TROPOS: RISK-AWARE SCHEDULING FOR DISTRIBUTED TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Computation–communication overlap schedulers rank candidate execution plans under point-estimated operation durations, then run the chosen plan for thousands of iterations in which durations vary with contention. Because durations add along each dependency path and iteration time is the maximum over competing paths, the objective is convex in the durations, and its point-estimate value under-predicts the expected iteration time by a Jensen gap that grows with the number of near-critical paths and with the tail weight of the durations. We formulate overlap scheduling as minimizing the conditional value-at-risk (CVaR) of a stochastic max-plus completion time under memory constraints and show that the deterministic objectives of RoundPipe, LAER-MoE, FCP, HCMS, and NEST are its point-mass special case. TROPOS makes this objective trainable with a log-sum-exp relaxation of the max-plus recursion whose error is bounded and one-sided, a contention-aware duration model with a log-normal–Pareto residual fitted from traces, and tail-focused importance sampling; a heterogeneous graph attention network trained on the resulting loss emits a schedule in one forward pass of 1.24 ms. On Qwen3 dense models from 1.7B to 32B parameters on eight RTX 4090 GPUs, TROPOS lowers P99 iteration latency by 21.0–35.7% and raises throughput by 14–48% relative to RoundPipe, and by a further 8–17% at P99 relative to RoundPipe given the same fitted duration model. Trained only on dense graphs, it lowers P99 on an unseen Qwen3-235B-A22B mixture-of-experts workload by 13.7% relative to LAER-MoE zero-shot and by 20.5% after 50 adaptation steps. The code is available at https://anonymous.4open.science/r/TROPOS.

1 INTRODUCTION

Computation–communication overlap keeps accelerators busy while gradients, activations, and expert tokens move between devices. An overlap scheduler decides where each operation runs and in which order, and it ranks candidate schedules with a cost model built from profiled durations (Luo et al., 2026; Yuan et al., 2026; Zhao et al., 2026; Liu et al., 2026b; Wang et al., 2026). The schedule then runs for thousands of iterations under durations that differ from the profile: kernels contend for streaming multiprocessors and PCIe lanes, collectives wait for their slowest participant, and offloaded tensors arrive late. On the eight-GPU node used in this paper, the profiled cost model under-predicts RoundPipe's measured P99 iteration time by 18.9%, and that P99 exceeds the median by 46% on every model (Section 5.2). Synchronous training cannot skip a slow iteration, so this tail enters throughput in full.

Prior schedulers treat this variability as a profiling problem and answer it with re-profiling, safety margins, or per-operation distributions whose means they then schedule against. We locate the error in the objective itself. Iteration time is a maximum over execution paths of sums of durations, a polynomial in the max-plus (tropical) semiring, and therefore convex in the duration vector. By Jensen's inequality, every point-estimate objective is a lower bound on the expected iteration time, and Proposition 3 bounds the gap by $\sigma\sqrt{2\ln N}$ for perturbation scale σ and N competing paths. Aggressive overlap creates many near-critical paths, so the schedules a deterministic search prefers are exactly those whose cost estimates deserve the least trust, and the bias at P99 is at least the bias in the mean. The objectives of RoundPipe, LAER-MoE, FCP, HCMS, and NEST are all maxima of affine functions of the durations (Proposition 1), and each is recovered from our formulation when the duration distribution collapses to a point mass (Proposition 2). Figure 1 shows the mechanism on

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.

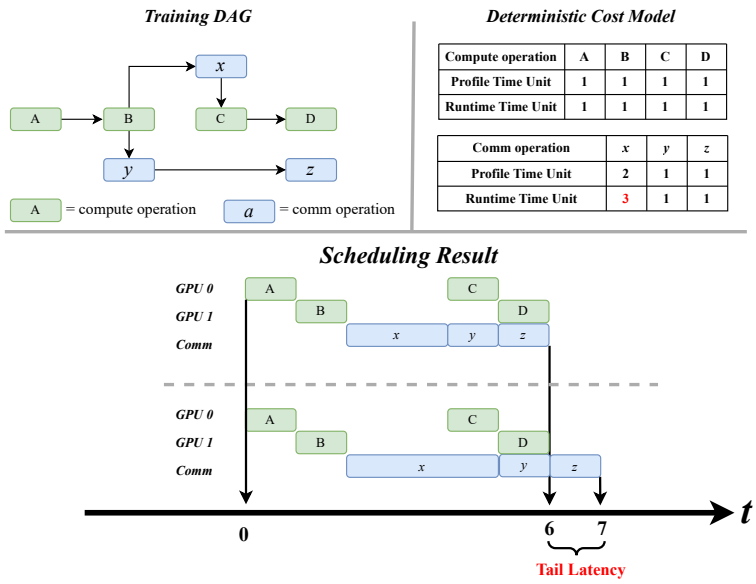


Figure 1: **A schedule tuned to profiled durations exposes the slowdown of any operation on its critical path.** Compute operations A–D take one time unit; transfer x is profiled at two units (top) and takes three at run time (bottom), which delays C, D, y, and z and moves the finish from time 6 to time 7.

a small graph; Figure 2 shows the gap growing with the number of near-critical paths, faster under heavy tails.

We therefore treat overlap scheduling as risk minimization: TROPOS minimizes the conditional value-at-risk (CVaR) of iteration time under memory constraints and amortizes the optimization into a heterogeneous graph attention network that emits a schedule in one forward pass of 1.24 ms, within 1.6% of the P99 reached by an iterative solver that needs 286 ms per plan. Our contributions are:

- **A distributional formulation with an analysis of the deterministic case.** We model iteration time as a stochastic max-plus completion time and minimize its CVaR under memory constraints. Five deterministic overlap objectives are the point-mass case of this problem, and each is optimistic by a Jensen gap of at most $\sigma\sqrt{2\ln N}$ in the mean and $\sigma\sqrt{2\ln(N/\alpha)}$ at tail level α under sub-Gaussian perturbations.
- **A differentiable, tail-aware objective and an amortized solver.** A log-sum-exp max-plus evaluator with a one-sided error bound (Theorem 1), straight-through relaxations of cut, placement, packing, and ordering decisions, a contention-aware heavy-tailed duration model, and tail-focused importance sampling train a graph policy that emits schedules in one forward pass.
- **Evidence on real training runs.** On Qwen3 dense models from 1.7B to 32B parameters, TROPOS lowers P99 iteration latency by 21.0–35.7% and raises throughput by 14–48% relative to RoundPipe, and by a further 8–17% at P99 relative to RoundPipe with the same fitted duration model; trained on dense graphs only, it lowers P99 on an unseen MoE workload by 13.7% zero-shot and by 20.5% after 50 adaptation steps. Ablations isolate each component, and a failure analysis identifies over-partitioning as the main residual error.

2 Related Work

Parallelism and pipeline schedules under deterministic cost models. Large-model training partitions computation and state across data, tensor, pipeline, and expert parallelism (Shoeybi et al.,

2019; Rajbhandari et al., 2020; Huang et al., 2019; Narayanan et al., 2019; Lepikhin et al., 2021; Duan et al., 2026); automatic planners search the joint space of partitions and placements (Jia et al., 2019; Tarnawski et al., 2020; Zheng et al., 2022; Um et al., 2024; Yuan et al., 2024); and pipeline schedules with fewer bubbles (Qi et al., 2024b;a; Ao et al., 2024), communication-efficient training (Wang et al., 2024; Sun et al., 2024; Douillard et al., 2025; Charles et al., 2025), and MoE systems that overlap expert dispatch with computation (Dai et al., 2024; He et al., 2022; Hwang et al., 2023) follow the same recipe: the cost of a plan is a deterministic function of the plan computed from profiled or modeled durations, and the search minimizes that function.

