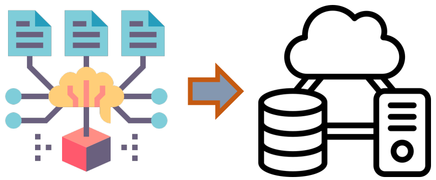
ILLINOIS
IBM Research



On the *Promise* and *Challenges* of **Foundation Models** for Cloud Systems Management

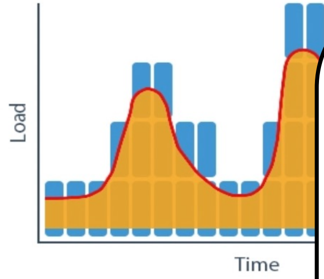
Haoran Qiu[1], Weichao Mao[1], Chen Wang[2], Hubertus Franke[2]
Zbigniew Kalbarczyk[1], Tamer Basar[1], Ravishankar Iyer[1]
[1]UIUC [2]IBM Research

ML for Systems Workshop at NeurIPS 2023

How do we make ML for systems useful?

Data center ma
- Power mar
- Job schedu

Cluster orches
- Container



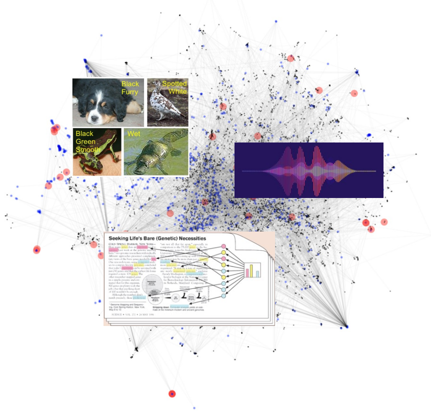
Resource manag
- Resource co
- Autoscaling
- Load balanci

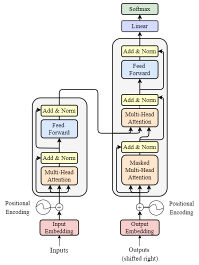
Networking tasks
- Congestion control
- Adaptive video

The recipe:
- 1 Agent
- 1 Task
- 1 Environment
  -> Infra + Workload
- 1 Policy/Model

Great for research and local setup; not great for actually **usable**, **deployable** models!

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





"Foundation Models"

Finetuning or Prompting

What would it take to bring this recipe to systems?

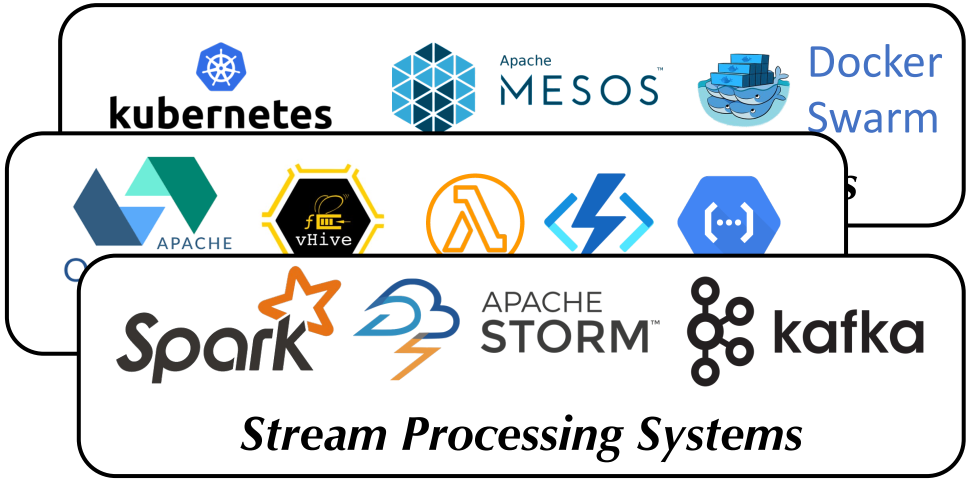
Foundation Model:

A model that **someone else** might actually **use** and **deploy**

Toward general-purpose ML4Sys models

Important design questions:

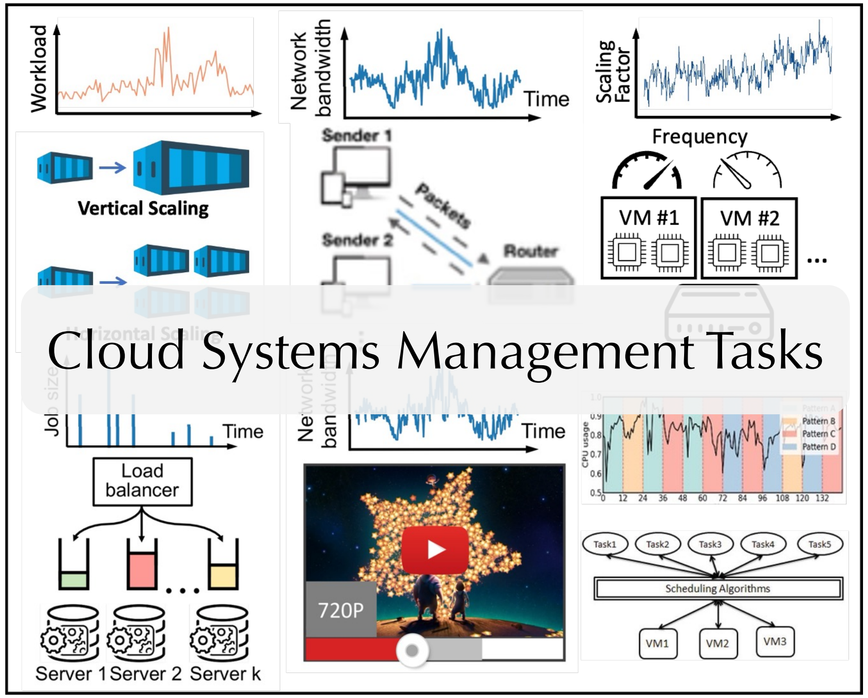
- ***What kind of systems to support?***
  - Must support many systems
  - Must train on many systems
- ***What kind of data should the model use?***
  - Existing ML4Sys model training/testing data
  - Monitoring data and system logs
- ***What should the model do?***
  - Solve a generic enough task (e.g., fundamentally lots of systems tasks are scheduling)
  - Delicate trade-off between generality and specialization
- ***How should the model be used/deployed?***
  - Zero-shot? Prompted? Few-shot? Fine-tuning? Or all of these?

