

Incentives and Evidence in Learned Service Orchestration

Syed Izhan Khilji[†], Alireza Furutanpey^{*†‡}, Schahram Dustdar[†]

[†]Distributed Systems Group (DSG), TU Wien, Vienna, Austria

{i.khilji, dustdar}@dsg.tuwien.ac.at

[‡]Coovally.ai, Barcelona, Spain

a.furutanpey@coovally.ai

Abstract—Reinforcement learning for service orchestration has been the subject of sustained research for over a decade, yet it is not used in production at scale. The usual explanation is that learned controllers degrade under delayed and noisy telemetry, workload shifts, and uncontrolled tenants. We test whether existing evidence supports that explanation. We re-evaluate three highly influential RL-based orchestration systems spanning resource allocation, DAG scheduling, and autoscaling, using pre-registered predictions about comparative degradation under production-relevant perturbations and paired inference with family-wise error correction. Across the tests, most predicted performance reversals do not occur. Diagnostic analyses show that these outcomes often reflect comparator collapse, artefact limitations, or evaluation choices rather than evidence that learned controllers tolerate the perturbations. One apparent advantage under observation lag falls by roughly fortyfold against a Kubernetes HPA-equivalent controller. Another widely cited result cannot be reconstructed from its released artefact, and the strongest reproducible margin is far smaller than the published headline. Conclusions also reverse under changes in perturbation magnitude and evaluation mode. Based on these results and patterns identified in broader literature, we identify a methodological and institutional problem. Publication and review incentives favour benchmark gains against convenient comparators, even when those gains provide little evidence of deployment performance. We argue that the problem is not solely technical, i.e., a new benchmark or novel quantitative evaluation criteria alone will not correct this. Rather, it is institutional, so learned orchestration needs production-grade comparators, registered perturbation models, separate operational metrics, and publication criteria that reward reproducible operational evidence over nominal benchmark gains. Without these changes, the literature can grow without establishing whether learning improves orchestration.

Index Terms—service orchestration, reinforcement learning, distributed systems, computing continuum, robustness, reproducibility, pre-registered evaluation, benchmarking

I. INTRODUCTION

Service orchestration coordinates the runtime decisions that keep a distributed system running, such as how many replicas of a service to run, where to place them, how to route requests among them, and how to divide finite compute under shifting demand. These decisions set cost, latency, and reliability, and they are made continuously as the workload changes. A controller can make them reactively, from the state it observes, or it can anticipate where demand is going and act before it

arrives. Whether the second, learned from data, beats the first has been an open question for over a decade.

Production control is reactive and hand-tuned. Google's Autopilot scales from moving-window statistics and a supervised recommender [1]. Borg packs tasks by priority under engineered rules [2], [3]. The default scalers in Kubernetes, the most widely deployed orchestrator, compare one utilisation signal against a threshold. These controllers are robust and cheap to run, but they act on the present observation, optimise one step at a time, and are retuned by hand for each deployment. Reinforcement learning is the dominant proposed alternative for a concrete reason. A hand-tuned rule does not adapt to the workload in front of it, acts one step at a time, and cannot express the coupled high-dimensional choices, which job on which machine, that scheduling and placement require. A learned policy can do all three. It anticipates rather than reacts, it is trained on the workload rather than tuned by hand, and it can represent decisions no simple rule captures. The published methods report large simulator improvements over hand-tuned heuristics for resource allocation [4], cluster scheduling [5], and auto-scaling [6]. That promise has sustained a decade of work. Learned orchestration has not reached production with a noticeable impact. Production environments expose learned policies to noisy and delayed telemetry, biased reward proxies, abrupt workload shifts, and uncontrolled tenants. Each condition can degrade learned-policy performance [7], [8], [9], [10]. The unresolved question is whether these conditions disadvantage learned policies relative to the controllers operators actually deploy.

Current evidence cannot determine whether learning improves orchestration for two reasons. First, deep reinforcement learning results are sensitive to implementation and evaluation choices [11], [12], and an advantage measured against a weak baseline may not hold up against a controller that operators would actually deploy. Second, existing evaluations rarely expose learned controllers to delayed telemetry, heavy-tailed workloads, or adversarial observation perturbations under reproducible protocols. Learned policies are vulnerable to adversarial inputs [7], [10], [13], while orchestration evaluations rarely test them under such inputs. They also omit the heavy-tailed, bursty, and skewed workloads observed in production traces [14], [15] and addressed explicitly in large-scale system design [16]. We demonstrate the first problem by re-evaluating

*Corresponding author

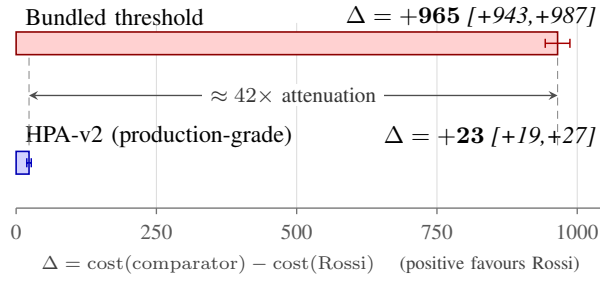


Fig. 1: Comparator choice attenuates Rossi’s apparent advantage under $k = 10$, s observation lag from $\Delta = +965$ against the bundled threshold controller to $+23$ against an HPA-v2-equivalent controller, approximately $42\times$. Positive Δ favours Rossi; whiskers show 95% paired-bootstrap confidence intervals over 30 windows.

the seminal work by Rossi et al. for autoscaling [6] under a 10s observation lag using 30 paired workload windows. We hold the learned controller, workload windows, perturbation, and cost metric fixed and change only the comparator. The HPA-v2-equivalent controller implements a tolerance deadband and scale-down stabilisation; its implementation and configuration sensitivity are detailed in Section IV-D. Against the unstabilised threshold controller bundled with the simulator, Rossi appears better by 965 cost units. Against the HPA-v2-equivalent controller, the estimated advantage falls to 23 units, an attenuation of approximately $42\times$ (Figure 1). The HPA-v2-equivalent controller is also cheaper in the clean, heavy-tailed, and adversarial conditions. The large lag advantage is therefore primarily evidence of comparator collapse rather than evidence that the learned controller degrades less under delayed telemetry than a production-representative autoscaler.

This result motivates our central question: *do current evaluation practices distinguish lower learned-policy degradation under specified perturbations from comparator failure, artefact limitations, and evaluation choices?* We address this question by re-evaluating DeepRM [4], Decima [5], and Rossi [6], which span resource allocation, DAG scheduling, and container autoscaling. We pre-register nine directional predictions that the learned policies will degrade more than their comparators under production-relevant perturbations and evaluate them using paired statistical inference with family-wise error correction (Section IV). Seven of the nine predictions fail, but these failures do not establish that the learned policies tolerate the tested perturbations. For DeepRM and Rossi, observation lag degrades the bundled comparators more than the learned policies, so the comparisons primarily measure comparator failure. Decima’s released artefact cannot reconstruct the testbed comparison supporting its reported 21% improvement, and the margin against the strongest comparator exposed by the artefact is 3.0%. The conclusions are also sensitive to evaluation choices: Decima’s lag verdict reverses at one perturbation magnitude (Section IV-E), and Rossi’s

lag verdict reverses between frozen-checkpoint and online evaluation.

Our core contribution is to show why current evidence cannot determine whether learning improves orchestration and to identify the minimum requirements for a valid test. Learned policies must be compared with deployed controllers under production perturbations using reproducible protocols and operational outcomes. Until those requirements are met, reported gains do not establish progress.