Computation–communication overlap. Overlap schedulers order and place operations so that communication hides behind computation (Zhang et al., 2017; Luo et al., 2026; Yuan et al., 2026; Zhao et al., 2026; Liu et al., 2026b; Wang et al., 2026; Cui & Pericas, 2026; Maziane et al., 2026); complementary work shrinks or reshapes the communication itself (Liu et al., 2026a; Ma et al., 2026; Jin et al., 2026). The objectives are the fill-and-drain makespan of a round-robin pipeline (Luo et al., 2026), the slowest worker or device (Zhao et al., 2026; Liu et al., 2026b), the critical path of a multi-stream pipeline (Yuan et al., 2026), or a bottleneck-stage dynamic program (Wang et al., 2026). Each is a maximum of sums of point-estimated durations (Proposition 1), evaluated at a single duration vector and reported in mean or steady-state time. TROPOS keeps the graph model of this line of work and changes what is optimized: a tail statistic of the completion-time distribution, with durations that depend on the schedule through contention.

Random durations, stragglers, and tail latency. The distribution of the longest path in a graph with random edge lengths is the classical PERT problem (Malcolm et al., 1959; Clark, 1961); computing it is #P-hard even for series-parallel graphs (Hagstrom, 1988), which is why TROPOS smooths and samples. Tail latency in large systems is dominated by rare slow components (Dean & Barroso, 2013), and distributed SGD mitigates stragglers reactively with backup workers or work reassignment (Chen et al., 2016; Harlap et al., 2016). CVaR is a coherent risk measure with a convex variational form (Rockafellar & Uryasev, 2000; 2002), and sample-based CVaR optimization with importance sampling is established in reinforcement learning (Tamar et al., 2015; Chow & Ghavamzadeh, 2014). These works analyze a fixed schedule or react at run time; TROPOS chooses the schedule against the distribution in advance.

Learned schedulers. Device placement and cluster scheduling have been learned with policy gradients over graph encoders (Mirhoseini et al., 2017; Addanki et al., 2019; Mao et al., 2019; Zhou et al., 2020), in the broader program of learning combinatorial optimization over graphs (Khalil et al., 2017) and amortizing repeated optimization into a network (Amos, 2023). These policies receive a scalar reward from a deterministic simulator or a measured mean, so operations off the realized critical path receive no signal. TROPOS differentiates through a smoothed cost model, so the policy receives gradients from every near-critical path and from the tail of the duration distribution, and it inherits the transfer property of graph policies (Table 2).

3 Risk-Aware Overlap Scheduling

3.1 Problem Formulation

We model one training iteration as a task graph $G = (V, E)$ whose nodes are operations (kernels, transfers, collectives, expert dispatch) and whose edges are data dependencies. A scheduling decision $z \in \mathcal{Z}$ fixes stage cut points, device or expert placement, communication windows, the execution order on each resource, and continuous knobs such as chunk granularity. Ordering induces resource edges $E_{\mathrm{ord}}(z)$ between consecutive operations on the same non-preemptive resource, and $G_z = (V, E \cup E_{\mathrm{ord}}(z))$ must be acyclic.

Durations are random: $\xi \sim P_\phi$ collects the perturbations of the execution environment, and $d = g_\phi(z, \xi) \in \mathbb{R}_{\geq 0}^{|V|}$ is the duration vector, where g_ϕ (Section 4.3) encodes how schedule, contention, and perturbations jointly set each duration; $\mathcal{D}_\phi(z)$ denotes the law of d. With $P_z(v)$ the predecessors of v in G_z, the completion times of v and of the iteration are

$$C_v = d_v + \max_{u \in P_z(v)} C_u \quad (\max_{u \in \varnothing} C_u := 0), \qquad T(z, d) = \max_{v \in V} C_v = \max_{p \in \mathcal{P}_z} \sum_{v \in p} d_v, \tag{1}$$

where $\mathcal{P}_z$ is the set of source-to-sink paths of G_z. The right-hand side of (1) is a polynomial in the max-plus (tropical) semiring, which gives TROPOS its name; as a function of d it is a maximum of affine functions, hence convex and piecewise affine.

Minimizing $T(z, \hat{d})$ at a point estimate $\hat{d}$ favors typical iterations over the slow ones that set the pace of synchronous training, so we optimize a tail risk measure. Fix a tail probability $\alpha \in (0, 1)$ ($\alpha = 0.01$ targets P99) and assume $\mathbb{E}[T(z, d)] < \infty$ for every feasible z. The conditional value-at-risk (CVaR) of T has the variational form of Rockafellar & Uryasev (2000),

$$F_\alpha(z, \tau) = \tau + \frac{1}{\alpha}\, \mathbb{E}_{\xi \sim P_\phi}\Big[\big(T(z, g_\phi(z, \xi)) - \tau\big)_+\Big], \qquad \rho_\alpha(z) = \inf_{\tau \in \mathbb{R}} F_\alpha(z, \tau), \tag{2}$$

where $(x)_+ = \max\{x, 0\}$, F_α is convex in τ, and the $(1 - \alpha)$-quantile of T attains the infimum, so no separate quantile estimate is needed. The risk-aware scheduling problem is

$$\min_{z \in \mathcal{Z}_{\text{feas}},\, \tau \in \mathbb{R}} F_\alpha(z, \tau), \qquad \mathcal{Z}_{\text{feas}} = \{z \in \mathcal{Z} : \mathrm{Mem}_r(z) \le M_r \ \ \forall r \in \mathcal{R}\}, \tag{3}$$

where $\mathrm{Mem}_r(z)$ is the peak memory that z places on resource r and $\mathcal{Z}$ enforces dependency, placement, and parameter-range constraints.

3.2 Why Point Estimates Miss Tail Risk

Deterministic overlap schedulers are the point-mass case of (3), and that case is biased in a fixed direction. Proofs are in Appendix A.

Proposition 1 (Max-plus representation). *Let $J(z, d)$ be built from the non-negative costs d_v by finitely many serial additions and maxima, as are longest-path completion time and bottleneck-stage time. Then for fixed z there is a finite family $\mathcal{P}$ of index sets with*

$$J(z, d) = \max_{p \in \mathcal{P}} \sum_{v \in p} d_v, \tag{4}$$

so J is convex and piecewise affine in d. The objectives of RoundPipe, LAER-MoE, FCP, HCMS, and NEST are of this form.

Proposition 2 (Deterministic scheduling as the point-mass case). *If $\mathcal{D}_\phi(z) = \delta_{\hat{d}}$ for every z, then for every $\alpha \in (0, 1)$*

$$\rho_\alpha(z) = T(z, \hat{d}) \qquad \textit{and} \qquad \min_{z \in \mathcal{Z}_{\text{feas}},\, \tau \in \mathbb{R}} F_\alpha(z, \tau) = \min_{z \in \mathcal{Z}_{\text{feas}}} T(z, \hat{d}), \tag{5}$$

so (3) is the problem solved by the schedulers of Proposition 1. For a general $\mathcal{D}_\phi(z)$,

$$\lim_{\alpha \uparrow 1} \rho_\alpha(z) = \mathbb{E}[T(z, d)], \qquad \lim_{\alpha \downarrow 0} \rho_\alpha(z) = \operatorname{ess\,sup} T(z, d). \tag{6}$$

Proposition 3 (Jensen gap and CVaR gap). *Fix z, let $N = |\mathcal{P}_z|$, and write the length of path p as $L_p(d) = \mathbb{E}[L_p(d)] + \xi_p$. If $\mathbb{E}\|d\|_1 < \infty$, then $T(z, \mathbb{E}d) \le \mathbb{E}[T(z, d)]$. If in addition each ξ_p is sub-Gaussian with parameter σ (the ξ_p may be dependent), then*

$$0 \le \mathbb{E}[T(z, d)] - T(z, \mathbb{E}d) \le \sigma\sqrt{2 \ln N}, \qquad \rho_\alpha(z) - T(z, \mathbb{E}d) \le \sigma\sqrt{2 \ln(N/\alpha)}. \tag{7}$$

The lower bound is Jensen's inequality for the convex map $d \mapsto T(z, d)$: every scheduler of Proposition 1 is optimistic, and since $\rho_\alpha(z) \ge \mathbb{E}[T(z, d)]$ its bias at tail level α is at least its bias in the mean. The upper bounds grow with σ and with N, which is largest for the aggressive overlap schedules these methods search for. They require sub-Gaussian path perturbations, whereas the lower bound needs only integrability; under the Pareto component of the fitted residuals, the measured gap grows faster than $\sqrt{\ln N}$ (Figure 2a).