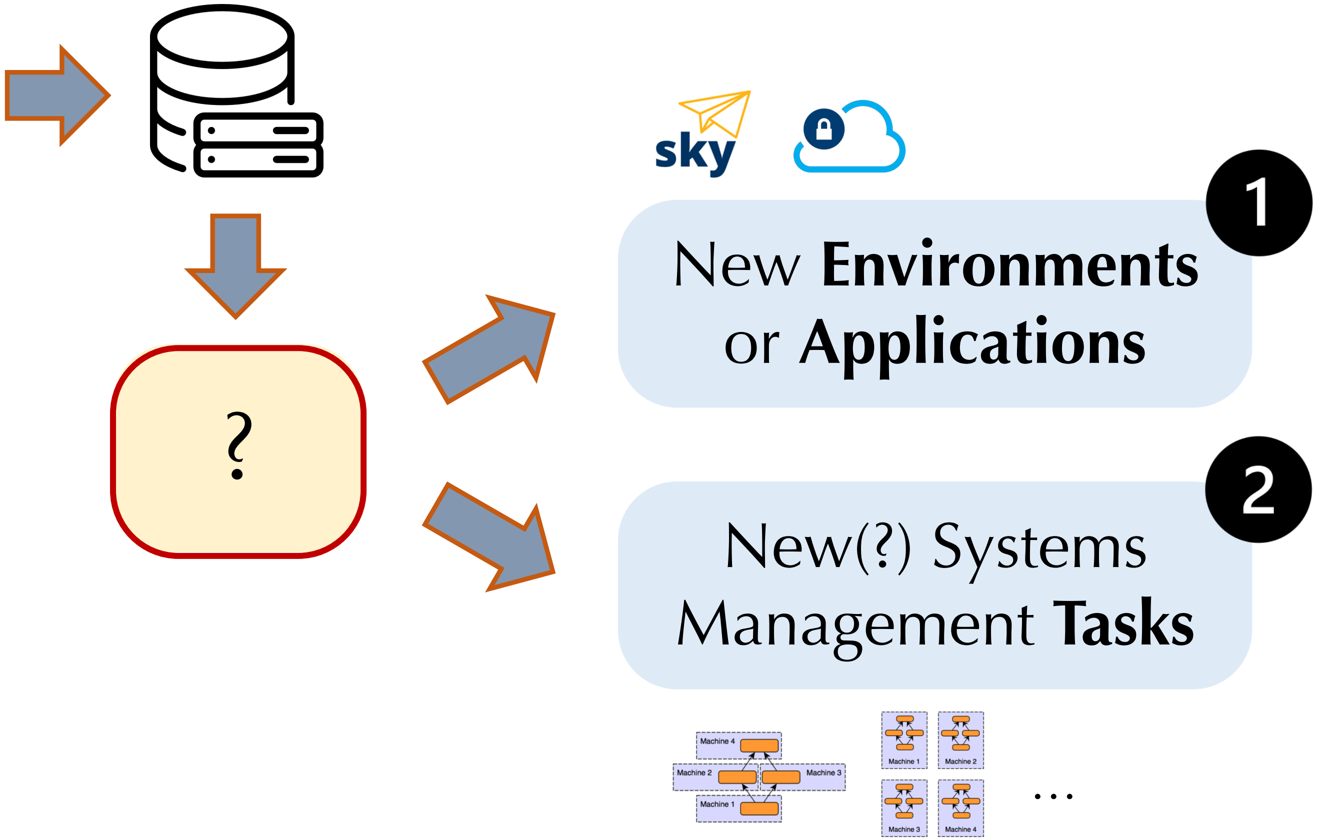


…

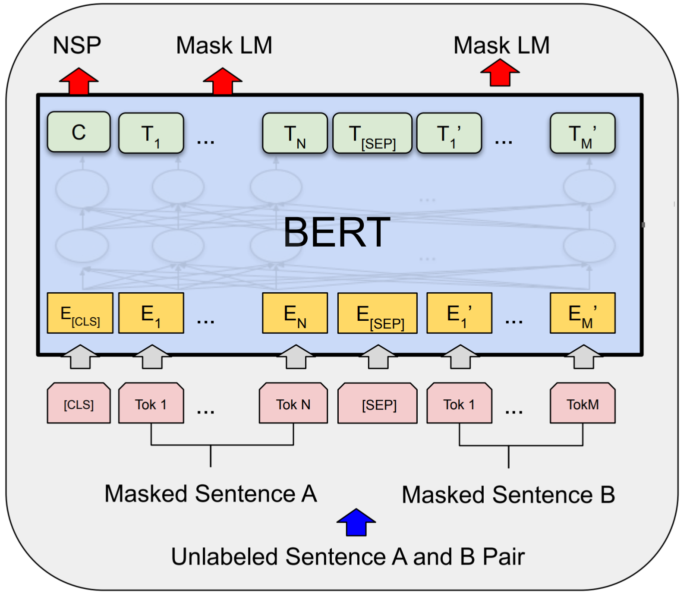
What do we actually want to learn?

What objectives can we use to learn general "common sense" policies from diverse data sources that apply to many systems management scenarios?

