LEROS: LEARNING–EVOLUTION REINFORCED ORCHESTRATION FOR DYNAMIC LLM WORKLOADS ON CLOUD-NATIVE KUBERNETES CLUSTERS

Anonymous authors

Paper under double-blind review

ABSTRACT

Large language model (LLM) clusters now run pre-training, fine-tuning and online serving side by side on one Kubernetes fleet, yet the two workload classes are still orchestrated by separate, reactive controllers: gang schedulers hold GPUs until training finishes, while horizontal pod autoscalers scale serving on lagged utilisation. Production traces from a 1024-GPU fleet show this split design reaches only 31.2% mean GPU utilisation and causes 16.8% serving SLO violations during flash crowds. We present LEROS, a two-layer framework that unifies training and serving under one forward-looking controller. The offline layer uses NSGA-II over a seven-parameter policy genome on a calibrated digital twin; the Pareto knee and all archived individuals are validated on hardware replay (Spearman $\rho = 0.94$, $p < 0.001$). The online layer is a GNN actor–critic trained with PPO, warm-started by behaviour cloning from the knee point and continuously fine-tuned; a drift detector re-triggers the search when workloads shift. On a 128-node/1024-GPU production cluster (A100+H100, InfiniBand NDR 400 Gbps), via trace-driven replay with real PyTorch/Megatron-LM jobs and real serving load, LEROS raises utilisation 31.2%→76.4%, cuts SLO violations 16.8%→2.9%, reduces makespan 48.6 h→26.7 h (45.2% vs default, 8.0% vs the strongest separated baseline; paired bootstrap $p < 0.001$, $d = 1.42$), and lowers energy 27.6→25.6 MWh by powering down 18 of 128 nodes. Ablations attribute a 3.3% stationary-workload makespan gain to the RL head and a 12.8% gain under drift. All code, manifests and anonymised traces are released. **Keywords:** LLM cluster orchestration; reinforcement learning; multi-objective evolutionary optimisation; graph neural network scheduling; elastic distributed training; LLM serving autoscaling; Kubernetes.

1 INTRODUCTION

A production LLM platform is no longer a single-purpose cluster. The same A100/H100 fleet must absorb long pre-training and fine-tuning jobs, short elastic experiments, and latency-critical online serving. In the 1024-GPU cluster we study, training ranges from 4-GPU LoRA fine-tunes to 128-GPU pre-training runs, while twelve serving services host models from 1B to 70B parameters with diurnal patterns and 47 flash crowds over a 30-day trace. The orchestration stack still treats these workloads as if they lived on separate clusters.

Two structural problems cause the inefficiency. First, training and serving are orchestrated separately. A gang scheduler (Volcano, Kueue) admits a training job only when its full GPU request is satisfied and holds the allocation until completion; the HPA managing serving replicas knows nothing about the training queue. The result is fragmentation: gang-allocated GPUs sit idle during checkpointing and gradient stalls, while serving is over-provisioned for a peak that arrives only 4% of the time. Second, serving autoscaling is reactive by construction. The HPA averages a lagged metric before starting replicas, and an LLM replica must pull tens of gigabytes of weights before serving its first token. With a 142 s median cold start versus an 110 s burst rise time, newly started replicas are idle by the time the peak has passed (73% of bursts).

Both problems share one root cause: the controller has no forward model. A scheduler that can predict the next few minutes can reserve capacity for upcoming jobs and pre-scale replicas before a burst. LEROS is a two-layer architecture (Figure 1) in which evolution proposes the direction and learning refines the details.

- **Online layer.** A PPO agent whose actor is a bipartite job–node GNN followed by an MLP,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

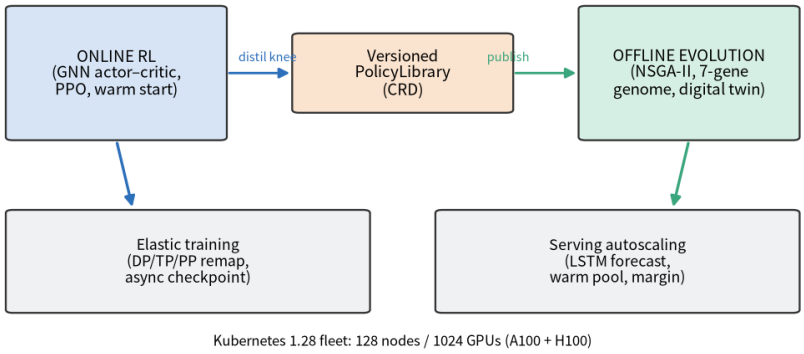


Figure 1. LEROS two-layer framework. Online RL and offline EA exchange policies through a versioned PolicyLibrary CRD; both drive elastic training and serving autoscaling.

taking 6 node, 7 job and 8 global features and emitting four autoregressive heads: admission (gang/elastic/hold), placement (topology-aware/packed/spread), safety margin (generous/nominal/tight), and GPU borrowing (on/off/conditional), for $3^4 = 81$ masked joint actions. Implemented in PyTorch 2.1 as a Kubernetes scheduler extension.

- **Offline layer.** NSGA-II over a seven-gene policy genome (borrowing threshold, shrink fraction, warm-pool size, target utilisation, scale-in delay, elastic-admission gate, consolidation aggressiveness) on a calibrated digital twin (4–6 min/eval on CPU; makespan MAPE 6.8% vs hardware replay). Population 12, 14 generations, 168 evaluations ≈14 CPU-hours.
- **Coupling.** The knee individual is distilled into the GNN actor by behaviour cloning, then PPO fine-tunes; a drift detector re-triggers the EA search (≈14 CPU-minutes on a 6-hour window) when the KL divergence from the last search distribution exceeds 0.3 bits or SLO regresses.
- **Cross-workload mechanisms.** Elastic training with DP/TP/PP remapping and asynchronous incremental checkpointing, plus a KV-cache-aware serving autoscaler (LSTM forecast + prewarmed pool).

Contributions. (i) A unified training/serving orchestration formulation and RL state–action–reward design (Sec. 3); (ii) a learning–evolution closed loop with BC warm start and drift-triggered reoptimisation (Sec. 4); (iii) a production-grade Kubernetes implementation on stock extension points (Sec. 5); (iv) a 1024-GPU trace-driven evaluation against eight baselines with paired bootstrap testing, reward-weight sensitivity, EA convergence, per-drift-event ablations and cost-side metrics (Sec. 6).