4 TROPOS

Problem (3) is non-smooth in d and z, its objective is #P-hard to evaluate even for a fixed schedule (Hagstrom, 1988), the durations depend on the schedule through contention, and the events that define ρ_α are rare. Sections 4.1–4.4 address these four obstacles and Section 4.5 amortizes the optimization into a graph policy.

4.1 Smooth Critical-Path Evaluation

We replace every maximum in (1) by the log-sum-exp with temperature $\varepsilon > 0$ (Nesterov, 2005), $\mathrm{smax}_\varepsilon(x_1, \dots, x_m) = \varepsilon \ln \sum_{j=1}^{m} e^{x_j/\varepsilon}$ with $\mathrm{smax}_\varepsilon(\varnothing) := 0$, which gives

$$\widetilde{C}_v^\varepsilon = d_v + \mathrm{smax}_\varepsilon\big(\{\widetilde{C}_u^\varepsilon\}_{u \in P_z(v)}\big), \qquad \widetilde{T}^\varepsilon(z, d) = \mathrm{smax}_\varepsilon\big(\{\widetilde{C}_v^\varepsilon\}_{v \in V}\big). \tag{8}$$

One evaluation costs $O(|E|)$, the same as the hard recursion.

Theorem 1 (Smoothing error). *For every schedule z and duration vector d,*

$$0 \le \widetilde{T}^\varepsilon(z, d) - T(z, d) \le \varepsilon \big(L \ln \Delta + \ln |V|\big), \tag{9}$$

where L is the number of edges on the longest path of G_z and $\Delta = \max_v |P_z(v)|$.

The bias is one-sided and opposite to the point-estimate bias of Proposition 3. The gradient $\partial\, \mathrm{smax}_\varepsilon / \partial x_j = \mathrm{softmax}(x/\varepsilon)_j$ spreads over predecessors in proportion to how close each is to critical, so near-critical paths receive signal even when they are not the argmax. We anneal ε from 1.0 to 0.03 with a cosine schedule; the hard maximum fails to converge (Table 5).

4.2 Relaxed Scheduling Decisions

The categorical parts of z (stage cut points, device and expert placement, chunk-to-window packing) are sampled with straight-through Gumbel-Softmax (Jang et al., 2017; Maddison et al., 2017; Bengio et al., 2013): the forward pass uses a one-hot decision, so (8) runs on an executable schedule, and the backward pass uses the softmax gradient. Expert-to-device logits are projected onto soft assignment matrices by Sinkhorn normalization (Mena et al., 2018). Execution order on a shared resource is relaxed by a soft priority matrix $\Pi_z \in [0, 1]^{|V| \times |V|}$,

$$[\Pi_z]_{vu} = \big(A_v^\top A_u\big)\ \mathrm{sigm}\left(\frac{s_u - s_v}{\tau_p}\right), \tag{10}$$

where A_v is the soft resource assignment of v, s_v is a learned priority score, sigm is the logistic function, and τ_p is a temperature. $[\Pi_z]_{vu}$ is the probability that u precedes v on a shared resource and replaces the hard ordering edges $E_{\mathrm{ord}}(z)$ by weighted predecessors in (8). The Gumbel and priority temperatures follow the same cosine schedule as ε, so training ends with schedules that decode to hard, feasible orders (0.0% constraint violations, Table 8).

4.3 Contention-Aware Durations

Operations that overlap in time share streaming multiprocessors, memory bandwidth, PCIe lanes, and host threads, so a schedule changes the durations it is evaluated on. Let $\mathcal{A}_z(v)$ be the operations whose execution intervals overlap that of v under z, estimated from the relaxed schedule and current timing estimates, and let $u_r(v) \in [0, 1]$ be the usage of resource r by v. We model

$$d_v = d_v^{\mathrm{solo}}\Big(1 + \sum_{r \in \mathcal{R}} \gamma_r^{(v)} \sum_{u \in \mathcal{A}_z(v)} u_r(u)\Big) + \epsilon_v, \qquad \epsilon_v \sim (1 - \pi_v)\,\mathrm{LN}(\mu_v, \varsigma_v^2) + \pi_v\,\mathrm{Pareto}(x_m, \kappa_v), \tag{11}$$

where d_v^{solo} is the duration of v in isolation, $\gamma_r^{(v)} \ge 0$ are contention sensitivities, and the residual mixes a log-normal (LN) body with a Pareto tail of scale x_m, shape $\kappa_v > 2$ (finite variance), and weight π_v. All parameters ϕ are fitted from execution traces (Appendix B); this model is the map g_ϕ of Section 3.1. With both terms it predicts the measured P99 of Qwen3-1.7B to within 2.4%, against 18.9% for point estimates (Table 10).

4.4 Tail-Focused Sampling

For small α, only a fraction α of Monte Carlo draws from $\mathcal{D}_\phi(z)$ falls in the tail that (2) averages over, so the estimate of F_α and of its gradient has high variance (Tamar et al., 2015). We draw the K duration samples from the exponentially tilted proposal

$$q_\eta(d) \propto p_\phi(d) \exp\big(\eta\, \mathbb{E}_{z \sim \pi_\theta}[\widetilde{T}^\varepsilon(z, d)]\big), \qquad \eta > 0, \tag{12}$$

which up-weights slow realizations, and correct with self-normalized likelihood ratios $\omega_k \propto p_\phi(d^{(k)})/q_\eta(d^{(k)})$ scaled to sum to K, so the intractable normalizer of q_η cancels. Smoothing the hinge with $\mathrm{softplus}_{\varepsilon'}(x) = \varepsilon' \ln(1 + e^{x/\varepsilon'})$ gives the differentiable estimator

$$\widehat{F}_\alpha(z, \tau) = \tau + \frac{1}{\alpha K} \sum_{k=1}^{K} \omega_k \, \mathrm{softplus}_{\varepsilon'}\big(\widetilde{T}^{\varepsilon}(z, d^{(k)}) - \tau\big), \qquad d^{(k)} \sim q_\eta. \tag{13}$$

At $K = 8$ the tilted proposal halves the variance of the CVaR estimate and lowers that of its gradient by 45% relative to naive sampling (Table 11).