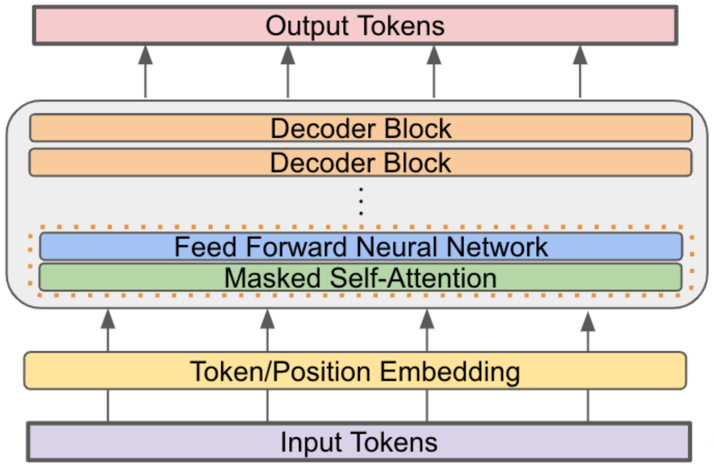




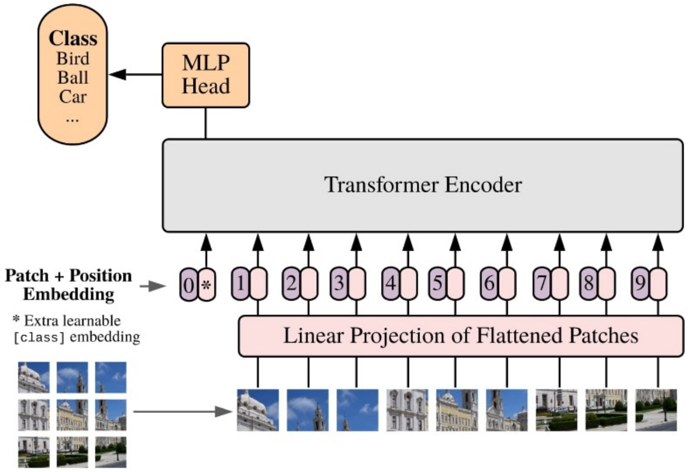
How does unsupervised learning usually work?



BERT: Masked token prediction



LLMs: Next token prediction



ViT: Masked patch prediction

Training on completion of incompletion data (easily obtained), "Fill in the blanks"

This is great because it **does not require strong supervision** (e.g., labels or classes) and therefore can use all available data!

Learning general policies from diverse data

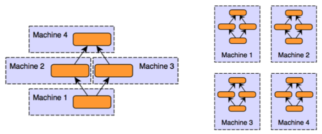
?