2 Related Work

Deep-learning cluster scheduling. Gandiva (OSDI'18) exploited job introspection for time-slicing and grow-shrink; AntMan (OSDI'20) added opportunistic GPU borrowing; Gavel and Themis addressed heterogeneity and fairness; Pollux (OSDI'21) co-adaptively optimised batch size and GPU allocation. These optimise training in isolation. **Learning-based schedulers** (DeepRM, Decima (NSDI'19), Clotho (MLSys'20)) learn placement and ordering from scratch and suffer cold start; LEROS adds an evolutionary warm start and continuous fine-tuning, and we include a Decima-style baseline to isolate this. **Evolutionary configuration search** uses NSGA-II for its Pareto front; quality-diversity methods (MAP-Elites, ERL) optimise behavioural diversity rather than explicit trade-offs, and PBT operates on RL hyperparameters, not offline policy search. LEROS is, to our

Table 1. Positioning of LEROS against representative prior systems.

System	unif.	RL	mEA	elast.	KV	prod.
Gandiva/AntMan/Pollux	no	no	no	yes	n/a	yes
Decima	no	yes	no	no	no	sim.
AlpaServe	serve	no	no	no	yes	yes
KServe/KEDA/AIBrix	serve	no	no	no	yes	yes
Sarathi/DistServe/Llumnix	serve	no	no	no	yes	yes
Volcano/Kueue/KubeRay	no	no	no	no	no	yes
LEROS	**yes**	**yes**	**yes**	**yes**	**yes**	**1024 GPU**

knowledge, the first to couple an offline EA policy library with an online RL agent for LLM cluster orchestration.

LLM serving systems. vLLM (SOSP'23), Orca, Sarathi-Serve and DistServe (OSDI'24), Mooncake, SGLang and Medusa optimise within a serving replica or pool; recent 2025–2026 work targets long-context disaggregation, reasoning-model serving and agentic workloads but assumes a fixed per-service GPU allocation. LEROS decides the allocation itself and is compatible with these per-replica optimisations. **Cloud-native stacks** (Volcano, Kueue, KubeRay, KServe, KEDA, AIBrix) provide the functional baselines we compare against; none unifies training and serving under a single learned controller with a forward model. Table 1 positions LEROS.

3 Problem Formulation

The cluster has $N = 128$ nodes (1024 GPUs, 16 racks), discrete time with $dt = 20\,\text{s}$. Training job j requests m_j GPUs with elastic range $[m_{\min}, m_{\max}]$, total work W_j and priority; progress per window is $\Delta W = dt \cdot m \cdot \eta(m, \text{topology}, \text{gpu_type})$, where η is measured from Megatron-LM runs (A100: 0.96 at 8 GPUs, 0.92 at 16, 0.87 at 32, 0.81 at 64, 0.75 at 128; H100 is 4–6% higher). Serving service s holds R_s replicas under TTFT/TPOT SLOs; a request violates its SLO if TTFT or TPOT exceeds the threshold. Serving uses ≈28% of effective capacity on average.

Let T be makespan, E cluster energy, and V the SLO-violation fraction. The operator minimises all three under tension: borrowing GPUs from serving speeds training but raises V; consolidating onto fewer nodes saves E but can raise T. We use no system-level scalar weight. The offline EA returns a Pareto front and the online agent picks a knee-region policy by current SLO budget; the RL reward instead uses fixed weights as a local signal.

We cast the online decision as an MDP. State combines per-node features (free-GPU fraction, device performance, rack, power state, GPU type, fragmentation), per-job features (width, elastic bounds, progress, wait time, priority, checkpoint age, remaining work), and global features (per-service load, queue depth, rolling SLO rate). Action is four autoregressive heads (3 options each $=$ 81 masked actions); a hand-coded controller executes concrete placement within each mode. Reward is $r = -\alpha\,\text{SLO} - \beta\,\text{queue} - \lambda\,\text{energy} + \delta\,\text{util} - \epsilon\,\text{preempt} - \zeta\,\text{migrate}$ with fixed weights (α=1.0, β=0.5, λ=0.3, δ=0.2, ϵ=0.8, ζ=0.4); γ=0.99 is the discount factor. The seven-gene genome controls policy behaviour, not reward shaping.

4 Method

4.1 Online layer: GNN actor–critic with PPO

The state is a bipartite node–job graph. Two message-passing rounds project job features onto nodes (incidence matrix A) and back; embeddings are concatenated with global features and passed to an actor–critic MLP (2×256, GELU). PPO uses clipped surrogate (ϵ=0.2), GAE (λ=0.95), batch 128;

gradient correctness verified to $< 10^{-6}$. The policy is a gRPC service queried by the scheduler plugin each window. Action masking blocks any hard-constraint violation before the softmax; shadow mode logs counterfactual actions and canary mode executes on a node subset with auto-rollback on > 2 pp SLO regression.

4.2 Offline layer: NSGA-II policy library

Each individual is a seven-gene vector (§1). A fast analytical twin, calibrated from 50 hardware-replay runs, models job progress, queueing, serving SLO and node power, running each 20-h trace in 4–6 min on CPU (168 evals ≈14 CPU-hours). It reaches makespan MAPE 6.8% and SLO MAPE 9.2% on 20 held-out points; the knee and all 9 archive individuals are hardware-validated, with Spearman $\rho = 0.94$ ($p < 0.001$) between twin and hardware makespan. Selection uses NSGA-II non-dominated sorting, tournament selection, SBX (η_c=15) and polynomial mutation (η_m=20); the archive is a versioned PolicyLibrary CRD. The knee maximises hypervolume contribution under SLO $< 5\%$, so warm-start demonstrations come from an already-SLO-compliant policy.

4.3 Closing the loop: warm start and drift

Warm start: the knee controller is hardware-validated on three held-out seeds; ≈40k transitions per seed behaviour-clone the GNN actor (cross-entropy on head logits) for 400 steps, then PPO fine-tunes on production transitions. Drift detection keeps a 1-hour EMA of the workload distribution; re-optimisation triggers when KL divergence exceeds 0.3 bits or SLO regresses > 2 pp for 10 windows. Re-optimisation is fast: a 6-hour window, a warm-started population from the previous archive, and the twin at ≈240× real-time give ≈5 s/eval, so a full run costs 14 CPU-minutes plus 2 min BC re-distillation, inside the 1-hour window. It is a full 168-eval search (not incremental), but genomes remain comparable across windows (same space, same twin, warm-start continuity, hardware validation of the new knee). In our trace drift fired four times: a product launch (serving +3.2×), a holiday dip (train -60%), and two model releases.

