Online Adaptive Resource Allocation for Dynamic Stream Processing in Cloud Environments using Deep Reinforcement Learning

IEEE International Conference on Communications (ICC)
(24 - 28, May 2026, Glasgow, Scotland, UK)

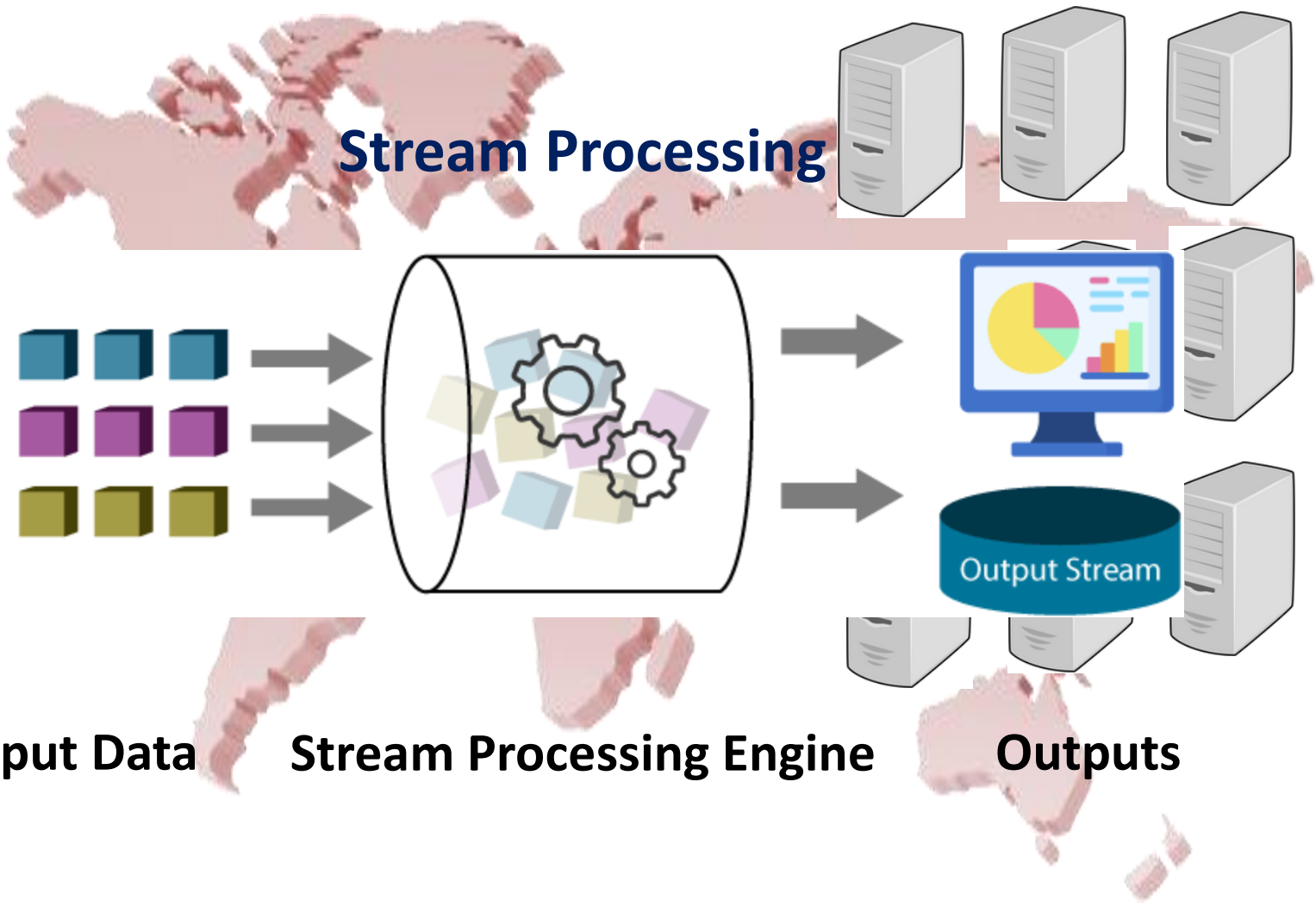
Jinwei Liu*, Sunday J. Awine*, Rui Gong†

*Dept. of Computer and Information Sciences, Florida A&M University, Tallahassee, FL, USA
†Dept. of Informatics and Mathematics, Mercer University, Macon, GA, USA

Talk Outline

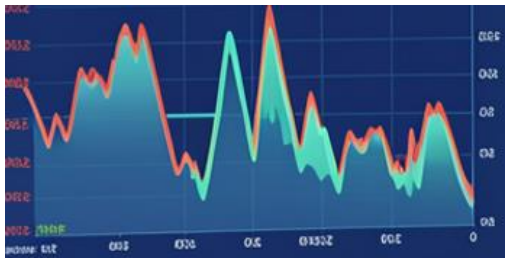
- **Introduction**
- System Model and Problem Formulation
- Design of H-MADRL
- Performance Evaluation
- Conclusions and Future Work