II. BACKGROUND

We introduce the terminology and action-space perspective that the remainder of this work builds on.

A. Reinforcement learning

A reinforcement-learning agent performs actions in an environment modelled as a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, with states $\mathcal{S}$, actions $\mathcal{A}$, transition kernel $P(s' | s, a)$, reward $r(s, a, s')$, and discount $\gamma \in (0, 1)$ [17]. It learns a policy $\pi(a | s)$ that maximises the expected discounted return $\mathbb{E}[\sum_t \gamma^t r_t]$. Orchestration methods span value-based families such as, tabular [18] and deep (DQN [19], Double-DQN [20]), policy gradients (PPO [21]), and actor-critic methods (SAC [22]). These families, as standardly formulated, assume a stationary or slowly drifting process, a scalar extrinsic reward, and a tractable action space. Orchestration violates all three.

B. Orchestration through its actions

The literature characterises an orchestration task by its state and reward. The action space is more informative. An orchestration action is a call to a control plane, with a precondition, a cost, and an effect that are fixed in advance. A replica cannot be removed when none exist. A pod cannot be placed on a node without capacity. Routing weights must sum to one. The action space therefore encodes what a policy can do and which formulations are well posed.

The two halves of the problem are not equally hard to model. The dynamics are opaque, because the effect of an action propagates through queueing, contention, and tenant behaviour that no simulator reproduces fully, so a model of them must be learned from data. The actions are not opaque, because their local semantics are given by API contract and are known before any data is collected (Figure 2). A standard formulation ignores this difference and learns both halves from data with no prior, which places the one structure the problem hands us, the action space, on the side that must be inferred. Domain knowledge enters most cheaply on the action side.

C. Learning for systems

Reinforcement learning for cluster management and scheduling was advanced by Mao *et al.* [4], [5], with antecedents in hybrid autonomic control [23], classification-driven cluster management [24], and a broad networking survey by Luong *et al.* [25]. The Park benchmark [26] collects twelve systems problems under one interface and notes how

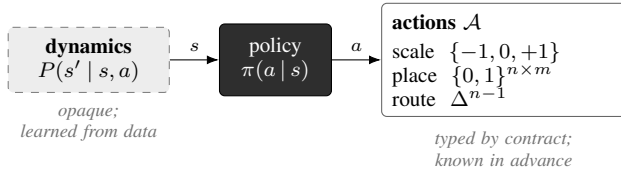


Fig. 2: Orchestration provides asymmetric prior knowledge. System dynamics must be inferred from data, whereas the local semantics and feasibility constraints of actions are specified by the control-plane API. Generic RL formulations typically do not encode this action-space structure.

learning for systems differs from learning for games. The three methods we study are the most-cited of this line, but the practices we examine are not unique to them. More recent microservice systems, such as FIRM [27] and Sinan [28], compare against bespoke baselines in a simulator or a single cluster, the arrangement we examine here. Across the line, learning beats heuristics in simulation but has not displaced the production elasticity heuristics surveyed in [29], [30], even where the deployment infrastructure exists.

III. ORCHESTRATION TASKS AND THE LIMITS OF CURRENT EVIDENCE

We first organise the task classes by the properties that affect whether reinforcement learning can improve control over heuristic or supervised baselines, namely action-space structure, observability, feedback delay, reversibility, and operational risk. We then argue that the field’s evidence cannot establish whether it does.

A. A problem-centred taxonomy

Table I profiles the task classes that account for most published RL for orchestration work. Each class is defined by the action-space structure, the state it observes, how long it waits before the effect of an action is visible, how reversible its actions are, and the operational risk of taking a wrong action. The table points to three implications. First, difficulty rises on two axes at once. Standard algorithms are designed for complex decision problems or non-stationary environments, and handling both together remains hard [31], yet orchestration presents both. Second, the classes where reinforcement learning has its most credible results, load balancing and simple scaling, are the classes where conventional heuristics already perform well [29], [30], so the benefit rarely justifies the operational cost of a learned component [1], [32]. Third, as Section II-B argued, the action space is more tractable than the dynamics, yet standard formulations treat the two alike and approximate every action with no structural prior. Each class has a distinct action geometry, monotone discrete, continuous simplex, or combinatorial assignment, that a generic parameterisation discards.

B. Comparators are selected for tractability

Across this literature, the baselines are chosen for tractability, not for the controllers operators run. The Rossi auto-

scaler [6] ships with a simulator that bundles a threshold controller, the control class Kubernetes deploys by default, yet its published comparisons are against static deployments and other learned variants, and the bundled threshold is left unevaluated. Decima [5] reports its 21% improvement against tuned heuristics on a live Spark testbed, with Graphene among its strongest simulator baselines, yet its released simulator exposes static partitioning and FIFO as the classical comparators and ships no executable Graphene, so the artefact can reproduce neither comparison. The pattern is not particular to orchestration. In recommender systems, most published neural methods fail to beat classical baselines once those baselines are tuned [33], and within reinforcement learning itself, simple linear policies match deep methods on the benchmarks that established them [34]. A field that selects comparators for availability inherits this failure mode by default. Section IV measures the cost. The Rossi advantage is attenuated by roughly fortyfold against a production-grade controller, and the Decima artefact cannot verify the claim for which it is cited.

C. Production analogues of adversarial RL failure modes

Each failure mode attributed to orchestration matches an adversarial machine learning attack class with a documented failure for deep reinforcement learning, and the preconditions of each hold in production with no adversary present. The modes are non-stationarity from workload drift and abrupt shifts in demand, the simulation-to-reality mismatch from simulators that omit contention and tail-latency dynamics [35], reward misspecification from biased proxies, exploration risk in irreversible settings [36], amplified by exploration bonuses that reward novelty itself [37], [38], and cluster heterogeneity that gives the same nominal task a different MDP on every cluster [14], [15], [39]. Table II states the correspondence.

Each row yields a directional prediction that, under realistic perturbation, the learned policy degrades more than the baseline. We test three, each instantiated on all three methods, giving the nine cells of Section IV.

P1 (observation lag). Production telemetry is sampled and aggregated with delay, and per-counter jitter is comparable to the perturbation budgets in adversarial RL [7], [8]. We predict that learned orchestrators degrade under realistic lag, and more than a non-learning baseline on the same telemetry.

P2 (workload tail). A release, a marketing event, or a tenant onboarding changes the transition kernel without changing the agent’s representation of it, the operational form of environment poisoning [9], [10] and of evasion through distribution shift [8]. We predict that a policy trained on one workload distribution degrades on another in proportion to the distance between them. We instantiate this as a service-time shift from exponential to Pareto-tailed ($\alpha = 1.5$), calibrated to the Google cluster workload [14], and for Decima as a reweighting of its DAG workload toward heavier-tailed production size statistics.

P3 (adversarial observation). A bounded FGSM-style perturbation of the policy input [40] should not only change the action but reverse the aggregate outcome.

TABLE I: Problem-centred taxonomy of RL for service orchestration by action-space structure, observability, feedback delay, reversibility, operational risk, and production status. The final column indicates whether RL has displaced heuristic or ML-augmented control in production.