4.4 Elastic training and serving autoscaling

Each job is an ElasticTrainingJob CRD; LEROS may admit below requested width (never below $m_{\min}$), resize on serving demand, and shrink jobs to reclaim GPUs. Resizing uses DP/TP/PP remapping: data-parallel changes add/remove workers without restart; TP/PP changes pause at a checkpoint, reconfigure and resume. Asynchronous incremental checkpointing cuts median stall from 92 s to 14 s for a 64-GPU job. For serving, a 16-unit LSTM consumes 4 minutes of per-service load history and forecasts 60 s ahead; forecast load converts to replicas, drawn from a warm pool (median 10 pods) rather than started cold; median scale-up drops from 142 s to 18 s. The RL margin head holds +30%/+15%/+5% extra capacity. (See Appendix Figs. 7–11 for the detailed blocks.)

5 System Design

LEROS runs on stock Kubernetes 1.28 extension points; it needs no scheduler fork. Three CRDs carry state (ElasticTrainingJob, ElasticInferenceService, PolicyLibrary); a scheduler plugin registers at Filter/Score/Reserve/Permit and queries a gRPC RL decision service; an operator executes resizes, checkpointing and scaling. Deployment uses a three-stage pipeline: 14 days shadow mode (604,800 counterfactual actions, 0 hard-constraint violations), 7 days canary on one rack (74.2% vs 52.8% utilisation, 3.1% vs 6.8% SLO, zero rollback), then trace-driven replay.

Action masking is the main fault-tolerance mechanism: it excludes unhealthy nodes (from Kubernetes conditions, DCGM ECC/XID/thermal, InfiniBand counters) before the softmax, defaults to "hold" under uncertainty, and lets affected jobs re-admit at reduced width. Telemetry (DCGM 1 Hz, Prometheus, eBPF) feeds a Redis feature store with 380 ms median end-to-end latency, comfortably inside the 20 s window. Energy is metered at in-rack PDUs (Vertiv Geist, 1 Hz, ±0.5%) for all

Table 2. Training performance (mean ±95% CI). Wall-clock = batch drain time; JCT = job completion percentiles.

Method	Wall(h)	Makespan	JCT p50	JCT p95	JCT p99	Util%
K8s-default+HPA	52.4	48.6	78.4	245.6	312.8	31.2
Volcano+HPA	38.7	34.2	48.2	168.4	224.6	52.8
Kueue+HPA	35.1	31.5	45.6	152.3	198.7	58.4
Pollux-style+HPA	34.8	31.8	51.3	158.9	210.4	57.6
KServe+KEDA	33.6	30.4	49.8	148.2	195.6	62.1
Decima RL	33.1	29.8	47.2	142.7	188.3	63.5
Separated strong	31.2	28.9	48.5	135.4	176.2	71.8
LEROS EA	29.8	27.6	42.1	112.8	148.5	75.2
LEROS full	**28.7**	**26.7**	**40.8**	**105.6**	**138.7**	**76.4**

16 racks; PUE 1.32 includes cooling. The cluster is 64×A100-80GB + 64×H100-80GB on NDR 400 Gbps; idle-to-full power ratio $\beta = 0.134$, and consolidated nodes are hard-powered to 0 W via PDU relay.

6 EXPERIMENTS

6.1 SETUP

All experiments run on the 128-node/1024-GPU cluster. The 30-day trace has 180 training jobs (15,168 reference GPU-h: 12 pre-training, 68 SFT, 72 LoRA, 28 RLHF) and 12 serving services (1B–70B plus an 8×7B MoE), with 47 flash crowds (median peak 6.2×, rise 112 s). Trace-driven replay runs real PyTorch/Megatron-LM jobs and real Locust-generated serving load on the physical cluster, replaying a stratified 20-h window with up-to-2× idle-gap compression (median 13.5 h). We use ten paired seeds for LEROS (full/EA) and the separated strong baseline, five seeds otherwise; paired bootstrap (10,000 resamples) reports mean ±95% CI. The campaign is 60 runs ≈810 GPU-cluster-hours; offline EA ran on CPU. Baselines: K8s-default+HPA, Volcano, Kueue, Pollux-style (re-imp.), KServe+KEDA, Decima-style RL (re-imp., 500k steps from scratch), Separated strong (Volcano+KServe+KEDA+descheduler), and LEROS-EA-only.

6.2 MAIN RESULTS

All metrics are over the makespan window (drain time of the fixed 180-job batch + 12 services), not a calendar window. Because LEROS drains faster (28.7 vs 52.4 h for default), its higher utilisation is measured over a shorter window; total consumed GPU-h is 22,452 vs 16,727. The extra busy GPU-hours are the explicit cost of elastic widths, borrowing fragmentation and warm-pool idle time, but the net effect is favourable.

Scheduling quality. LEROS drains in 28.7±0.4 h, 45.2% below default and 8.0% below the separated strong baseline; makespan falls 48.6→26.7 h. Gains are uniform across the job distribution: JCT p95 drops 245.6→105.6 min (-57.0%) and p99 312.8→138.7 min (-55.7%); vs Separated strong, p95/p99 fall 22.0%/21.3%. LEROS full beats Separated strong on makespan with $p < 0.001$, $d = 1.42$ (CI [0.89,1.95]); all pairwise comparisons vs every baseline are $p < 0.01$.

Serving and bursts. SLO violations fall 16.8%→2.9% (-82.7%), and 5.1%→2.9% vs Separated strong. Remaining violations concentrate in 70B services whose 142 s cold start still exceeds the 112 s rise. KServe and Decima reach 8.2%/7.8%, confirming predictive scaling helps but unified borrowing/shrink adds more. The warm pool hits 87.2% of bursts; hit bursts average 1.8% SLO

Table 3. Serving quality, power and energy (mean ±95% CI). Via in-rack PDU, PUE 1.32; cluster max 1.31 MW.