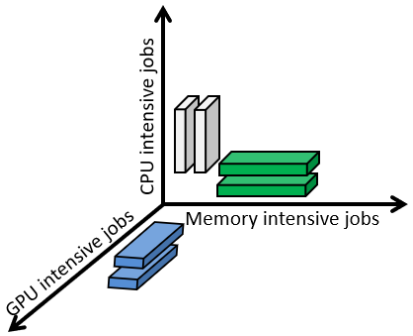
Introduction



Motivation

- Dynamic workload challenges
  - Dynamic stream processing faces fluctuating input rates, heterogeneous workloads, and constrained resources





- Resource allocation issues
  - Static/threshold autoscaling leads to under/over-provisioning, SLA violations, and latency spikes

- Need for intelligent solutions
  - Need: Online, topology-aware, proactive resource allocation that balances throughput, latency, and cost in real time

Research Challenges

- Multi-tenancy and node heterogeneity
  - Resource contention
  - Node heterogeneity
  - Isolation requirements
- Evolving DAG topologies
  - Dynamic operator graphs
  - Shifting bottlenecks
  - Dependency complexity
- Bursty and deceptive workloads
  - Bursty traffic
  - Deceptive patterns

This Research

- **Research Objective:** develop a Hierarchical Multi-Agent Deep Reinforcement Learning (H-MADRL) framework that integrates PPO agents with Kubernetes orchestration and GNN-based topology-aware scaling to achieve autonomous, data-driven adaptation for dynamic stream processing in cloud environments

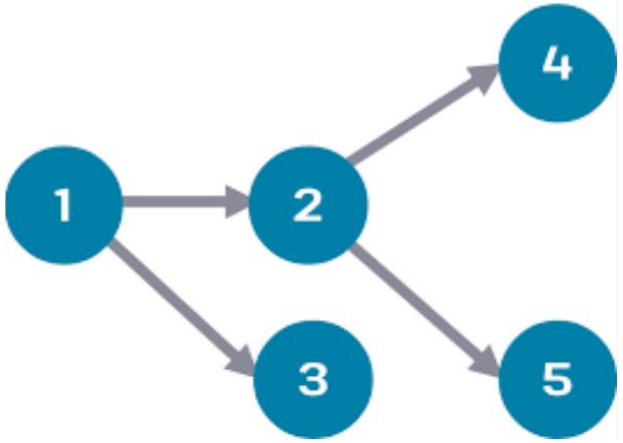
- **Key Contributions**
  - H-MADRL framework
  - Topology-aware scaling
  - Proactive autoscaling
  - Multi-objective reward
  - Cloud-native evaluation

Talk Outline

- Introduction
- **System Model and Problem Formulation**
- Design of H-MADRL
- Performance Evaluation
- Conclusions and Future Work

System Model

- **Stream topology model:** model the stream processing topology as a DAG $G = (V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ denotes the set of streaming operators and $E \subseteq V \times V$ represents the data dependencies between them. Each operator v_i processes incoming tuples at a rate $r_i(t)$ at time t. The system maintains the following key metrics:
  - C_t: number of TaskManager replicas (resource cost component)
  - $\mathcal{U}_t$: average CPU utilization
  - $\mathcal{L}_t$: average processing latency
  - $\mathcal{T}_t$: normalized throughput



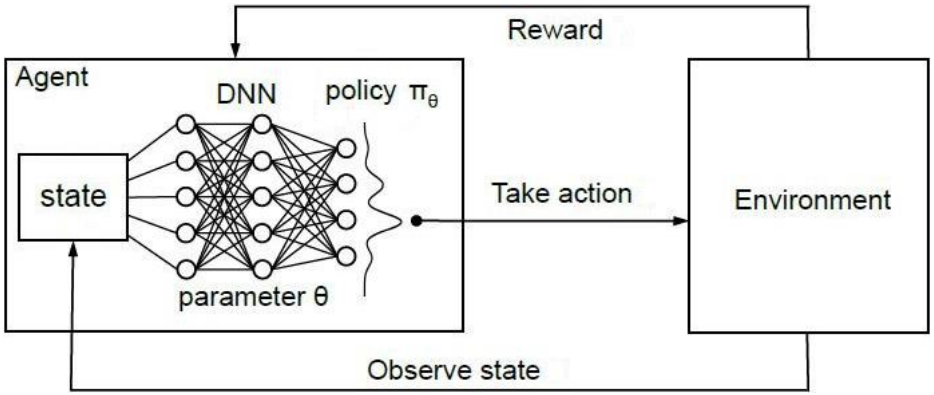
Directed Acyclic Graph (DAG)

Problem Formulation

- **Problem formulation:** The PPO (Proximal Policy Optimization) agent formulates resource allocation as a sequential decision-making problem with a multi-objective reward that balances throughput, latency, and cost. The agent's objective is to maximize the multi-objective reward.
  - The reward function at time t is defined as

$$\mathcal{R}_t = \eta_T \times \mathcal{T}_t - \eta_L \times \mathcal{L}_t - \eta_C \times C_t, \qquad (1)$$

    where η_T, η_L, and η_C are tunable weighting coefficients that determine the relative importance of each objective.
  - State and action
    - $s_t = [\mathrm{CPU}_t, \mathrm{Latency}_t, \mathrm{Throughput}_t, \mathrm{Backlog}_t, \mathrm{Replicas}_t]$
    - $a_t \in$ {scale-in, hold, scale-out}