	Task class	Action-space structure	State signal	Observation lag	Reversibility	Risk	Production RL status
	Horizontal/vertical scaling	Discrete adjustment $\{-1, 0, +1\}$	Utilisation, queue depth	seconds–minutes	partial	medium	ML-augmented heuristic (Autopilot)
	Load balancing and routing	Continuous simplex Δ^{n-1}	Rates, response times	seconds	high	low	Heuristic control in production
	Resource allocation	Constrained continuous $\mathbb{R}_{\geq 0}^{m \times d}$	Demands, capacity	seconds–minutes	partial	high	Simulation evidence only
	Service placement and migration	Binary assignment $\{0, 1\}^{n \times m}$	Topology, affinity	minutes–hours	low	high	Heuristic control in production
	SLO-driven control	Mixed discrete and continuous controll knobs	SLO metrics	variable	very low	very high	Largely underexplored

TABLE II: Each common orchestration phenomenon matches an adversarial machine learning attack class with a documented failure for deep reinforcement learning. The final column marks whether the corresponding prediction is tested in Section IV.

#	Orchestration phenomenon	Adversarial-ML analogue	Tested
1	Noisy / lagged telemetry	Observation perturbation [7], [8]	Yes (P1)
1*	Hostile observation noise	Targeted adversarial input [7]	Yes (P3)
2	SLOs as proxy rewards	Reward poisoning [9]	No (Section V)
3	Co-tenant interference	Adversarial co-agents [13]	No
4	Abrupt workload shifts	Environment poisoning [9], [10]	Yes (P2)

Two rows are not tested. Co-tenant interference is the weakest analogue. Co-tenants are independent workloads competing for shared resources, not adversaries optimised against the orchestrator, so the correspondence is distribution shift rather than adversarial optimisation, and we name it as the next perturbation class to register. Reward poisoning points to something deeper than a robustness result. An SLO is an operational invariant the system must hold [41], and encoding it as one term of a scalar reward to be traded off against others is a category error, independent of any noise on the measurement [42]. We return to reward design in Section V, and the Rossi cell in Section IV shows how a scalar weighting can make a comparison uninterpretable.

Two kinds of threat cut across these rows, and the field’s evaluations apply neither. One is deliberate. An adversary chooses inputs to drive the policy to a costly action, and the P3 cell is the analogue, in which a bounded perturbation corrupts the policy’s decisions but, in these simulators, not the aggregate outcome (Section IV-C). The other is emergent. It arises from the workload itself with no adversary, and it

is the more common. Load skew, heavy-tailed service times, hot keys, and bursty correlated arrivals are properties of real workloads that production traces report as the norm [14], [15], and that large-scale systems are explicitly engineered to tolerate [16]. The robust-MDP framework formalises this as optimisation over an uncertainty set of such patterns [43], [44], [45], and the P2 cell is one point from that set, whose weak and method-specific outcome (Section IV-C) marks the class as under-explored rather than safe. Orchestration benchmarks instead use benign synthetic arrivals, Poisson and exponential, so a method can look strong while never facing the worst-case structure it must survive.

The conditions for failure, naturally occurring, deliberately adversarial, and emergent, are present and almost entirely untested. Whether they produce the predicted degradation is the question of Section IV. The argument is not any single correspondence but the cumulative one, that production orchestration, by accident, assembles the conditions under which deep reinforcement learning is documented to fail [7], [8], [10].

IV. EMPIRICAL EVIDENCE

We test the nine predictions of Section III-C on the three methods in Table III, each a canonical example of orchestration RL in one task class. Each is tested against the strongest non-learning comparator its released artefact ships, for two reasons. Where the artefact implements the published comparison, as for DeepRM, this tests the published claim on its own terms. Where it does not, as for Decima, it is the strongest comparison the released evidence permits, and Section IV-A documents the narrowing. A separate, pre-registered re-evaluation against a production-grade controller follows in Section IV-D. The primary result is the difference observed between the two comparator classes, not the failure of the initial predictions, and this comparison was specified before any perturbed run rather than constructed after a null result.

a) *Protocol*: A prediction is *pre-registered* if its (method, perturbation, anchor) cell, comparator, metric, perturbation

TABLE III: The three methods, and the simulator-grade comparator bundled with each. The limitations of these comparators as proxies for production controllers are the subject of Section IV-D.

Method	Task class	Bundled comparator
DeepRM [4]	Resource allocation	Tetris*, the packing heuristic of [46] as implemented in the DeepRM source, and SJF
Rossi [6]	Horizontal / vertical scaling	Threshold controller in the RLAD simulator, with no stabilisation features
Decima [5]	DAG cluster scheduling	<code>dynamic_partition</code> , static partitioning in the public simulator

magnitude, directional claim, statistical test, and rejection rule were written down and frozen in a timestamped pre-registration document before any perturbed run for that cell. For each cell we draw 30 paired evaluation seeds (DeepRM, Decima) or 30 non-overlapping windows of the official workload (Rossi), and compute $\Delta_i = \text{metric}_i(\text{comparator}) - \text{metric}_i(\text{RL})$, where the metric is the one defined by the original paper (mean slowdown, total cost, mean job completion time). A positive Δ means the learned method is better, and the predicted degradation is $\Delta < 0$. Confidence intervals are 95% paired percentile bootstraps over 5000 resamples. The reported p -values are paired sign-flip values over 10^5 flips, one-sided in the observed direction, corrected across the nine predictions by Holm–Bonferroni at $\alpha = 0.05$, following the call for statistically defensible deep-RL evaluation [12]. Anchors are calibrated to the literature. P1 lag is set at $k = 10$ s from Borg telemetry P95 values [2], and at $\lambda = 1.0$ for Decima, the upper limit of the range for which its simulator is calibrated. P2 is set at Pareto $\alpha = 1.5$, and for Decima at DAG-size tail weight $w = 0.5$, which reweights its TPC-H workload toward the heavier-tailed DAG statistics of the Alibaba production trace. P3 is an FGSM perturbation at $\varepsilon = 0.05$. P1 and P2 perturb the environment and apply to the learned policy and the comparator identically. P3 targets the learned policy’s observation alone while the comparator reads the true state, the threat model in which the attacker corrupts the representation the defender computes on [7]. The pre-registration document, perturbation specifications, controller implementations, and evaluation scripts accompany this submission as supplementary material and are released publicly¹.

A. Reproduction gates

A perturbation test is informative only on a method whose clean behaviour reproduces. DeepRM passes a strict 30-seed gate with no silent admission drops, reaching mean slowdown 36.5 against Tetris* 61.0 ($\Delta = +24.5$, CI $[+22.7, +26.3]$), with the authors’ source-aligned values (19.1, 23.3, 44.8) also reproduced under the original protocol. Rossi matches all six metrics of its Table I within sub-percent relative error, the

worst case being 0.36%, well inside the pre-registered 15% gate. The gate also fixes the evaluation mode. Rossi is a tabular controller that learns online, and a frozen-checkpoint variant fails the clean gate, posting cost 608.9 against the reproduced 204.1, and reverses the P1 verdict ($\Delta = -233.7$ at $k = 10$, against +965.1 online). We therefore evaluate the published online controller and release the frozen sweep as a superseded sensitivity. A verdict that flips with the evaluation mode is itself the implementation-dependence documented for deep reinforcement learning [47], [11], here surfacing in the comparison rather than the training. Decima is the exception. Trained under the authors’ released code for the full 10,000-epoch budget, our checkpoint improves on the README-exposed `dynamic_partition` comparator by 3.0% in mean JCT, against the 21% the paper reports over tuned heuristics on a 25-node Spark testbed. The released artefact is the event-driven simulator, not the testbed. Its reference evaluation exposes static partitioning and FIFO as the classical schemes and ships no executable Graphene, so neither the headline comparison nor the strongest-baseline one can be reconstructed from it. We therefore report the Decima cells as a narrowed comparison, consistent with the broader finding that implementation choices and ablated comparators explain much of the difference between published RL results and independent re-implementations [11].

