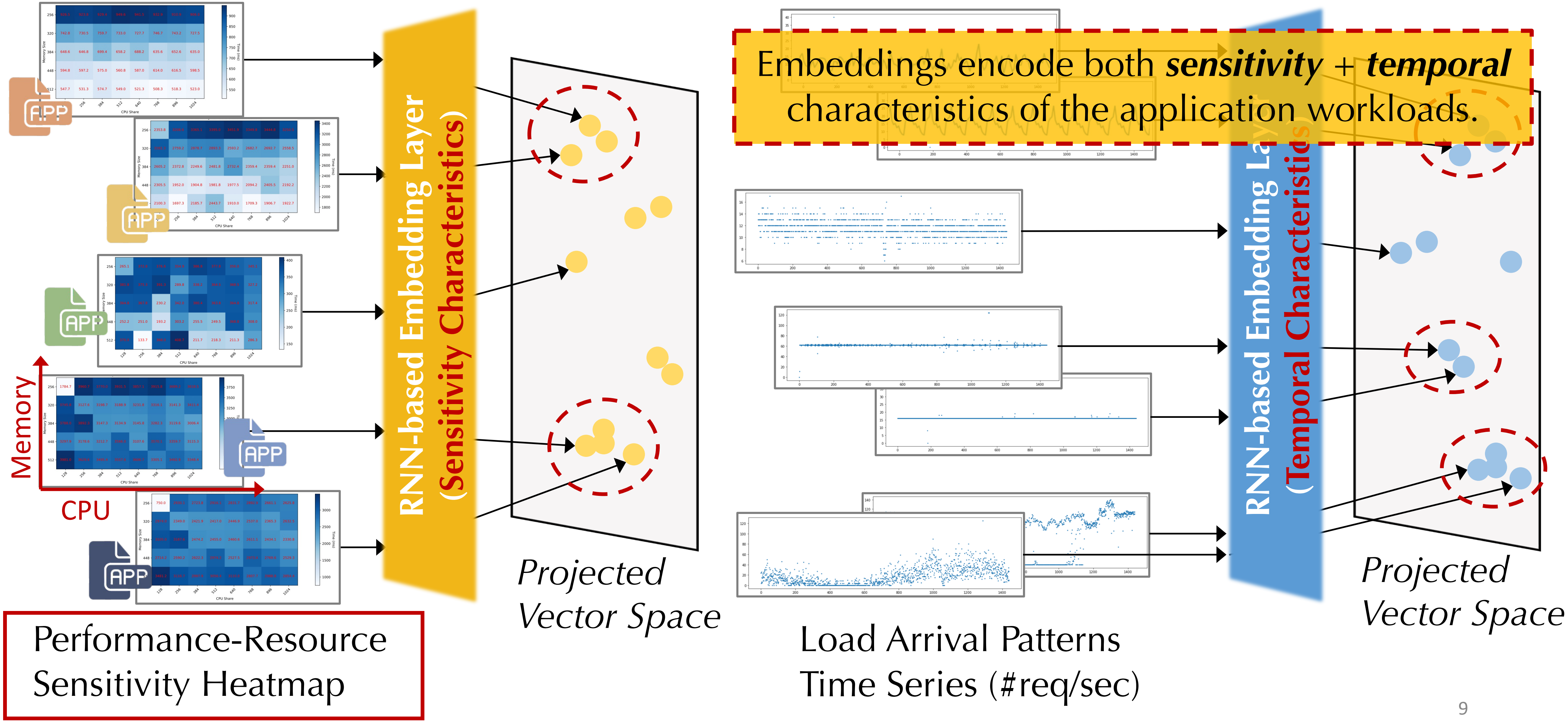
Meta-learning - "Learning to learn"

- Generalize to novel samples
- Fast adaptation based on similarity
- Combined with RL -> learned policy conditioning on the embeddings