PPO-Based Controller

- **PPO-based autoscaling policy for global controller and heuristic CPU-threshold autoscaling**

Algorithm 1: PPO-Based Autoscaling Policy for Global Controller

Input: System state
$s_t = [\text{CPU}_t, \text{Latency}_t, \text{Throughput}_t, \text{Backlog}_t, \text{Replicas}_t]$;
Action space $a_t \in \{\text{scale-in}, \text{hold}, \text{scale-out}\}$;
Reward $\mathcal{R}_t = \eta_T \mathcal{T}_t - \eta_L \mathcal{L}_t - \eta_C \mathcal{C}_t$ Eq. (1);
PPO clipping factor ϵ, discount γ, GAE parameter λ.
Output: Updated policy parameters θ, value parameters ϕ.

Initialize policy π_θ and value network V_ϕ
for *each training episode* $k = 1 \dots K$ **do**
  Collect rollout trajectories $\tau = \{(s_t, a_t, \mathcal{R}_t, s_{t+1})\}_{t=1}^{T}$
  Compute $\mathcal{R}_t$ via Eq. (1)
  Estimate advantages with GAE:

$$\hat{A}_t = \sum_{i=0}^{\infty} (\gamma\lambda)^i [\mathcal{R}_{t+i} + \gamma V_\phi(s_{t+i+1}) - V_\phi(s_{t+i})]$$

  Compute PPO loss:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

  where $r_t(\theta) = \pi_\theta(a_t|s_t) / \pi_{\theta_{\text{old}}}(a_t|s_t)$
  Update θ, ϕ via gradient ascent/descent
  Synchronize global and local agents
end
Deploy final policy π_{θ^*} to Kubernetes autoscaler

Algorithm 2: Heuristic CPU-Threshold Autoscaling Baseline

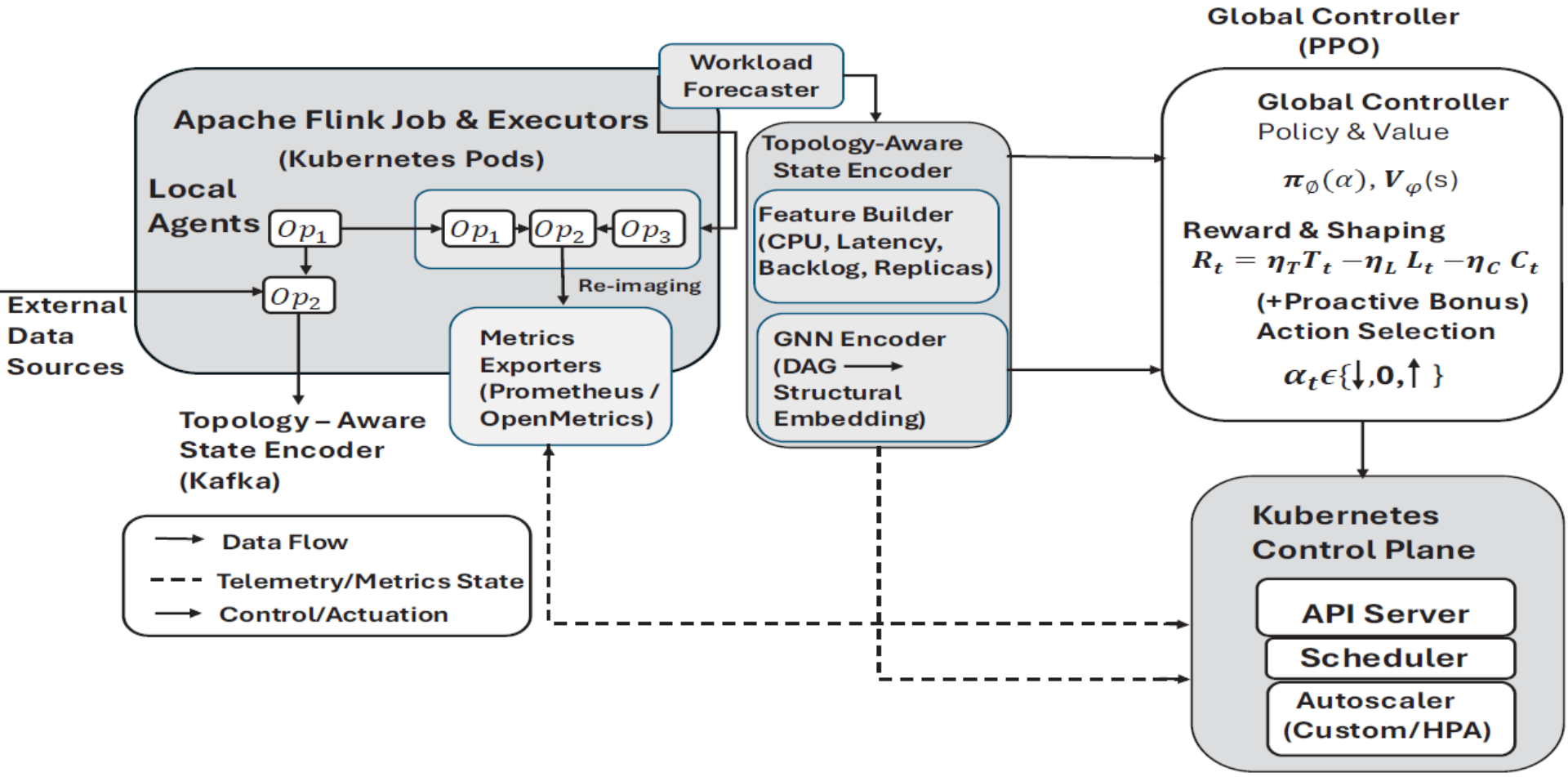
Input: CPU utilization u_t per TaskManager; thresholds $\theta_\uparrow$, $\theta_\downarrow$; Monitoring interval Δt (seconds).

