AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems

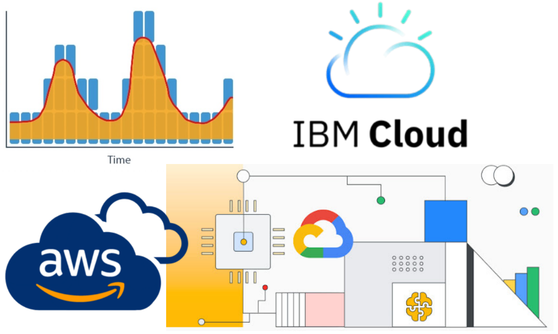
Haoran Qiu[1], Weichao Mao[1], Chen Wang[2], Hubertus Franke[2], Alaa Youssef[2]
Zbigniew T. Kalbarczyk[1], Tamer Basar[1], Ravishankar K. Iyer[1]

[1] UIUC [2] IBM Research

ATC '23

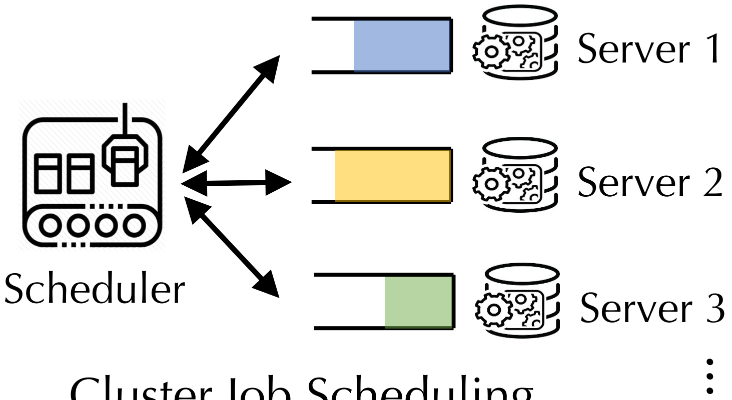
Cloud Systems: Natural Arena for RL

- Full of sequential decision-making processes
  - E.g., resource management, job scheduling, congestion control, etc.
- Hard to model, mostly rely on human-engineered heuristics
  - RL enables using DNNs to express the (1) complex dynamics with raw and noisy signals (2) policies
- Abundant data generated in modern cloud systems: monitoring measurements, systems metrics, workload performance, etc.
  - E.g., Prometheus for Kubernetes, Monarch (Google), Scuba (Meta), etc.



Resource Management

Congestion Control



Cluster Job Scheduling

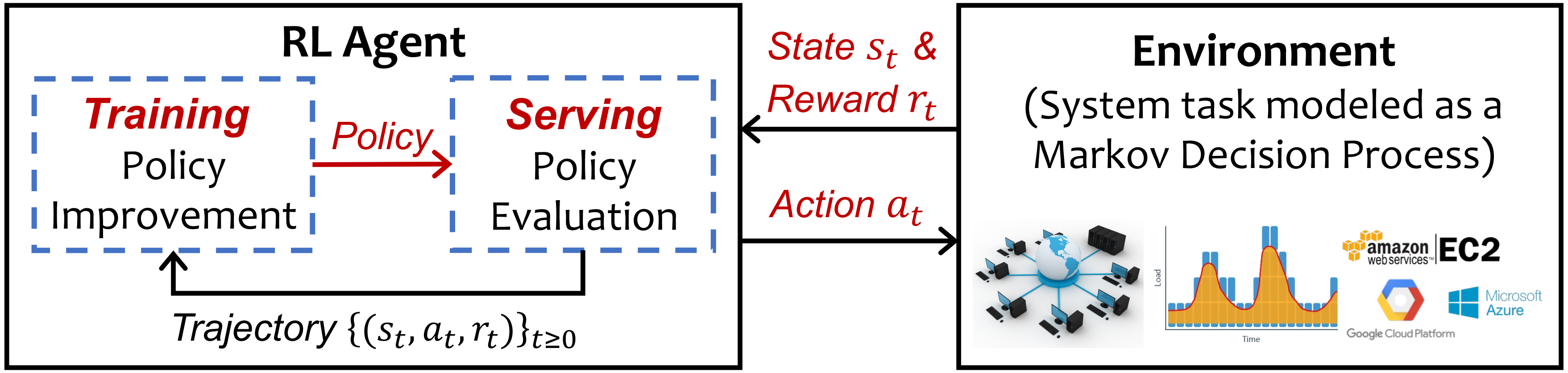
Examples of RL in Cloud Systems

- Cluster Management and Scheduling
  - Job scheduling (SIGCOMM 2019, NeurIPS 2019, HotNets 2016), Process scheduling (ICML 2020), Device placement (ICLR 2018)
- Networking and Video Streaming
  - Congestion control (ICML 2019, AAAI 2021, SIGCOMM 2022), Adaptive video streaming (SIGCOMM 2017)
- Database Optimization
  - Query optimization (VLDB 2019), Index structure (SIGMOD 2018)
- **Resource Management and Autoscaling [Our Focus]**
  - MIRAS (ICDCS 2019), **FIRM (OSDI 2020)***, A-SARSA (ICWS 2020), ADRL (TPDS 2021), Q-learning-based Autoscaler (CCGrid 2021), SOL (ASPLOS 2022), **SIMPPO (SoCC 2022, NeurIPS 2022)***, DeepScaling (SoCC 2022)

*H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, et. al. *SIMPPO: A Scalable and Adaptive Online Learning Framework for Serverless Resource Management*. SoCC 2022.
*H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, R. K. Iyer. *FIRM: An Intelligent Fine-Grained Resource Management Framework for SLO-Oriented Microservices*. OSDI 2020.

Cloud Systems Management with RL: A Primer

- RL agent interacts with an environment, step by step taking observations (s_t), making actions (a_t), receiving rewards (r_t)
- Optimize for specific workloads (e.g., small jobs, low load, periodicity, high scaling factor) by continuing to learn and maximizing the reward
- Direct real benefit by aligning the objectives with reward functions (i.e., agent performance): Meeting SLOs & Higher cluster utilizations

