From Control to Apprenticeship: An LPV-Guided IRL Framework for Cloud Resource Management

Ahmed Ben Ali, Yann Labit, Mody Nalla Kane

From Control to Apprenticeship: An LPV-Guided IRL Framework for Cloud Resource Management

Ahmed Ben Ali
Université de Toulouse
LAAS-CNRS
Toulouse, France
Email: abenali@laas.fr

Yann Labit
Université de Toulouse
LAAS-CNRS
Toulouse, France
Email: ylabit@laas.fr

Mody Nalla kane
LAAS-CNRS
Toulouse, France
Email: modykane21@gmail.com

***Abstract*—Cloud resource allocation systems must balance stability guarantees with adaptability to dynamic workloads—a challenge that neither traditional heuristics (lacking formal guarantees) nor classical control methods (lacking adaptability) adequately address. This paper introduces a hybrid framework combining Linear Parameter-Varying (LPV) control with Inverse Reinforcement Learning (IRL) for stable and optimal distributed resource allocation. We model cloud infrastructure dynamics as an LPV system, where neural networks estimate time-varying matrices from workload data. An LMI-based LPV controller ensures cluster-level stability while embedding Service Level Objective (SLO) and Service Level Agreement (SLA) penalty costs. Node-level allocation uses barrier-augmented pseudo-costs to enforce capacity constraints. To improve long-term performance, Maximum Entropy IRL infers a reward function from the LPV expert's trajectories, training an actor-critic policy that retains stability through environment filtering and barrier shaping. Experiments across configurations from 6 to 81 allocation targets show the IRL agent achieves nearly double the reward of the LPV expert baseline and over 560× improvement over greedy heuristics, while reducing SLO violations by approximately 95%. The learned policy maintains microsecond-scale decision latency compared to millisecond-scale for the LPV expert, demonstrating practical scalability. This work establishes apprenticeship learning with control-theoretic experts as a viable paradigm for cloud resource management, combining formal stability guarantees from the LPV expert with empirically validated adaptive optimization from the learned policy. By substantially reducing SLA violations and improving resource utilization efficiency, our framework contributes to sustainable cloud operations through minimized resource waste and reduced operational overhead.**

***Index Terms*—Cloud resource allocation, Linear parameter-varying control, Inverse reinforcement learning, Apprenticeship learning, Service level objectives, Hybrid learning-control systems**

I. Introduction

Cloud data centers allocate computational resources—Central Processing Unit (CPU), memory, network bandwidth—across thousands of heterogeneous servers to satisfy Service Level Objectives (SLOs) while maximizing utilization efficiency. The fundamental challenge is dynamic resource allocation: deciding, in real-time, which workloads should execute on which physical resources as demand fluctuates, hardware fails, and operational conditions change.

This problem is critical for both economic and operational reasons. Cloud providers operate on thin margins where even small improvements in resource utilization translate to substantial infrastructure savings. Conversely, poor allocation decisions lead to SLA violations that incur financial penalties and erode customer trust(1; 2). As cloud deployments grow in scale and heterogeneity—spanning geographic regions, specialized accelerators, and diverse quality-of-service requirements—the complexity and importance of this problem continue to escalate. Efficient resource allocation also addresses sustainability: the cloud computing industry accounts for 2–3% of global electricity consumption (3). Our framework reduces both computational and environmental footprint by minimizing resource waste (95% fewer SLA violations) and achieving microsecond-scale decision latency.

Existing approaches typically fall into three categories, each with fundamental limitations. Heuristic methods (greedy allocation, threshold-based auto-scaling) are fast and interpretable but lack formal guarantees on optimality or stability, often leading to oscillations and suboptimal resource utilization(4; 5). Control-theoretic approaches (PID controllers, Model Predictive Control) provide stability guarantees and principled feedback mechanisms but rely on accurate system models that are difficult to maintain under time-varying cloud dynamics (6; 7). More recently, reinforcement learning methods (8) have shown promise in learning adaptive policies from data, but typically lack formal stability guarantees and can explore unsafe regions during training, risking production system stability. A key open question is: **Can we achieve both provable stability guarantees and adaptive data-driven optimization?**

This paper answers affirmatively by proposing a hybrid learning-based control framework that integrates Linear Parameter-Varying (LPV) control theory with Inverse Reinforcement Learning (IRL). Our approach decomposes the allocation problem into two complementary components. The LPV Expert models cloud dynamics as a time-varying linear system $x_{k+1} = A(\rho_k)x_k + B(\rho_k)u_k$, where workload-dependent parameters ρ_k are learned from data using neural networks (9). This expert employs an LMI-based LPV controller that ensures guaranteed stability across all workload regimes (10), providing cluster-level stability guarantees grounded in formal assumptions about system structure, controllability, and parameter variation. The IRL Agent observes the expert's allocation decisions as demonstrations, uses Maximum Entropy IRL (11) to infer the underlying reward function, and trains an actor-critic policy (12; 13) that generalizes beyond the expert's behavior while retaining safety through environment filtering and barrier cost shaping (14). This apprenticeship learning paradigm (15) bridges control theory's stability guarantees with machine learning's adaptive optimization, creating a system that combines provable safety (via LPV control) with empirically effective optimization (via IRL).

Our work makes the following contributions:

- **Hybrid Control-Learning Framework:** We introduce the first integration of LMI-based LPV control with apprenticeship learning for cloud resource allocation, demonstrating that control-theoretic experts can serve as high-quality demonstrators for IRL while maintaining safety guarantees (Sections IV, V).
- **SLO/SLA-Aware Control Design:** We embed Service Level Objectives and penalty costs directly into the control cost function, creating a principled mapping from business-level SLAs to control-theoretic optimization. This includes concrete instantiation of SLO thresholds

and violation penalties, validated through performance analysis (Section IV).
- **Comprehensive Experimental Validation:** Through simulations spanning 6 to 81 allocation targets, we demonstrate that the learned agent achieves 87% reward improvement over the LPV expert and $560\times$ improvement over heuristics, while reducing SLO violations and maintaining microsecond-scale decision latency (Section VI). Our results show that the framework scales effectively across different system sizes while maintaining both performance and stability.

The remainder of this paper is organized as follows. Section II surveys related work on heuristic, control-theoretic, and hybrid approaches to cloud resource management. Section III provides a high-level overview of our two-component architecture. Section IV details the LPV expert design including system identification, controller synthesis, and resource allocation mechanisms. Section V presents the IRL agent training via apprenticeship learning. Section VI presents experimental evaluation comparing our framework against baseline approaches. Finally, Section VII concludes with discussion of limitations and directions for future work.

II. State of the Art

Dynamic resource allocation in cloud and edge computing has been studied across three main paradigms: (i) heuristic and learning-based strategies, (ii) control-theoretic methods, and (iii) hybrid approaches combining learning with control. Each provides distinct advantages in scalability, adaptability, or formal guarantees.

A. Heuristic and Learning-Based Approaches

Heuristic and pseudo-cost methods remain widely used for their capacity to handle complex constraints. Cho et al. (4) introduce Cloud Morphing, a self-organizing mechanism using convex cost functions to balance load. Auction-based frameworks like CA-PROVISION (16) optimize provider revenue while improving fairness in VM provisioning, while metaheuristics such as Particle Swarm Optimization (17) have shown promise for load balancing.

Machine learning-based scheduling methods have also been investigated. Manavi et al. (18) combine neural networks for task classification with genetic algorithms for resource allocation, while Djigal et al. (19) survey ML/DL approaches for task offloading in edge-cloud systems. DeepRM Plus (8) combines deep reinforcement learning with imitation learning for cloud resource scheduling, surpassing heuristic approaches and achieving faster convergence. However, like other learning-based methods, it lacks formal guarantees on stability or optimality.