while *system is running* **do**
  Retrieve u_t via Prometheus API
  if $u_t > \theta_\uparrow$ *and replicas* $< R_{\max}$ **then**
    `kubectl scale deployment --replicas = replicas + 1`
  else if $u_t < \theta_\downarrow$ *and replicas* > 1 **then**
    `kubectl scale deployment --replicas = replicas - 1`
  else
    Maintain current replica count
  end
  Sleep Δt seconds
end

Talk Outline

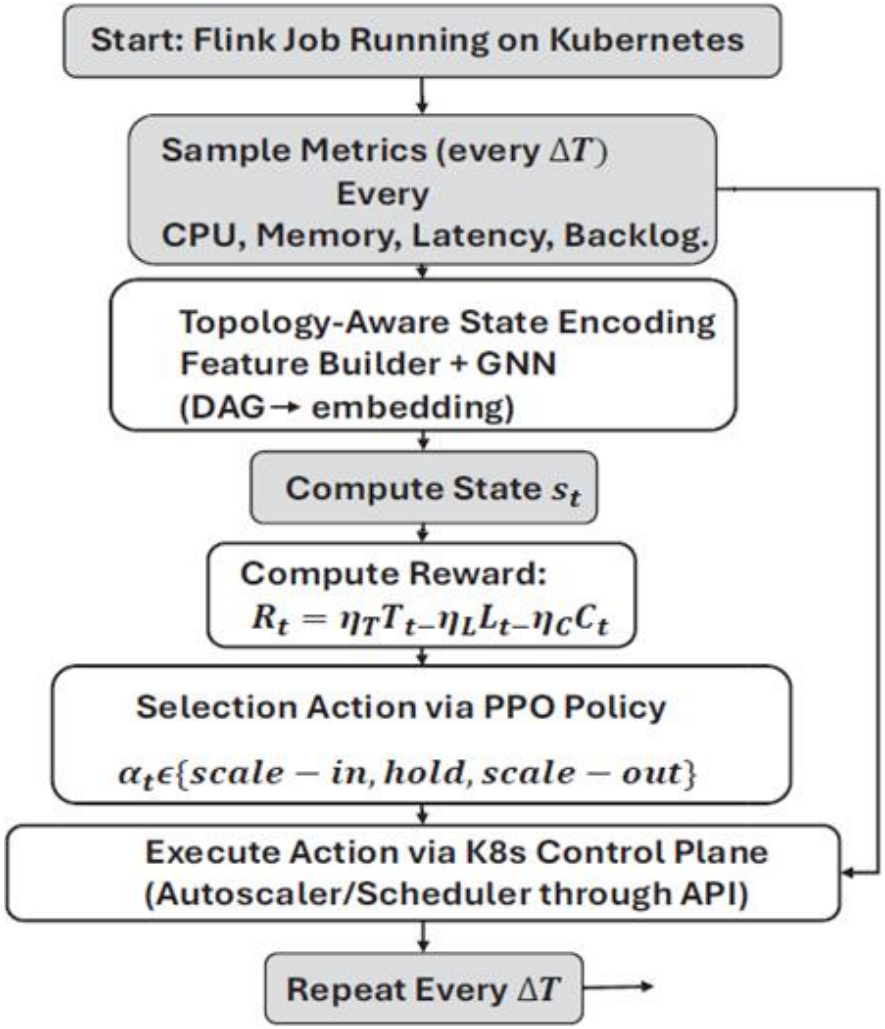
- Introduction
- System Model and Problem Formulation
- **Design of H-MADRL**
- Performance Evaluation
- Conclusions and Future Work

Design of H-MADRL



System architecture of the H-MADRL autoscaling framework. Operator telemetry is fused with topology via a GNN encoder and augmented by a Transformer forecaster. The global PPO controller optimizes the multi-objective reward in Eq. (1) and issues scaling actions via the Kubernetes control plane

Design of H-MADRL (cont.)