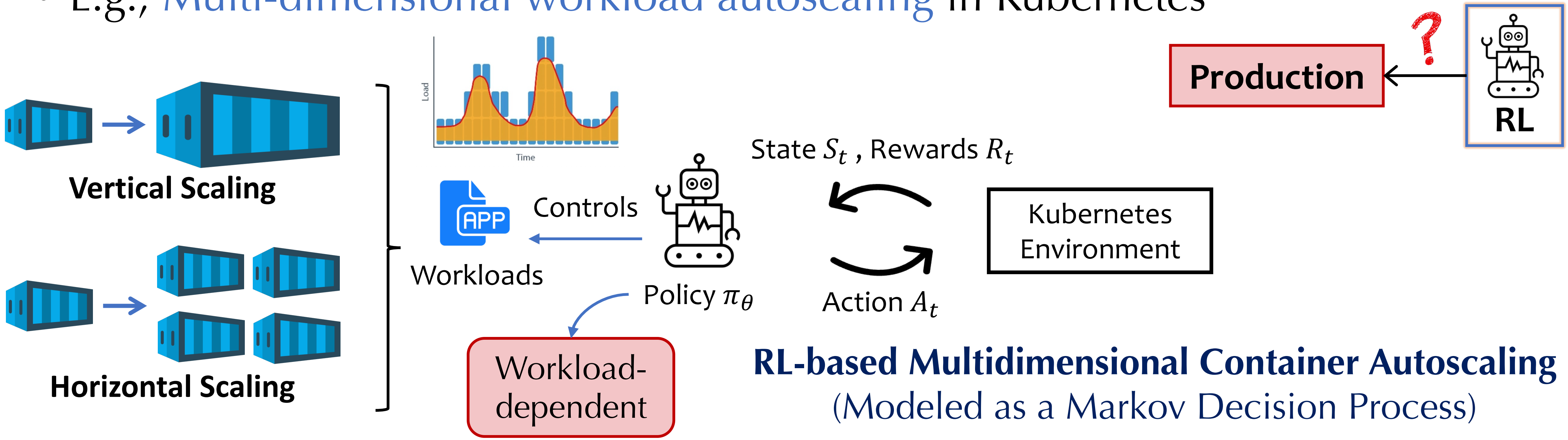


Goal: Maximize the expected cumulative reward $\mathbb{E}[\sum_{t=0}^{T} \gamma^t \cdot r_t]$

(in any trajectory with T steps)

A Framework for Running RL in Production is Missing

- Bridge RL model development and advances to production
- Allow robust and reliable deployment of RL-based controllers in real cloud systems
- **Goal: To provide a framework for managing and running RL-based controller in production cloud systems**
  - E.g., Multi-dimensional workload autoscaling in Kubernetes



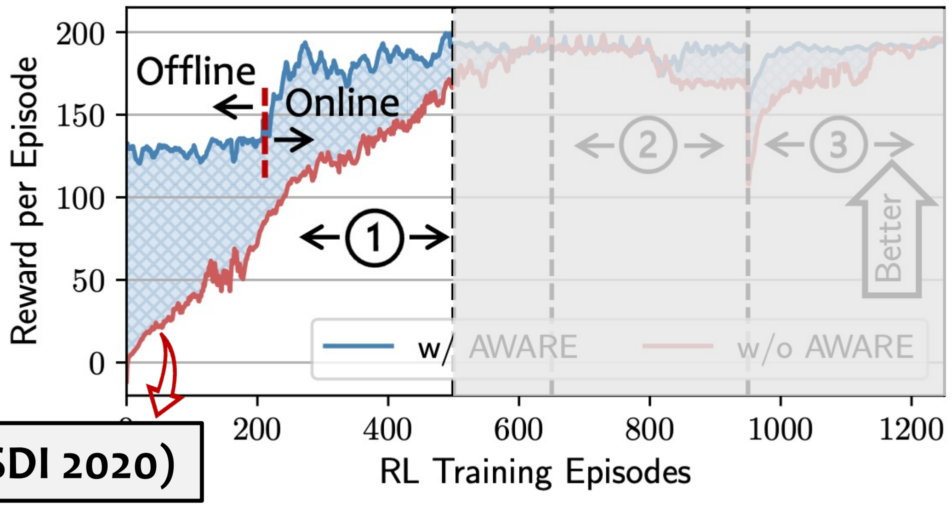
RL-based Multidimensional Container Autoscaling
(Modeled as a Markov Decision Process)

What are the Challenges?

Enabling built-in intelligence in cloud systems with less manual intervention while achieving high robustness and self-adaptation (in both training/inference)

Challenge #1: In the early training stages, RL agents tend to generate poor autoscaling decisions

- Lower than baseline rewards (i.e., worse agent performance) and more SLO violations
- Solution: Reliable RL exploration with offline training (i.e., bootstrapping) + online training & inference



FIRM (OSDI 2020)

RL Episodes	EP #1-100	EP #101-200	EP #201-300	EP #301-400
CPU Util	-32.3% ± 14%	-42.9% ± 15%	-22.1% ± 12%	-10.0% ± 6%
Memory Util	-28.8% ± 11%	-30.5% ± 10%	-26.5% ± 8%	-7.8 % ± 2%
SLO Violations	56.1 ± 14x	22.2 ± 7x	12.7 ± 5x	10.1 ± 3x

Overprovisioning -> CPU & memory utils deficit compared w/ baseline

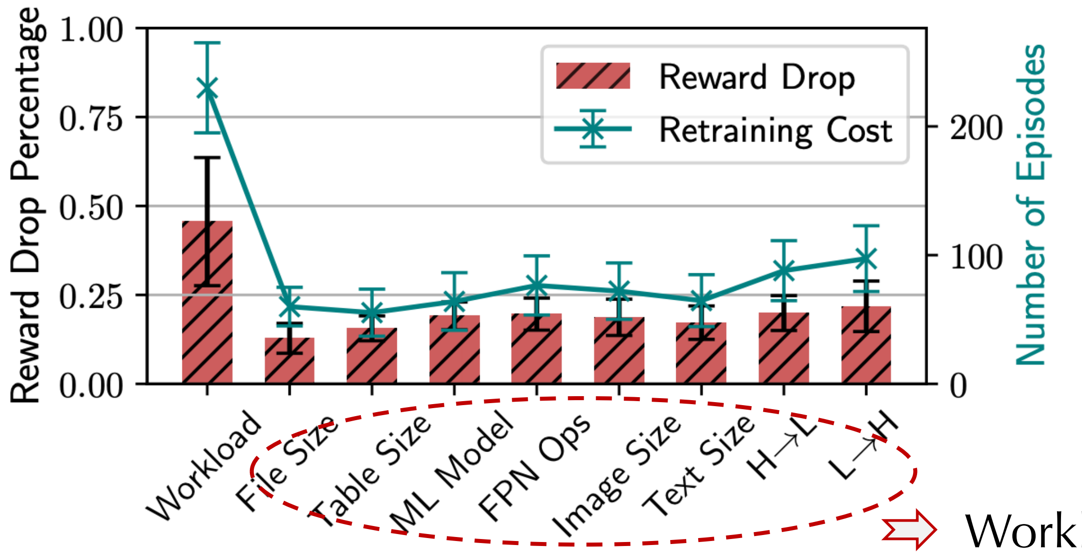
Unable to re-scale properly for workloads changes -> SLO violations

What are the Challenges?