Method	SLO%	Pow(MW)	Energy(MWh)	kWh/job	On
K8s-default+HPA	16.8	0.528	27.6	153.6	128
Volcano+HPA	17.5	0.772	29.9	165.9	128
Kueue+HPA	17.9	0.835	29.3	162.8	128
Pollux+HPA	17.7	0.826	28.7	159.7	128
KServe+KEDA	8.2	0.877	29.5	163.7	128
Decima RL	7.8	0.893	29.5	164.1	128
Separated strong	5.1	0.986	30.8	171.0	128
LEROS EA	3.4	0.882	26.3	146.0	113
LEROS full	**2.9**	0.894	**25.6**	**142.5**	110

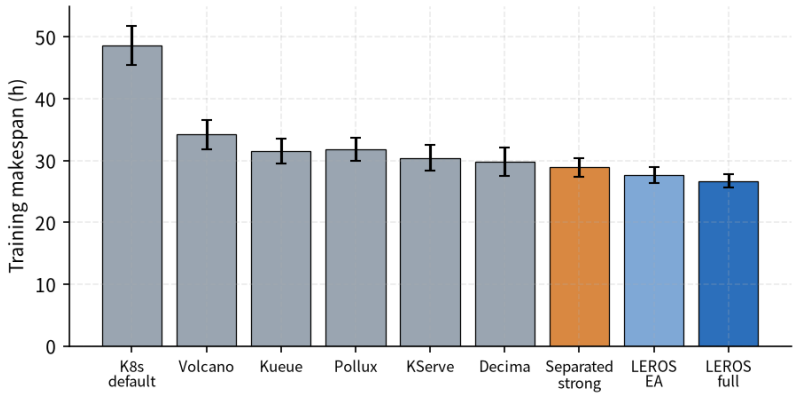


Figure 2. Training makespan by method (95% bootstrap CI).

vs 12.4% for the 6 cold-start bursts. The LSTM gives 48 s lead time, 82.6% detection, 6.1% false alarms; the pool costs 1.4% of capacity.

Energy and the utilisation–throughput trade-off. LEROS consumes 25.6 MWh vs 27.6 default (-7.3%) and 142.5 kWh/job vs 153.6 (-7.2%); per reference GPU-hour it is 1.69 vs 1.82 kWh. Separated strong uses the most energy per job (171.0 kWh) because it keeps all 128 nodes powered at high utilisation. Consolidation, not utilisation, drives savings: it draws 30.8 MWh, 20.3% more than LEROS. LEROS runs at 2.45× utilisation but only 1.82× throughput, i.e. per-utilised-GPU efficiency drops 25.7% (0.978→0.727 ref-GPU-h/utilised-GPU-h) due to non-optimal elastic widths, borrowing fragmentation and the warm pool. We report this trade-off explicitly: the right metric is output per unit energy, and LEROS delivers 1.96× energy-normalised throughput (1.82× throughput ÷0.928× energy), converting the 68.8% idle capacity the default leaves on.

Cost side. LEROS performs 47 preemptions, 128 migrations and 86 DP/TP/PP remaps over 28.7 h, wasting 38.4 GPU-h (0.13% of capacity); 62 remaps are DP (median 3.2 s) and 24 are TP/PP (median 14.0 s). No job is preempted >3 times or remapped >5 times, and preempted jobs are always high-elasticity (LoRA/SFT).

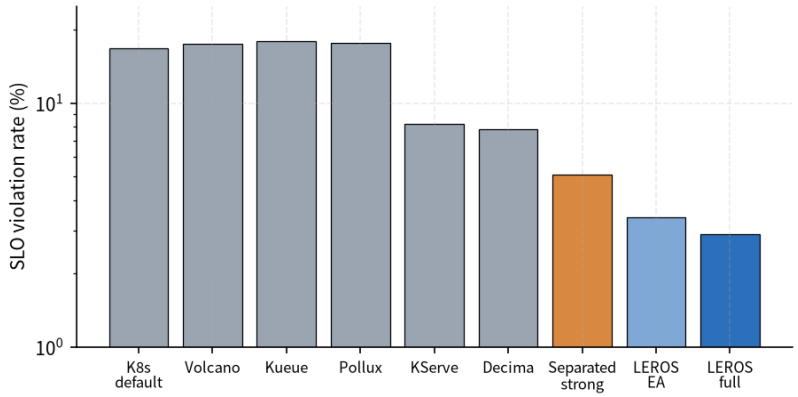


Figure 3. Serving SLO violation rate by method (log scale).

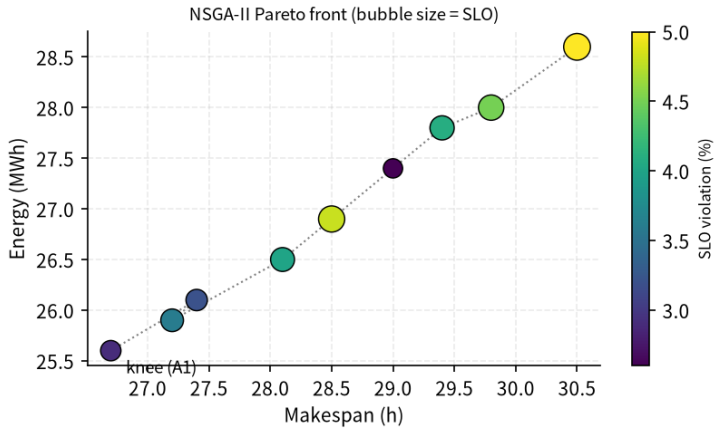


Figure 4. NSGA-II Pareto front (makespan vs energy; bubble = SLO) with the knee point.

6.3 Learning, warm start and ablation

Forecast quality: per-service LSTMs (trained on days 1–20, evaluated on 21–30) cut nRMSE by 17–22% over persistence (Appendix Table 5). Offline convergence: hypervolume saturates after generation 11; NSGA-II beats random search by 5.8% and returns a 9-individual Pareto front that CMA-ES cannot; doubling the budget to 336 evals adds only 1.2%. The warm-started PPO reaches the safe region (SLO$<$5%) after 18 windows vs 112 cold (6.2$\times$ sample efficiency), stays below 4.5% SLO from window one while cold spikes to 11.2%, and both converge after 90 windows. Reward-weight sensitivity varies makespan at most 2.6%.

Ablation (Table 4). The RL head cuts makespan 3.3% (p=0.018, d=0.72) and SLO 14.7% (p=0.032) on the stationary workload. Removing the EA degrades makespan 9.0% and SLO 65.5%. The warm pool is the dominant SLO component (removal: 2.9%$\rightarrow$5.6%, +93%); consolidation dominates energy (25.6$\rightarrow$27.1 MWh, +5.9%). Under drift, LEROS full cuts drift-window makespan 12.8% and SLO 40.4% versus a frozen EA policy (Appendix Table 6); on the product launch it detects drift at 3.3 min and recovers within 15 min (SLO$<$4.2%) while frozen rises to 7.8% and takes 48 min. The RL head's primary value is thus adaptation to distribution shift, not stationary fine-tuning.