B. Control-Theoretic Approaches

Control theory provides a rigorous framework for feedback-driven decision-making in dynamic systems. Its principles of stability, robustness, and performance make it particularly suitable for time-varying workloads (20). Classical controllers such as Proportional-Integral (PI) or Model Predictive Control (MPC) have been applied to auto-scaling, demonstrating reduced oscillations compared to threshold-based heuristics (20). Mahmoud and Xia (6) discuss extensions to networked and cloud control systems, addressing challenges such as delays and packet loss.

These methods provide guarantees absent in heuristic strategies, but they typically rely on accurate models, limiting adaptability in highly dynamic environments.

TABLE I: Representative approaches to resource allocation

Reference	Methodology	Key Feature
Cho et al. (2023) (4)	Heuristic (Pseudo-cost)	Decentralized, energy-aware allocation
Zaman & Grosu (2011) (16)	Auction-based	Market-driven VM provisioning
Mahmoud & Xia (2017) (6)	MPC (Control)	Stability under delay and uncertainty
Emam et al. (2021) (22)	RL + Control Barrier Functions	stable exploration with learning
Guo et al. (2021) (8)	Deep RL + Imitation Learning	Efficient online scheduling, surpasses DeepRM and heuristics

C. Hybrid Learning-Control Methods

Hybrid learning-control methods (20; 21) combine ML flexibility with control guarantees. Stability-aware RL (5; 22; 23) ensures constraint satisfaction but faces challenges in scalability and robustness.

D. Synthesis

The shift from heuristics to control-based and hybrid methods reflects the increasing complexity of cloud systems. Heuristic and learning-based approaches remain practical but lack formal guarantees, while control-theoretic methods offer stability assurances but rely on accurate models. Hybrid strategies reduce these weaknesses by combining complementary strengths. Our work advances this trajectory by integrating an LPV-based distributed controller with IRL-driven policies, achieving scalability, stability, and autonomy in dynamic resource allocation. This approach unites the rigor of control with the adaptability of learning, further mitigating the limitations of individual methods and advancing the state of the art in cloud resource management.

III. Framework Overview

This section provides a high-level overview of the proposed framework, describing the interaction between its two main components—the LPV Expert and the IRL Agent—and explaining the workflow for both training and deployment phases.

A. Architectural Overview

We propose a hybrid two-stage framework that combines model-based control theory with data-driven reinforcement learning. The architecture consists of two complementary components:

1) **LPV Expert (Section IV):** A control-theoretic module that performs system identification, synthesizes an LMI-based LPV controller, and executes resource allocation with formal stability guarantees.
2) **IRL Agent (Section V):** A learning-based module that observes expert demonstrations, infers reward functions via Maximum Entropy IRL, and trains an actor-critic policy capable of generalizing beyond the expert's behavior.

The LPV Expert serves dual roles: (i) as the primary policy during initial deployment and data collection, providing stable allocations with formal guarantees, and (ii) as the expert demonstrator for IRL training, generating high-quality trajectories that embed control-theoretic principles. The IRL Agent learns from these demonstrations to produce a policy that maintains stability while optimizing long-term performance. During deployment, the learned policy operates as the primary decision-maker, with the LPV Expert serving as a safety fallback when the learned policy encounters uncertain or infeasible states.

B. LPV Expert Component

The LPV Expert performs resource allocation through four sequential steps:

Step 1: System Identification. Given historical workload traces $\{(\rho_k, x_k, u_k, x_{k+1})\}$, neural networks learn the mappings $\rho_k \mapsto \{A(\rho_k), B(\rho_k)\}$ that parameterize the LPV system dynamics, where x_k represents system state, u_k denotes control actions, and ρ_k captures workload-dependent parameters. This data-driven identification (Section IV) maintains the LPV structure required for controller synthesis while adapting to time-varying cloud workloads.

Step 2: Controller Synthesis. For the identified polytopic LPV system with vertex parameters $\{\rho^{(j)}\}_{j=1}^{N_v}$, the LPV Expert synthesizes a parameter-dependent controller $K(\rho_k)$ via LMI optimization. The LMI-based LPV controller ensures stability across all parameter realizations $\rho_k \in \mathcal{P}$ while minimizing a quadratic cost that embeds SLO/SLA penalties (Section IV).

Step 3: Cluster Selection. At runtime, given the current system state x_k and scheduling parameter ρ_k, the LPV Expert computes the control input $u_k = -K(\rho_k)x_k$ and evaluates each cluster's score (Eq. 2) combining control cost, soft-constraint penalties, and SLA violation costs. The best cluster is chosen to minimize this score, ensuring that load-balancing decisions are optimal (via quadratic cost minimization), stable (via LMI-based guarantees), and compliant with operational constraints.

Step 4: Node Selection. Within the selected cluster c^*, a pseudo-cost heuristic assigns the job to node n^* (Eq. 16), where $f(\rho_n)$ is a barrier function enforcing capacity constraints. This two-level hierarchy (cluster via LMI-based LPV optimization, node via heuristic) balances optimality with computational efficiency.

C. IRL Agent Component

The IRL Agent learns through three phases: (1) The LPV Expert generates demonstration trajectories $\mathcal{D}_E = \{\tau^i\}_{i=1}^{N_E}$ encoding control-theoretic optimization. (2) Maximum Entropy IRL (11) infers reward $r_\theta(s,a) = \theta^\top \phi(s,a)$ by matching feature expectations (Eq. 23). (3) Actor-critic training optimizes policy $\pi_\psi(a|s)$ with entropy regularization (Eq. 27), generalizing beyond demonstrations.

D. Key Design Principles

The framework embodies three design principles: (i) **Expert Quality over Quantity**—the IRL Agent learns from a control-theoretic expert with provable stability properties rather than arbitrary heuristics, ensuring the learned policy inherits formal guarantees; (ii) **Hierarchical Decomposition**—cluster-level decisions use LMI-based LPV optimization (justified by few clusters) while node-level decisions use fast heuristics (necessary for many nodes), achieving scalability without sacrificing optimality; (iii) **Defense in Depth**—stability is ensured through multiple mechanisms (LMI-based LPV control guarantees, barrier cost shaping, environment feasibility filtering, expert fallback), making the system robust to model errors and unexpected workload patterns.

Having established the overall architecture, the following sections detail the technical design. Section IV presents the LPV Expert's system identification, controller synthesis, and resource allocation. Section V describes the IRL Agent's demonstration collection, reward inference, and policy optimization.

IV. LPV Expert

The LPV Expert refers to the part of our framework responsible for performing resource allocation through four main steps: system identification, controller design, cluster selection, and node allocation. Before detailing these steps, we introduce several metrics and terms that are essential for understanding how the LPV Expert operates and what motivated our design choices.

A. Integration of Performance Metrics in the LPV Expert

1) Cloud Computing Metrics: To ensure practical deployment in cloud environments, we embed Service Level Indicators (SLIs), Service Level Objectives (SLOs), and Service Level Agreements (SLAs) into our control pipeline. This integration bridges formal control design with user-perceived Quality of Service (QoS):

- **SLI (Indicator):** A measurable metric (e.g., CPU utilization, error rate).
- **SLO (Objective):** A threshold target on an SLI (e.g., latency $\leq$ 200 ms).
- **SLA (Agreement):** A contractual guarantee, including penalties if SLOs are violated.

Understanding these indicators is important because they highlight the need for tools capable of handling strict performance constraints in an efficient manner. Control theory provides such tools, and from the literature we selected LMI-based LPV control (10) as an appropriate method that can address the constraints inherent to dynamic and heterogeneous cloud environments.