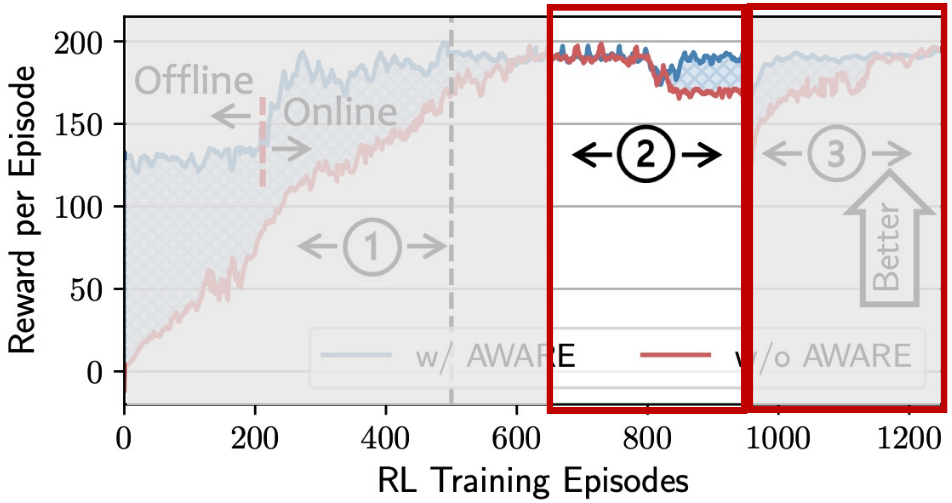
Enabling built-in intelligence in cloud systems with less manual intervention while achieving high robustness and self-adaptation (in both training/inference)

Challenge #2: During policy-serving stage, RL agent performance degrades when workloads are updated

- Solution: Continuous monitoring + Retraining detection & trigger mechanism



Workload changes leads to 21.8% reward drops



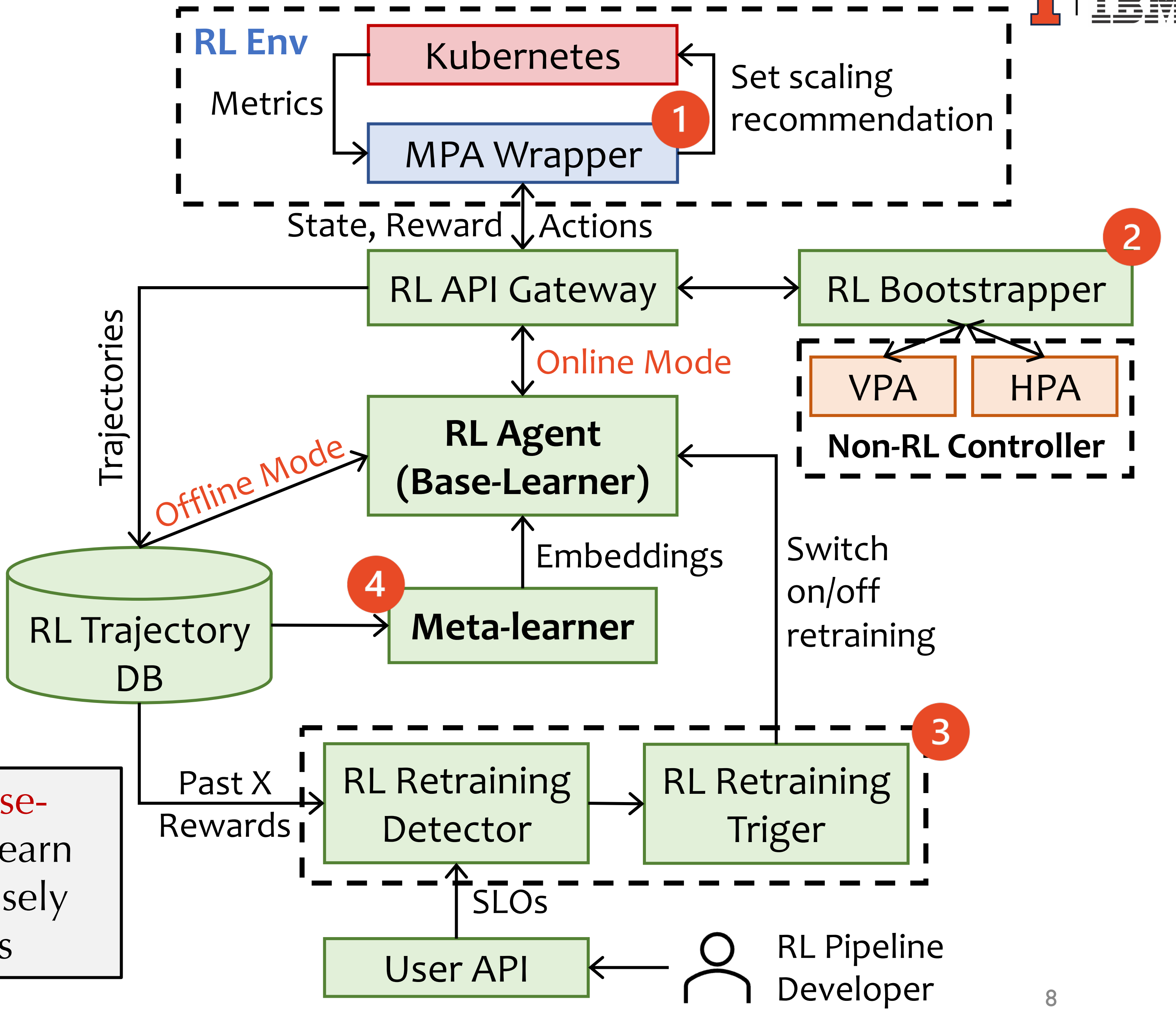
Challenge #3: Trained policies are application-specific, costly to adapt to new applications

- 45.6% reward degradation (~230 eps retraining)
- Solution: Meta-learning for fast model adaptation

AWARE Overview

Key Components:

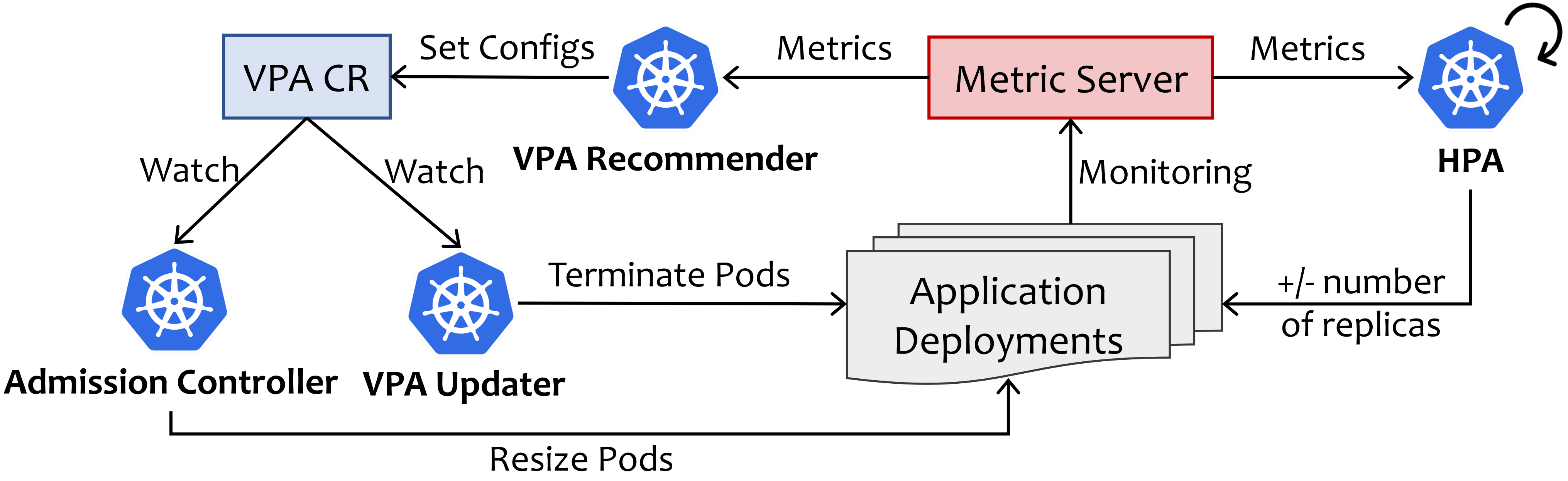
1. An MPA (multi-dimensional pod autoscaling) system for RL
2. Offline training (via an RL bootstrapper) followed by online training & inference
3. An RL retraining detection & trigger module
4. A meta-learning module for fast model adaptation