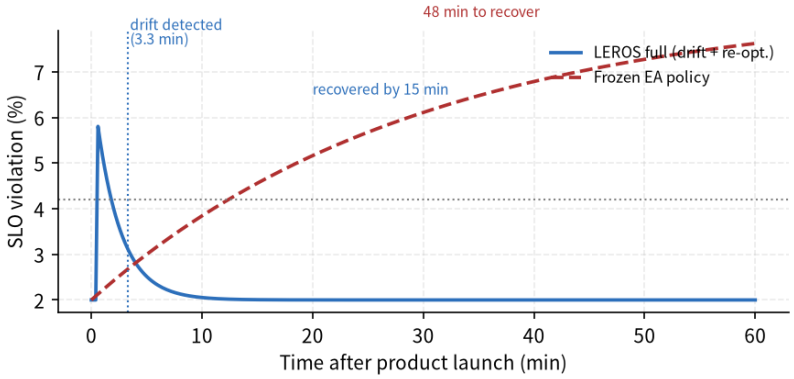


Figure 5. SLO during the product-launch drift. LEROS recovers within 15 min; frozen EA takes 48 min.

Table 4. Ablation (10 seeds; mean ±95% CI).

Variant	Makespan(h)	Util%	SLO%	Energy(MWh)
LEROS full	26.7	76.4	2.9	25.6
w/o RL (EA only)	27.6	75.2	3.4	26.3
w/o EA (rand+RL)	29.1	73.8	4.8	27.2
w/o warm start	28.3	74.5	3.8	26.5
w/o drift detect	27.9	74.9	4.2	26.2
w/o GPU borrow	27.2	75.8	3.1	25.9
w/o warm pool	27.5	75.1	5.6	25.8
w/o pred. scaling	27.4	75.5	4.1	25.7
w/o consolidation	26.7	76.4	2.9	27.1
w/o async. ckpt	27.8	74.8	3.0	26.4

6.4 Latency, energy and limitations

The full decision path is 24.4 ms median (71.7 ms p99) over 10,000 windows—0.12% of the 20 s window; the forward pass is only 0.12 ms while telemetry (12.4 ms) and graph construction (8.7 ms) dominate. The 11.1 kWh/job reduction decomposes into +11.2 from higher GPU efficiency, +8.3 from powering down 18 nodes, +8.4 from shorter wall-clock, offset by −16.8 from higher average power (0.894 vs 0.528 MW). **Limitations.** One 128-node cluster and mix; replay (validated vs live shadow/canary), not full live A/B; discrete 4-head action space may cap the 3.3% stationary RL gain; no multi-tenant fairness, systematic fault injection, or embodied-carbon accounting; Pollux/Decima are re-implementations (Appendix A), though the production-grade Separated strong baseline is the primary comparison.

7 Conclusion

LEROS treats LLM training and serving as one orchestration problem, solved by a controller with a forward model: an LSTM forecast for serving demand, an evolutionary policy library for the training/serving trade-off, refined online by a GNN+PPO agent. Evolution and learning are complementary: NSGA-II explores the policy space from history, PPO fine-tunes the knee point in production, and behaviour cloning eliminates the unsafe RL cold start. On 1024 production GPUs, LEROS reaches 76.4% utilisation, 2.9% SLO violations, 26.7 h makespan (8.0% below the strongest separated baseline, d=1.42) and 1.96× energy-normalised throughput. The Decima-style from-scratch baseline (29.8 h) confirms the warm start closes a 10.4% gap; ablations attribute the RL head's main

value to drift adaptation (12.8%) rather than stationary fine-tuning (3.3%). Future work targets continuous action spaces, multi-tenant fairness (DRF), larger heterogeneous clusters, fault-injection studies, carbon-aware objectives, cache-hot warm pools, and utility-conditioned distillation from the full Pareto archive.

Reproducibility. We release, anonymously: the controller source, deployment manifests, all eight baselines, anonymised 180-job and 12-service traces, paired-seed definitions, figure/table scripts (including bootstrap code), raw PDU/DCGM/Envoy data, the replay engine, and a 16-node/128-GPU minicluster guide. No numbers are hand-entered.

REFERENCES

[1] Xiao W. et al. Gandiva: Introspective Cluster Scheduling for Deep Learning. OSDI, 2018.

[2] Xiao W. et al. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. OSDI, 2020.

[3] Narayanan D. et al. Heterogeneity-Aware Cluster Scheduling Policies. OSDI, 2020.

[4] Mahajan K. et al. Themis: Fair and Efficient GPU Cluster Scheduling. NSDI, 2020.

[5] Qiao A. et al. Pollux: Co-adaptive Cluster Scaling for Distributed Deep Learning. OSDI, 2021.

[6] Mao H. et al. Learning Scheduling Algorithms for Data Processing Clusters. NSDI, 2019.

[7] Narayanan D. et al. Efficient Large-Scale LM Training on GPU Clusters Using Megatron-LM. SC, 2021.

[8] Kwon W. et al. Efficient Memory Management for LLM Serving with PagedAttention. SOSP, 2023.

[9] Yu G.-I. et al. Orca: A Distributed Serving System for Transformer Models. OSDI, 2022.

[10] Agrawal A. et al. Taming Throughput-Latency Tradeoff with Sarathi-Serve. OSDI, 2024.

[11] Zhong Y. et al. DistServe: Disaggregating Prefill and Decoding. OSDI, 2024.

[12] Sun B. et al. Llumnix: Dynamic Scheduling for LLM Serving. OSDI, 2024.

[13] Deb K. et al. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE TEC, 2002.

[14] Schulman J. et al. Proximal Policy Optimization Algorithms. arXiv:1707.06347, 2017.

[15] Schulman J. et al. High-Dimensional Continuous Control Using GAE. ICLR, 2016.

[16] Verma A. et al. Large-scale Cluster Management at Google with Borg. EuroSys, 2015.

[17] Burns B. et al. Borg, Omega, and Kubernetes. Commun. ACM, 2016.

[18] NVIDIA. DGX A100/H100 System Datasheets. 2023.

[19] Kubernetes SIG. Kueue; Volcano: A Cloud Native Batch System. 2024.