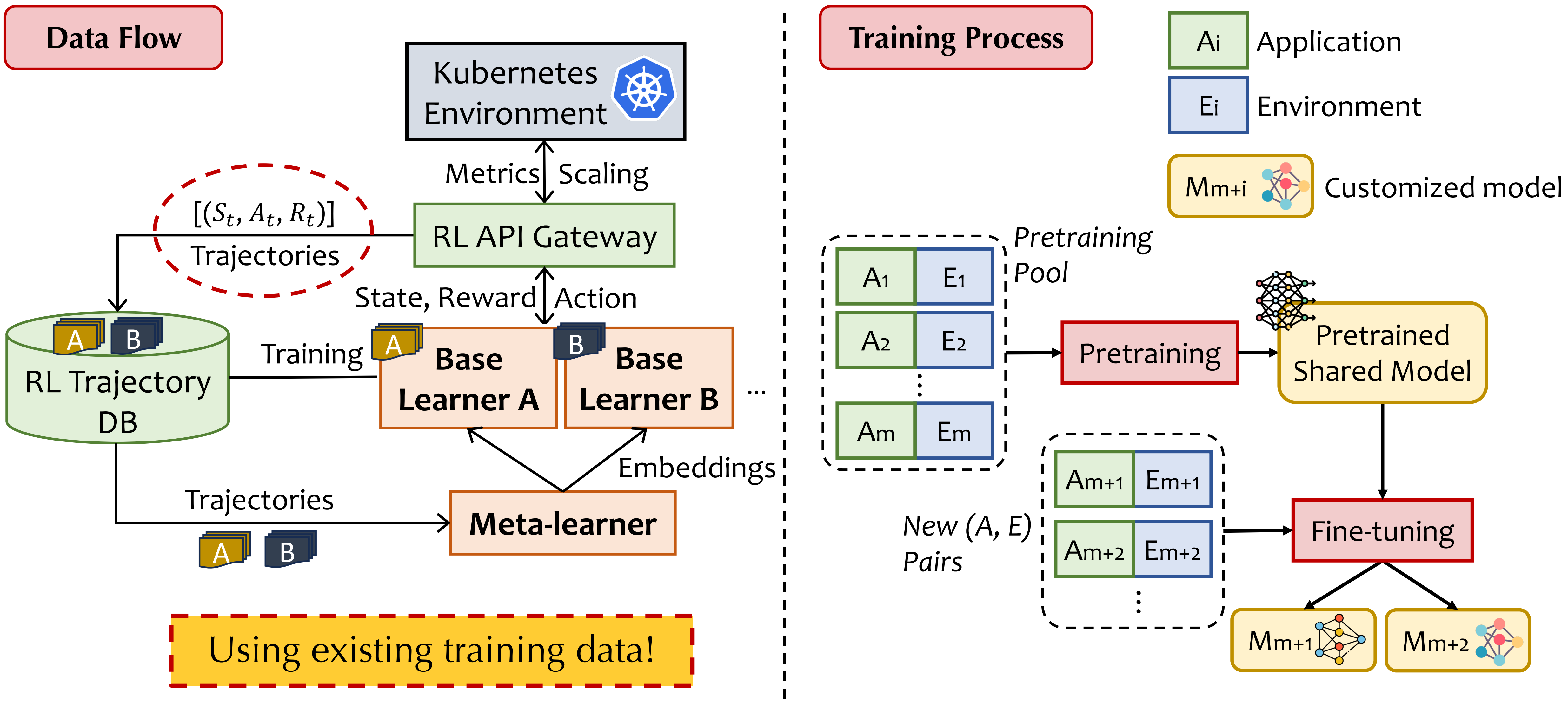




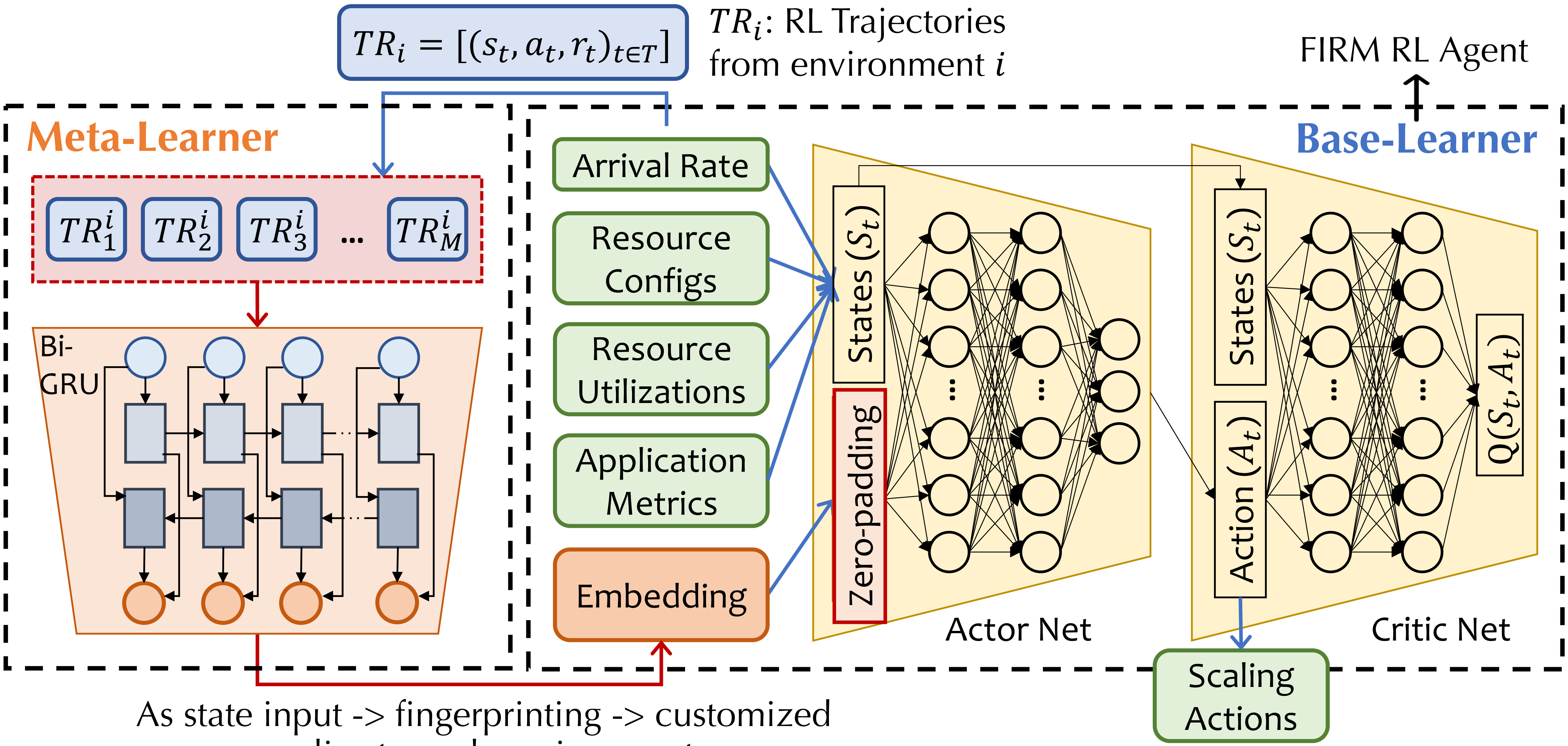
Interpreting Embeddings from Systems Perspective



Pretraining and Fine-tuning with Meta Learner



Meta Learner Design and Model Architecture



As state input -> fingerprinting -> customized policy to each environment

Evaluation

- **Does FLASH provide fast model adaptation to new workloads?**
  - What is the value of embedding-based meta-learning?
- Using FIRM [1]'s RL model as the base-learner
- Setup:
  - Generated 1000 synthetic applications
  - 16 represented production serverless function segments [2] (e.g., CPU-intensive jobs, image manipulation, text processing, web serving, ML model serving, I/O services)
  - Pretrained the meta-learner on 200 applications and tested on the remaining ones