4.5 Amortized Graph Scheduler

Solving (3) by iterative optimization for each new graph takes 286 ms on Qwen3-1.7B (Table 13), too slow to run before every iteration. TROPOS instead trains a policy π_θ that maps $(G, \phi, \psi(h), \alpha)$ to a schedule in one forward pass, where $\psi(h)$ summarizes the gate-input hidden states h of an MoE layer as an early estimate of expert load (absent for dense models). The policy is a heterogeneous graph attention network (Wang et al., 2019) with type-specific projections for compute, communication, gate, and expert nodes and a learned bias per edge type. After $L_{\mathrm{gnn}} = 3$ layers, output heads emit the decisions of Section 4.2, the priority scores s_v, and the CVaR threshold τ_θ (Appendix B). The training loss is the estimator (13) averaged over a distribution of workloads, with a penalty on memory violations and an entropy term $\mathcal{H}$ on the categorical heads,

$$\mathcal{L}(\theta) = \mathbb{E}_{G,\phi}\Big[\widehat{F}_\alpha\big(\pi_\theta(G, \phi, \psi(h), \alpha), \tau_\theta\big)\Big] + \lambda_{\mathrm{mem}} \sum_{r \in \mathcal{R}} \big(\mathrm{Mem}_r(\pi_\theta) - M_r\big)_+ + \lambda_{\mathrm{ent}}\, \mathcal{H}(\pi_\theta), \tag{14}$$

where minimizing over τ_θ jointly with θ realizes the inner infimum of (2); Algorithm 1 (Appendix B) summarizes training and deployment. At deployment the categorical heads are decoded by argmax and each resource is ordered by its priority scores; forward pass and decoding take 1.24 ms on Qwen3-1.7B, and the decoded schedules reach a P99 within 1.6% of the iterative solver.

5 Experiments

5.1 Setup

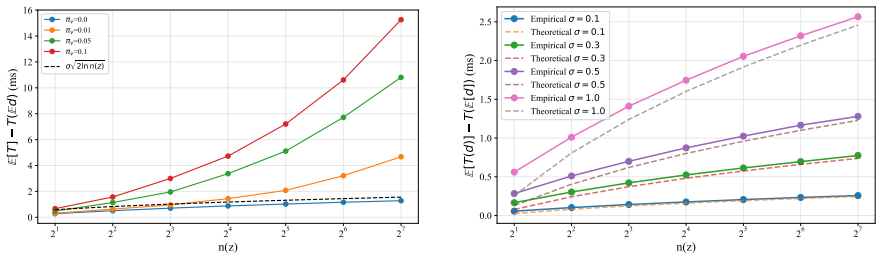
Hardware, runtime, and workloads. All experiments run on one server with eight NVIDIA RTX 4090 GPUs (24 GB each) on PCIe 4.0 in two NUMA domains, a setting with pronounced contention and cross-socket variability (Appendix B). Qwen3 dense models (Yang et al., 2025) with 1.7B, 4B, 8B, 14B, and 32B parameters are trained with full-parameter updates and CPU offloading. Batch size, micro-batch count, and recomputation granularity are tuned per model for the highest stable throughput and held fixed across methods, so latencies are comparable within a model but not across models. Qwen3-235B-A22B with LoRA adapters (Hu et al., 2021) tests transfer to an MoE graph with routing, expert parallelism, and all-to-all collectives that the scheduler never saw in training.

Metrics. We record the latency of every measured iteration after warm-up and report its mean, P50, P95, and P99, with training throughput in tokens per second. Each TROPOS end-to-end number is the mean of three runs (standard deviations in Table 1 and Appendix C).

Baselines and configuration. RP is RoundPipe (Luo et al., 2026), which schedules against profiled point durations. RP+Var is RoundPipe given the fitted duration model of Section 4.3 while keeping its deterministic objective; the difference between RP+Var and TROPOS isolates the CVaR objective and the amortized solver from better duration modeling. LAER-MoE (Liu et al., 2026b) is the MoE baseline. All planners run in the same execution stack, so only the scheduling policy differs (Appendix B). Unless stated otherwise, TROPOS uses $\alpha = 0.2$, a 3-layer HAN with hidden size 128, $K = 32$ samples, and cosine-annealed temperatures (Table 4).

5.2 Does the Problem Exist? The Jensen Gap in Practice

Figure 2 tests Proposition 3 on execution graphs constructed to have $n(z)$ near-critical paths. Under sub-Gaussian perturbations (Figure 2b) the gap $\mathbb{E}[T(z, d)] - T(z, \mathbb{E}d)$ grows as $\sqrt{\ln n(z)}$ for all four



(a) (b)

Figure 2: **Point-estimate cost models are optimistic, and more so with more competing paths.** Gap $\mathbb{E}[T(z,d)] - T(z,\mathbb{E}d)$ versus the number of near-critical paths $n(z)$. (a) Log-normal–Pareto residuals with Pareto weight π_v; the dashed line is the sub-Gaussian bound $\sigma\sqrt{2\ln n(z)}$ of Proposition 3, which heavy tails exceed. (b) Sub-Gaussian perturbations of scale σ: empirical gaps (solid) against the theoretical prediction (dashed).

Table 1: **End-to-end iteration latency and throughput on Qwen3 dense models** (eight RTX 4090, full-parameter training with CPU offloading). Best per model in bold. TROPOS mean and P99 are mean ± standard deviation over three runs; ΔP99 is relative to RP. Batch configurations differ across models, so compare methods within a model.

Model	Method	Mean (s)	P50 (s)	P95 (s)	P99 (s)	Thr. (tok/s)	ΔP99
Qwen3-1.7B	RP	21.000	20.412	27.846	29.862	50132	–
	RP+Var	18.460	18.430	24.390	25.680	55384	−14.0%
	TROPOS	**18.060**±0.471	**17.700**	**22.090**	**23.590**±0.853	**57026**	−21.0%
Qwen3-4B	RP	48.063	46.720	63.720	68.340	22915	–
	RP+Var	41.190	40.450	54.560	57.460	27248	−15.9%
	TROPOS	**39.410**±1.046	**38.620**	**48.350**	**51.650**±1.847	**29106**	−24.4%
Qwen3-8B	RP	75.731	73.610	100.380	107.650	12874	–
	RP+Var	63.610	62.030	84.320	88.870	16053	−17.4%
	TROPOS	**59.830**±1.664	**58.180**	**73.270**	**78.230**±2.964	**17421**	−27.3%
Qwen3-14B	RP	71.000	69.012	94.146	100.962	7943	–
	RP+Var	56.090	54.990	74.020	78.400	10186	−22.3%
	TROPOS	**49.700**±1.284	**48.650**	**61.470**	**64.930**±2.331	**11352**	−35.7%
Qwen3-32B	RP	60.026	58.320	79.560	85.320	3487	–
	RP+Var	45.660	44.780	60.700	64.510	4621	−24.4%
	TROPOS	**42.000**±1.104	**41.120**	**52.160**	**55.480**±1.936	**5174**	−35.0%

values of σ and stays below the bound $\sigma\sqrt{2\ln n(z)}$. Under the fitted log-normal–Pareto residuals (Figure 2a) the gap exceeds that bound as soon as the Pareto weight π_v is positive, and at $\pi_v = 0.1$ and $n(z) = 2^7$ it is ten times the bound. On the real workload, the point-estimate model under-predicts the measured P99 of Qwen3-1.7B by 18.9% and the mean by 8.7% (Table 10), and RoundPipe's P99 exceeds its median by 46% on every model in Table 1.

5.3 End-to-End Latency and Throughput

Table 1 reports the main result. TROPOS lowers every latency statistic and raises throughput on all five models: relative to RoundPipe, P99 falls by 21.0–35.7%, mean latency by 14.0–30.0%, throughput rises by 13.8–48.4%, and the P99-to-median ratio drops from 1.46 to 1.33–1.35. RP+Var recovers 14.0–24.4% at P99, the value of the fitted duration model alone; TROPOS removes a further 8.1–17.2% on top of it, the value of the CVaR objective and the amortized solver. The gains are largest on the two largest models, which have longer critical paths and more concurrent communication under offloading, hence more near-critical paths—the regime in which Proposition 3 predicts the largest optimism of deterministic schedules. Model FLOPs utilization rises from 31.2–41.7% to 35.6–59.5%, and peak GPU memory grows by 0.5–0.8 GB within the 24 GB budget (Appendix C).