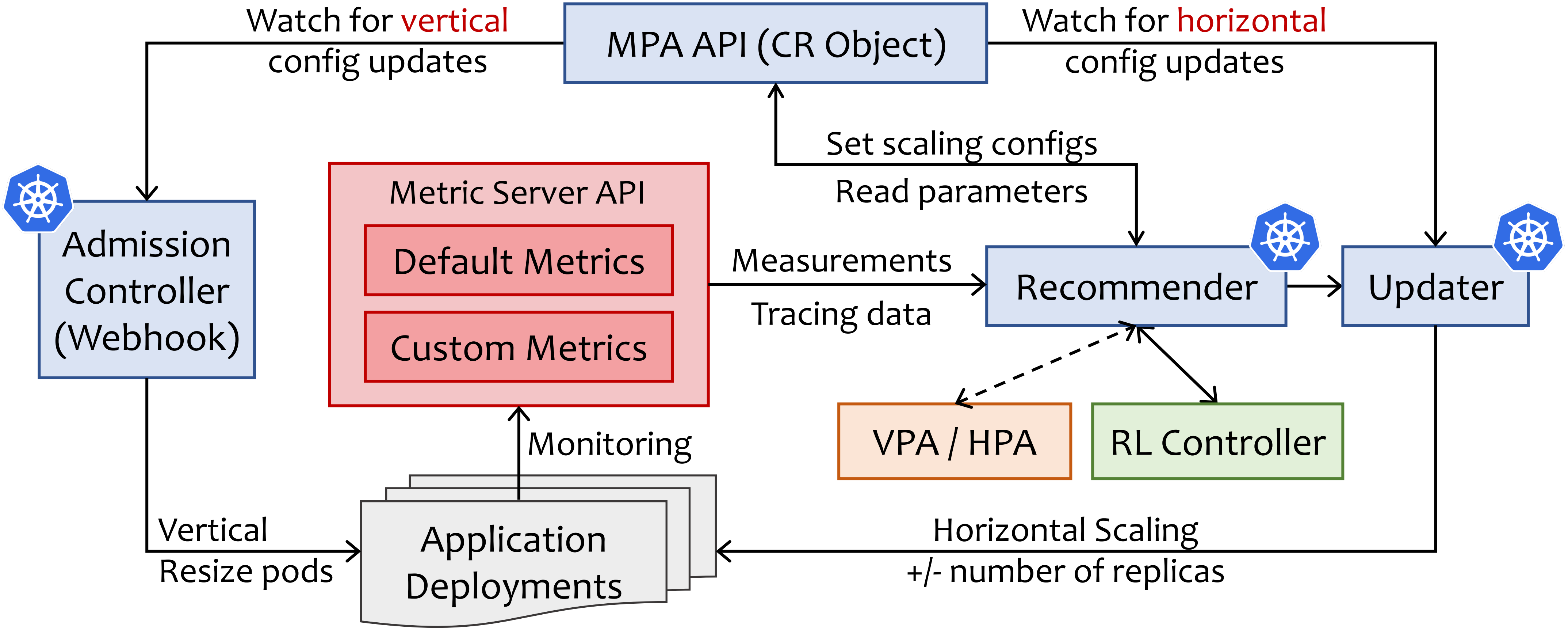
Key Idea: Models the RL agent as a base-learner and creates a meta-learner to learn to generate embeddings that can precisely differentiate and represent applications

Multi-dimensional Pod Autoscaling (MPA)

- Open-source Framework: A system design that allows general workloads on Kubernetes to use RL-based autoscalers such as FIRM
  - Reusing HPA/VPA as a fallback to RL to have a default autoscaling algorithm
  - Scaling recommendation is separated from actuation
  - Supports customized plug-and-play multi-dimensional autoscaling algorithms