----

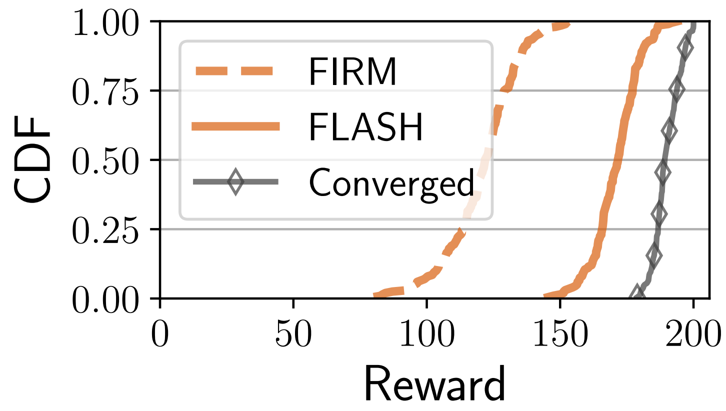
[1] FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Microservices. Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer. OSDI 2020.

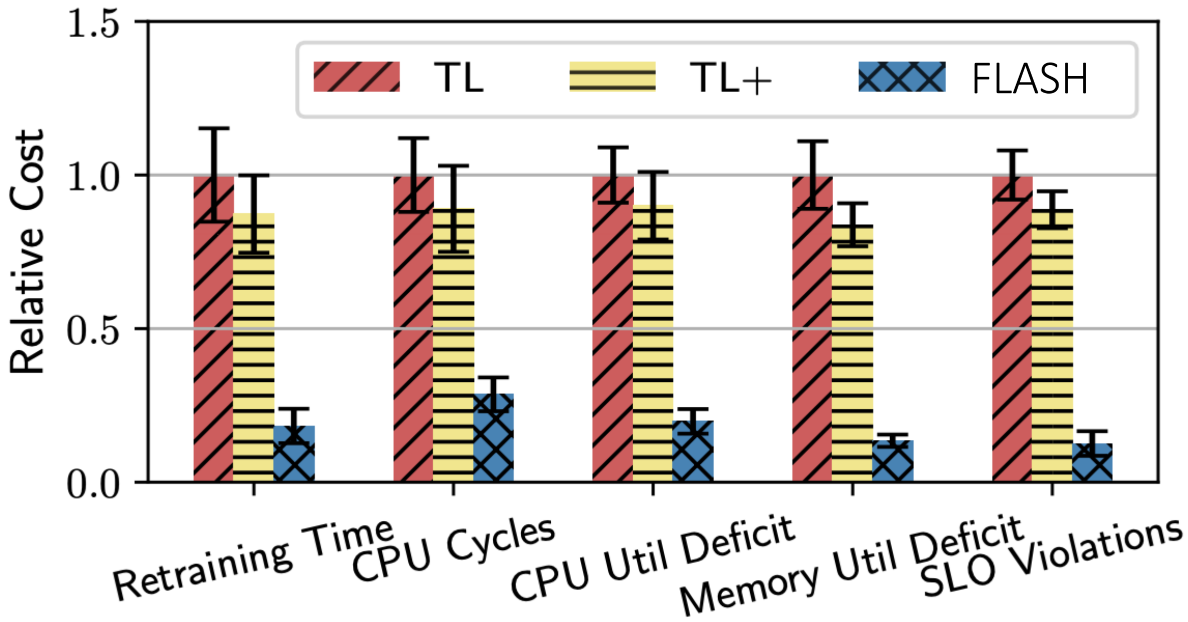
[2] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L. Abad, and Alexandru Iosup. *Serverless Applications: Why, When, and How?* IEEE Software, 38(1):32– 39, 2021.

Robustness to Application variability

When encountering novel applications, FLASH:

- Reduces the performance (reward) degradation by 2x
- Adapts 5.5× faster than transfer learning
  - TL: Transfer learning with parameter sharing
  - TL+: w/ handcrafted application fingerprints
- Reduces CPU and memory utilization deficit by 4.6× and 6.2×
- Reduces SLO violations by 7.1×