Table 2: **Transfer from dense to MoE graphs** on Qwen3-235B-A22B with LoRA. The scheduler is trained on dense graphs only; the last row adapts the scheduler for 50 steps on MoE traces.

Method	Mean (s)	P50 (s)	P95 (s)	P99 (s)
LAER-MoE	42.541	41.736	52.318	58.907
TROPOS, zero-shot	35.861	35.124	44.907	50.836
TROPOS, 50 adaptation steps	**33.872**	**33.209**	**41.184**	**46.815**

Table 3: **Risk objective versus mean objective and external baselines.** All rows share one execution stack and hardware. TROPOS-Mean replaces only the CVaR loss. OOM marks runs killed after host memory was exhausted under CPU offloading. Bold marks the best P99 per model.

Model	Method	Mean (s)	P99 (s)	Thr. (tok/s)	ΔP99 vs. TROPOS
1.7B	TROPOS	18.060	23.590	57026	–
	TROPOS-Mean	17.102	**22.859**	60219	+3.1%
	NEST	20.918	29.587	50287	−25.4%
	PyTorch FSDP	40.427	76.043	25475	−222.4%
4B	TROPOS	39.410	51.650	29106	–
	TROPOS-Mean	37.894	**50.617**	30270	+2.0%
	NEST	43.512	62.603	25704	−21.2%
	PyTorch FSDP	108.437	206.284	10578	−299.4%
8B	TROPOS	59.830	78.230	17421	–
	TROPOS-Mean	58.657	**77.604**	17769	+0.8%
	NEST	63.572	92.041	15867	−17.6%
	PyTorch FSDP	204.731	406.739	5091	−419.9%
14B	TROPOS	49.700	**64.930**	11352	–
	TROPOS-Mean	54.615	69.475	10330	−7.0%
	NEST	53.189	74.826	10598	−15.3%
32B	TROPOS	42.000	**55.480**	5174	–
	TROPOS-Mean	50.299	63.525	4320	−14.5%
	NEST	44.211	58.917	4915	−6.2%

5.4 Transfer to an Unseen MoE Workload

Table 2 tests whether a policy trained only on chain-like dense graphs transfers to the branching graph of Qwen3-235B-A22B, with gate, all-to-all dispatch, parallel expert, and all-to-all combine nodes. Without any update to θ, TROPOS lowers P99 relative to LAER-MoE from 58.91 s to 50.84 s (13.7%); fifty gradient steps of scheduler adaptation on MoE traces, with the model weights untouched, bring it to 46.82 s (20.5%). The zero-shot gain indicates that the policy has learned graph properties that carry over, such as keeping high-variance operations off near-critical paths; the gap to the adapted policy measures what is specific to MoE routing.

5.5 Does the Risk Objective Pay Off?

To isolate the CVaR objective we compare against **TROPOS-Mean**, which keeps the solver, duration model, and runtime identical and replaces only the CVaR loss with the sample mean of iteration time, and against **NEST** (Wang et al., 2026) and **PyTorch FSDP** with CPU offloading (Appendix B).

Table 3 shows a crossover rather than a uniform win. Below 8B the two objectives are within noise, with TROPOS-Mean ahead by 3.1%, 2.0%, and 0.8% at P99; at 14B and 32B the ordering reverses and TROPOS wins by 7.0% and 14.5%. The sign change tracks what Proposition 3 bounds: the Jensen gap scales with the perturbation magnitude σ and the number of competing paths N, and both grow with model size as more stages, experts, and concurrent collectives make more paths near-critical. Where the frontier is narrow there is little tail risk to exploit and the mean objective, whose estimator has lower variance, is better conditioned; from roughly 14B on, mean-based scheduling optimizes a quantity that increasingly understates realized iteration time. The crossover point would move with α.

Against NEST, TROPOS is ahead at P99 on every model, by 20.3%, 17.5%, 15.0%, 13.2%, and 5.8% from 1.7B to 32B; the margin narrows at 32B because NEST's bottleneck-stage objective fits that partition more closely, and TROPOS retains a 3.4 s advantage. FSDP is far slower wherever it completes and exceeds host memory at 14B and 32B, which bounds the comparison rather than the method.

5.6 Which Components Matter

Component ablation. Table 5 in Appendix C removes or replaces one component at a time on Qwen3-1.7B. Two components carry most of the gain: point-estimated durations return P99 to the RoundPipe level (29.86 s, against 23.59 s for the full system), and optimizing the hard maximum does not converge (30.13 s). Dropping either half of the duration model costs 1.2–1.5 s of P99; the remaining relaxations and solver variants cost 1.1–5.2 s, with greedy argmax decoding the most damaging at 5.2 s. The non-amortized iterative solver reaches a P99 lower by 0.38 s and plans 230 times slower.

Risk level and numerical equivalence. Sweeping the tail probability α on Qwen3-1.7B (Figure 4, Appendix C), $\alpha = 0.5$ averages over half the distribution and accepts schedules with a heavy tail (P99 of 29.18 s), while $\alpha = 0.2$ improves every statistic, including the mean, because the schedules that lose most under slow realizations have many near-critical paths. Below 0.2 the estimator (13) rests on a handful of extreme samples and all statistics degrade (P99 of 29.42 s at $\alpha = 0.05$). We use $\alpha = 0.2$; the α of the objective need not equal the reported quantile. TROPOS changes only execution order and placement, so model outputs differ from RoundPipe's only by floating-point reordering: the relative difference stays within $\pm 10^{-3}$ and centered on zero across iterations.

5.7 Cost, Failure Modes, and Boundaries

Planning costs 1.24 ms per iteration against 41.7 ms for a greedy CPU planner and 286 ms for the iterative solver (Table 13); the smoothed recursion and the K samples add $O(K|E|)$ offline work. The dominant failure mode is over-partitioning: on Qwen3-4B LoRA fine-tuning the surrogate favored 24 stages against 16–18 for the best measured schedules, because the duration model under-prices activation movement and synchronization at stage boundaries, and 20.8% of the schedules predicted to improve P99 made it worse. A partition penalty with $\lambda_{\text{part}} = 0.03$ lowers this rate to 8.3% and improves mean and P99 (Appendix D); the residual failures are high-contention cases with non-constant boundary cost. Two boundaries remain: all measurements come from one eight-GPU PCIe node with consumer GPUs, and the transfer result covers one MoE workload. The prototype also controls SM quotas and tile sizes only through runtime knobs and does not rewrite vendor kernels (Appendix E).

6 Conclusion

Overlap schedulers have treated run-time variability as a profiling problem. Section 3.2 locates the error in the objective: iteration time is a maximum over paths of sums of durations, so a point-estimate objective is a convex function evaluated at the mean and, by Jensen's inequality, optimistic by an amount that grows with the number of near-critical paths and the tail weight of the durations. TROPOS instead optimizes the CVaR of the completion-time distribution through one-sided smoothing, a contention-aware heavy-tailed duration model, tail-focused sampling, and a graph policy that plans in 1.24 ms. On Qwen3 models from 1.7B to 32B parameters this lowers P99 by 21.0–35.7% and raises throughput by 14–48% relative to RoundPipe, and the policy transfers zero-shot to an unseen MoE graph.

Three questions remain open: a Jensen-gap bound under the Pareto tails that the fitted residuals show, where the measured gap already exceeds the sub-Gaussian bound of Proposition 3; a policy that conditions on partial execution state and reacts to slowdowns as well as anticipating them; and a schedule-conditioned model of stage-boundary costs. The evaluation should also extend to multi-node clusters, where network variability dominates.