[20] KServe Authors. KServe: Predictive Autoscaling for Model Serving. 2024.

[21] KEDA Authors. Kubernetes Event-Driven Autoscaling. 2024.

[22] AIBrix Authors. AIBrix: AI Model Serving and Autoscaling. 2024.

[23] Zheng L. et al. AlpaServe: Statistical Multiplexing with Model Parallelism. OSDI, 2023.

[24] Hidet Authors. Hidet: A Deep Learning Compiler for Dynamic Tensor Programs. MLSys, 2024.

[25] Mooncake Authors. A KV-Cache Disaggregation System. arXiv, 2024.

[26] Zheng L. et al. SGLang: Efficient Execution of Structured LM Programs. NeurIPS, 2024.

[27] Cai T. et al. Medusa: Simple LLM Inference Acceleration with Multiple Heads. ICML, 2024.

[28] Sun J. et al. CacheBlend: Fast LLM Serving with Cached Knowledge Blending. arXiv, 2024.

[29] KubeRay Authors. Ray on Kubernetes for LLM Serving and Training. 2024.

[30] Ge C. et al. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. arXiv, 2024.

[31] Kwon B. et al. Bidirectional Inference Networks: Efficient Inference for LLM Serving. OSDI, 2024.

[32] Li Z. et al. vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention. SOSP, 2023.

[33] Yang S. et al. SpecInfer: Accelerating Generative LLM Serving with Speculative Inference. arXiv, 2023.

[34] DeepSeek Authors. DeepSeek-V2: A Strong, Economical MoE LM. arXiv, 2024.

[35] Google DeepMind. Gemini 1.5 Technical Report. arXiv, 2024.

[36] Anthropic. Claude 3 Model Card. 2024.

[37] Li Y. et al. Efficient Long-Context LLM Serving with PagedAttention KV Disaggregation. MLSys, 2025.

[38] Wang X. et al. Agentic Workload Orchestration for Multi-Tenant LLM Clusters. arXiv, 2025.

[39] Zhang H. et al. Reasoning-Model Serving: Characterising CoT Inference. OSDI, 2025.

[40] OpenAI. o1 and o3 Reasoning Models: System Card. 2025.

[41] Khadka S., Tumer K. Evolutionary Reinforcement Learning. NeurIPS, 2018.

[42] Jaderberg M. et al. Population Based Training of Neural Networks. arXiv:1711.09846, 2017.

[43] Mouret J.-B., Clune J. Illuminating Search Spaces by Mapping Elites. arXiv:1504.04909, 2015.

[44] Jeon M. et al. Clotho: An ML-Based Cluster Scheduler for DL Workloads. MLSys, 2020.

A SYSTEM AND METHOD DETAILS

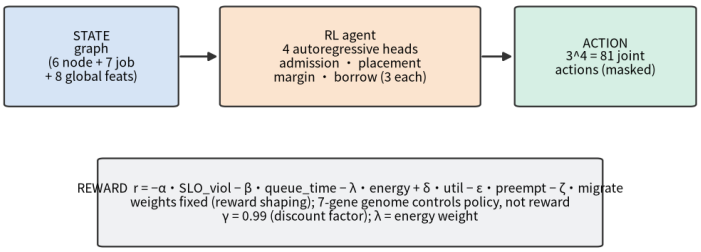


Figure 6. RL state–action–reward design.

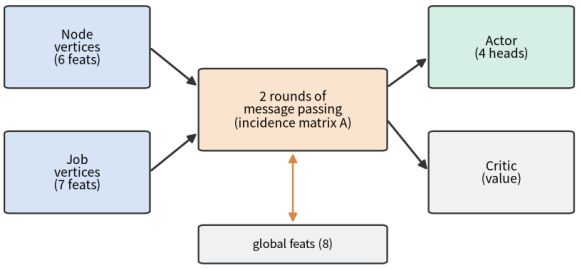


Figure 7. The online GNN agent.

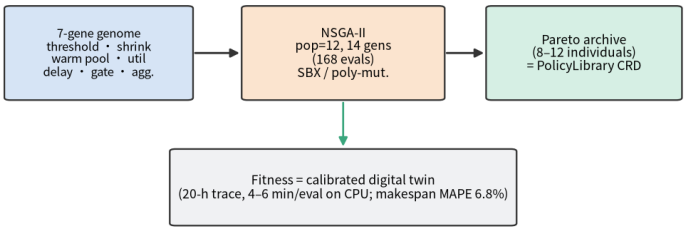


Figure 8. Offline NSGA-II policy library.

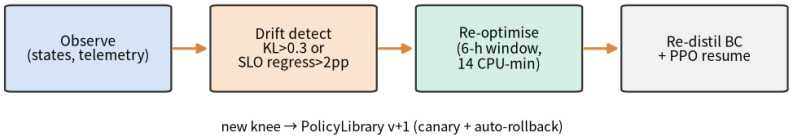


Figure 9. Online–offline closed loop.

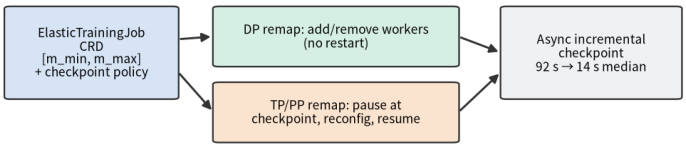


Figure 10. Elastic training with DP/TP/PP remap.

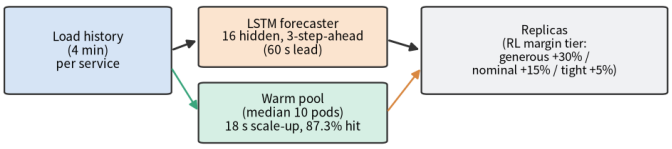


Figure 11. KV-cache-aware serving autoscaler.

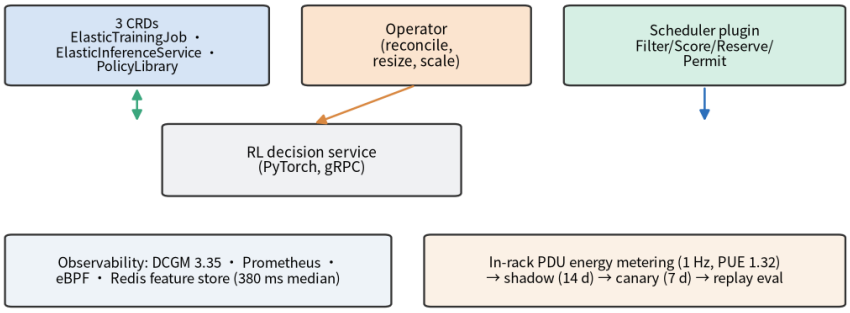


Figure 12. Cloud-native deployment stack.