B. Outcomes

Table IV reports every cell. Seven of nine predictions are falsified. The two confirmations (Rossi P2, P3) are not perturbation-induced crossovers. The comparator already beats the learned controller in the clean cell ($\Delta_{\text{clean}} = -85.8$ on all 30 windows), and the perturbation only widens an existing divergence.

In none of the three methods is the learned policy robust enough for an operator to rely on. For Rossi, the +965 P1 magnitude is the bundled threshold’s bang-bang oscillation under stale telemetry, a baseline artefact that attenuates roughly $40\times$ against a stabilisation-windowed controller (Section IV-D). For DeepRM, the magnitude is sensitive to the stale-action injection rule, under which the Tetris* comparator falls through to a no-op in 68.6% of perturbed steps. An alternative injection protocol reduces it by 63% (Figure 3), so the cell reflects the perturbation harness as much as the method. For Decima, neither explanation applies. At the calibrated anchor, its lag-robustness survives even a stronger dependency-aware comparator (Section IV-D), though Section IV-E shows that this particular verdict is sensitive to the lag magnitude, and the difficulty the Decima cells expose is not a weak baseline but a published comparison that cannot be reproduced from the released artefact. The remaining cells, dissected in Section IV-C, divide along the same line, between artefacts of weak comparators and method behaviour that the released code does not let an independent party verify. Three falsified P1 predictions rest on three different causes, and none of them supports the conclusion that the method is sound. The count is uninformative because the comparisons behind it are.

¹<https://github.com/izhan19717/Learned-Service-Orchestration.git>

TABLE IV: The nine pre-registered predictions and their anchor outcomes. $\Delta = \text{metric}(\text{comparator}) - \text{metric}(\text{RL})$, with the per-method metric named in the second column. A positive Δ means the learned method is better, and the predicted degradation is $\Delta < 0$. The quantity p_H is the Holm–Bonferroni adjusted p -value across the family of nine.

ID	Method (metric)	Perturbation	Anchor	Δ	95% CI	p_H	Outcome
P1-DR	DeepRM (slowdown)	Observation lag	$k = 10$	+195.7	[+185.1, +205.9]	$< 10^{-3}$	falsified
P2-DR	DeepRM (slowdown)	Workload tail	$\alpha = 1.5$	+91.8	[+79.9, +104.6]	$< 10^{-3}$	falsified
P3-DR	DeepRM (slowdown)	FGSM observation	$\varepsilon = 0.05$	+23.6	[+21.7, +25.4]	$< 10^{-3}$	falsified
P1-Ro	Rossi (cost)	Observation lag	$k = 10$ s	+965.1	[+943.1, +986.5]	$< 10^{-3}$	falsified [†]
P2-Ro	Rossi (cost)	Service-time tail	$\alpha = 1.5$	−106.0	[−113.1, −98.7]	$< 10^{-3}$	confirmed (widens)
P3-Ro	Rossi (cost)	Bucket-flip	$\varepsilon = 0.05$	−91.0	[−95.4, −86.8]	$< 10^{-3}$	confirmed (widens)
P1-De	Decima (JCT)	Observation lag	$\lambda = 1.0$	+716,396	[+653,206, +779,842]	$< 10^{-3}$	falsified
P2-De	Decima (JCT)	Workload tail	$w = 0.5$	+8,044	[+1,138, +16,525]	0.033	falsified [‡]
P3-De	Decima (JCT)	FGSM node features	$\varepsilon = 0.05$	+2,001	[+1,057, +3,019]	$< 10^{-2}$	falsified

[†] The magnitude is a property of the bundled threshold. Against an HPA-equivalent controller it drops by about 40×, and the verdict survives only marginally (Section IV-D). [‡] Survives Holm at the boundary under the one-sided convention, but not under the two-sided sensitivity convention ($p_{H,2s} \approx 0.066$).

C. The cross-method pattern

Three observations recur, each carrying the caveats above.

Observation lag breaks the weaker comparators harder than the learned policies, but not uniformly and not always because the comparator is weak. P1 is falsified in all three methods. For Rossi and DeepRM the mechanism is the comparator. The Rossi threshold lacks production stabilisation, and the Tetris* no-op fall-through under the injection rule inflates the DeepRM magnitude. Decima is the exception that disciplines the claim, because at the calibrated anchor its lag advantage holds even against a stronger dependency-aware scheduler (Section IV-D), although that advantage is itself magnitude-sensitive (Section IV-E). We therefore do not claim that lag uniformly breaks heuristics, nor that learned policies are uniformly robust. What the three cells share is narrower. In no case does the bundled comparison let a reader conclude how the method would fare against a controller an operator would run.

Adversarial perturbations land on the decision, not the outcome. P3 is falsified in all three, and the diagnostic is consistent. The attack bites at the policy output, changing DeepRM’s argmax in 69.8% of states in the locked P3 diagnostic (75.2% in the re-evaluation pass of the action-redundancy ablation, Section IV-F), dropping Decima’s target-action probability from 0.341 to 0.279, and flipping Rossi’s bucketed control on 20.9% of steps, yet it does not reverse the aggregate metric. The locus of the apparent robustness is therefore the action space rather than the state encoding, in every method. This locates *where* the robustness sits, not yet *why*. The ablation in Section IV-F tests one candidate mechanism and does not support it, and the open question that remains motivates the action-space view of Section II-B.

Workload-tail effects are weak and method-specific. P2 is the closest to a mixed result. DeepRM rejects with a large positive Δ . Rossi degrades as predicted, but only on top of pre-existing clean dominance. Decima rejects at the boundary, on an aggregate driven by a small number of outlier seeds,

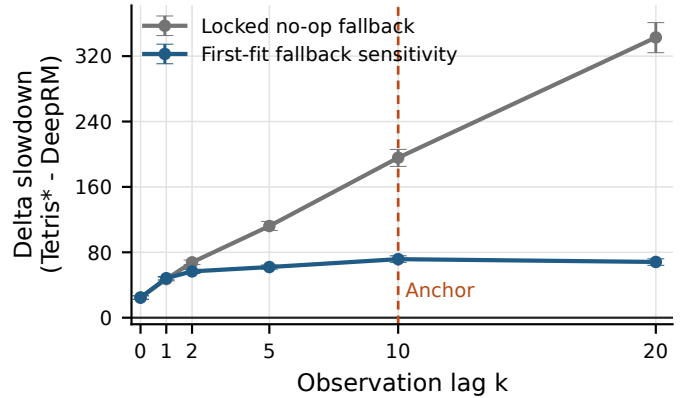


Fig. 3: The DeepRM P1 magnitude is largely an artefact of how a stale action is injected. Under the pre-registered no-op fallback (grey) the Tetris* comparator falls through to a no-op and its cost grows without bound in the lag. Under a first-fit fallback (blue) the same comparator stays competent, and the difference at the $k = 10$ anchor (dashed) falls by about 63%. The directional verdict is unchanged across the grid, but the magnitude is not. $\Delta = \text{slowdown}(\text{Tetris}^*) - \text{slowdown}(\text{DeepRM})$.

with only 16 of 30 seeds favouring Decima (Figure 4). The class generalises in neither direction.