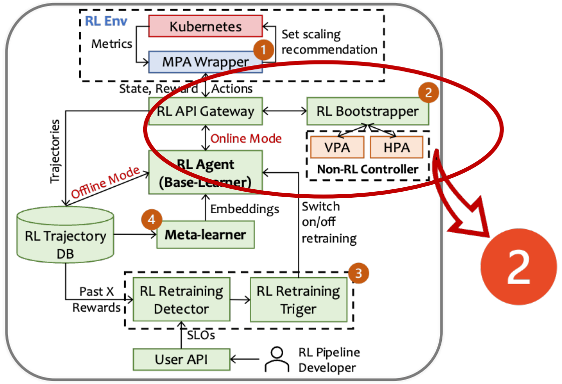
MPA Design Overview



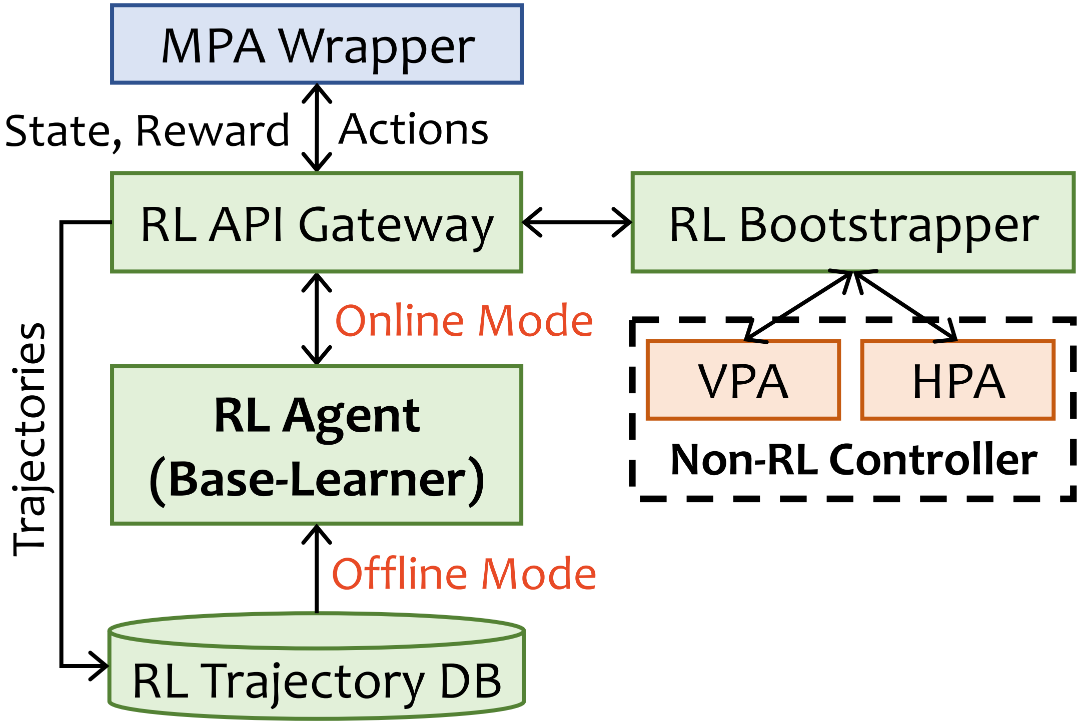
Kubernetes Operators

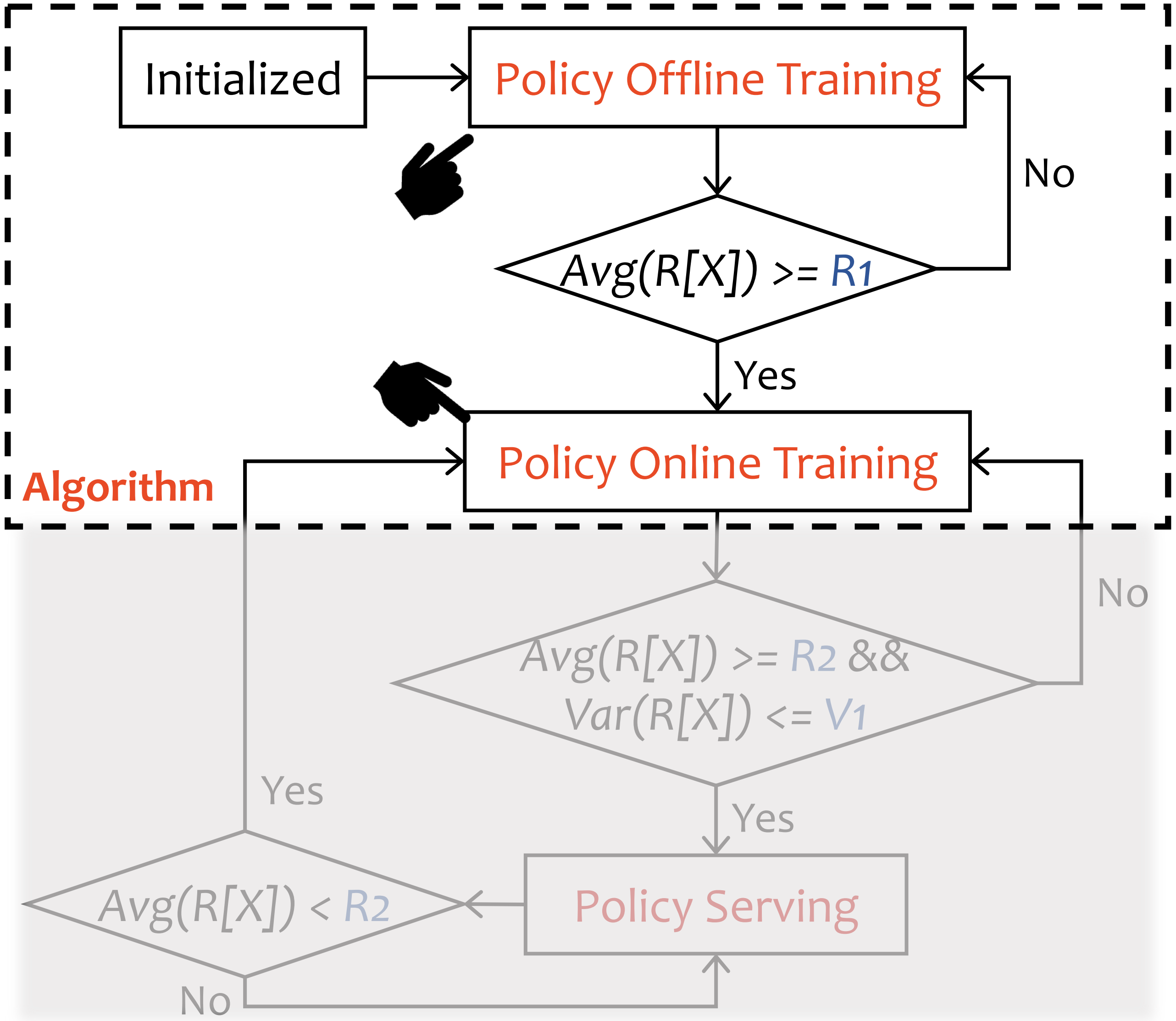
* MPA: Multi-dimensional Pod Autoscaler

RL Training Bootstrapping



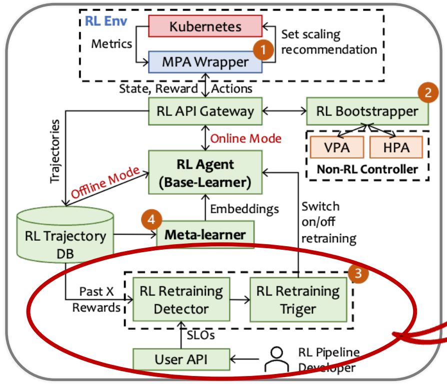
An RL bootstrapper that combines offline training with online training & inference