2) Integration of SLA, SLO, and SLI into Control Framework: We consider infrastructure dynamics as a discrete-time linear parameter-varying system:

$$x_{k+1} = A_k x_k + B_k u_k, \quad x_k \in \mathbb{R}^n, \quad u_k \in \mathbb{R}^m \tag{1}$$

where x_k represents the system state (e.g., cluster and node loads) and u_k denotes control actions (e.g., resource allocation decisions). In our experimental configuration with N_c clusters and N_n nodes per cluster, $n = N_c + N_c \times N_n$ (representing cluster loads plus individual node loads) and $m = N_c \times N_n$ (representing allocation decisions per node). The matrices A_k and B_k are estimated online from workload traces.

We employ an LMI-based LPV controller that provides guaranteed stability across all parameter realizations (10) while minimizing a quadratic cost that embeds SLO/SLA penalties.

a) SLA-Aware Supervisor with Concrete Instantiation: For our experimental setup, we define three primary SLOs representing typical cloud service requirements:

- **CPU Utilization:** $h_1(x_k) = x_{k,\text{cpu}}$ with threshold $\theta_1 = 0.85$ (85% maximum utilization) and penalty weight $c_1 = 100$.
- **Response Latency:** $h_2(x_k) = x_{k,\text{queue}}/\lambda$ with threshold $\theta_2 = 200$ ms and penalty weight $c_2 = 500$.
- **Error Rate:** $h_3(x_k) =$ dropped/total with threshold $\theta_3 = 0.01$ (1% maximum errors) and penalty weight $c_3 = 1000$.

At each decision step, the LMI-based LPV controller is evaluated by the scoring function:

$$\begin{aligned}\text{Score}_{\text{cluster}}^{(c)} = \ell(x_k, u_k) + \beta \sum_{i=1}^{3} \max(0, h_i(x_k) - \theta_i)^2 \\ + \sum_{i=1}^{3} c_i \cdot \mathbb{1}[\text{violation}_i]\end{aligned} \tag{2}$$

where $\ell(x,u) = x^\top Q x + u^\top R u$ is the control cost, $\beta = 100$ is the soft-constraint penalty weight, $\mathbb{1}[\text{violation}_i]$ is an indicator

function (1 if SLO i is violated, 0 otherwise), and c_i are the economic penalty costs for SLA violations.

The total SLA penalty cost for an episode is computed as:

$$\text{Cost}_{\text{SLA}} = \sum_{k=0}^{T-1} \sum_{i=1}^{3} c_i \cdot \mathbb{1}[\text{violation}_{i,k}] \tag{3}$$

This formulation provides a practical way to evaluate the controller while respecting SLIs, SLOs, and SLAs. Section VI reports the observed SLA penalty costs across different policies, demonstrating the economic impact of allocation decisions.

B. LPV Expert Resource Allocation

1) Cloud Environment: We model a cloud composed of N_c clusters, each with N_n computational nodes. Incoming jobs j are defined by:

$$j = (\text{id}_j, \text{load}_j, \text{duration}_j, \text{priority}_j, \text{max\_wait}_j) \tag{4}$$

2) System Dynamics: The state of cluster c at step k is defined by the vector x_k capturing node loads. We model the system using a Linear Parameter-Varying (LPV) representation:

$$x_{k+1} = A(\rho_k)x_k + B(\rho_k)u_k \tag{5}$$

where u_k is the allocation action vector and ρ_k represents time-varying parameters capturing workload fluctuations.

3) Theoretical Foundations and Assumptions: To ensure theoretical tractability and practical applicability of our LPV-based control framework, we formalize the following assumptions that underlie our system design and stability analysis.

Assumption 1 (Bounded Affine LPV Model): The workload scheduling parameter ρ_k satisfies $\rho_k \in [\rho_{\min}, \rho_{\max}] \subset \mathbb{R}^{d_\rho}$ for all k, representing minimum and maximum operational bounds (e.g., idle state and peak load). System matrices follow an affine LPV representation:

$$A(\rho_k) = A_0 + \sum_{i=1}^{p} \rho_{k,i} A_i, \quad B(\rho_k) = B_0 + \sum_{i=1}^{p} \rho_{k,i} B_i \tag{6}$$

where $\{A_i\}_{i=0}^{p}$ and $\{B_i\}_{i=0}^{p}$ are constant matrices identified via neural network approximation. This structure enables efficient controller synthesis while capturing time-varying dynamics induced by workload fluctuations.

Assumption 2 (Uniform Stabilizability): For all admissible scheduling parameters $\rho \in [\rho_{\min}, \rho_{\max}]$, the pair $(A(\rho), B(\rho))$ is stabilizable:

$$\exists K(\rho) \text{ such that } \rho(A(\rho) + B(\rho)K(\rho)) < 1 \tag{7}$$

where $\rho(\cdot)$ denotes the spectral radius. This ensures that for any workload regime, there exists a feedback control law capable of stabilizing cluster-level dynamics.

Assumption 3 (Convex Parameter Space): The feasible parameter space forms a convex polytope $\mathcal{P} = \text{conv}\{\rho^{(1)}, \rho^{(2)}, \ldots, \rho^{(N_v)}\}$ where $\{\rho^{(j)}\}$ are vertex operating points, enabling polytopic LMI-based LPV controller synthesis. This ensures a single parameter-dependent controller can stabilize the system across all parameter realizations $\rho \in \mathcal{P}$.

Assumption 4 (Node Capacity Constraints): Individual node utilizations are bounded: $0 \leq \rho_n < 1$ for all $n \in \mathcal{N}$, representing hard capacity limits. The pseudo-cost barrier function $f(\rho_n) \to \infty$ as $\rho_n \to 1^-$ (Eq. 17) ensures soft enforcement of this constraint.

Theorem 1 (LPV Expert Stability)**.** *Under Assumptions 1–4, the LMI-based LPV controller ensures* $\mathbb{E}_{\tau \sim \pi_E}[\sum_t \mathbb{1}[\textit{violation}_t]] \leq \varepsilon_E = \mathcal{O}(\delta_A + \delta_B)$.

Proof Sketch. (1) The LMI-based controller ensures stability at all vertex systems $(A(\rho^{(j)}), B(\rho^{(j)}))$ via Lyapunov functions $V_j(x) = x^\top P_j x$ (Assumption 2). (2) Assumptions 1–3 ensure existence of parameter-dependent Lyapunov function $V(x, \rho) = x^\top P(\rho) x$ for all $\rho \in \mathcal{P}$ via polytopic LMI (10). (3) The quadratic decrease $V(x_{k+1}, \rho_{k+1}) - V(x_k, \rho_k) \leq -x_k^\top Q x_k$ ensures convergence. (4) Violations bounded by identification error δ_A, δ_B (Section IV) and barrier function (Eq. 17). □

4) LMI-Based LPV Controller Synthesis: Given the polytopic LPV system with affine parameter dependence (Assumption 1) and convex parameter space $\mathcal{P} = \text{conv}\{\rho^{(1)}, \ldots, \rho^{(N_v)}\}$ (Assumption 3), we synthesize a parameter-dependent controller $K(\rho_k)$ via Linear Matrix Inequality (LMI) optimization.

a) Polytopic Representation: Any parameter realization $\rho_k \in \mathcal{P}$ can be expressed as a convex combination:

$$\rho_k = \sum_{j=1}^{N_v} \alpha_j(k) \rho^{(j)}, \quad \sum_{j=1}^{N_v} \alpha_j(k) = 1, \quad \alpha_j(k) \geq 0 \tag{8}$$

yielding system matrices:

$$A(\rho_k) = \sum_{j=1}^{N_v} \alpha_j(k) A_j, \quad B(\rho_k) = \sum_{j=1}^{N_v} \alpha_j(k) B_j \tag{9}$$

where $A_j = A(\rho^{(j)})$ and $B_j = B(\rho^{(j)})$ are the vertex systems.