DRL-based autoscaling flowchart. At each interval, the PPO agent observes s_t, computes the reward $\mathcal{R}_t$, selects an action, and executes the decision via the Kubernetes control plane

Talk Outline

- Introduction
- Problem Statement
- Proposed Approach: H-MADRL
- Design of H-MADRL
- **Performance Evaluation**
- Conclusions and Future Work

Performance Evaluation

- **Methods for comparison**
  - **Traditional static provisioning (Static)**: a traditional static resource allocation approach with fixed replicas; ignores workload variability
  - **CPU-threshold heuristic (Heuristic):** this baseline scales up or down based on fixed utilization thresholds, without considering queue delays or topology dynamics. While computationally lightweight, this approach often lags in response to bursty workloads or multidimensional Bottlenecks
- **Comparative Analysis of Autoscaling Approaches**

Approach	Topology Awareness	Proactivity	Coordination	Overhead	Key Characteristics
Static	No	Reactive	None	Low	Fixed replicas; ignores workload variability; no adaptation
Heuristic	Partial	Reactive	Basic	Medium	CPU thresholds; lightweight but myopic; limited adaptation
DRL (Ours)	Full (GNN)	Proactive	Coordinated	Medium	PPO + GNN + Transformer; topology-aware; adaptive

Experimental Setup

- **Experimental configuration**

Component	Configuration	Description
Platform	Kubernetes + Apache Flink	Cloud-native stream processing with container orchestration
Workload	Dynamic/bursty streams (e.g., 1,000-5,000 events/sec)	Simulated and live workloads with varying intensity
Agents	GNN-DRL (control), MLP-DRL (ablated)	Proactive vs Reactive scaling strategies
Baselines	Static, CPU-threshold heuristic	Traditional static resource allocation approach and a deterministic CPU threshold heuristic
Evaluation	Latency, throughput, cost, variance	Multi-objective performance assessment

Evaluation of H-MADRL

- **GNN and proactive policy validation**
  - **GNN ablation**

TABLE I: Comparative Statistical Summary of Agent Performance Under Dynamic Topologies.

Group	Mean Latency	Std. Dev.	Mean Throughput	Std. Dev.	Mean Cost
GNN-DRL (Control)	0.0286	0.0047	0.3494	0.0340	1.000
MLP-DRL (Ablated)	0.0374	0.0060	0.2972	0.0399	0.7917

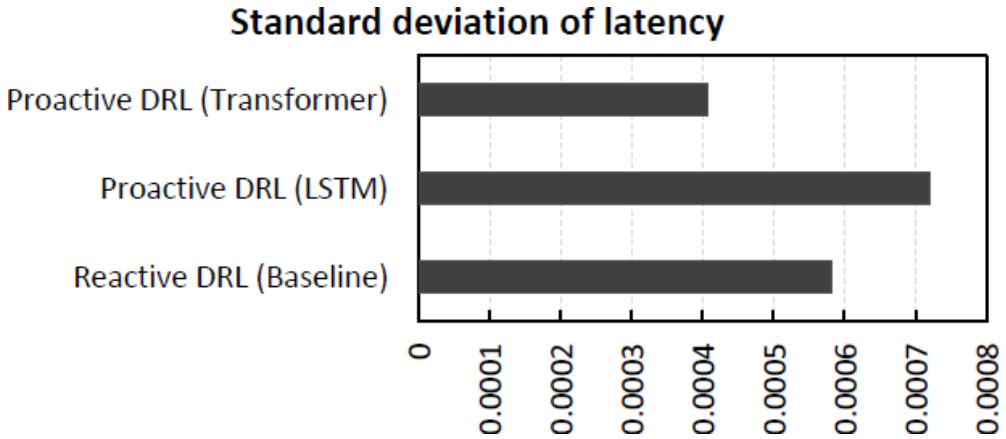
Result: The GNN-DRL agent achieved a 23.5% reduction in mean latency and a 17.7% increase in throughput; this improvement is especially significant under dynamic workloads where operator relationships shift over time

Evaluation of H-MADRL (cont.)

- **GNN and proactive policy validation**
  - **Proactive scaling with Transformer**

TABLE II: Proactive Action Count and Policy Validation

Agent	Mean Latency	Mean Throughput	Mean Cost	Std. Latency	Proactive Actions
Reactive DRL (Baseline)	0.362873	0.998592	0.998	0.000582042	0
Proactive DRL (LSTM)	0.363865	0.995697	0.994	0.000720055	35
Proactive DRL (Transformer)	0.362625	0.998588	0.998	0.000408169	37



Quantifying QoS Stability: Transformer agent achieves 30% lower σ_L than Reactive Baseline