D. Re-evaluation against stronger comparators

The Rossi P1 result rests on a baseline whose bang-bang collapse is partly an artefact of the bundled implementation. To test whether it survives a comparator with production-grade stabilisation, we implemented a Kubernetes HPA-equivalent controller following the `autoscaling/v2` specification, with a proportional recommendation under a 10% tolerance deadband, a 300 s scale-down stabilisation window, immediate scale-up, default rate policies, a 15 s sync period, replica bounds $[1, 10]$, and a 50% target utilisation. We re-ran the four Rossi cells under the same paired protocol. Three pre-

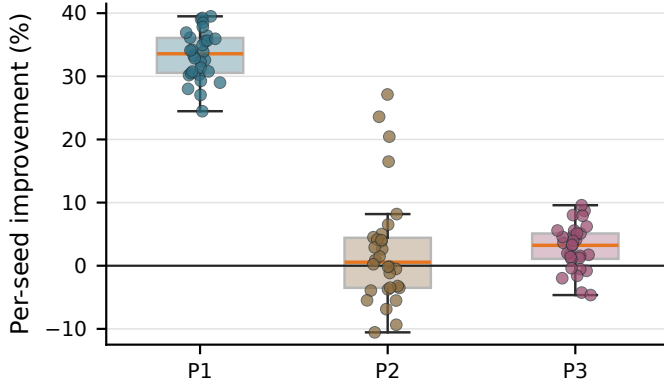


Fig. 4: Per-seed paired improvement for the three Decima cells, against `dynamic_partition`. P1 lag is positive on all 30 seeds, ranging from 24% to 40%. P2 tail has a median near zero with a few large positive outliers carrying the aggregate, the pattern behind its boundary rejection and the reason it generalises in neither direction. P3 is narrow and small. Positive means Decima is the cheaper policy.

TABLE V: Re-evaluating the four Rossi cells against a production-grade HPA-equivalent controller. $\Delta = \text{cost}(\text{HPA-v2}) - \text{cost}(\text{Rossi})$, where negative means HPA-v2 is cheaper. The bundled-threshold Δ is shown for comparison.

Cell	Δ (bundled)	Δ (HPA-v2)	95% CI (HPA-v2)
Clean	-85.8	-31.6	[-36.4, -26.8]
P1 lag $k = 10$ s	+965.1	+23.0	[+18.8, +27.2]
P2 tail $\alpha = 1.5$	-106.0	-52.4	[-60.0, -45.0]
P3 bucket-flip $\varepsilon = 0.05$	-91.0	-36.8	[-41.7, -32.5]

implementation sanity gates confirmed the controller matches the specification before any perturbation cell was run.

Table V reports the outcome. The lag-induced difference shrinks by roughly $40\times$ when the bundled threshold is replaced by HPA-v2, and Figure 5 shows why. The production-grade controller does not oscillate under lag. The directional verdict survives in the narrow sense that Rossi is cheaper on all 30 lagged windows, but the magnitude is now the same order as its clean-cell advantage rather than an order larger. Across the four cells HPA-v2 wins three (clean, tail, adversarial) at -31.6 to -52.4 cost units, and loses only under lag. The published Rossi advantage therefore survives against an operationally realistic baseline under one perturbation class, and only by a small margin.

a) *Robustness to the controller’s configuration:* The 50% target and 300 s window of Table V are one configuration, and the outcome could depend on it. We therefore varied the target utilisation over $\{40, 50, 60, 70\}\%$ and the scale-down window over $\{300, 0\}$ s and re-ran all four cells. Rossi retains the lag cell only at the most cautious settings, $+62$ at a 40% target with full stabilisation, $+23$ at the reported point, and $+12$ at 40% without stabilisation. At a 50% target

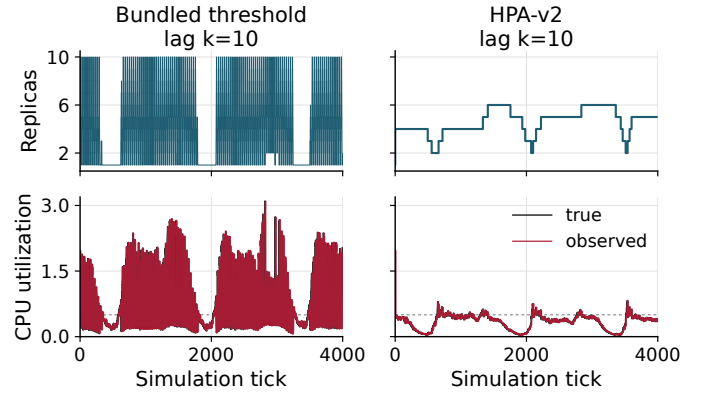


Fig. 5: The mechanism behind the $40\times$ difference, under observation lag $k = 10$ s. The bundled threshold (left) oscillates bang-bang between the replica bounds, with true utilisation repeatedly spiking to roughly $3\times$ overload. The HPA-equivalent controller (right) tracks the same workload within a narrow replica band and, after the initial transient, maintains a near-50% utilisation. The HPA-v2 configuration uses a 300s scale-down stabilisation window and a 10% tolerance deadband, which suppress scale-down responses to isolated lagged-low measurements.

without stabilisation, and at every target of 60% or above, the production controller wins even under lag (Figure 6). Across all eight configurations, the lag-cell divergence never exceeds $+62$ cost units, at least fifteen times below the bundled threshold’s $+965$ collapse. The only setting in which Rossi regains the clean cell is the most cautious one, a 40% target with full stabilisation, where the adversarial cell also becomes the grid’s only zero-straddling interval. The artefact is the bundled comparator’s collapse, not the production controller’s tuning, and no configuration reproduces it.

b) *The same check reframes the Decima result:* The bundled Decima comparator, `dynamic_partition`, has the symmetric weakness of never reallocating. We re-ran the four Decima cells against a dependency-aware shortest-remaining-work (SRW) scheduler that selects the arrived job with the least estimated remaining work, then its least-remaining ready node, and allocates executors work-conservatively. Decima beats this stronger comparator in all four cells, including under lag, by 1.4×10^5 (clean), 1.8×10^6 (lag), 4.1×10^5 (tail), and 1.5×10^5 (adversarial) mean-JCT units. These results delimit what the Decima cells are evidence of. They are not evidence that a better baseline overturns Decima, because against two classical schedulers it is not overturned. They are evidence of the other failure documented here. The comparison the published headline rests on, against tuned heuristics on a live testbed, is absent from the released artefact, so the reported 21% advantage cannot be evaluated from it. Against the strongest classical comparator its reference evaluation exposes, the margin is 3.0%. A method can be strong while its central published claim remains unverifiable. Decima exhibits both, and only the second is a claim this paper needs.

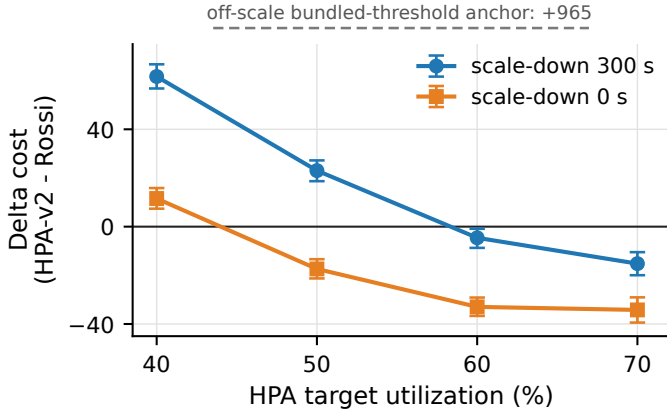


Fig. 6: Rossi’s lag-cell advantage across HPA-v2 configurations. Every configuration sits far below the bundled-threshold collapse (dashed, +965). Rossi retains the cell only at the most cautious settings, a 40% target or 50% with full stabilisation. At a 50% target without stabilisation, and at any target of 60% and above, the production controller wins even under lag. The grid maximum of +62 puts the bundled comparison’s overstatement above $15\times$ at every configuration.