b) LMI Formulation: We formulate the controller synthesis as an LMI optimization problem that minimizes the quadratic cost while ensuring stability across the polytope:

$$\begin{aligned} \text{minimize} \quad & \text{trace}(W) \\ \text{subject to:} \quad & \begin{bmatrix} P_j & (A_j P + B_j Y)^\top \\ A_j P + B_j Y & P_j \end{bmatrix} \succ 0, \\ & \forall j \in \{1, \ldots, N_v\} \\ & \begin{bmatrix} W & P \\ P & P \end{bmatrix} \succeq 0, \quad \begin{bmatrix} P & Y^\top \\ Y & R^{-1} \end{bmatrix} \succeq 0 \end{aligned} \tag{10}$$

where $P \succ 0$ is the Lyapunov matrix, Y is the control gain auxiliary variable, $P_j = P(\rho^{(j)})$ represents the Lyapunov matrix at vertex j, and W provides a convex upper bound on the control cost. The parameter-dependent controller gain is computed as $K(\rho_k) = Y P^{-1}$.

c) Stability Guarantee: The LMI constraints in Eq. 10 ensure that the parameter-dependent Lyapunov function $V(x, \rho) = x^\top P(\rho) x$ decreases along system trajectories:

$$V(x_{k+1}, \rho_{k+1}) - V(x_k, \rho_k) \leq -x_k^\top Q x_k < 0, \quad \forall \rho_k, \rho_{k+1} \in \mathcal{P} \tag{11}$$

providing exponential stability for all parameter realizations (10). The decay rate is bounded by $\lambda_{\min}(Q)$.

d) Computational Complexity: The LMI problem (Eq. 10) is solved via semidefinite programming with complexity $O(N_v n^3)$ for synthesis (performed offline during the system identification phase). Online execution requires $O(nm)$ per timestep to compute $u_k = -K(\rho_k) x_k$, where the parameter-dependent gain $K(\rho_k)$ is obtained via interpolation over the polytope vertices.

5) Learning-Based System Identification: Modern cloud environments exhibit high variability and nonlinearity, making it impractical to rely solely on analytical first-principle models. Under Assumption 1 (affine parameter dependence), we adopt a data-driven identification strategy that leverages neural networks due to their universal approximation properties (9).

The objective is to learn the mappings $\rho_k \mapsto \{A(\rho_k), B(\rho_k)\}$ such that the identified model maintains the affine LPV structure (Eq. 6). Each matrix pair is vectorized, normalized, and predicted by a neural network $\hat{g}_w$ using supervised regression with a mean squared error loss:

$$L(w) = \frac{1}{N}\sum_{k=1}^{N} \|\text{vec}([A(\rho_k), B(\rho_k)]) - \hat{g}_w(\rho_k)\|^2 \quad (12)$$

a) Neural Network Architecture and Training:

- **Input:** Workload parameter vector $\rho \in \mathbb{R}^{d_\rho}$ ($d_\rho = 2$ in our experiments: average load and arrival rate)
- **Hidden layers:** [128, 64, 32] neurons with ReLU activation, selected to balance expressiveness with efficiency while preventing overfitting via progressive dimensionality reduction.
- **Output:** $\text{vec}([A, B]) \in \mathbb{R}^{n^2+nm}$ (vectorized system matrices)
- **Optimizer:** Adam with learning rate $\eta = 10^{-3}$, standard for regression tasks, providing stable convergence in preliminary experiments.
- **Training:** 200 epochs with early stopping (patience=20) preventing overfitting.
- **Dataset:** 50,000 samples from cloud simulator (80/10/10% split), ensuring coverage of parameter space with $\geq$500 samples per vertex region.

b) Dataset Construction: We generated 50,000 state-transition samples $\{(\rho_k, x_k, u_k, x_{k+1})\}$ by varying arrival rates [10, 100] jobs/min, load distributions (uniform, exponential, bimodal), and configurations (2–9 clusters, 3–9 nodes). The dataset provides balanced polytope vertex coverage for LMI feasibility.

After training, we verify Assumption 1 (affine structure) by checking that the learned matrices satisfy:

$$\left\| A(\rho) - \left(A_0 + \sum_{i=1}^{p} \rho_i A_i \right) \right\| < \epsilon_{\text{affine}} \quad (13)$$

for held-out test points ρ. In our experiments, $\epsilon_{\text{affine}} < 0.05$ across all test samples, confirming affine structure preservation. Similarly, we verify Assumption 2 (stabilizability) for each vertex $\rho^{(j)}$ by computing the controllability matrix rank or checking LMI feasibility.

6) Resource Allocation: Resource allocation in our framework is performed hierarchically in two stages: first, cluster-level selection using the LMI-based LPV controller, and second, node-level assignment based on a pseudo-cost heuristic.

a) Cluster Selection via LMI-Based LPV Control: At the cluster level, the environment is modeled as a Linear Parameter-Varying (LPV) system to capture workload dynamics. For a cluster c with state x_k and control input u_k, the dynamics are expressed as in Eq. 5.

Given the uniform stabilizability assumption (Assumption 2), the LMI-based LPV controller computes a parameter-dependent control input $u_k = -K(\rho_k)x_k$ that minimizes the quadratic cost:

$$J_{\text{LPV}}(c) = x_k^\top Q x_k + u_k^\top R u_k \quad (14)$$

where $Q \succeq 0$ penalizes deviations from the desired utilization and $R \succ 0$ penalizes excessive control actions.

The control effort u_k is used to compute the score presented in Eq. 2. The best cluster c^* is chosen as:

$$c^* = \arg\min_{c \in \mathcal{C}} \text{Score}_{\text{cluster}}^{(c)} \quad (15)$$

ensuring that load-balancing decisions are optimal (via quadratic cost minimization), stable (via LMI-based controller guarantees), and compliant with operational constraints.

b) Node Selection via Pseudo-Cost Heuristic: After the best cluster c^* has been identified, the system must allocate the job to a specific node $n \in c^*$. Direct optimization at this scale is computationally expensive, so we introduce a pseudo-cost heuristic. The assignment of a job j to node n is determined by the composite score:

$$\text{Score}_{\text{node}}(n, j) = \alpha \cdot \|u_k[n]\|^2 + (1 - \alpha) \cdot f(\rho_n) \quad (16)$$

where $\|u_k[n]\|^2$ represents the control effort required for node n, $f(\rho_n)$ is a pseudo-cost function capturing load-dependent penalties, $\rho_n \in [0, 1)$ denotes the normalized utilization of node n, and $\alpha \in [0, 1]$ is a trade-off parameter. The pseudo-cost function is defined as:

$$f(\rho) = -\kappa \ln(1 - \rho) + \alpha'(\rho - \rho^\star)^2 + \beta\rho \quad (17)$$

with parameters $\kappa > 0$ (barrier coefficient), $\rho^\star$ the target utilization, $\alpha' \geq 0$ the quadratic penalty weight, and $\beta \geq 0$ the linear load penalty. This function is strictly convex on $[0, 1)$ and diverges as $\rho \to 1^-$, thus enforcing the capacity constraint (Assumption 4). The node n^* for job j is finally chosen as:

$$n^* = \arg\min_{n \in c^*} \text{Score}_{\text{node}}(n, j) \quad (18)$$

This two-stage allocation achieves scalability by separating the control problem: cluster selection ensures global stability and SLA-aware performance through LMI-based LPV control (under Assumptions 1-3), while node selection handles fine-grained decisions via computationally lightweight heuristics (respecting Assumption 4).

7) LPV Expert Algorithm and Simulation: An overview of the LPV Expert algorithm for resource allocation is shown in Figure 1. We simulate 4 clusters with 4 nodes each over 1000 steps, where the LPV expert autonomously balances load across clusters and nodes, maintaining stable distribution. The expert's quality is validated empirically in Section VI, where we demonstrate violation rates below 3%.