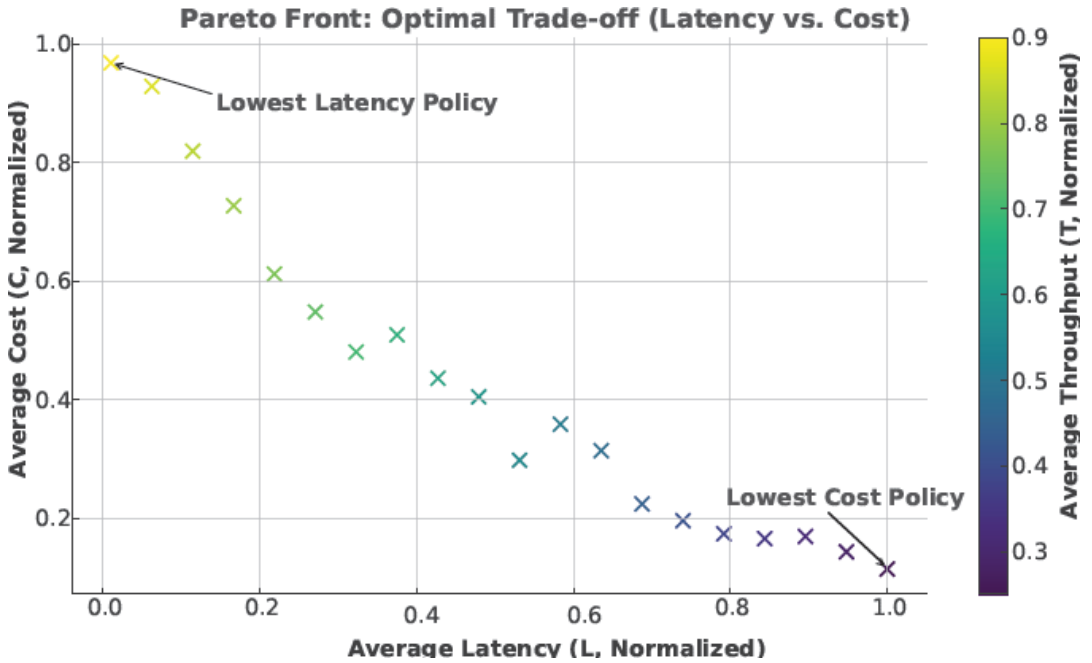
Result: Transformer architecture outperforms LSTM due to superior ability to capture complex, multi-periodic workload dependencies; the Transformer-augmented DRL agent achieves the lowest latency variance, representing a 30% reduction in volatility compared to the Reactive Baseline

Evaluation of H-MADRL (cont.)

- **Multi-objective reward optimization**

TABLE III: Pareto-optimal policies (Trade-offs).

Trial	η_T	η_L	η_C	$\mathcal{T}_t$	$\mathcal{L}_t$	$\mathcal{C}_t$
39	0.8177	0.9701	0.1154	0.9416	0.2312	0.6133
27	0.7099	0.6548	0.3592	0.9388	0.3105	0.5186
12	0.3542	0.4491	0.9818	0.8997	0.3801	0.4355

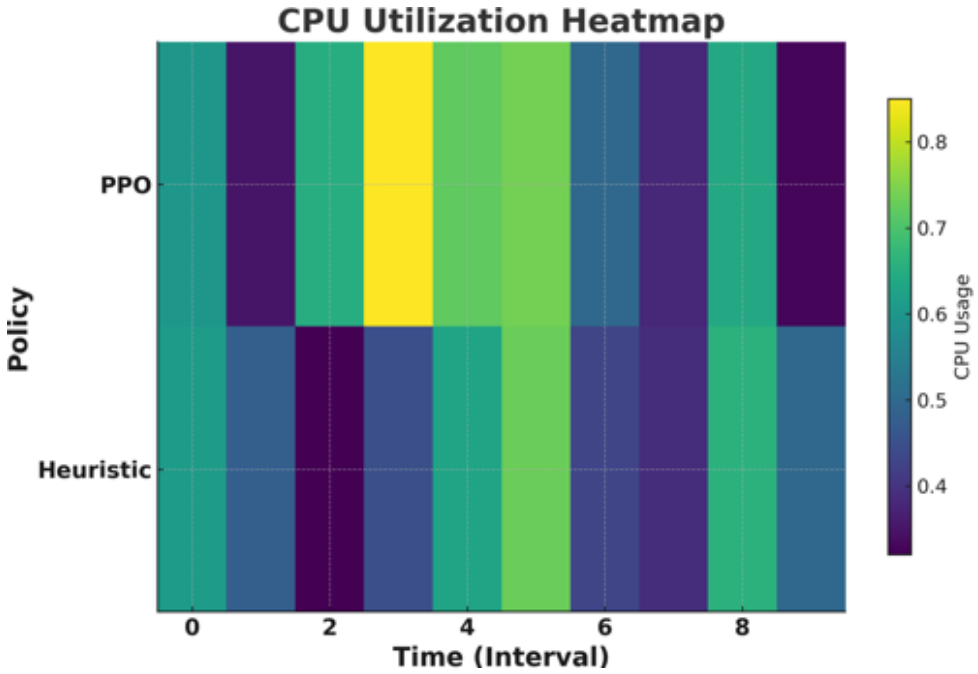


Pareto front: optimal trade-off latency vs. cost.