E. Magnitude sensitivity

A conclusion drawn at a single anchor per perturbation could depend on the choice of that anchor. We therefore pre-registered a sweep of each perturbation over a grid of magnitudes and re-evaluated the fixed policies at every point. We committed in advance to report any verdict that reverses inside the realistic range, even where the reversal favours the explanation we argue against. Across the three methods the anchor verdicts are stable in sign with one exception. DeepRM stays ahead of both bundled comparators at every lag and adversarial magnitude in the realistic range, with the difference widening as the comparators degrade (Figure 7), and the one severe point, $\varepsilon = 0.2$, drove a non-terminating rollout and is reported as noncompletion outside the confirmatory range. For Rossi the result of Section IV-D strengthens. The HPA-equivalent controller stays within a small band of the learned policy across the entire lag grid. Its largest difference, about 101 cost units, is a tenth of the bundled threshold’s $k = 10$ collapse and stays below the pre-registered 25% alarm. The bundled collapse is itself an onset, with the threshold competitive at $k \leq 2$ and diverging only from $k \geq 5$ (Figure 8). The Rossi tail and adversarial verdicts are sign-stable across their grids.

The exception is Decima under observation lag, reported as registered (Figure 9). The verdict is not sign-stable inside the calibrated range. At $\lambda = 0.25$ the static comparator is significantly cheaper than Decima, with a Holm-adjusted p of 0.003 and 24 of 30 paired seeds favouring the comparator, while at the $\lambda = 1.0$ anchor and beyond Decima leads by large margins. We audited the lag sampler and found no rounding artefact, so the reversal is not a sampler artefact. Because each magnitude draws its own perturbation rather

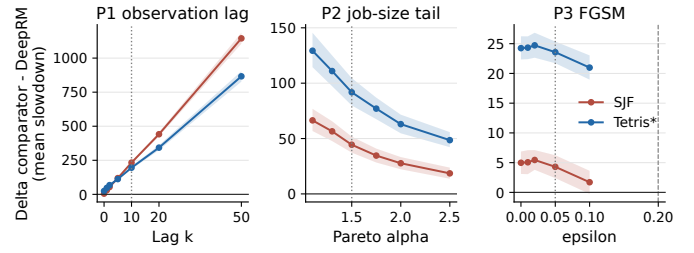


Fig. 7: DeepRM under the pre-registered magnitude sweeps, against the locked Tetris* and SJF comparators. The falsification direction is stable across the realistic range, the learned policy is ahead at every lag and every adversarial magnitude, and the divergence widens as the comparators degrade. The severe point $\varepsilon = 0.2$ is a non-terminating evaluation and is omitted. $\Delta = \text{metric}(\text{comparator}) - \text{metric}(\text{DeepRM})$, and the dotted line marks the calibrated anchor.

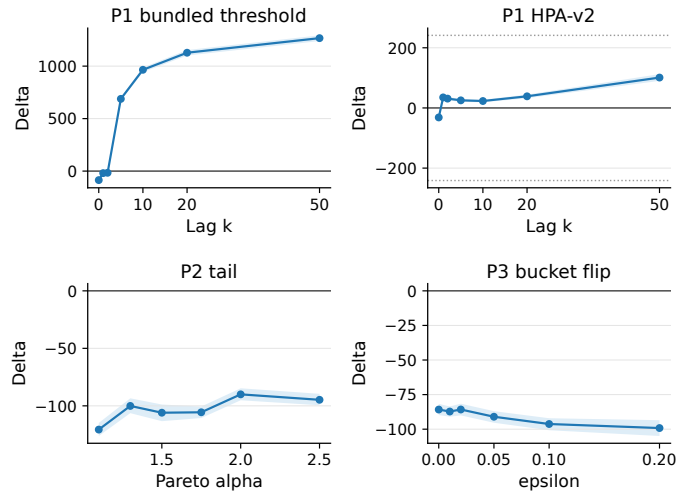


Fig. 8: Rossi across the pre-registered perturbation sweeps. The bundled threshold diverges for $k \geq 5$, while HPA-v2 remains close to Rossi across the lag grid (top). The tail and adversarial verdicts remain stable in sign (bottom). Dotted lines mark the 25% alarm threshold. $\Delta = \text{metric}(\text{comparator}) - \text{metric}(\text{Rossi})$.

than a scaled version of one draw, it is not a point on a smooth response curve, and we make no monotonicity claim for this cell. We therefore no longer state the Decima lag result as a magnitude-robust falsification, and the claim we retain is its anchor-level robustness against stronger comparators (Section IV-D). Decima’s other grids carry no such reversal. P3 is sign-stable across the full ε range, and P2 is marginal at the anchor and Holm-significant from $w \geq 0.75$. A pre-registered comparison that changes sign on the choice of a single magnitude demonstrates that single-anchor evaluation cannot settle the question it asks.

F. Two supporting analyses

Autocorrelation sensitivity (Rossi). The 30 Rossi windows are slices of one trajectory, so the paired bootstrap’s inde-

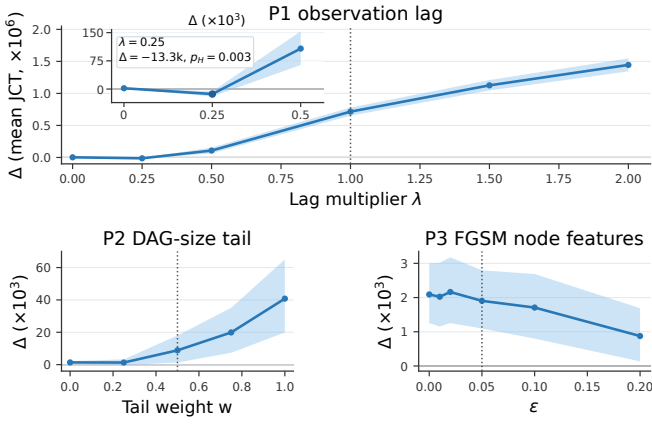


Fig. 9: Decima under the pre-registered magnitude sweeps, against the locked `dynamic_partition` comparator. P2 and P3 are sign-stable, with the P2 margin growing past its marginal anchor and the P3 falsification holding across the full ε grid. P1 is not. The inset shows the pre-registered $\lambda = 0.25$ cell, where the comparator is significantly cheaper ($\Delta = -13,305$, CI $[-19,833, -6,168]$, $p_H = 0.003$, 24 of 30 seeds) while every other magnitude favours Decima by up to 1.4×10^6 JCT units. A verdict that reverses on the choice of a single magnitude is the failure mode single-anchor evaluation cannot detect. $\Delta = \text{JCT}(\text{dynamic_partition}) - \text{JCT}(\text{Decima})$, and dotted lines mark the calibrated anchors.

pendence assumption is only approximate. A pre-registered block-bootstrap analysis confirms the verdicts. Lag-1 auto-correlations are below 0.25 in all four cells, no Ljung-Box independence null rejects, and moving-block intervals at $L \in \{5, 10\}$ never change a cell's zero-containment status. The block sign-flip test is uninformative at $N = 30$, because its discreteness floors the p -value at 0.125 for $L = 10$, so the moving-block intervals are the better summary. In the P1 cell those intervals are centred more than 100 standard errors from zero.