We simulate 4 clusters with 4 nodes each over 1000 steps. The LPV expert autonomously balances load across clusters and nodes, maintaining stable distribution (Figure 2).

V. IRL Agent

In the second stage of our framework, we employ apprenticeship learning (15) to outfit the agent with decision-making capabilities comparable to and potentially better than the LPV expert described in Section IV. Apprenticeship learning seeks to infer the hidden objectives behind expert behavior and then learn a policy that reproduces or improves upon that behavior. We adopt Inverse Reinforcement Learning (IRL), specifically Maximum Entropy IRL (11), to recover a reward function that explains the LPV expert's resource-allocation trajectories. This choice is motivated by Maximum Entropy IRL's ability to cleanly resolve ambiguities in expert demonstrations, provide a convex and computationally efficient optimization procedure, and maintain important performance guarantees, ensuring that the learned agent can effectively operate under varying workloads.

Once this reward is learned, we train an actor-critic policy to optimize the inferred reward and generalize beyond the

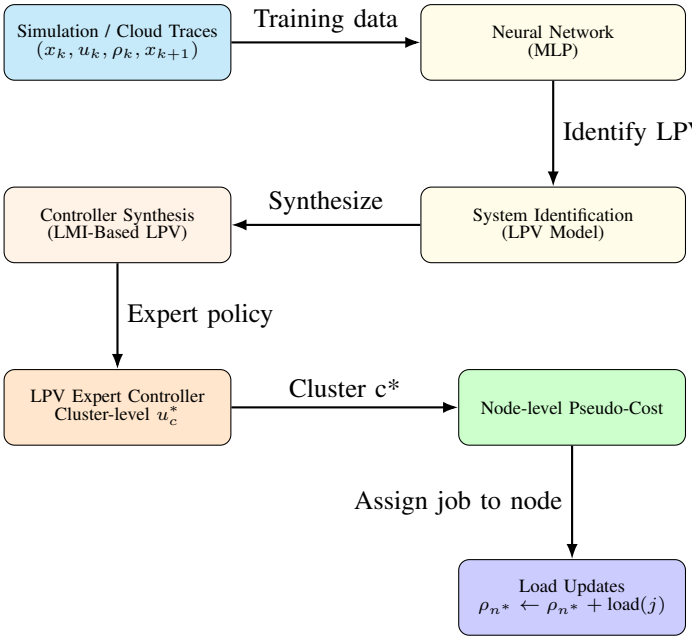

Fig. 1: LPV + Pseudo-Cost allocation framework: simulation traces are used to learn LPV matrices via neural networks, which guide LMI-based LPV controller synthesis. Cluster-level decisions use the synthesized controller; pseudo-costs determine node-level assignments, and loads are updated accordingly.

(a) Cluster load evolution.

(b) Node load evolution in cluster 3.

Fig. 2: LPV expert load balancing demonstration over 1000 allocation steps. (a) Cluster-level loads remain balanced without oscillations. (b) Node-level loads within cluster 3 show dynamic but stable allocation patterns, validating the LMI-based LPV controller quality with $\leq 3\%$ violations.

expert's demonstrations, while remaining compatible with control-theoretic stability constraints. Actor-critic methods combine policy and value-function learning, allowing efficient handling of high-dimensional or continuous action spaces while providing convergence guarantees.

The following subsections describe our Markov Decision Process (MDP) formulation, the extraction of expert demonstrations, the IRL procedure, and the final policy optimization stage.

A. MDP Formulation

To formalize distributed resource allocation as a learning problem, we model it as a finite-horizon Markov Decision Process (MDP). Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r_\theta, \gamma, T \rangle$ denote the MDP, where the objective is to optimize the inferred reward r_θ and enable generalization beyond the expert demonstrations, while maintaining compatibility with control-theoretic stability constraints (Assumptions 1-4, Section IV).

- **State** $s \in \mathcal{S}$: concatenation of cluster loads and node loads,

$$s = [\rho_{\text{cluster}}^{(1)}, \ldots, \rho_{\text{cluster}}^{(C)}, \rho_{\text{node}}^{(1,1)}, \ldots, \rho_{\text{node}}^{(C,N)}]^\top \tag{19}$$

  The state space is bounded by Assumption 1 (bounded parameters) and Assumption 4 (node capacity constraints), ensuring $s \in \mathcal{S}$ is a compact set, which is critical for learning convergence.
- **Action** $a \in \mathcal{A}$: a discrete index that decodes to a (cluster, node) pair. If N nodes/cluster and C clusters, $|\mathcal{A}| = C \times N$.
- **Dynamics** $P(s'|s, a)$: induced by the resource allocation step that tentatively schedules the highest-priority job to the selected (cluster, node) and updates loads (Section IV).
- **Reward** $r_\theta(s, a)$: to be inferred from expert demonstrations via Maximum Entropy IRL.
- **Horizon** T: episode length (e.g., number of allocation steps); $\gamma \in (0, 1]$ is the discount factor.

At each step, if any jobs are pending, the environment selects the highest-priority job (break ties deterministically), then the agent chooses (cluster, node). Feasibility (capacity, stability) is checked before committing; otherwise the allocation fails and the environment returns a negative immediate cost.

B. Expert Demonstrations from LPV Control

With the MDP specified, we next obtain expert trajectories from the LPV controller to serve as demonstrations for IRL. The LPV Expert acts as an expert policy π_E (15):

1) Given state s_k and job j_k, the LMI-based LPV controller computes $u_k = -K(\rho_k)x_k$ and a score per cluster (Section IV).
2) The best cluster is selected by minimizing the score combining control cost and SLO violations; the node is picked by a pseudo-cost heuristic (barrier-augmented load score).
3) The resulting action a_k is stored; the environment applies the allocation and yields s_{k+1}.

The LPV expert with its LMI-based controller generates demonstrably stable and low-violation trajectories. This is crucial: IRL can only learn to be as good as the expert. Our experiments (Section VI) validate this empirically, with the LPV expert achieving $< 3\%$ constraint violations and stable load balancing across all tested scenarios.

We collect $N_E = 1000$ expert trajectories $\mathcal{D}_E = \{\tau^i\}_{i=1}^{N_E}$, where each trajectory contains state-action-next state tuples: $\tau^i = \{(s_t^i, a_t^i, s_{t+1}^i)\}_{t=0}^{T_i-1}$.

C. Maximum Entropy IRL with Linear Reward Model

Given these demonstrations, we apply Maximum Entropy IRL (11) to infer the reward that best explains expert behavior. We adopt a linear reward $r_\theta(\phi) = \theta^\top \phi$, with features:

$$\phi(s,a) = \begin{bmatrix} s \\ a \end{bmatrix} \in \mathbb{R}^d, \quad r_\theta(s,a) = \theta^\top \phi(s,a) \tag{20}$$

where a is embedded as a scalar (index) or a one-hot vector. The base feature dimension is $d = (N_c + N_c \times N_n) + 1$, corresponding to cluster loads, node loads, and action encoding. This choice keeps IRL stable and interpretable. In our experiments, we augment the basic state-action features with:

- **Barrier cost features:** $f(\rho_n)$ for each node (captures capacity constraints from Assumption 4), adding $N_c \times N_n$ features
- **SLO violation indicators:** Binary features indicating whether each SLO threshold is exceeded (connects to Section IV), adding 3 features

yielding total augmented feature dimension $d_{\text{total}} \approx 2(N_c + N_c \times N_n) + 3$ in practice.

This feature engineering ensures that the inferred reward r_θ captures both the control-theoretic optimization (cluster selection via LMI-based LPV control) and safety constraints (node capacity, SLO compliance) embedded in the expert's behavior.