Ethics Statement

This work improves the efficiency of distributed training through scheduling and does not involve human subjects, personal data, or new datasets. The experiments use publicly released pretrained models (Qwen3) and open-source software (PyTorch, NCCL) on hardware owned or leased by the authors' institutions. We do not foresee misuse specific to this method beyond the general effects of cheaper model training. All authors have read and adhere to the ICLR Code of Ethics.

Reproducibility Statement

Appendix A gives complete proofs of Propositions 1–3 and Theorem 1, including the constructions that map each baseline objective onto the max-plus model. Appendix B specifies the hardware and interconnect topology, the runtime and baselines, the per-model training configuration, the graph encoding, the network architecture and output heads, and the training hyperparameters (Table 4). Appendix C reports run-to-run variance over three runs, all ablations with their additional metrics, algorithmic complexity, model FLOPs utilization, and peak memory; Appendix D reports the failure analysis. The code is available at https://anonymous.4open.science/r/TROPOS.

The Use of Large Language Models

We used large language models to edit the language and to reorganize the presentation of this manuscript. All technical content, derivations, experiments, and conclusions are the authors' own.

References

Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Placeto: Learning generalizable device placement algorithms for distributed machine learning. In *Advances in Neural Information Processing Systems*, 2019.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023.

Sun Ao, Weilin Zhao, Xu Han, Cheng Yang, Xinrong Zhang, Zhiyuan Liu, Chuan Shi, and Maosong Sun. Seq1F1B: Efficient sequence-level pipeline parallelism for large language model training. *arXiv preprint arXiv:2406.03488*, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2013.

Zachary Charles, Gabriel Teston, Lucio Dery, John Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 106706–106737. Curran Associates, Inc., 2025. doi: 10.52202/085713-3559. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/99acb4c087266e80b547aed79247266b-Paper-Conference.pdf.

Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016.

Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for CVaR optimization in MDPs. In *Advances in Neural Information Processing Systems*, 2014.

Charles E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2): 145–162, 1961.

Minyu Cui and Miquel Pericas. Resource-aware computation-communication overlap for multi-gpu ml workloads, 2026. URL https://arxiv.org/abs/2606.09200.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming diloco with overlapping communication: Towards a distributed free lunch, 2025. URL https://arxiv.org/abs/2501.18512.

Jiangfei Duan, Shuo Zhang, Zerui Wang, Lijuan Jiang, Wenwen Qu, Qinghao Hu, Guoteng Wang, Qizhen Weng, Hang Yan, Xingcheng Zhang, et al. Efficient training of large language models on distributed infrastructures: a survey. *Vicinagearth*, 3(1):9, 2026.

Jane N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147, 1988.

Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. Addressing the straggler problem for iterative convergent parallel ML. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pp. 98–111, 2016.

Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 120–134, 2022.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, 2019.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In *Proceedings of Machine Learning and Systems*, 2023.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017. URL https://arxiv.org/abs/1611.01144.

Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Proceedings of Machine Learning and Systems*, 2019.

Yuze Jin, Xin Zhe Khooi, Ruyi Yao, and Mun Choon Chan. Symphony: Taming step misalignments in the network for ring-based collective operations, 2026. URL https://arxiv.org/abs/2604.16880.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Man Liu, Xingchen Liu, Xingjian Tian, Bing Lu, Shengkai Lyu, Shengquan Yin, Wenjing Huang, Zheng Wei, Hairui Zhao, Guangming Tan, and Dingwen Tao. Taco: Efficient communication compression of intermediate tensors for scalable tensor-parallel llm training. In *Proceedings of the 35th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '26, pp. 235–248, New York, NY, USA, 2026a. Association for Computing Machinery. ISBN 9798400726408. doi: 10.1145/3806645.3807584. URL https://doi.org/10.1145/3806645.3807584.

Xinyi Liu, Yujie Wang, Fangcheng Fu, Xuefeng Xiao, Huixia Li, Jiashi Li, and Bin Cui. Laermoe: Load-adaptive expert re-layout for efficient mixture-of-experts training. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 1055–1072, New York, NY, USA, 2026b. Association for Computing Machinery. ISBN 9798400723599. doi: 10.1145/3779212.3790180. URL https://doi.org/10.1145/3779212.3790180.

Yibin Luo, Shiwei Gao, Huichuan Zheng, Youyou Lu, and Jiwu Shu. Efficient training on multiple consumer gpus with roundpipe, 2026. URL https://arxiv.org/abs/2604.27085.

Shuang Ma, Chon Lam Lao, Zhiying Xu, Zhuang Wang, Ziming Mao, Delong Meng, Jia Zhen, Jun Wu, Ion Stoica, Yida Wang, and Yang Zhou. Uccl-zip: Lossless compression supercharged gpu communication, 2026. URL https://arxiv.org/abs/2604.17172.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017.

Donald G. Malcolm, John H. Roseboom, Charles E. Clark, and Willard Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 270–288, 2019.

Yassine Maziane, Ammar Mahran, Artavazd Maranjyan, and Peter Richtárik. Loscar-sgd: Local sgd with communication-computation overlap and delay-corrected sparse model averaging, 2026. URL https://arxiv.org/abs/2605.20866.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *International Conference on Learning Representations*, 2018.

Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2430–2439, 2017.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103 (1):127–152, 2005.

Penghui Qi, Xinyi Wan, Nyamdavaa Amar, and Min Lin. Pipeline parallelism with controllable memory. In *Advances in Neural Information Processing Systems*, 2024a.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *International Conference on Learning Representations*, 2024b.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–41, 2000.

R. Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471, 2002.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Weigao Sun, Qin Zhen, Weixuan Sun, Shidi Li, Dong Li, Xuyang Shen, Yu Qiao, and Yiran Zhong. Co2: Efficient distributed training with full communication-computation overlap. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39055–39077, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/a98846e9d9cc01cfb87eb694d946ce6b-Paper-Conference.pdf.

Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the CVaR via sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2993–2999, 2015.

Jakub Tarnawski, Amar Phanishayee, Nikhil R. Devanur, Divya Mahajan, and Fanny Nina Paravecino. Efficient algorithms for device placement of DNN graph operators. In *Advances in Neural Information Processing Systems*, 2020.

Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. Metis: Fast automatic distributed training on heterogeneous GPUs. In *2024 USENIX Annual Technical Conference*, pp. 563–578, 2024.

Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Xiaoxia Wu, Connor Holmes, Zhewei Yao, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. ZeRO++: Extremely efficient collective communication for large model training. In *International Conference on Learning Representations*, 2024.

Irene Wang, Vishnu Varma Venkata, Arvind Krishnamurthy, and Divya Mahajan. Nest: Network- and memory-aware device placement for distributed deep learning. In A. Chowdhery and Z. Jia (eds.), *Proceedings of Machine Learning and Systems*, volume 8, pp. 332–352. MLSys, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/file/949c07bf8756d51642c40b541c7306a8-Paper-Conference.pdf.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, WWW '19, pp. 2022–2032, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313562. URL https://doi.org/10.1145/3308558.3313562.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chao Yuan, Pan Li, Yingnan Sun, and Jing Liu. Hcms: Head-chunked multi-stream pipeline for communication-computation overlap in long-sequence parallel attention, 2026. URL https://arxiv.org/abs/2607.01817.

Tailing Yuan, Yuliang Liu, Xucheng Ye, Shenglong Zhang, Jianchao Tan, Bin Chen, Chengru Song, and Di Zhang. Accelerating the training of large language models using efficient activation rematerialization and optimal hybrid parallelism. In *2024 USENIX Annual Technical Conference*, pp. 545–561, 2024.

Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 181–193, Santa Clara, CA, July 2017. USENIX Association. ISBN 978-1-931971-38-6. URL https://www.usenix.org/conference/atc17/technical-sessions/presentation/zhang.

Yilong Zhao, Xiaonan Nie, Kan Zhu, Shuang Ma, Zhichao Lai, Hongxiang Hao, Yang Zhou, Baris Kasikci, and Ion Stoica. Unleashing scalable context parallelism for foundation models pre-training via fcp. In A. Chowdhery and Z. Jia (eds.), *Proceedings of Machine Learning and Systems*, volume 8, pp. 2129–2142. MLSys, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/file/e01c431bbb83153632c0dcfaf8ccda0a-Paper-Conference.pdf.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation*, pp. 559–578, 2022.

Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu, Phitchaya Mangpo Phothilimthana, Shen Wang, Anna Goldie, Azalia Mirhoseini, and James Laudon. Transferable graph optimizers for ML compilers. In *Advances in Neural Information Processing Systems*, 2020.

APPENDIX

APPENDIX CONTENTS

A Proofs . . . 15
A.1 Proof of Proposition 1 (Max-plus representation) . . . 15
A.2 Proof of Proposition 2 (Deterministic scheduling as the point-mass case) . . . 18
A.3 Proof of Proposition 3 (Jensen gap and CVaR gap) . . . 19
A.4 Proof of Theorem 1 (Smoothing error) . . . 19
B Detailed Experiment Settings . . . 20
B.1 Hardware . . . 20
B.2 Baselines . . . 20
B.3 Training Parameter Settings . . . 20
B.4 Graph Encoding . . . 21
B.5 Network Architecture and Output Heads . . . 21
B.6 Training Hyperparameters . . . 22
C Additional Experiment Results . . . 23
C.1 Component Ablation . . . 23
C.2 Risk Level and Numerical Equivalence . . . 23
C.3 Tail-Latency Stability Analysis . . . 23
C.4 Ablation Study . . . 23

C.5 Algorithmic Complexity 26
C.6 MFU 26
C.7 Peak GPU Memory Usage 26
D Error Analysis 27
E Discussion 27

A Proofs

A.1 Proof of Proposition 1 (Max-plus representation)

Proof. The first claim is closure under the two operations. A single cost d_v has the form $\max_{p\in\mathcal{P}} \sum_{v\in p} d_v$ with $\mathcal{P} = \{\{v\}\}$. If two objectives have this form with families $\mathcal{P}$ and $\mathcal{Q}$, their maximum has it with $\mathcal{P} \cup \mathcal{Q}$, and their serial sum equals $\max_{(p,q)\in\mathcal{P}\times\mathcal{Q}} \sum_{v\in p\cup q} d_v$, where p and q are disjoint because serial composition joins distinct operations. Finitely many such steps therefore yield (4) for a finite family $\mathcal{P}$, a maximum of affine functions of d, hence convex and piecewise affine. For the second claim we instantiate the model for each method: we exhibit the task graph or index family, the decision z, and the point-estimated duration vector $\hat{d}$ under which the method's objective equals $\max_{p\in\mathcal{P}} \sum_{v\in p} \hat{d}_v$.

RoundPipe(Luo et al., 2026). RoundPipe runs a pipeline with N stages and M micro-batches; each micro-batch executes one forward and one backward pass through all stages, and stages are dispatched round-robin over the GPUs. We construct a time-expanded graph in which the fill and drain phases appear as paths.

For each stage $s \in \{1, \dots, N\}$, micro-batch $m \in \{1, \dots, M\}$, and phase $\phi \in \{\text{fwd}, \text{bwd}\}$, define operation $v_{s,m,\phi}$. The vertex set is

$$V_{\text{RP}} = \{v_{s,m,\phi} : 1 \le s \le N,\ 1 \le m \le M,\ \phi \in \{\text{fwd}, \text{bwd}\}\}.$$

The edge set contains data-dependency and resource-ordering edges:

- **Data dependencies**: forward-to-backward within each micro-batch and stage: $(v_{s,m,\text{fwd}}, v_{s,m,\text{bwd}})$ for all s, m; forward-to-forward across stages: $(v_{s,m,\text{fwd}}, v_{s+1,m,\text{fwd}})$ for $s < N$; backward-to-backward across stages in reverse: $(v_{s+1,m,\text{bwd}}, v_{s,m,\text{bwd}})$ for $s < N$.
- **Resource ordering**: operations on the same stage must execute serially. For each stage s, we add ordering edges $(v_{s,m,\phi}, v_{s,m',\phi'})$ where (m, ϕ) precedes (m', ϕ') in the pipeline schedule.

Thus $G_{\text{RP}} = (V_{\text{RP}}, E_{\text{RP}})$ is a time-expanded pipeline graph with $2MN$ operations.

For each operation $v_{s,m,\phi}$, RoundPipe uses profiled mean duration:

$$\hat{d}_{v_{s,m,\phi}} = \frac{1}{R} \sum_{r=1}^{R} \tau_{s,\phi,r},$$

where $\tau_{s,\phi,r}$ is the measured execution time of phase ϕ on stage s in profiling run r.

Evaluating $T(z, \hat{d}_{\text{RP}})$ on this graph gives the pipeline makespan under data dependencies and per-stage serialization; the fill and drain phases are the longest path through the $2MN$ operations. RoundPipe is therefore the max-plus model with G the time-expanded pipeline graph and $\hat{d}$ the profiled mean stage-phase durations.

LAER-MoE(Liu et al., 2026b). LAER-MoE separates each device-side MoE execution into communication followed by expert computation. Let there be P devices. For each device p, define two vertices c_p and e_p, where c_p denotes communication and e_p denotes expert computation. Construct

$$V_{\text{LAER}} = \{c_p, e_p : 1 \le p \le P\},$$

and

$$E_{\mathrm{LAER}} = \{(c_p, e_p) : 1 \le p \le P\}.$$

Thus each device has a serial subgraph

$$c_p \to e_p.$$

Let there be Q experts and let $z = (A, R)$, where $A \in \{0,1\}^{Q\times P}$ is the expert layout matrix and $R \in \mathbb{R}_{\ge 0}^{Q\times P}$ is the token-routing (load) matrix. Here $A_{ep} = 1$ means expert e is materialized on device p, and R_{ep} is the amount of tokens of expert e routed to device p.

Let b_{ep} be the communicated parameter or token volume associated with expert e on device p, let B_p be the effective communication bandwidth of device p, and let α_p be the launch/synchronization latency. Then the communication point estimate is

$$\hat{d}_{c_p}(z) = \alpha_p + \frac{1}{B_p}\sum_{e=1}^{Q} A_{ep} b_{ep}.$$

Let q_e be the per-token expert computation cost for expert e. The computation point estimate is

$$\hat{d}_{e_p}(z) = \sum_{e=1}^{Q} q_e R_{ep}.$$

Therefore

$$\hat{d}_{\mathrm{LAER}}(z) = \left\{\hat{d}_{c_p}(z), \hat{d}_{e_p}(z)\right\}_{p=1}^{P}.$$

Since each device executes communication before computation and the iteration is limited by the slowest device, the instantiated objective is

$$T(z, \hat{d}_{\mathrm{LAER}}) = \max_{1\le p\le P}\left(\hat{d}_{c_p}(z) + \hat{d}_{e_p}(z)\right).$$

Thus LAER-MoE is represented by a device-parallel collection of communication-to-computation chains.