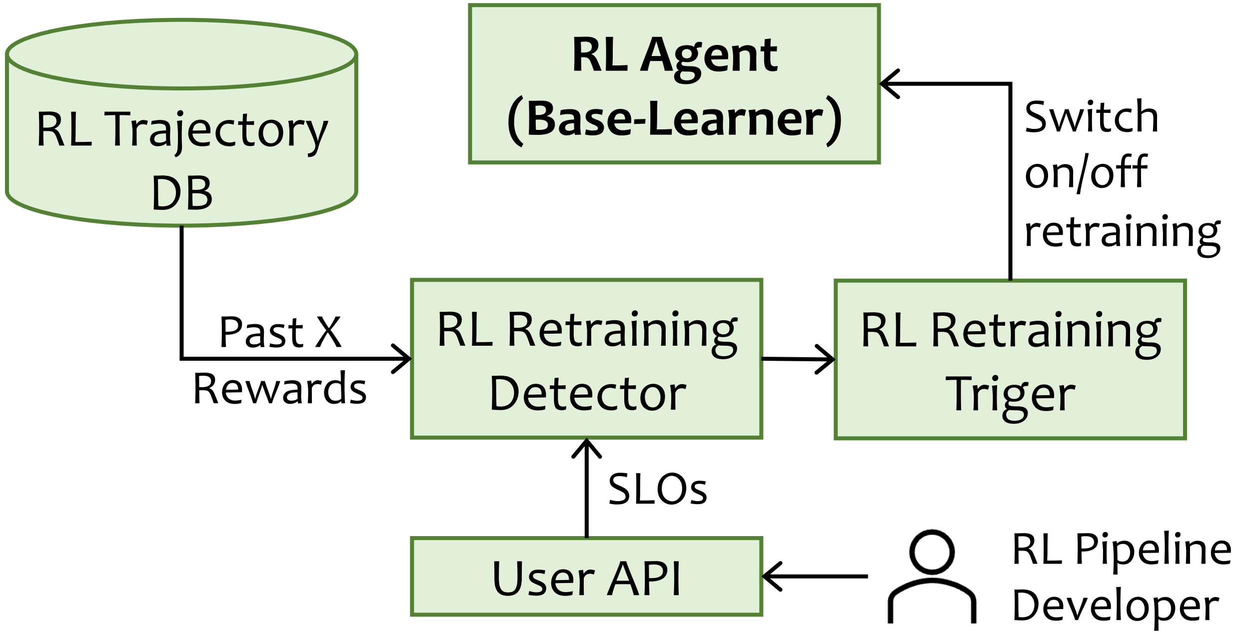


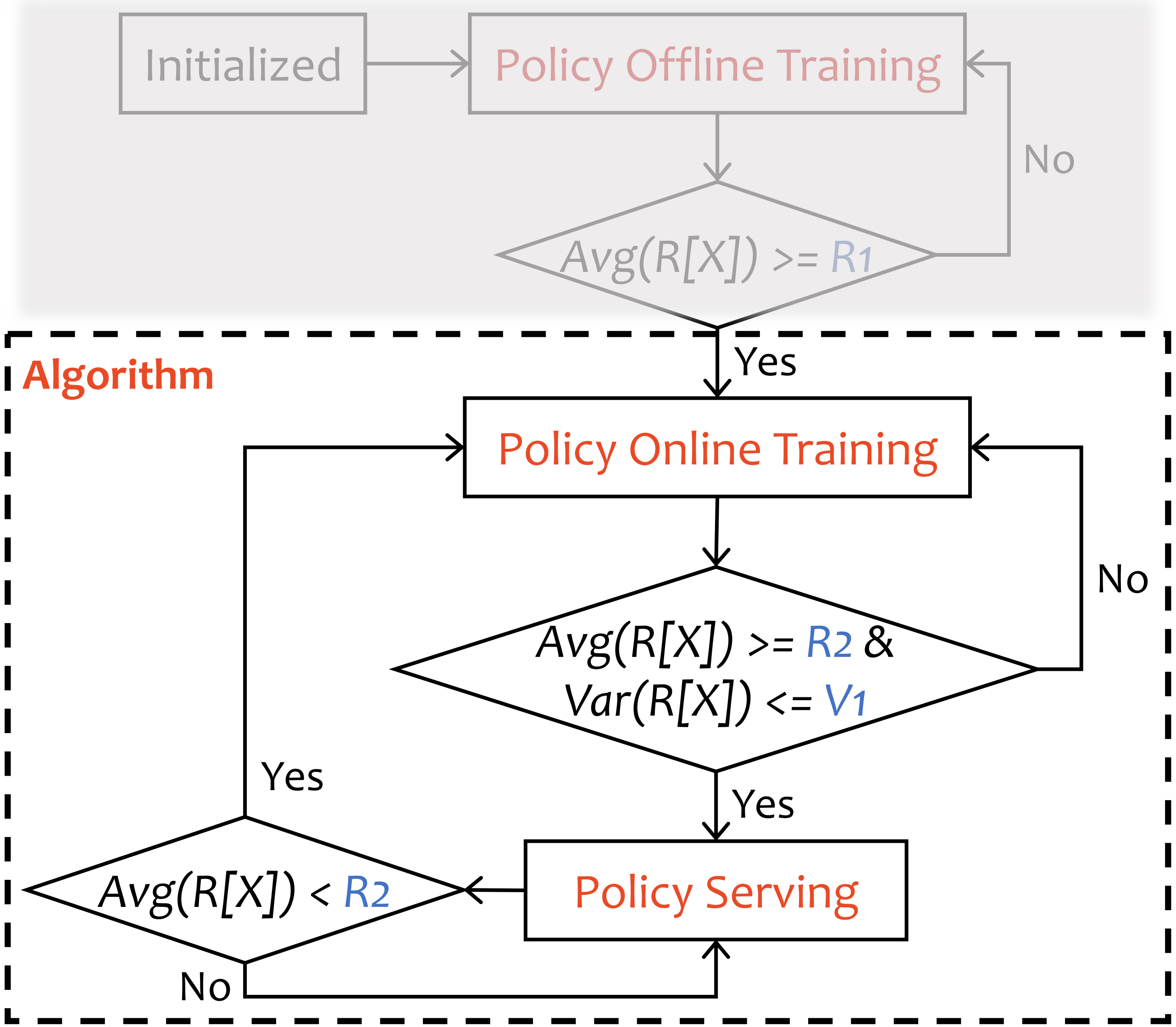
* R1 and R2 are calculated based on user-specified SLOs