1

New **Environments** or **Applications**

2

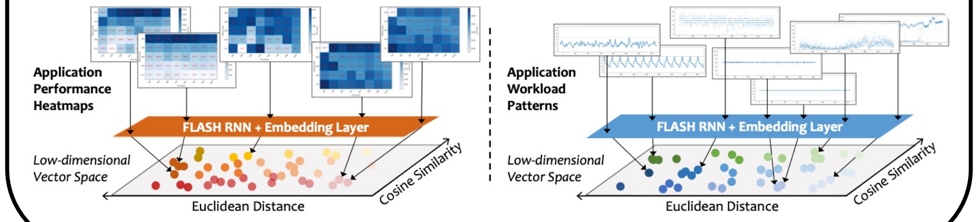
New(?) Systems Management **Tasks**



…

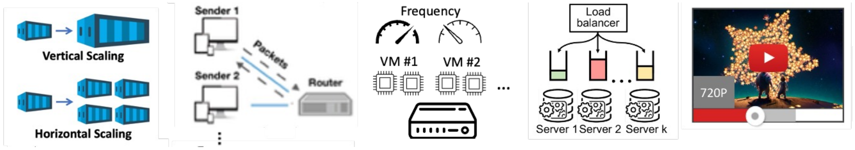
Current Work: FLASH

- *Embedding*-based meta-learning
- Adaptation from *embedding pairing*



Ongoing Work

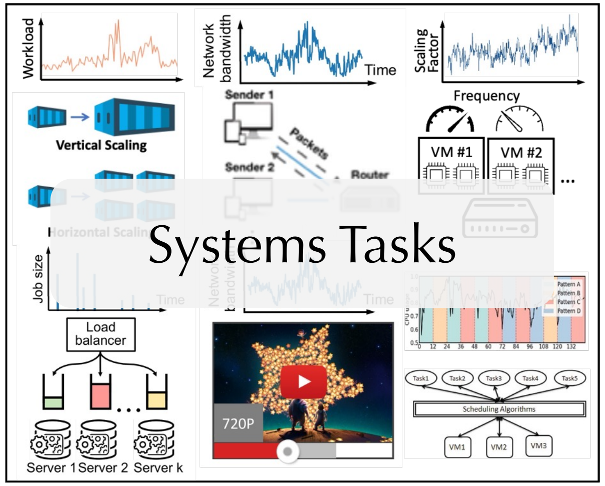
- *Generic pre-training* with fill-in blanks in state-decision trajectories
- Adaptation across tasks

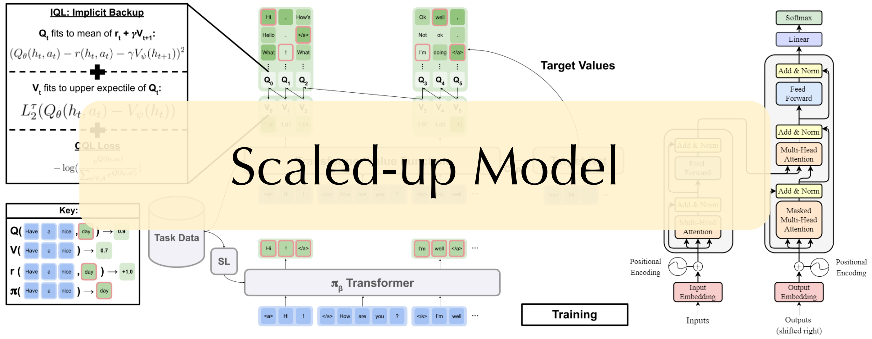


Generalization to entirely new(?) tasks

What objectives can we use to learn general "common sense" policies from diverse data sources that apply to many systems management scenarios? ⇨ Next ??? Prediction

Which bit of data can we "fill in" such that the prediction is not too hard and can generalize across different tasks, yet forces learning useful stuff? ⇨ **State-Decision Trajectories**





- Given current observations
- Predict current decision
- Predict next observation
- Predict time steps to goal

- Very general objective -> can use **any trajectory data**
- Captures systems ***dynamics*** and "***common sense***" of what actions lead to what outcomes

Adapting to diverse application and environments

Major components in ML for Sys:

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


contributed articles

DOI:10.1145/3364684

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

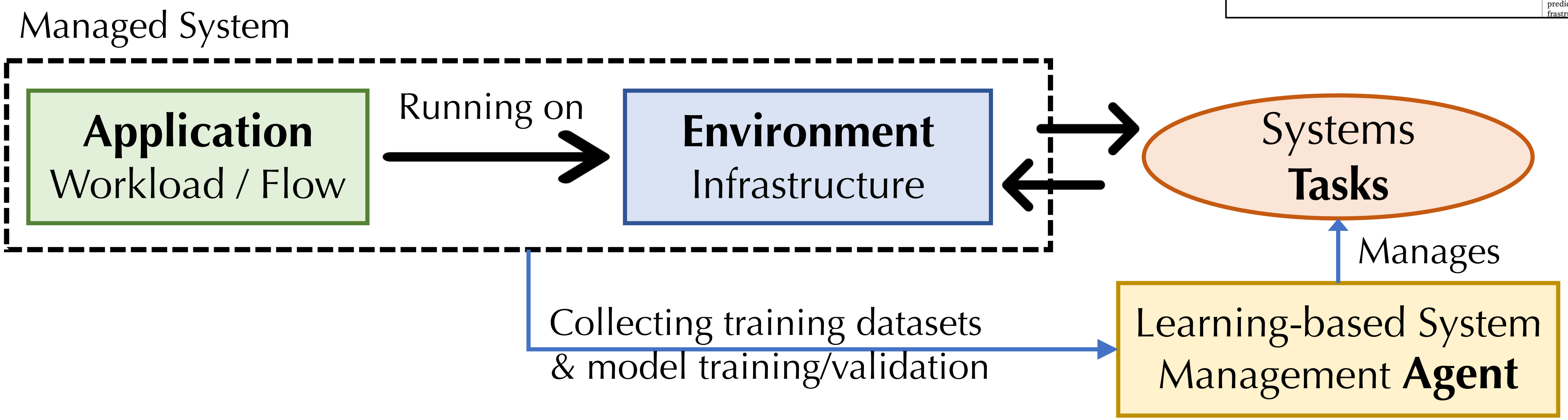
BY RICARDO BIANCHINI, MARCUS FONTOURA, ELI CORTEZ, ANAND BONDE, ALEXANDRE MUZIO, ANA-MARIA CONSTANTIN, THOMAS MOSCIBRODA, GABRIEL MAGALHAES, GIRISH BABLANI, AND MARK RUSSINOVICH