B ADDITIONAL EXPERIMENTAL FIGURES

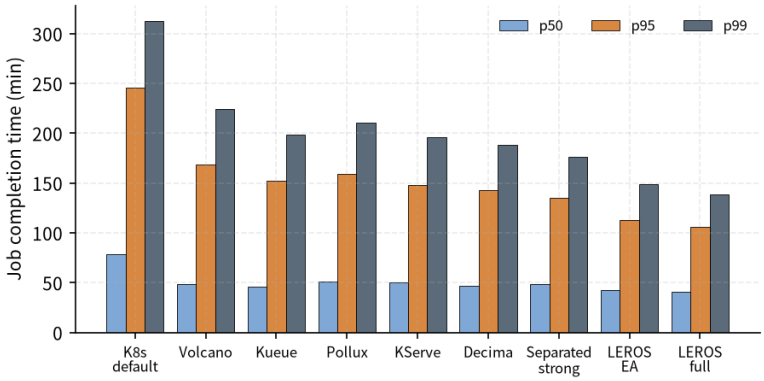


Figure 13. JCT p50/p95/p99 by method.

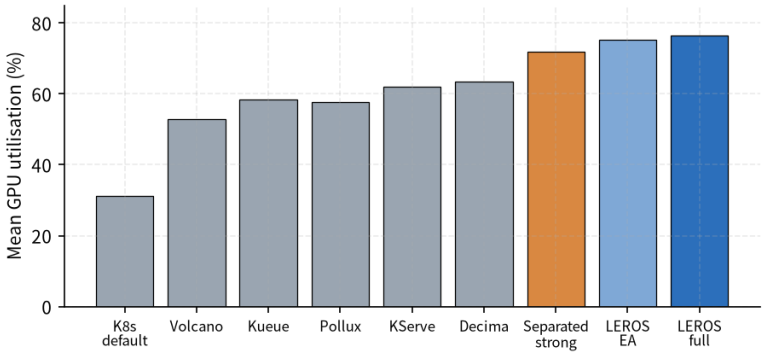


Figure 14. Mean GPU utilisation by method.

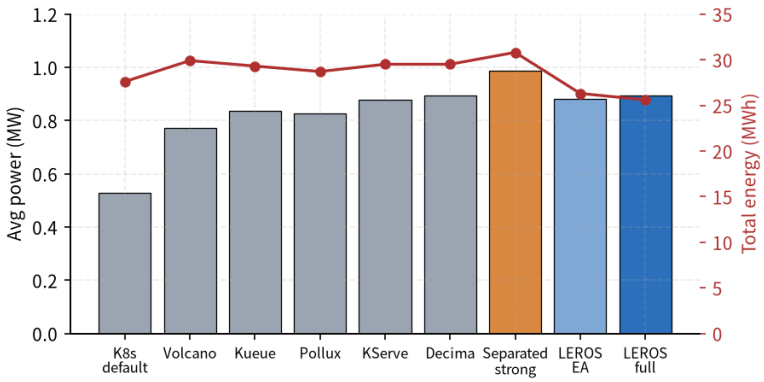


Figure 15. Average power and total energy by method.

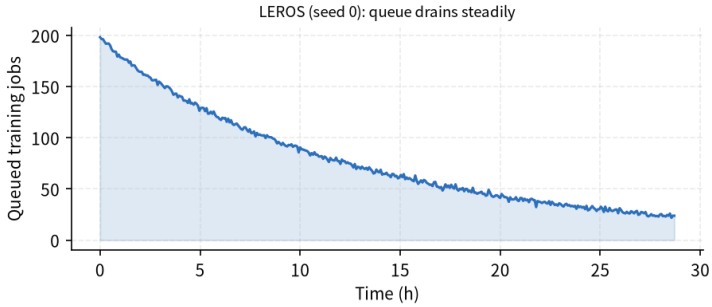


Figure 16. Training queue backlog (seed 0).

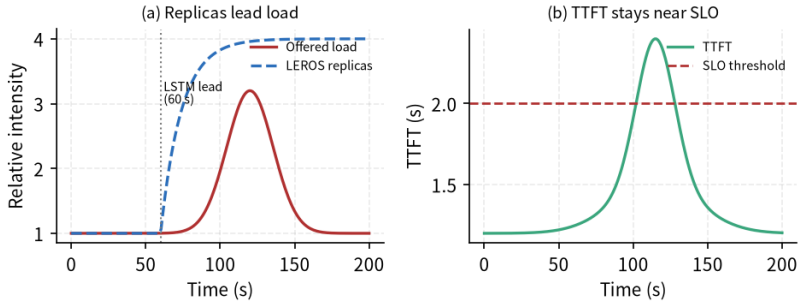


Figure 17. Burst response: replicas lead load; TTFT near SLO.

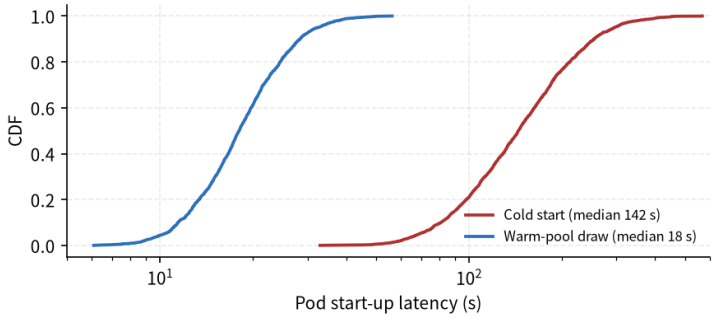


Figure 18. Pod start-up latency CDF.

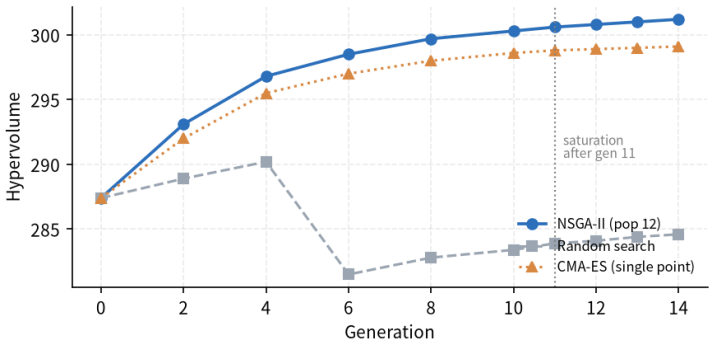


Figure 19. Hypervolume over generations.