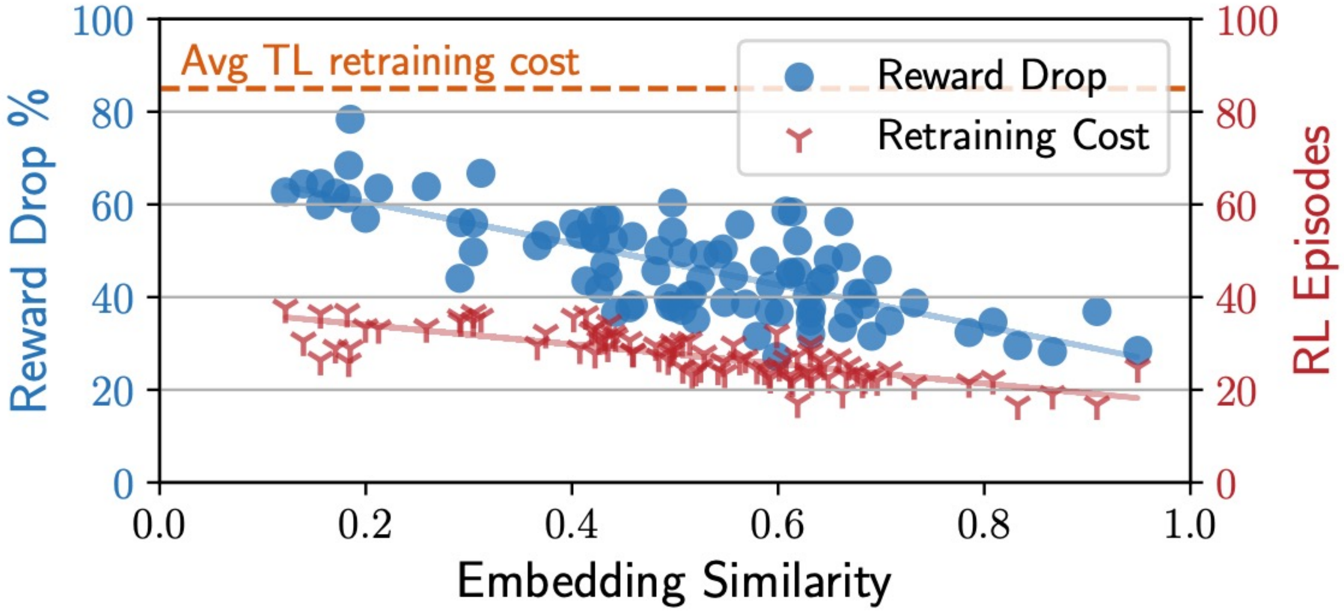
Predictability of Adaptation Overhead

Two embeddings: e_i, e_j

- ***Similarity metric***: $S(e_i, e_j) = (1 - ED(e_i, e_j) + CS(e_i, e_j))/2$

  Euclidean Distance **Cosine Similarity**

- Can be used for predictability of the adaptation cost / performance drop



Summary: A Foundation Model Recipe in ML for Sys?

Summary

- **FLASH**: Fast ML model adaptation across ***applications*** and ***environments***
  - Resource configuration / Autoscaling / Power management / Congestion control
- **Embedding-based meta learning**
  - Base learner and meta learner abstraction
  - Unified API for both supervised learning and reinforcement learning
  - Interpretability of the embedding and predictability of adaptation cost
- Source code available: https://gitlab.engr.illinois.edu/DEPEND/flash

Next?

- **Model size and complexity**
  - Larger models (e.g., transformers) have larger capability + better generalizability
  - Higher training / fine-tuning cost and inference overhead -> detrimental for real-time tasks
- **Adaptation across tasks?**
  - E.g., Decision Transformers

Haoran Qiu[1], Weichao Mao[1], Archit Patke[1], Shengkun Cui[1], Saurabh Jha[2]
Chen Wang[2], Hubertus Franke[2], Zbigniew T. Kalbarczyk[1], Tamer Basar[1], Ravishankar K. Iyer[1]