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.

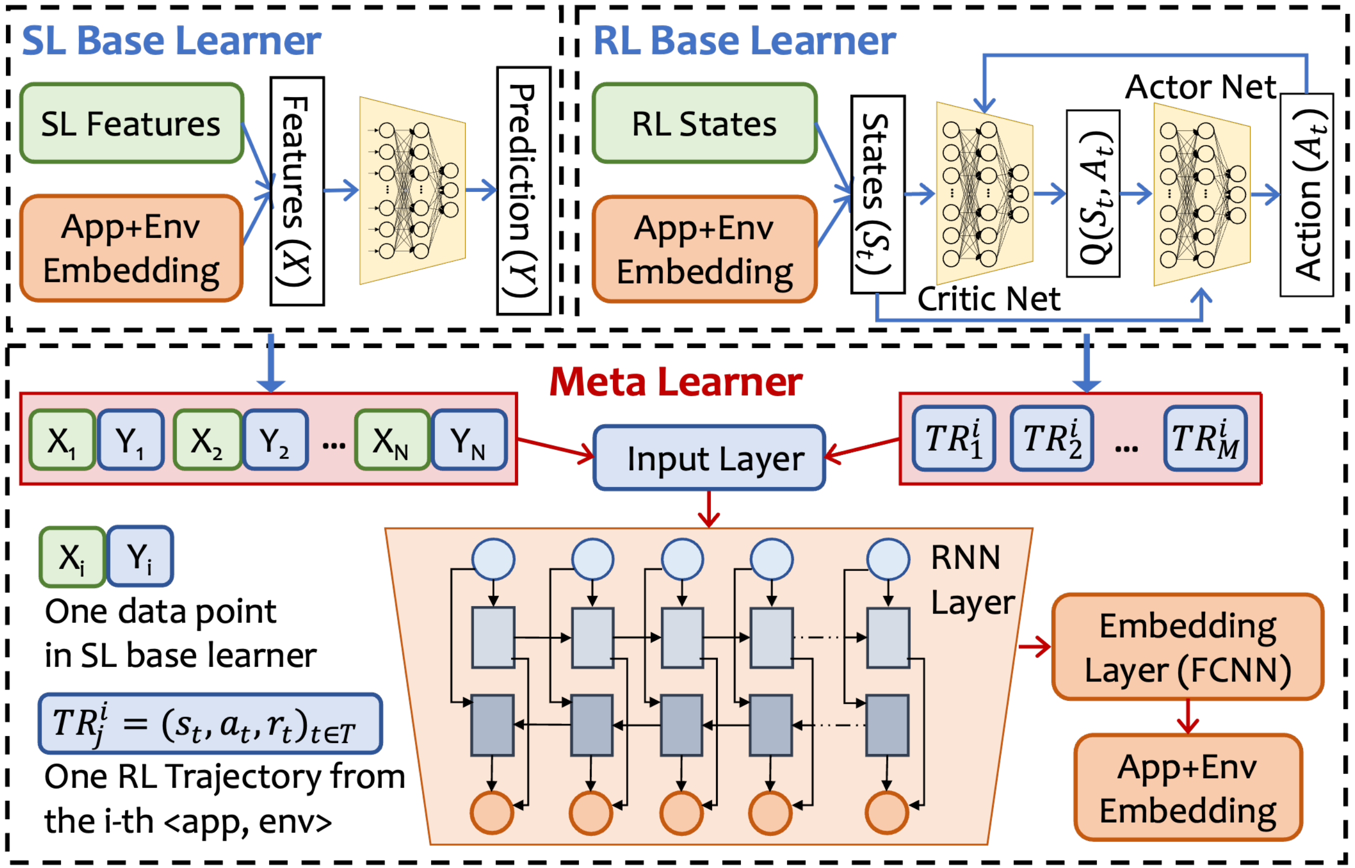
Public cloud providers are starting to explore ML-based resource management in production.[9,14] For example, Google uses neural networks to optimize fan speeds and other energy knobs.[14] In academia, researchers have proposed using collaborative filtering—a common technique in recommender systems—in 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.