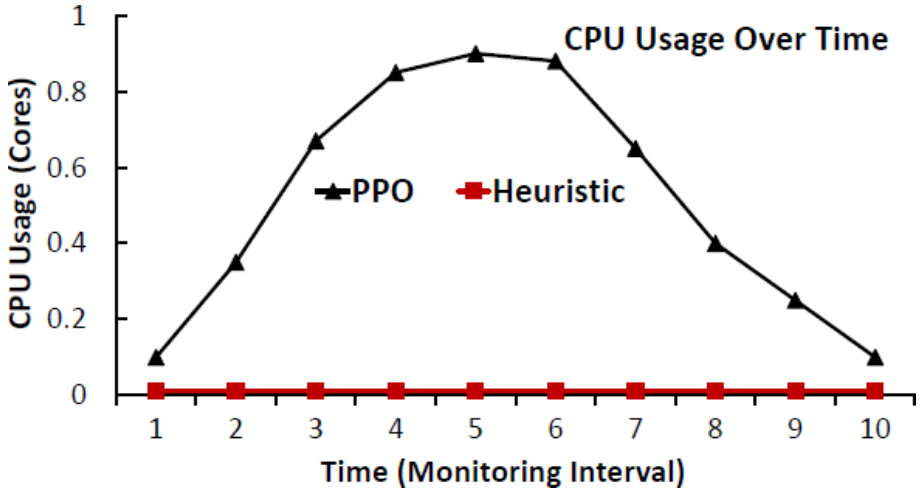
Result: DRL can be precisely tuned; Trial 39 (η_L high) achieves minimum latency at the highest cost; Trial 12 (η_C high) achieves minimum cost with slightly higher latency

Evaluation of H-MADRL (cont.)

- **Adaptive Scaling and Utilization Comparison**



(a) Heatmap showing CPU Utilization under PPO-based DRL and heuristic scheduling approaches

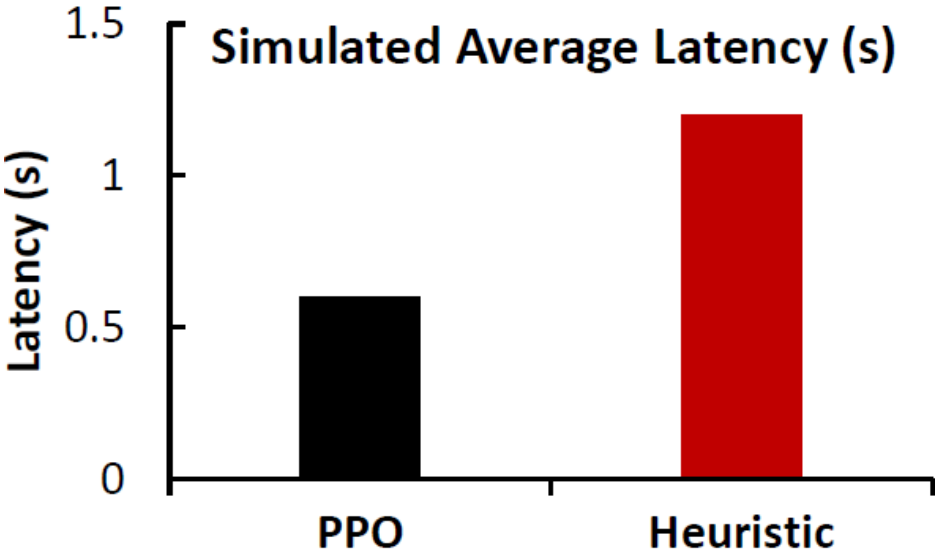


(b) Comparison of CPU usage trends over time for PPO-based vs heuristic-based scheduling strategies

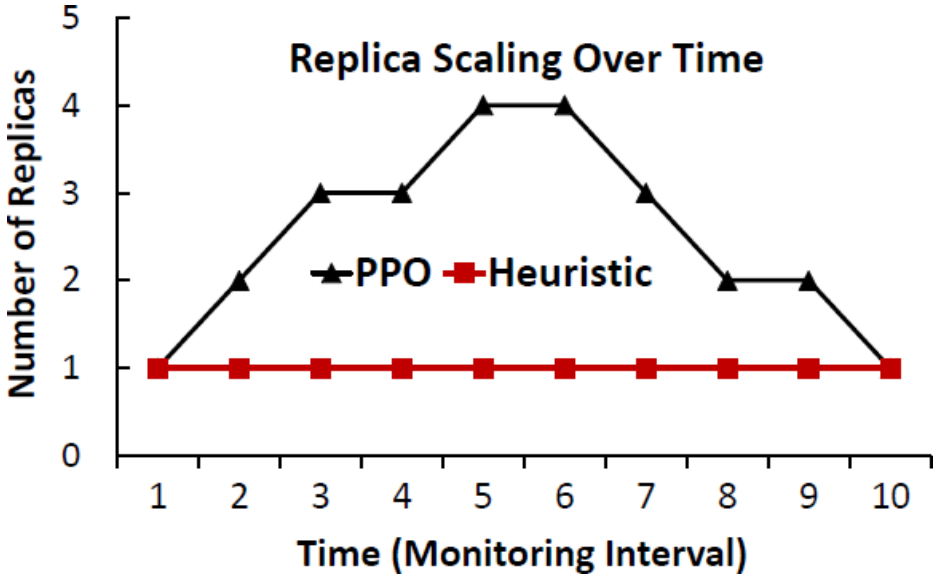
Result: Heuristic method consistently underutilizes resources (low CPU usage) and fails to detect workload pressure; the PPO model actively manages CPU engagement, demonstrating greater workload awareness

Evaluation of H-MADRL (cont.)