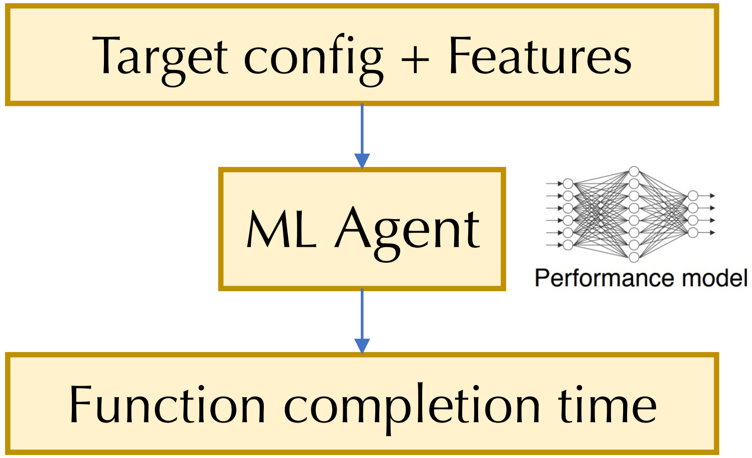
Thank you!

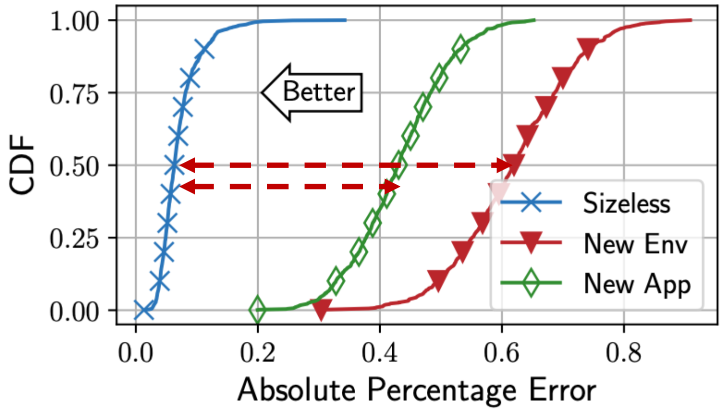
ILLINOIS IBM Research NSF

Backup Slides

Also Required in Other Tasks

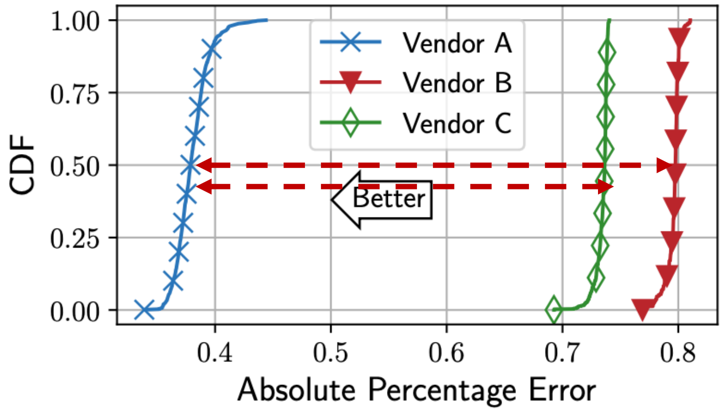
- **Resource Configuration Search**
  - Sizeless (Middleware 2021), Supervised Learning





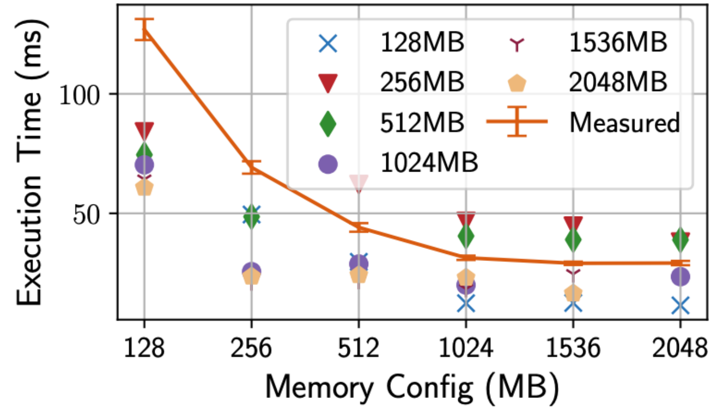
(a) OpenWhisk dataset.

9.9× and 16x increase in median APE for new app / env



(b) CloudBandit dataset.