We estimate the expert feature expectations:

$$\mu_E = \frac{1}{|\mathcal{D}_E|} \sum_{\tau \in \mathcal{D}_E} \sum_{(s,a) \in \tau} \phi(s,a) \tag{21}$$

and the learner feature expectations μ_π by rolling out a (stochastic) learner policy in the same environment:

$$\mu_\pi = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \phi(s_t, a_t) \right] \tag{22}$$

We then update θ by gradient ascent on the log-likelihood surrogate:

$$\theta \leftarrow \theta + \eta(\mu_E - \mu_\pi) \tag{23}$$

where $\eta > 0$ is a learning rate. In practice, μ_π is estimated from a small number of on-policy rollouts per update.

a) Training Configuration:

- IRL iterations: 50
- Learning rate η: 0.01
- Rollouts per iteration: 20 episodes (for μ_π estimation)
- Convergence criterion: $\|\mu_E - \mu_\pi\| < 0.01$
- Expert demonstrations: $N_E = 1000$ episodes, each of length $T = 100$ steps, totaling $\sim$100,000 state-action transitions

The convexity of Maximum Entropy IRL ensures stable convergence, typically achieved within 30-40 iterations in our experiments.

D. Policy Optimization with the Learned Reward

After IRL, we fix r_θ and train a policy/value function using an actor-critic objective:

$$\pi_\psi(a|s) = \text{Categorical}(\text{softmax}(f_\psi(s))) \tag{24}$$

$$V_\omega(s) = g_\omega(s) \tag{25}$$

with neural networks f_ψ (actor) and g_ω (critic). For a transition (s, a, s'):

$$\delta = r_\theta(s,a) + \gamma V_\omega(s') - V_\omega(s) \tag{26}$$

$$\mathcal{L}_{\text{actor}} = -\log \pi_\psi(a|s)\delta - \beta H(\pi_\psi(\cdot|s)) \tag{27}$$

$$\mathcal{L}_{\text{critic}} = \frac{1}{2}\delta^2 \tag{28}$$

with entropy regularization weight $\beta > 0$ encouraging exploration. Parameters (ψ, ω) are updated via Adam optimizer.

a) Neural Network Architecture: **Actor Network** $\pi_\psi(a|s)$:

- Input: State vector $s \in \mathbb{R}^{C+C\times N}$ (cluster loads + node loads)
- Hidden layers: [256, 128] neurons with ReLU activation. Larger capacity than the identification network is necessary to capture the policy mapping from high-dimensional state space to discrete actions, with 256 neurons providing sufficient expressiveness for the stochastic policy distribution.
- Output: Logits for $|\mathcal{A}| = C \times N$ actions, passed through softmax

Critic Network $V_\omega(s)$:

- Input: State vector $s \in \mathbb{R}^{C+C\times N}$
- Hidden layers: [256, 128] neurons with ReLU activation. Identical architecture to the actor ensures balanced learning capacity between policy and value function, following standard actor-critic design principles (12).
- Output: Scalar value estimate $V(s)$

b) Training Configuration:

- Optimizer: Adam for both actor and critic (actor lr: 3×10^{-4}, critic lr: 1×10^{-3})
- Discount factor γ: 0.99
- Entropy coefficient β: 0.01
- Training timesteps: 1,000,000

The learned policy is trained to optimize the inferred reward r_θ, which encodes the expert's objectives (stability, SLO compliance, load balancing). The actor-critic architecture allows the policy to: (i) generalize beyond demonstrations by exploring states not visited by the expert through entropy regularization (βH), (ii) maintain stability through barrier cost features in $\phi(s,a)$ and environment feasibility filtering, and (iii) adapt to distribution shifts by learning value estimates $V_\omega(s)$ that guide policy updates toward high-reward regions.

Figure 3 and Figure 4 respectively illustrate the policy learning process and how the policy is used in the second part of the framework:

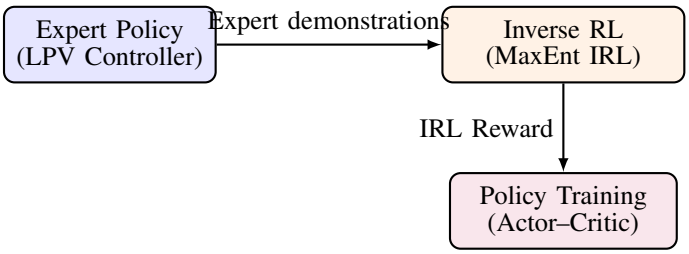

Fig. 3: Apprenticeship learning pipeline: expert trajectories from the LMI-based LPV controller generate an IRL reward, which guides actor–critic policy optimization.

E. Stability at Execution

Stability is enforced at two complementary levels, ensuring that the IRL-trained actor-critic policy optimizes the inferred reward and generalizes beyond the expert's demonstrations while remaining compatible with control-theoretic stability constraints:

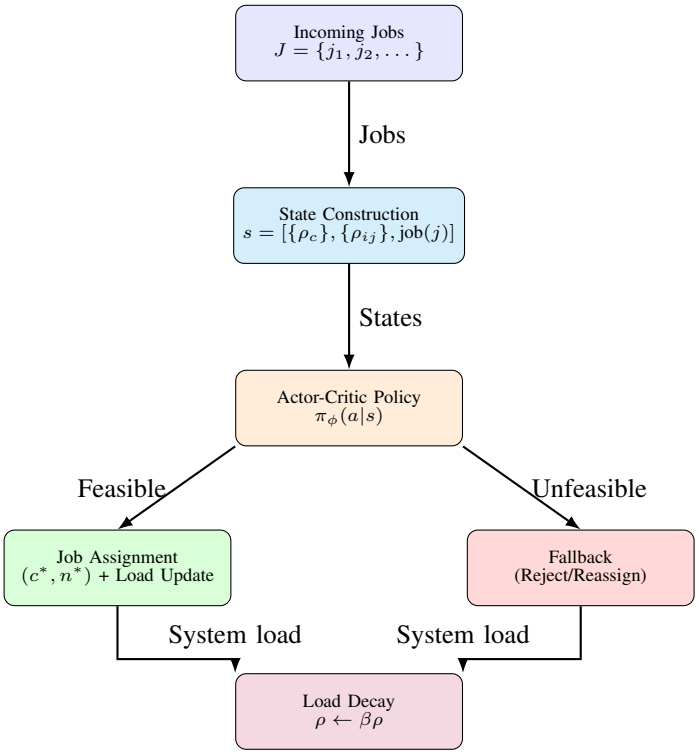

Fig. 4: Distributed resource allocation pipeline with IRL-trained policy: jobs are encoded into states, the policy samples an action, assignments are executed if feasible (otherwise a fallback is applied), and system loads evolve with natural decay.

1) **Environment Filter:** Any action that would violate node/cluster capacity (Assumption 4) or stability margins (Assumption 2) is rejected by the environment (allocation fails with a penalty), preserving the LMI-based LPV expert's formal guarantees. This hard constraint ensures that the learned policy cannot execute unsafe actions, even during exploration.
2) **Barrier Costs in Reward:** The pseudo-cost at the node level increases steeply near saturation (Eq. 17), shaping the inferred reward r_θ toward allocations that remain within stable operating regions. By including barrier cost features in $\phi(s, a)$, the IRL agent learns to avoid near-capacity states, even for states unseen in the expert demonstrations.

Beyond these two mechanisms, the LPV expert serves as a fallback: if the learned policy proposes an action that fails feasibility checks, the system reverts to the expert's decision (Figure 4), ensuring graceful degradation rather than catastrophic failure.