RL Retraining Detection and Trigger



3 An RL retraining detection & trigger module



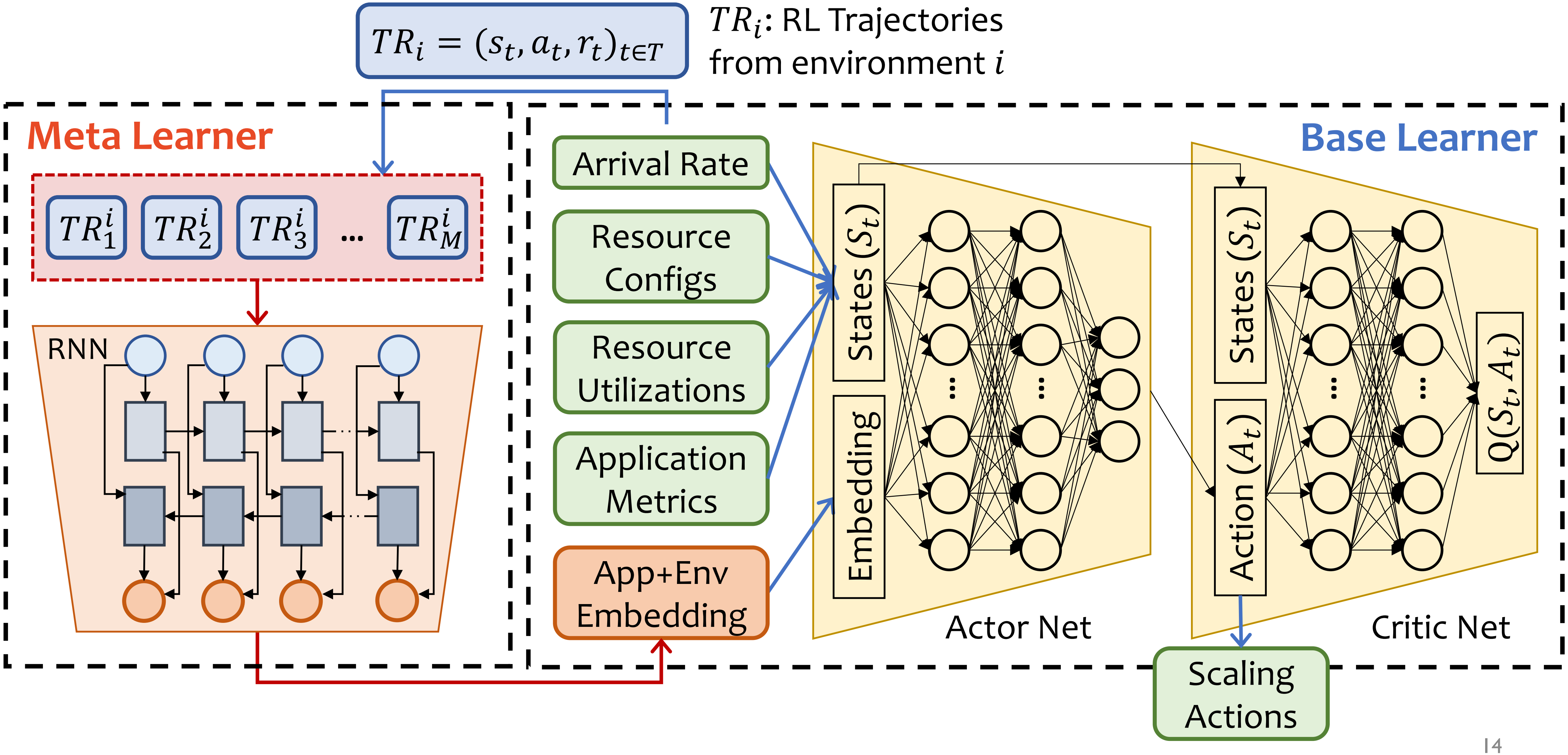


* R1 and R2 are calculated based on user-specified SLOs

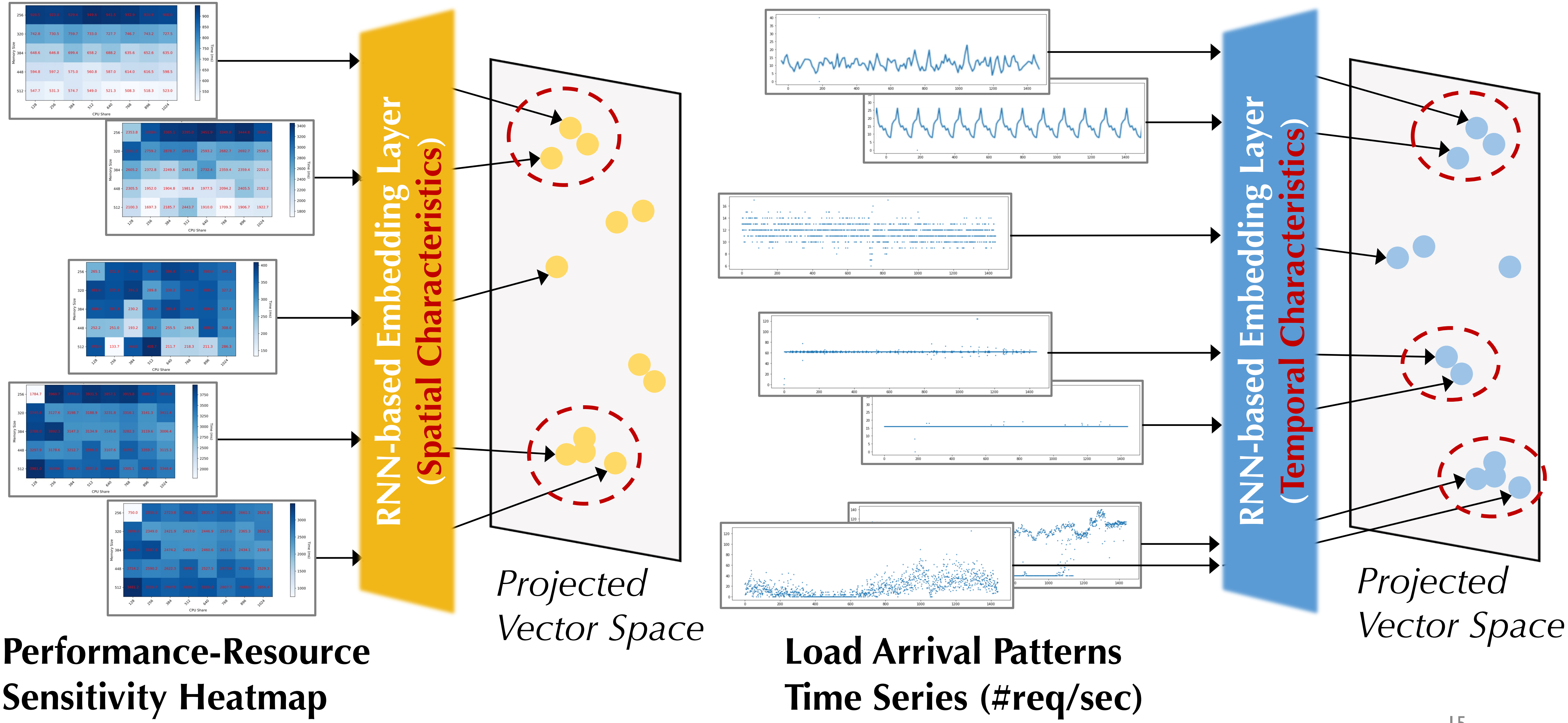
Fast Model Adaptation with Meta-learner

- **Goal**: To reduce RL model retraining time (cost) and adapt quickly to new application workloads (unseen during training)
- **Key Idea**: Model each RL agent as a base-learner and create a meta-learner to learn to generate an *embedding* that can accurately represent each environment
  - The embedding is fed to the base-learner (as state input) to differentiate one RL environment from another -> customized to each environment
- **Why meta-learning?**
  - "Learning to learn"
  - Capable of adapting well or generalizing to new environments that have never been encountered during training
  - Adaptation process requires only limited exposure to the new environment
  - A systematic framework that enables automatic adjusting of internal hidden states to learn (combined with RL -> learned policy conditioning on the application)