Action-redundancy ablation (DeepRM). The decision-versus-outcome separation in P3 invites a redundancy explanation, in which many near-equivalent actions per state mean that flipping the `argmax` rarely selects a catastrophic alternative. We test this by training DeepRM under restricted visible action sets $M \in \{10, 3, 1\}$ at the source-aligned budget. Reducing M from 10 to 3 keeps DeepRM competent against the matched comparator and produces no detectable increase in FGSM aggregate degradation, with deg_3 and deg_{10} both statistically indistinguishable from zero and the observed ratio opposite in sign to the prediction. The $M = 10$ condition reproduces the locked P3 aggregate exactly while sampling a different diagnostic state set, which is why its `argmax-change` rate (75.2%) differs from the locked diagnostic (69.8%, Section IV-C). The $M = 1$ condition fails the competency gate, which reflects training failure under extreme action restriction rather than a robustness mechanism. The redundancy explanation is not supported. The structural origin of the separation remains

open, and that is exactly the kind of question a typed action space (Section II-B) would make tractable by construction.

G. Limitations

The findings hold at the anchors tested, and Section IV-E shows the verdict signs are stable across magnitude in every cell but one, the Decima lag cell, whose sign we no longer claim as magnitude-robust. The scope is three methods, and two of them, DeepRM and Decima, share an author lineage. Each was selected as the canonical, most built-on instance of its task class, and breadth across groups is future work. The structural origin of the action-versus-outcome robustness difference (Section IV-C) is unresolved. Most consequentially, the public simulators capture queueing, packing, and arrivals but not the contention, tail-latency dynamics, micro-bursts, or scheduling jitter that dominate production performance. This bounds what the programme can claim, and it does so asymmetrically. It would undercut an absolute-performance claim, which we do not make. It does not undercut the claims we do make, because both kinds are independent of simulator fidelity. Whether an artefact can express the comparison its headline rests on is a property of the artefact, not of its fidelity, and a relative comparison between two policies on the same trajectory holds whatever that trajectory omits. Within this bound, three results remain. The pre-registered predictions do not survive against the bundled comparators, the strongest result attenuates by more than an order of magnitude against a production-grade controller robust to its configuration, and a second method withstands stronger comparators at its anchor while its published headline remains unverifiable.

V. DISCUSSION

A. Interpreting the adversarial correspondence

Production telemetry carries the magnitude of bounded observation perturbations, workload shifts have the structure of environment poisoning, and scalar proxies satisfy the pre-conditions of reward poisoning, so the correspondence in Table II describes operating conditions rather than hypothetical attacks. Our use of the adversarial-RL literature [7], [13], [8], [9], [10] is a reframing rather than a new attack or defence, and Section IV subjects that reframing to a pre-registered test on canonical orchestration methods. The test refutes the directional claim. The learned policy is not the party that degrades against the comparators the field uses, and the strongest case for it dissolves against a production-grade baseline.

B. Key performance indicators selection

The Rossi cell is a small instance of a general failure, in which a weighted-sum reward becomes the metric and the weights, not the behaviour, set its scale. The published cost weights SLA compliance at 0.90, resource cost at 0.09, and reconfiguration churn at 0.01, pricing churn two orders of magnitude below the SLA term. The released traces show the consequence. In the clean cell the learned policy issues roughly 1,500 reconfigurations per window against the

threshold controller's 56, a $27\times$ difference the scalar renders invisible. Re-pricing the churn to a still-modest 0.30 widens the threshold controller's clean-cell advantage over the learned policy from 99 to 487 cost units, a difference a reader of the published number never sees. Across the full re-pricing grid no cell's verdict flips, so the weight chooses the margin rather than the winner. A scalar cost hides the trade-off it encodes. Orchestration results should be reported in operational terms, with SLA violations, action churn, overload peaks, and tail latency kept distinct, so that a reader can apply the weighting their own deployment implies rather than inherit the authors'.

C. Limits of benchmark evidence

Current orchestration benchmarks model queueing, packing, and arrival processes but omit resource contention, cross-tenant interference, microbursts, scheduling jitter, and component failures. Performance gains measured without these conditions do not establish deployment readiness. A credible benchmark should include failure scenarios, delayed and noisy telemetry, heavy-tailed and bursty workloads, tenant churn, and bounded observation perturbations. It should also compare learned controllers with production-grade baselines.

D. Scope and requirements for valid evaluation

This paper neither proposes a comprehensive evaluation standard for learned orchestration nor presents a method that satisfies such a standard. It puts an effort into isolating the deficiencies that prevent current studies from supporting deployment-relevant conclusions and identifies the evidence that a valid evaluation would require. We argue that a valid evaluation requires four technical components. First, the evaluation infrastructure should combine a multi-tenant testbed with extensible simulation or emulation. End-to-end application suites such as DeathStarBench provide realistic service dependencies and request paths [48], while trace-driven frameworks such as faas-sim support controlled evaluation of scaling, placement, routing, and scheduling across heterogeneous containerised and serverless infrastructures [49]. Physical execution is still required to expose contention, scheduling jitter, and other effects that cannot be validated exclusively through simulation. Second, the workload corpus must cover several distributional and temporal profiles derived from production traces [14], [15]. Third, the comparator set must include both academic methods and deployed controllers, including DeepRM, Decima, default Kubernetes scalers, Autopilot's recommenders [1], and structurally constrained methods such as HPAQT [50]. Fourth, results must be reported by task class and by operational metric, in alignment with Table I.

These components also require changes to publication and review criteria. Evaluations should justify comparator selection and perturbation models before testing, report operational metrics separately, release sufficient material for reproduction, and treat negative results as evidence. Absent these criteria, successive benchmark improvements remain isolated results rather than cumulative evidence of operational validity.

E. Outlook

The findings above do not imply that learning has no role in orchestration. They indicate that the formulation should begin with the system's operational structure rather than a benchmark-defined Markov decision process. One proposed direction is a computing-continuum system defined by application requirements, observable metrics, infrastructure configurations, and admissible adaptations [39]. A Markov blanket links these elements, while equilibrium denotes an operating state consistent with the system requirements and available infrastructure. Under this formulation, learning estimates system dependencies and selects reconfigurations within a structure defined before training. It does not determine the operational objective or invent the actions through which the system is controlled.

Danilenka *et al.* provide an initial instantiation of this direction in heterogeneous and lifelong federated learning [51]. Their agents represent SLO fulfilment, configuration variables, and system measurements in a learned Bayesian network, then select configurations under data drift and hardware heterogeneity. The physical-testbed results show that high-level SLOs and explicit configuration choices can be combined with a learned environment model. They do not establish that active inference solves the service orchestration problem. The application scope is narrower, the configuration space is enumerated, and scalability, temporal modelling, and more complex SLO specifications remain open. Extending this direction to orchestration requires three commitments. Objectives should follow from operational contracts, represented as hard constraints through constrained MDPs [52] or as preferred outcome distributions, rather than as freely weighted scalar rewards. Actions should be typed against control-plane interfaces, including their preconditions, costs, reversibility, and feasibility. Evaluation should compare learned controllers with production-grade controllers under registered perturbations, with SLO violations, reconfiguration churn, overload, and tail latency reported separately. Robust-MDP methods [43], [44], [45], [53] provide one way to formalise uncertainty, but they do not determine the correct objective, action space, or evidence standard. Therefore, progress requires methods whose objectives and admissible actions are grounded in system contracts and whose operational validity is established against production-grade controllers under registered perturbations.

VI. CONCLUSION

We examined whether production conditions explain why reinforcement learning has not reached service orchestration at scale. Across three influential systems, the registered tests did not produce a stable verdict on learned control. Several apparent advantages arose from weak comparators or perturbation implementations. One advantage decreased by roughly fortyfold against a production-grade controller. Another published claim could not be reconstructed from the released artefact. Other conclusions changed with perturbation magnitude or evaluation mode. These results do not establish that learned controllers withstand production conditions, but

demonstrate that the available comparisons cannot reliably determine whether they do.