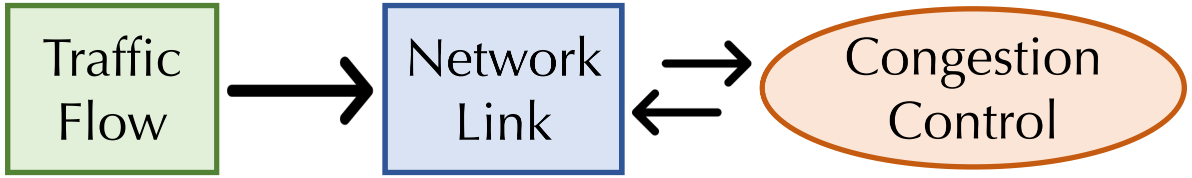
2.3× and 2.1x increase in median APE for new env

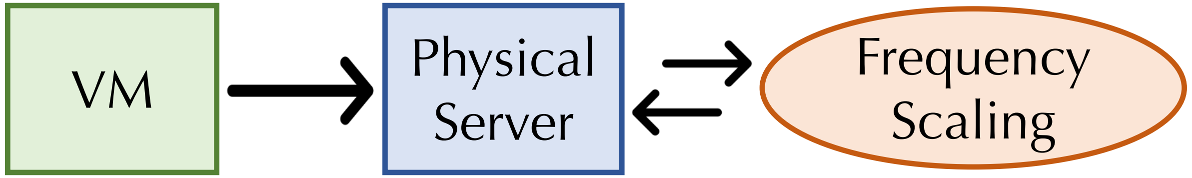


(c) An example of estimation error.

Also Required in Other Tasks

- **Congestion Control**
  - PCC (NeurIPS 2019), RL – PPO Algorithm
  - Requires the agent at the transport layer to adjust the sending rate based on measured network statistics
- **CPU Frequency Scaling**
  - SmartOC (ASPLOS 2022), RL – Q-Learning Algorithm
  - Requires the agent to balance the workload performance improvements with the extra power cost when increasing the frequency



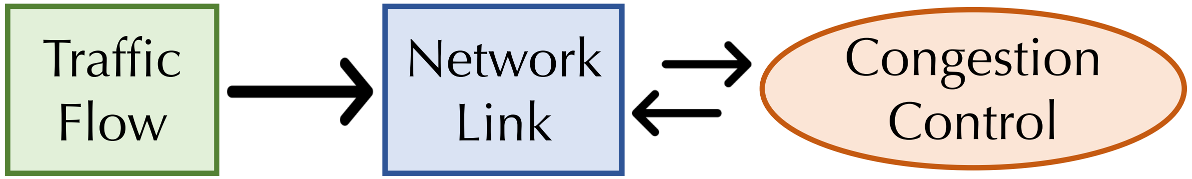


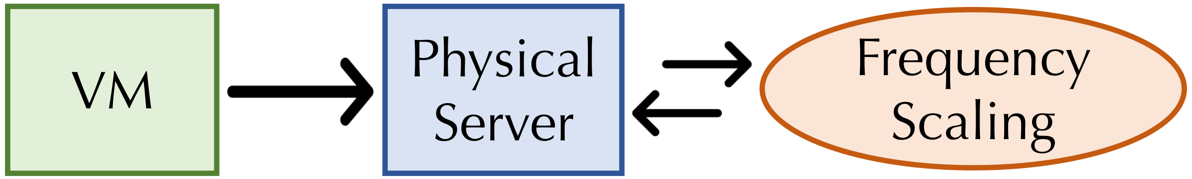
State Space (s_t)
Sending/receiving rate, Sending/receiving duration, avg RTT in a time window, min RTT, RTT inflation, RTT ratio, Ack/Sent latency inflation, Loss/Sent ratio
Action Space (a_t)
Sending rate in the current time window
Reward Function (r_t)
$r_t = \alpha \cdot Throughput_t + \beta \cdot Latency_t + \gamma \cdot Loss_t$