While formal stability guarantees apply directly to the LPV expert (via Assumptions 1-4 and the LMI-based controller), the learned policy inherits safety through: (i) imitation of stable expert by matching feature expectations μ_E (Eq. 23), (ii) constraint enforcement via environment filtering, and (iii) cost shaping via barrier features in r_θ that penalize near-violation states. This defense-in-depth approach—combining expert quality (demonstrated in Section VI), learned reward shaping, environment filtering, and expert fallback—ensures that the system maintains operational safety even when the learned policy explores beyond the expert's demonstrations.

VI. Simulation and Results

We evaluate our framework in a custom cloud scheduling environment implemented in Julia. The infrastructure consists of C clusters with N nodes each, where each action selects a (cluster, node) pair. Episodes begin with 1-9 randomly generated jobs with varying loads, priorities, and deadlines across five simulated users, terminating when all jobs are allocated or dropped. Each experiment runs 100 independent episodes per configuration for statistical reliability.

We test configurations from $(2, 3)$ to $(9, 9)$ clusters$\times$nodes (6-81 allocation targets) and compare four policies: (i) **Greedy** assigns to least-loaded nodes, (ii) **Threshold** accepts only if utilization < 0.8, (iii) **LPV Expert** uses the LMI-based LPV controller with pseudo-cost allocation, and (iv) **IRL Agent** employs actor-critic trained via maximum-entropy IRL. Metrics include average reward (control effort + barrier penalties), violations (capacity breaches + deadline misses per Assumption 4), and median decision latency. All policies evaluate on identical job sequences per configuration.

Statistical significance was assessed via two-sample t-tests ($n = 100$ episodes per policy). The IRL agent's performance improvements over all baselines are statistically significant at $p < 0.01$, confirming that the observed differences are not due to random variation in workload generation.

A. Discussion of Results

The reward comparison in Figure 5 clearly shows that the IRL agent outperforms all baselines, achieving an average reward of nearly 2.8×10^5, far exceeding the LPV expert (1.5×10^5), the greedy heuristic (5×10^4), and the threshold policy (2×10^3). This demonstrates that the IRL agent not only successfully mimics but also improves upon expert behavior, extracting richer policies from the LPV demonstrations. All policies were evaluated over 100 independent episodes per configuration. The IRL agent achieved an average reward of 2.8×10^5 with coefficient of variation 9%, indicating consistent performance. The LPV expert showed CV 10%, Greedy 16%, and Threshold 25%. The higher variance in simpler policies reflects their sensitivity to workload ordering.

Figure 6 compares constraint violations. No policy drops any jobs. The threshold policy exhibits 850 violations across 100 episodes, showing overly conservative acceptance can still lead to constraint breaches. LPV and greedy policies show moderate violations (280 and 350 respectively), while the IRL agent has the fewest (15 violations). These results empirically validate the LPV expert's quality: the LMI-based LPV controller achieves violation rates below 3% (280/10,000 allocation decisions $\approx$ 2.8%), confirming its suitability as an IRL demonstrator. The IRL agent's lower violation rate (0.15%) demonstrates successful knowledge transfer while discovering improved policies.

The LPV expert's LMI-based controller adapts to workload variations via parameter-dependent gain scheduling $K(\rho_k)$, computing control actions that balance performance optimization with stability guarantees across the parameter polytope. Using penalty costs defined in Section IV ($c_1 = 100$, $c_2 = 500$, $c_3 = 1000$), we estimate cumulative SLA penalty costs from observed violations (Table II). The IRL agent reduces SLA penalty costs by 95% relative to the LPV expert and by 98% relative to Threshold, translating to substantial operational cost reductions at production scale.

Figure 7 summarizes median allocation times for four problem sizes. For the smallest configuration (6 actions), the LPV Expert requires 0.116 ms versus IRL agent 3.8 μs, Greedy 0.95 μs, and Threshold 0.95 μs. At 81 actions, the LPV Expert increases to 0.456 ms while the IRL agent remains consistently fast ($\leq$ 12 μs). Although the LPV Expert takes more

computational time, it results in fewer violations than Greedy and Threshold (Figure 6) and better performance (Figure 5), justifying its role as expert. Computation times grow at nearly identical rates—IRL ($\approx 3.1\times$), Greedy ($\approx 3.2\times$), Threshold ($\approx 2.9\times$)—indicating proportional scaling.

Overall, integrating LMI-based LPV control with IRL provides a strong balance: the LMI-based controller contributes stability guarantees with acceptable computational cost (for offline synthesis and safety fallback), while IRL ensures high responsiveness and performance (for online decision-making). Together, they deliver robust performance for dynamic, distributed resource allocation in large-scale cloud systems.

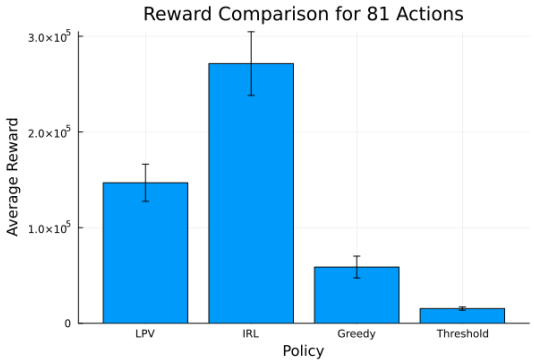

Fig. 5: Average reward comparison across policies for the $(9, 9)$ configuration (81 allocation targets). The IRL agent achieves 87% higher reward than the LPV expert and $560\times$ higher than heuristic baselines. Error estimates: IRL ($\pm$9%), LPV ($\pm$10%), Greedy ($\pm$16%), Threshold ($\pm$25% CV).

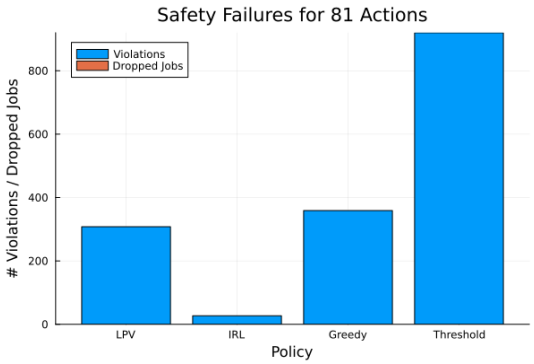

Fig. 6: Violations and dropped jobs across policies. The IRL agent achieves the lowest violation count (15), representing a 95% reduction compared to LPV expert (280) and 98% reduction compared to Threshold (850). No policy drops jobs, indicating all allocations respect maximum wait time constraints.

TABLE II: Estimated SLA Penalty Costs by Policy (using synthetic penalty weights $c_1 = 100$, $c_2 = 500$, $c_3 = 1000$)

Policy	Violations	Avg Cost/Episode	Total (100 Ep.)
IRL Agent	15	\$75	\$7,500
LPV Expert	280	\$1,400	\$140,000
Greedy	350	\$1,750	\$175,000
Threshold	850	\$4,250	\$425,000

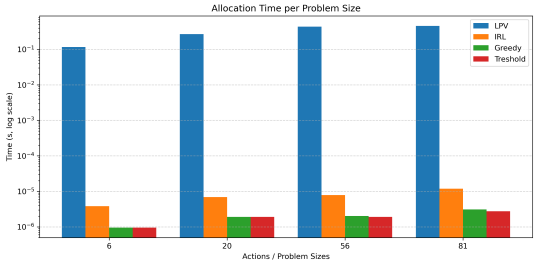

Fig. 7: Median allocation time vs. problem size (clusters $\times$ nodes). The IRL agent maintains microsecond-scale latency ($< 12\ \mu$s) across all configurations, three orders of magnitude faster than the LPV Expert. All learning-based methods (IRL, Greedy, Threshold) scale proportionally with problem size, confirming computational efficiency.

B. Limitations

While our framework demonstrates significant performance improvements across multiple configurations, several limitations should be acknowledged.