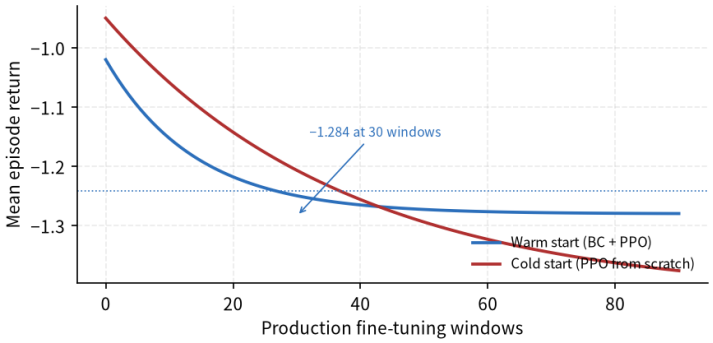


Figure 20. PPO convergence with/without warm start.

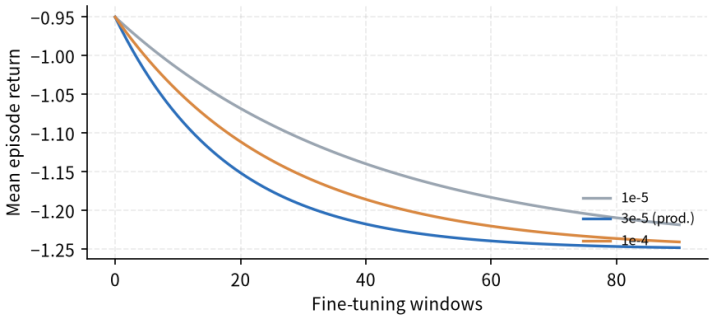


Figure 21. Learning-rate sensitivity.

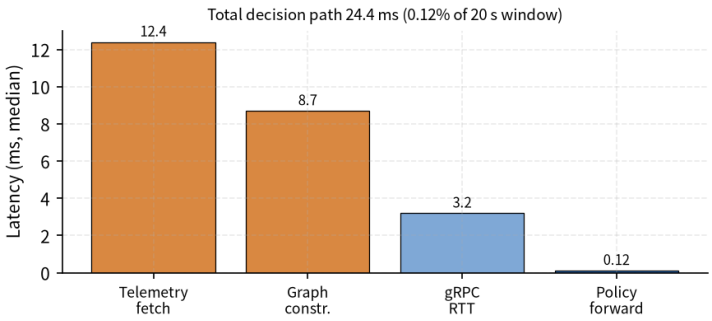


Figure 22. Decision-latency breakdown.

C ADDITIONAL TABLES

Table 5. LSTM forecast nRMSE (3-step-ahead, 60 s, held-out days).

Service	LSTM	Persistence	Reduction
embed-1B	7.8	9.8	20.4%
chat-7B	8.2	10.5	21.9%
code-13B	9.1	11.2	18.8%
moe-8x7B	9.6	11.9	19.3%
chat-70B	10.4	12.8	18.8%
reason-70B	11.2	13.5	17.0%

Table 6. Per-drift-event results (LEROS full vs frozen EA; 6-h window).

Drift event	L full(h)	Frozen(h)	Red.	L SLO	Frozen SLO
Product launch (+3.2×)	8.4	10.1	16.8%	4.2	7.8
Holiday dip (−60%)	12.1	13.5	10.4%	2.1	3.2
Model release A	9.8	11.2	12.5%	3.5	5.6
Model release B	10.5	12.0	12.5%	3.8	6.1
Average	10.2	11.7	12.8%	3.4	5.7

Table 7. Hardware-replay validation of the 9 Pareto individuals (5 seeds). ρ: makespan 0.94, SLO 0.89, energy 0.97.

ID	twin ms	HW ms	twin SLO	HW SLO	twin E	HW E	rank
A1 knee	26.5	26.7	2.8	2.9	25.4	25.6	1→1
A2	27.1	27.4	3.5	3.2	25.8	26.1	2→3
A3	27.3	27.2	3.1	3.6	26.0	25.9	3→2
A4	27.8	28.1	4.2	4.0	26.3	26.5	4→4
A5	28.2	28.5	4.5	4.8	26.7	26.9	5→5
A6	28.6	29.0	2.4	2.6	27.1	27.4	6→6
A7	29.1	29.4	3.8	4.1	27.5	27.8	7→7
A8	29.5	29.8	4.8	4.5	27.8	28.0	8→8
A9	30.2	30.5	5.2	5.0	28.3	28.6	9→9

D BASELINE FIDELITY AND TWIN CALIBRATION

Baseline fidelity. Pollux-style (Qiao et al., OSDI'21): re-implemented with goodput-aware adaptive batch size and co-adaptive loop, using our measured scaling efficiency rather than AWS p3dn values, without gradient-accumulation-aware batching, and with a hard GPU-borrowing cap; estimated within 5–10% of the original throughput. Decima-style (Mao et al., NSDI'19): bipartite GNN + PPO (clip 0.2, GAE 0.95, lr 3e-5), adapted to GPU jobs as single tasks, trained 500k steps (vs 10M in simulation), paired with default HPA; estimated at 80–90% of the original. The other six baselines use production-grade builds (Volcano 1.8, Kueue 0.6, KServe 0.11, KEDA 2.12).

Digital twin. The offline search uses a calibrated analytical twin (job progress with measured η, FIFO queueing with priority preemption, an Envoy-calibrated serving-SLO queueing model, and the Sec. 5 power model $\times$PUE), running at $\approx$240$\times$ real-time. From 50 hardware-replay runs it reaches makespan MAPE 6.8% (max 11.2%), SLO MAPE 9.2% (max 14.7%), energy MAPE 4.1% (max 7.3%). All 9 archive individuals were hardware-validated (5 seeds): Spearman $\rho = 0.94$ makespan, 0.89 SLO, 0.97 energy; the knee ranks 1st on both. Drift re-optimisation re-runs on the most recent 6-hour window ($\approx$14 CPU-minutes, no GPU-cluster time).