AWARE Design and Model Architecture



Interpreting "Embeddings" from Systems Perspective

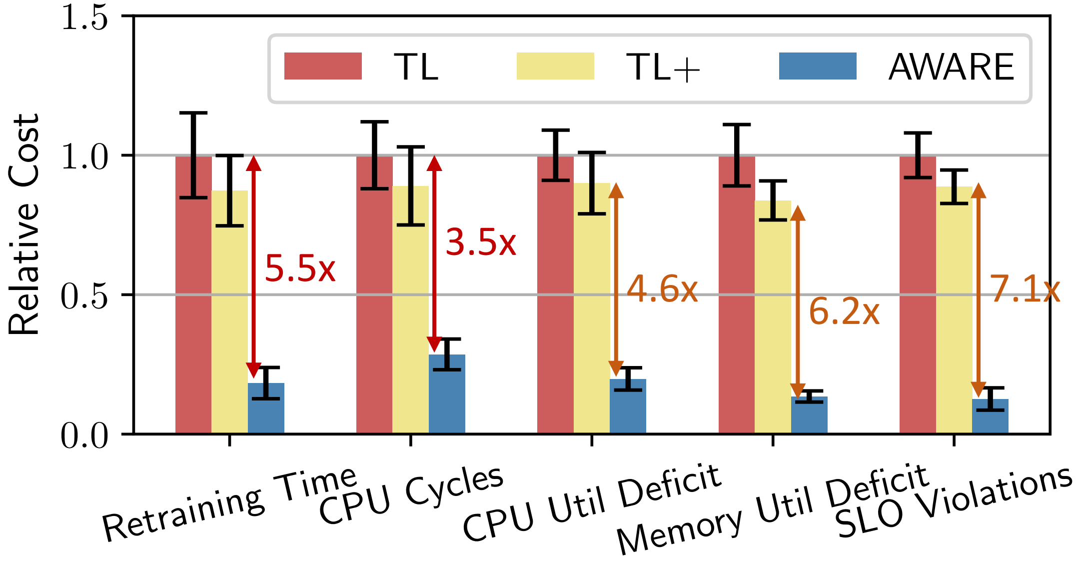


Evaluation

- **RQ1**: Does AWARE provide fast model adaptation to new workloads?
  - What is the value of meta-learning?
- **RQ2**: How does AWARE perform in online policy-serving when workload updates or load changes occur?
- **RQ3**: How does AWARE perform in the early stages of policy training, compared to RL agents without bootstrapping?
- Workload generation:
  - 16 represented production serverless function segments (e.g., CPU-intensive jobs, image manipulation, text processing, web serving, ML model serving, I/O services)
  - Generated 1000 synthetic applications by random selection and combination
- RL agent/algorithm (i.e., base-learner) implementation adopted from FIRM (OSDI 2020) – DDPG, an actor-critic RL algorithm
  - Reward function: $R(t) = \alpha \cdot RU(t) + (1 - \alpha) \cdot SP(t)$, where $SP(t) = \min(\frac{latency_{SLO}}{\text{latency}}, 1)$

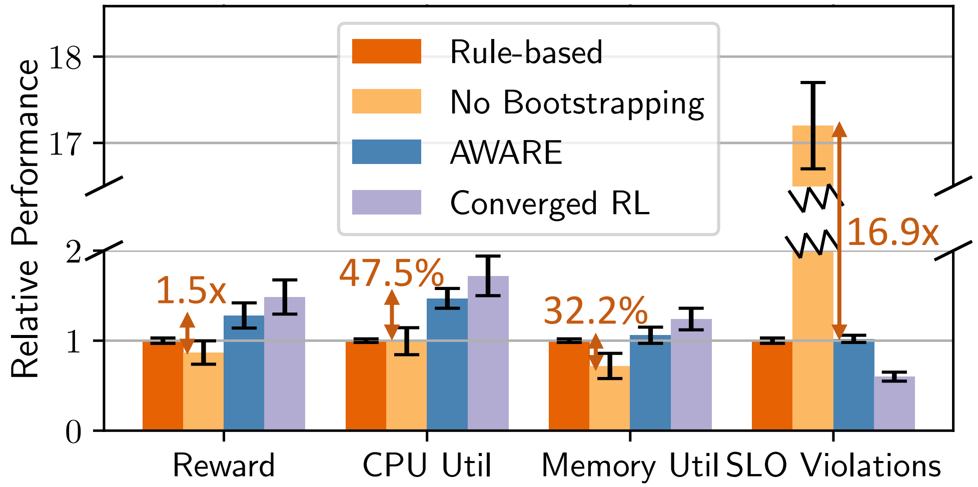
RQ1 – Fast Model Adaptation

- AWARE adapts 5.5× and 4.6× faster than TL and TL+
  - TL: Transfer learning with model parameter sharing
  - TL+: Transfer learning that includes additional features
- AWARE saves 68–72% CPU cycles
- AWARE reduces CPU and memory utilization deficit by 4.6× and 6.2×
- AWARE reduces SLO violations by 7.1×



RQ2 & RQ3 – Bootstrapping and Online Policy-serving

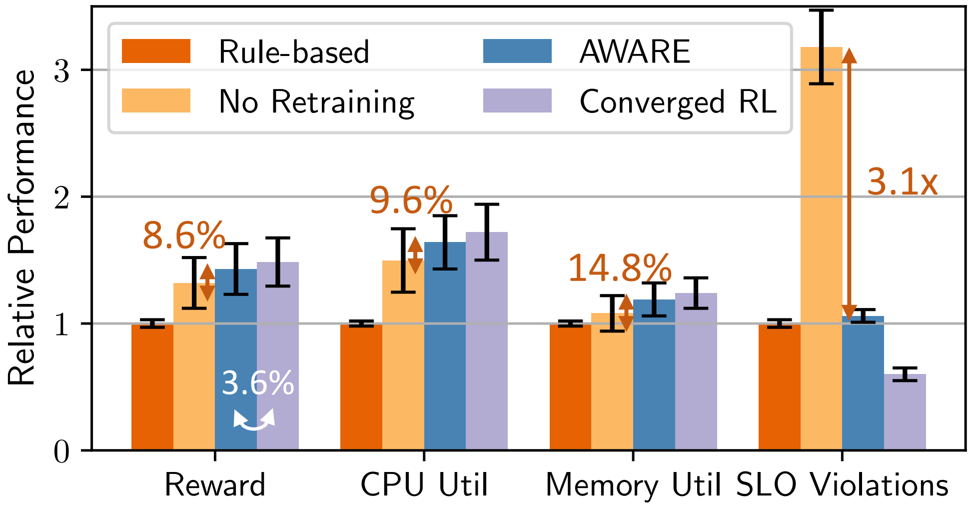
Bootstrapping



Compared to no-bootstrapping:

- AWARE had 47.5% and 32.2% higher CPU and memory utilization
- AWARE reduced workload SLO violations by 16.9×

Online Policy-serving



Compared to no-retraining:

- AWARE had 9.6% and 14.8% higher CPU and memory utilization
- AWARE reduced workload SLO violations by 3.1×

Summary and Future Work

- AWARE is an extensible framework for deploying and managing RL-based controllers in production systems
- AWARE provides (1) fast adaptation with meta-learning, (2) reliable RL exploration with bootstrapping, (3) robust online performance with timely retraining
- Demonstrated AWARE in workload autoscaling:
  - Adapts a learned autoscaling policy to new workloads 5.5× faster than the existing transfer-learning-based approach
  - Provides stable online policy-serving performance with less than 3.6% reward degradation
  - Helps achieve 47% and 32% higher CPU and memory utilization while reducing SLO violations by a factor of 16.9× during initial policy training
- Out-of-distribution cases (limitation of meta-learning)
  - Detection/classification + Fine-grained customization
- Future Work: Extend the meta-learning-based framework for other workload-aware ML4Sys cases as a general paradigm which supports fast model adaptation
  - Scheduling, resource config search, congestion control, power management, etc.

Thank you!

Haoran Qiu[1], Weichao Mao[1], Chen Wang[2], Hubertus Franke[2], Alaa Youssef[2]
Zbigniew T. Kalbarczyk[1], Tamer Basar[1], Ravishankar K. Iyer[1]

[1] UIUC [2] IBM Research

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Check out the paper for more details:
https://www.usenix.org/conference/atc23/presentation/qiuhaoran