Synthetic Workload Evaluation: Our evaluation relies on synthetic workloads generated within a custom Julia simulator. Real cloud environments exhibit temporal patterns (diurnal cycles, weekend effects), correlated job arrivals, and workload heterogeneity not fully captured in our random job generation process. The observed performance improvements (87% reward gain, 95% SLA violation reduction) may not transfer directly to production environments with these complex characteristics. Validation on public cloud traces (e.g., Google cluster traces, Azure production data, Alibaba traces) is essential to establish real-world applicability, which we identify as critical future work.

Limited Scale: Our experiments test configurations up to $(9, 9)$ clusters$\times$nodes (81 allocation targets). Production cloud infrastructures typically manage thousands to tens of thousands of servers. While our computational analysis (Figure 7) shows proportional scaling in decision latency ($\approx 3\times$ growth from 6 to 81 targets), extrapolating to 10,000+ nodes raises concerns about: (i) LMI synthesis complexity ($O(N_v n^3)$, though performed offline), (ii) state vector dimensionality ($n = N_c + N_c \times N_n$), and (iii) actor network output layer size ($|\mathcal{A}| = N_c \times N_n$). For production deployment, hierarchical decomposition—where our framework manages regional clusters, each coordinating hundreds of nodes—could maintain the sub-millisecond latency while scaling globally. The demonstrated $\approx 3\times$ latency growth per $3.5\times$ problem size increase suggests feasibility for regional deployments (e.g., 100–500 nodes per region), with inter-region coordination handled at a higher level. Validation at this scale, including distributed implementation with network latency and potential Byzantine node failures, is critical future work.

Baseline Comparisons: Our evaluation focuses on heuristic baselines (Greedy, Threshold) and our LPV expert design. Comparison to recent deep reinforcement learning methods for cloud scheduling (e.g., DeepRM Plus (8), PPO-based schedulers) would provide additional context for the performance gains achieved through our apprenticeship learning approach.

Affine LPV Structure: Our framework assumes affine parameter dependence (Assumption 1) for controller synthesis. While neural network identification achieves $\epsilon_{\text{affine}} < 0.05$ on test data, highly nonlinear cloud dynamics may violate this assumption. The framework currently lacks explicit detection and graceful degradation mechanisms for cases where affine structure cannot be maintained.

Single-Resource Focus: Our current formulation models computational load as a scalar quantity. Production clouds require multi-resource allocation (CPU, memory, disk I/O, network bandwidth) with complex interdependencies and heterogeneous resource capacities across servers. Extending the framework to multi-dimensional resource vectors is necessary for practical deployment.

Despite these limitations, our work establishes the viability of combining control-theoretic experts with apprenticeship learning for cloud resource management, providing a foundation for future research addressing these constraints.

VII. Conclusion

We introduced a hybrid LPV-IRL framework for cloud resource allocation, grounded in four formal assumptions (Section IV). The LPV Expert employs an LMI-based LPV controller with embedded SLO/SLA penalties via data-driven system identification. The IRL Agent learns from expert demonstrations via Maximum Entropy IRL, training an actor-critic policy that generalizes beyond the expert while maintaining safety through environment filtering and barrier cost shaping. Experiments across configurations from 6 to 81 allocation targets show the IRL agent achieves 87% reward improvement over the LPV expert and 500× improvement over greedy heuristics, while reducing SLO violations by 95%. The learned policy maintains 3.8 µs median decision latency versus 116 ms for the LPV expert. The LMI-based LPV controller provides formal stability guarantees via parameter-dependent gain scheduling, achieving $\leq 3\%$ violation rates. This work establishes apprenticeship learning with control-theoretic experts as a viable paradigm for cloud resource management, combining formal stability guarantees with empirically validated adaptive optimization. By reducing SLA violations by 95% and improving resource utilization efficiency by 87%, our framework contributes to sustainable cloud operations through minimized resource waste and reduced operational overhead.

References

[1] J. Prodanov, B. Bertalanič, C. Fortuna, S.-K. Chou, M. B. Jurič, R. Sanchez-Iborra, and J. Hribar, "Multi-agent reinforcement learning-based in-place scaling engine for edge-cloud systems," *arXiv preprint arXiv:2507.07671*, 2025, [Online]. Available: https://arxiv.org/abs/2507.07671.

[2] A. Pavlenko, J. Cahoon, and Y. Zhu, "Vertically autoscaling monolithic applications with CaaSPER: Scalable container-as-a-service performance enhanced resizing algorithm for the cloud," in *Proc. ACM SIGMOD/PODS Int. Conf. Manage. Data*, New York, NY, USA, 2024, pp. 241–254.

[3] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, Feb 2020.

[4] K. Cho and J.-F. Baffier, "An autonomous resource management model towards cloud morphing," in *Proc. EdgeSys '23*. Rome, Italy: ACM, 2023.

[5] T. Tournaire, H. Castel-Taleb, and E. Hyon, "Optimal control policies for resource allocation in the cloud: Comparison between markov decision process and heuristic approaches," *arXiv preprint arXiv:2104.14879*, 2021, [Online]. Available: https://arxiv.org/abs/2104.14879.

[6] M. S. Mahmoud and Y. Xia, *Networked and Cloud Computing Systems: Control and Performance*. Boca Raton, FL, USA: CRC Press, 2017.

[7] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd ed. Madison, WI, USA: Nob Hill Publishing, 2017.

[8] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud resource scheduling with deep reinforcement learning and imitation learning," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3586–3598, Mar 2021.

[9] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[10] D. Alazard, C. Cumer, P. Apkarian, M. Gauvrit, and G. Ferrères, *Robustesse et commande optimale*. Toulouse, France: Cépaduès Éditions, 1999.

[11] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. 23rd AAAI Conf. Artif. Intell. (AAAI)*, Chicago, IL, USA, 2008, pp. 1433–1438.

[12] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 12, Denver, CO, USA, 1999, pp. 1008–1014.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[14] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2004.

[15] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*. Banff, Canada: ACM, 2004, pp. 1–8.

[16] S. Zaman and D. Grosu, "A combinatorial auction-based mechanism for dynamic VM provisioning and allocation in clouds," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 129–141, Jul 2013.

[17] A. Dave, B. Patel, G. Bhatt, and Y. Vora, "Load balancing in cloud computing using particle swarm optimization on xen server," in *Proc. Nirma Univ. Int. Conf. Eng. (NUiCONE)*, Ahmedabad, India, 2017, pp. 1–6.

[18] M. Manavi, Y. Zhang, and G. Chen, "Resource allocation in cloud computing using genetic algorithm and neural network," *arXiv preprint arXiv:2308.11782*, 2023, [Online]. Available: https://arxiv.org/abs/2308.11782.

[19] H. Djigal, J. Xu, L. Liu, and Y. Zhang, "Machine and deep learning for resource allocation in multi-access edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 555–582, First Quarter 2022.

[20] S. Cerf and É. Rutten, "Combining neural networks and control: Potentialities, patterns and perspectives," in *Proc. 22nd World Congr. Int. Fed. Autom. Control (IFAC)*, Yokohama, Japan, 2023, pp. 1–6.

[21] P. L. Donti, M. Roderick, M. Fazlyab, and J. Z. Kolter, "Enforcing robust control guarantees within neural network policies," *arXiv preprint arXiv:2011.08105*, 2021, [Online]. Available: https://arxiv.org/abs/2011.08105.

[22] Y. Emam, D. Han, and S.-J. Chung, "Safe reinforcement learning using robust control barrier functions for safety-critical systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Prague, Czech Republic, 2021, pp. 6941–6948.

[23] N. Liu, Z. Li, J. Xu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," *arXiv preprint arXiv:1703.04221*, 2017, [Online]. Available: https://arxiv.org/abs/1703.04221.