- **Adaptive Scaling and Utilization Comparison**



(a) Simulated average latency under PPO-based DRL vs heuristic-based scheduling strategies

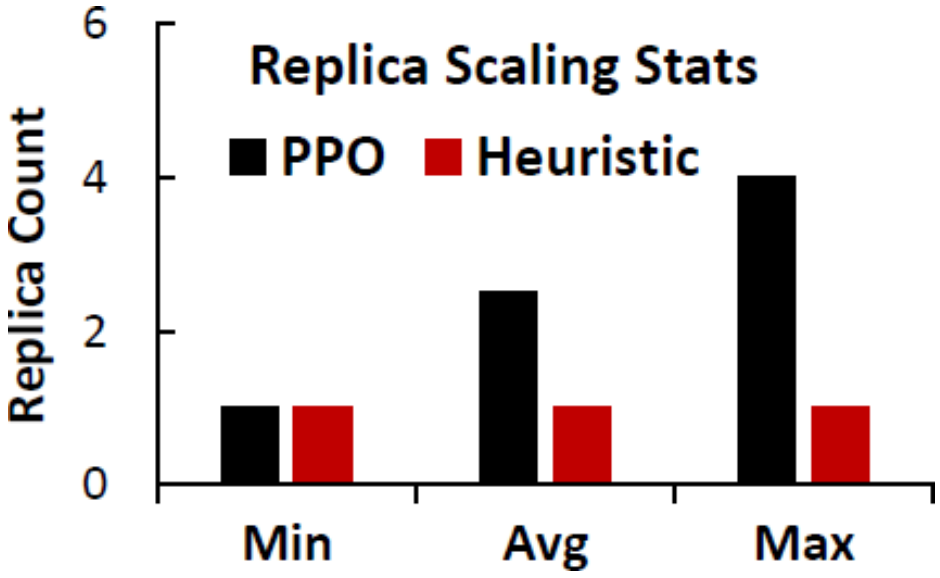


(b) Adaptive scaling behavior over time between DRL and heuristic methods

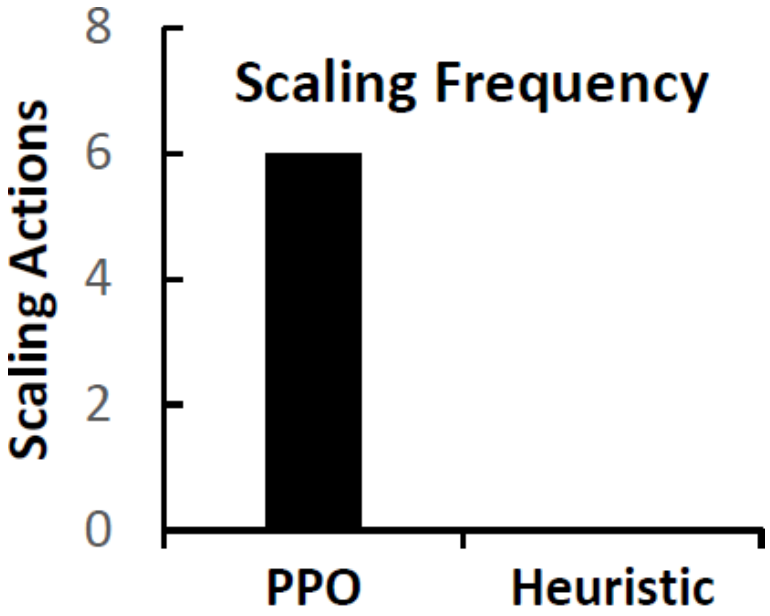
Result: PPO model achieves significantly lower average latency; the DRL agent scales replicas dynamically in response to stream load, while the heuristic remains static

Evaluation of H-MADRL (cont.)

- **Adaptive Scaling and Utilization Comparison**



(a) Minimum, average and maximum number of replicas triggered by PPO-based and heuristic models



(b) Total Scaling actions recorded for PPO vs heuristic approaches

Result: The PPO agent demonstrates adaptive scaling (range 1 to 4, $\bar{C}$ = 2.5), while the heuristic remains fixed at 1 replica; the PPO agent performs 6 scaling actions, confirming responsiveness, while the heuristic performs 0, indicating rigidity

Talk Outline

- Introduction
- Problem Statement
- Proposed Approach: H-MADRL
- Design of H-MADRL
- Performance Evaluation
- **Conclusions and Future Work**

Conclusions and Future Work

- **Our contributions**
  - Propose a Hierarchical Multi-Agent DRL (H-MADRL) autoscaling framework
  - Integrate PPO agents with Kubernetes orchestration and Graph Neural Networks (GNNs) for topology-aware scaling decisions
  - Integrate a Transformer-based forecasting model to enable proactive scaling, rewarding the agent for preemptive actions
  - Formulate a multi-objective reward using Bayesian/NSGA-II search to demonstrate tunable elasticity strategies
  - Conduct comprehensive cloud-native evaluation on Kubernetes + Flink

- **Future work**
  - Focus on a federated reinforcement learning extension to support knowledge sharing across multi-cloud deployments

Thank you!

Questions & Comments?

Jinwei Liu, Assistant Professor

jinwei.liu@famu.edu

Department of Computer and Information Sciences

Florida A&M University