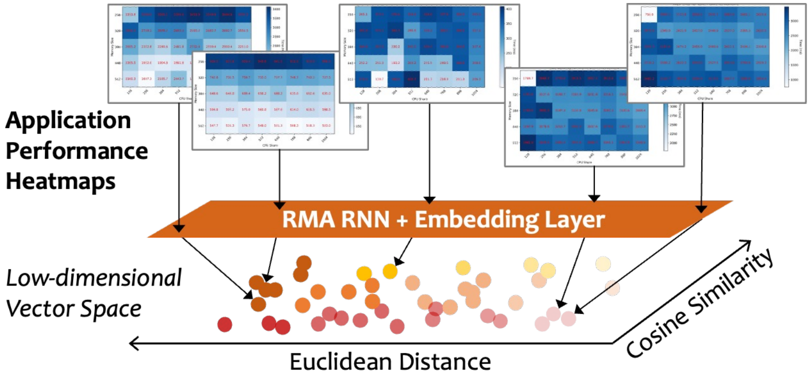
Despite these prior efforts and opportunities, it is currently unclear how best to integrate ML into cloud resource management. In fact, prior approaches differ in multiple dimensions. For example, in some cases, the ML technique produces insights/predictions about the workload or infrastructure; in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

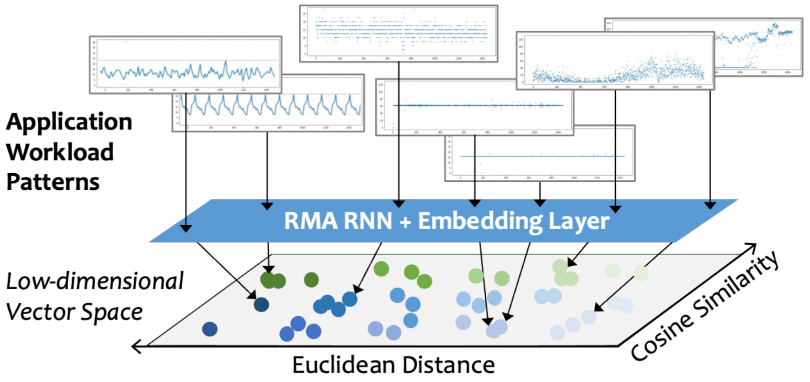


Adapting to diverse application and environments

Supervised Learning   Reinforcement Learning

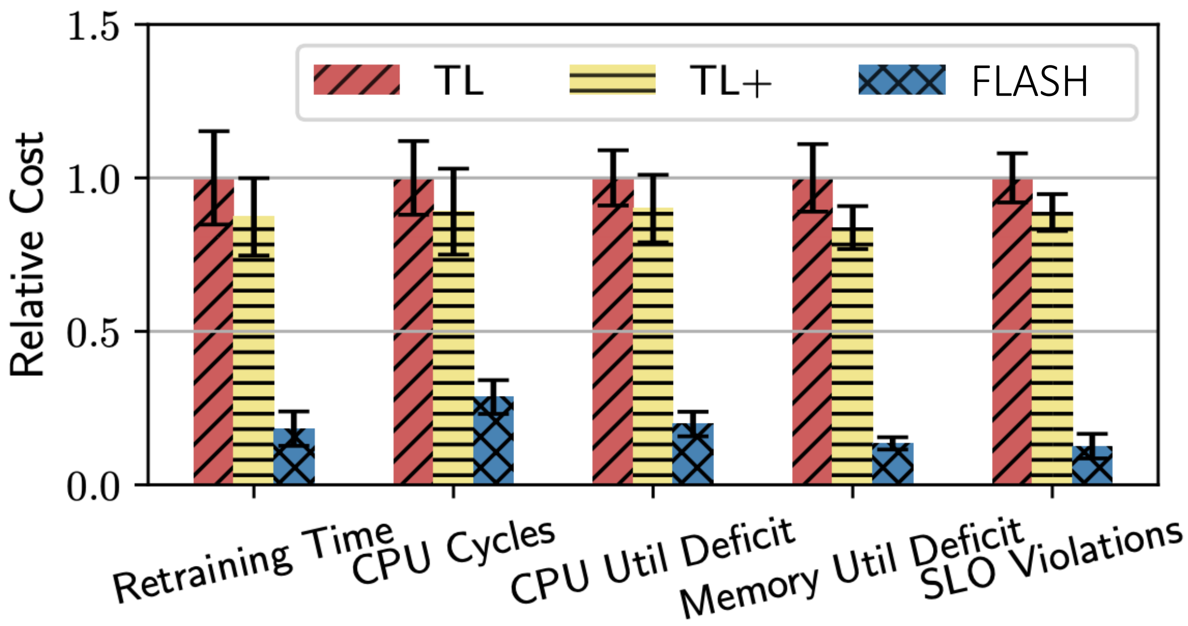


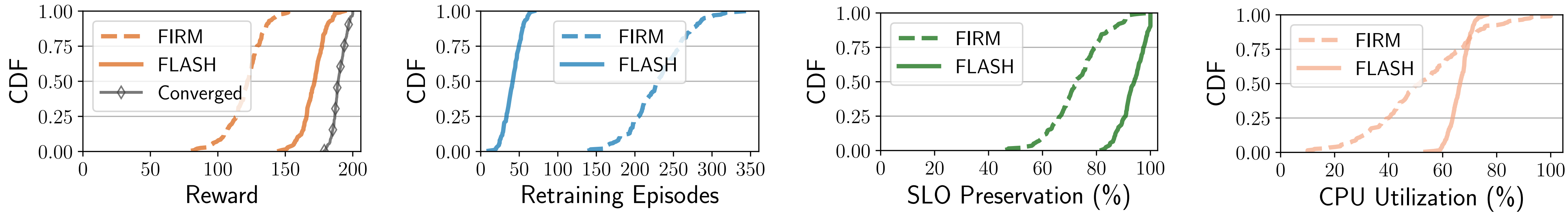




Robustness to application/environment variability

- FLASH adapts 5.5× faster than transfer learning
  - TL: Transfer learning with parameter sharing
  - TL+: TL + additional application fingerprints
- FLASH saves 68–72% CPU cycles
- FLASH reduces CPU and memory utilization deficit by 4.6× and 6.2×
- FLASH reduces SLO violations by 7.1×





Summary: A Foundation Model recipe in ML for Sys

Summary

- Meta-learning for fast model adaptation across ***applications*** and ***environments***
- Missing element prediction in state-decision trajectories for generalization across ***tasks***

Next?

- **Model size and complexity**
  - Larger models have larger capability + better generalizability
  - Higher training / fine-tuning cost and inference overhead -> detrimental for real-time tasks
- **Trade-offs between generalizability and heterogeneity**
  - Generalizability across both (1) cloud applications and environments and (2) systems tasks while still allowing the model to capture the heterogeneity of the various systems in a task
- **Risk of homogenization and bias**
  - Foundation (shared) models are singular points of failure that can radiate harm (e.g., security risks or biases) to downstream applications/tasks at scale