Based on these results and the broader literature, we identify a methodological and institutional problem. Research and review incentives favour benchmark improvements over available comparators, even when the comparison lacks an operational win condition and provides little evidence of deployment performance. This prevents results from accumulating into a reliable account of when learning improves orchestration. Addressing this problem requires changes to both evaluation practice and how contributions are judged. Learned controllers should be compared with production-grade controllers under registered perturbations. Operational outcomes should be reported separately rather than combined through author-selected scalar weights. Released artefacts should reproduce the claims for which a method is cited. Objectives and actions should also follow from system contracts and control-plane interfaces rather than from benchmark conventions. These requirements do not determine which learning method should replace current controllers. They establish the evidence needed to decide whether any method should be used. Until publication incentives reward production-relevant evidence, additional funding, computation, and publication volume will expand the literature without establishing whether reinforcement-learning-based anticipatory control improves service orchestration under production conditions.

ACKNOWLEDGMENT

We thank Alexander Knoll for providing us with the necessary hardware infrastructure. This work has been supported by the Marie Skłodowska-Curie Actions Doctoral Networks (DN) under call HORIZON-MSCA-DN-2024-01-01 (Project “SAILING”).

REFERENCES

- [1] K. Rzacda, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, and J. Wilkes, “Autopilot: Workload autoscaling at Google,” in *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*. ACM, 2020, pp. 1–16.
- [2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*. ACM, 2015, pp. 1–17.
- [3] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016.
- [4] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets)*. ACM, 2016, pp. 50–56.
- [5] H. Mao, M. Schwarzkopf, S. B. Venkatakrisnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proceedings of the ACM SIGCOMM 2019 Conference*. ACM, 2019, pp. 270–288.
- [6] F. Rossi, M. Nardelli, and V. Cardellini, “Horizontal and vertical scaling of container-based applications using reinforcement learning,” in *Proceedings of the IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 329–338.
- [7] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial attacks on neural network policies,” in *ICLR 2017 Workshop Track*, 2017.
- [8] H. Zhang, H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C.-J. Hsieh, “Robust deep reinforcement learning against adversarial perturbations on state observations,” in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020, pp. 21 024–21 037.
- [9] A. Rakhsha, G. Radanovic, R. Devidze, X. Zhu, and A. Singla, “Policy teaching via environment poisoning: Training-time attacks on reinforcement learning,” in *Proceedings of the 37th International Conference on Machine Learning (ICML)*. PMLR, 2020, pp. 7974–7984.
- [10] V. Behzadan and A. Munir, “Vulnerability of deep reinforcement learning to policy induction attacks,” in *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference (MLDM)*, ser. Lecture Notes in Computer Science, vol. 10358. Springer, 2017, pp. 262–275.
- [11] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, “Implementation matters in deep policy gradients: A case study on PPO and TRPO,” in *Proceedings of the 8th International Conference on Learning Representations (ICLR '20)*, 2020.
- [12] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, “Deep reinforcement learning at the edge of the statistical precipice,” in *Advances in Neural Information Processing Systems 34 (NeurIPS '21)*, 2021, pp. 29 304–29 320.
- [13] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, “Adversarial policies: Attacking deep reinforcement learning,” in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.
- [14] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and dynamicity of clouds at scale: Google trace analysis,” in *Proceedings of the 3rd ACM Symposium on Cloud Computing (SoCC)*. ACM, 2012.
- [15] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, “Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms,” in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. ACM, 2017, pp. 153–167.
- [16] J. Dean and L. A. Barroso, “The tail at scale,” *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.
- [17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
- [18] C. J. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992.
- [19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [20] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.
- [21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 1861–1870.
- [23] G. Tesauro, N. K. Jong, R. Das, and M. N. Bannani, “A hybrid reinforcement learning approach to autonomic resource allocation,” in *Proceedings of the 3rd IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2006, pp. 65–73.
- [24] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-efficient and QoS-aware cluster management,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2014, pp. 127–144.
- [25] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of deep reinforcement learning in communications and networking: A survey,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.
- [26] H. Mao, P. Negi, A. Narayan, H. Wang, J. Yang, H. Wang, R. Marcus, R. Addanki, M. Khani Shirkoohi, S. He, V. Nathan, F. Cangialosi, S. B. Venkatakrisnan, W.-H. Weng, S. Han, T. Kraska, and M. Alizadeh, “Park: An open platform for learning-augmented computer systems,” in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, pp. 2490–2502.
- [27] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, “FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020, pp. 805–825.

- [28] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: ML-based and QoS-aware resource management for cloud microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 167–181.
- [29] T. Llorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.
- [30] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–33, 2018.
- [31] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–25, 2021.
- [32] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28 (NIPS '15)*, 2015, pp. 2503–2511.
- [33] M. Ferrari Dacrema, P. Cremonesi, and D. Jannach, "Are we really making much progress? A worrying analysis of recent neural recommendation approaches," in *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys)*. ACM, 2019, pp. 101–109.
- [34] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018, pp. 1800–1809.
- [35] W. Zhao, J. P. n. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," *arXiv preprint arXiv:2009.13303*, 2020.
- [36] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
- [37] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 2778–2787.
- [38] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- [39] S. Dustdar, V. Casamayor Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4092–4105, 2023.
- [40] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.
- [41] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, and Y. Xiong, "SLOC: Service level objectives for next generation cloud computing," *IEEE Internet Computing*, vol. 24, no. 3, pp. 39–50, 2020.
- [42] V. Casamayor Pujol, B. Sedlak, T. Salvatori, K. Friston, and S. Dustdar, "Distributed intelligence in the computing continuum with active inference," *arXiv preprint arXiv:2505.24618*, 2025.
- [43] G. N. Iyengar, "Robust dynamic programming," *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.
- [44] A. Nilim and L. El Ghaoui, "Robust control of Markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.
- [45] W. Wiesemann, D. Kuhn, and B. Rustem, "Robust Markov decision processes," *Mathematics of Operations Research*, vol. 38, no. 1, pp. 153–183, 2013.
- [46] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the ACM SIGCOMM 2014 Conference*. ACM, 2014, pp. 455–466.
- [47] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI '18)*, 2018, pp. 3207–3214.
- [48] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zanis, A. Vizzini, D. Petrovska, A. Phanishayee, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud and edge systems," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019, pp. 3–18.
- [49] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar, "faas-sim: A trace-driven simulation framework for serverless edge computing platforms," *Software: Practice and Experience*, vol. 53, no. 12, pp. 2327–2361, 2023. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3277>
- [50] R. Mayerhofer, A. Morichetta, A. Furutanpey, and S. Dustdar, "HPAQT: Adaptive and interpretable high-level SLO-aware autoscaling with reinforcement learning," in *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '25)*, 2025, pp. 19:1–19:9.
- [51] A. Danilenka, A. Furutanpey, V. C. Pujol, B. Sedlak, A. Lackinger, M. Ganzha, M. Paprzycki, and S. Dustdar, "Adaptive active inference agents for heterogeneous and lifelong federated learning," *arXiv preprint arXiv:2410.09099*, 2024.
- [52] E. Altman, *Constrained Markov Decision Processes*. Boca Raton, FL: Chapman & Hall/CRC, 1999.
- [53] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, ser. PMLR, vol. 70, 2017, pp. 2817–2826.