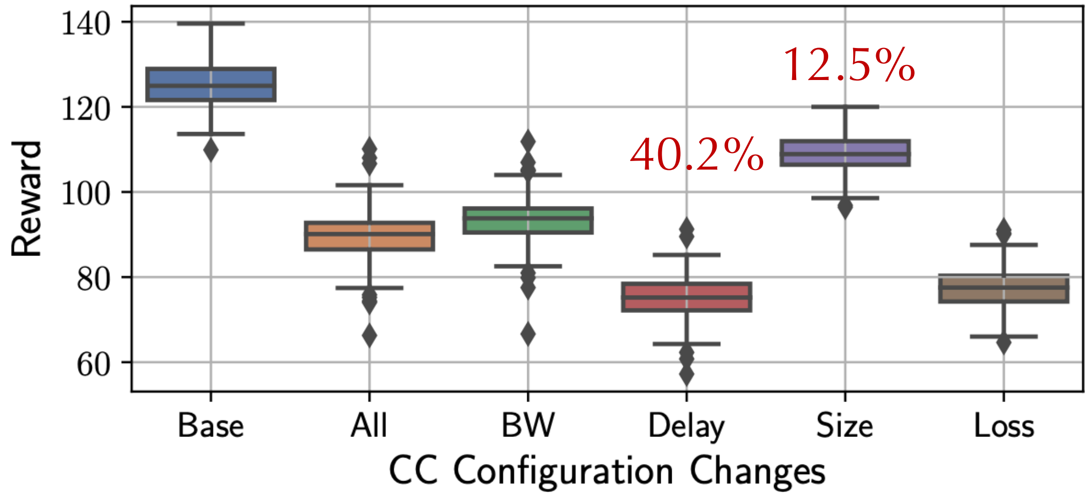
State Space (s_t)
Instructions per second (IPS), CPU usage, Measured core frequency, SLO Preservation Ratio (Latency, Throughput)
Action Space (a_t)
CPU core frequency (every second)
Reward Function (r_t)
$r_t = \alpha \cdot ((IPS_t - IPS_{t-1})/IPS_t)_{\Delta freq>0} + (1-\alpha) \cdot SP_t$

Also Required in Other Tasks

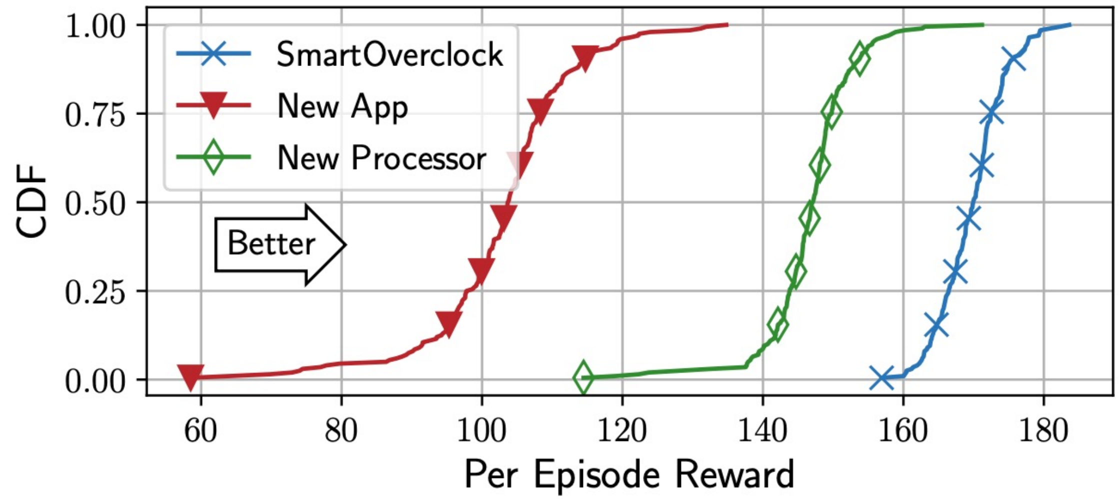
- **Congestion Control**
  - PCC (NeurIPS 2019), RL – PPO Algorithm
- **CPU Frequency Scaling**
  - SmartOC (ASPLOS 2022), RL – Q-Learning Algorithm







Avg: 32.8% degradation (p90),
27% degradation (p50)



44.2% degradation (new app)
17.3% degradation (new processor)