FCP(Zhao et al., 2026). FCP partitions sequences into fixed-size blocks and assigns blocks to workers. Let $\mathcal{B} = \{1, \dots, B\}$ be the set of blocks and $\mathcal{W} = \{1, \dots, P\}$ be the set of workers. The graph is a one-layer parallel-worker graph:

$$V_{\mathrm{FCP}} = \{w_p : 1 \le p \le P\}, \qquad E_{\mathrm{FCP}} = \varnothing.$$

Hence

$$G_{\mathrm{FCP}} = (V_{\mathrm{FCP}}, \varnothing).$$

Let

$$x_{bp} \in \{0, 1\}$$

indicate whether block b is assigned to worker p. Let a_b denote the attention computation cost of block b, let m_{bp} denote the amount of data communicated for block b when assigned to worker p, and let B_p be the effective communication bandwidth of worker p. The non-overlapped worker cost is

$$\bar{d}_{w_p}(z) = \sum_{b=1}^{B} x_{bp}\left(a_b + \frac{m_{bp}}{B_p}\right).$$

FCP overlaps communication and computation through block-level pipelining. Let $\eta_p \in (0, 1]$ be the effective overlap coefficient for worker p. Then the point estimate used by T is

$$\hat{d}_{w_p}(z) = \eta_p \bar{d}_{w_p}(z) = \eta_p \sum_{b=1}^{B} x_{bp}\left(a_b + \frac{m_{bp}}{B_p}\right).$$

Therefore

$$\hat{d}_{\mathrm{FCP}}(z) = \left(\hat{d}_{w_1}(z), \dots, \hat{d}_{w_P}(z)\right).$$

Since the workers execute in parallel and synchronize at the end of the attention module,

$$T(z, \hat{d}_{\mathrm{FCP}}) = \max_{1\le p\le P} \hat{d}_{w_p}(z).$$

Thus FCP is obtained by choosing G as a parallel-worker graph and $\hat{d}$ as the overlap-adjusted worker load.

HCMS(Yuan et al., 2026). HCMS partitions attention heads into C chunks and overlaps communication and computation using two CUDA streams. Define communication vertices

$$u_1, \dots, u_C$$

and computation vertices

$$a_1, \dots, a_C,$$

where u_c is communication for chunk c, and a_c is attention computation for chunk c. The vertex set is

$$V_{\text{HCMS}} = \{u_c, a_c : 1 \le c \le C\}.$$

The edge set contains stream-ordering and data-dependence edges:

$$E_{\text{HCMS}} = \{(u_c, u_{c+1}) : 1 \le c < C\} \cup \{(a_c, a_{c+1}) : 1 \le c < C\} \cup \{(u_c, a_c) : 1 \le c \le C\}.$$

Thus

$$G_{\text{HCMS}} = (V_{\text{HCMS}}, E_{\text{HCMS}})$$

is a coupled two-stream pipeline.

Let h_c be the number of heads in chunk c, H the total number of heads, T_{comm} the total all-to-all communication time without chunking, and T_{attn} the total attention-computation time without chunking. The chunk-level point estimates are

$$\hat{d}_{u_c}(z) = \frac{h_c}{H} T_{\text{comm}},$$

and

$$\hat{d}_{a_c}(z) = \frac{h_c}{H} T_{\text{attn}}.$$

To account for launch overhead, synchronization, and stream interference, let $\beta_c \ge 0$ be the per-chunk interference term. The effective computation vertex estimate is

$$\hat{d}^{\text{eff}}_{a_c}(z) = \hat{d}_{a_c}(z) + \beta_c.$$

Equivalently, the point-estimate vector is

$$\hat{d}_{\text{HCMS}}(z) = \left\{ \hat{d}_{u_c}(z), \hat{d}^{\text{eff}}_{a_c}(z) \right\}_{c=1}^{C}.$$

Substituting G_{HCMS} and $\hat{d}_{\text{HCMS}}$ into T gives the HCMS two-stream pipeline objective $T(z, \hat{d}_{\text{HCMS}})$. In the balanced-chunk case, $h_c = H/C$, this becomes the standard chunked pipeline estimate

$$\hat{d}_{u_c}(z) = \frac{T_{\text{comm}}}{C}, \qquad \hat{d}^{\text{eff}}_{a_c}(z) = \frac{T_{\text{attn}}}{C} + \beta_c.$$

Therefore HCMS is represented by a C-segment dual-stream graph with chunk-level communication and computation estimates.

NEST(Wang et al., 2026). NEST places contiguous layer groups onto devices using a dynamic program that minimizes the bottleneck stage time in a pipeline. Let the model have L layers. Construct a chain graph

$$V_{\text{NEST}} = \{\ell_1, \dots, \ell_L\},$$

and

$$E_{\text{NEST}} = \{(\ell_i, \ell_{i+1}) : 1 \le i < L\}.$$

Thus

$$G_{\text{NEST}} = (V_{\text{NEST}}, E_{\text{NEST}}).$$

NEST's decision variable z selects a partition of the chain into contiguous stages, the number of devices assigned to each stage, and the network locality level of the placement. For a candidate stage

$$S_{i:j} = \{\ell_i, \ell_{i+1}, \dots, \ell_j\},$$

let a be the number of devices assigned to the stage and let λ be the network-locality level. Define the point-estimated stage latency as

$$\hat{d}^{a,\lambda}_{i:j} = \hat{c}^{a}_{i:j} + \hat{r}^{a}_{i:j} + \hat{n}^{a,\lambda}_{i:j},$$

where $\hat{c}^{a}_{i:j}$ is the profiled or modeled computation time of the stage under the selected intra-stage parallelism, $\hat{r}^{a}_{i:j}$ is the backward/reduction communication estimate inside the stage, and $\hat{n}^{a,\lambda}_{i:j}$ is the inter-stage activation communication estimate induced by network level λ.

The memory estimate for placing $S_{i:j}$ at pipeline position s is a constraint, not a duration:

$$\widehat{\mathrm{Mem}}(S_{i:j}, s) = \sum_{\ell \in S_{i:j}} (2W_\ell + O_\ell + A_\ell) + (s-1)R_{i:j},$$

where W_ℓ denotes layer weights, O_ℓ optimizer states and gradients, A_ℓ current activations, and $R_{i:j}$ stashed activation data for the stage. A candidate stage is feasible only if

$$\widehat{\mathrm{Mem}}(S_{i:j}, s) \le \mathrm{Cap}\,.$$

Objective. NEST minimizes the bottleneck stage time. For a partition z into k stages $S_{i_1:j_1}, \ldots, S_{i_k:j_k}$ with device counts a_s and locality levels λ_s,

$$F_{\mathrm{NEST}}(z) = \max_{s=1}^{k} \hat{d}^{a_s,\lambda_s}_{i_s:j_s} = \max_{s=1}^{k} \big(\hat{c}^{a_s}_{i_s:j_s} + \hat{r}^{a_s}_{i_s:j_s} + \hat{n}^{a_s,\lambda_s}_{i_s:j_s}\big),$$

which the dynamic program

$$\mathrm{DP}[j,k] = \min_{\substack{i<j \\ \text{feasible}}} \max\Big\{\mathrm{DP}[i,k-1], \hat{d}^{a,\lambda}_{i+1:j}\Big\}$$

minimizes over partitions whose stages satisfy the memory constraint. Taking one node per cost term of each stage, $\{c_s, r_s, n_s\}$, and the index family $\mathcal{P} = \{\{c_s, r_s, n_s\}\}_{s=1}^{k}$, we have $F_{\mathrm{NEST}}(z) = \max_{p\in\mathcal{P}} \sum_{v\in p} \hat{d}_v$, the form in Proposition 1. This objective differs from the source-to-sink makespan of the chain graph G_{NEST}, which would sum the stage latencies; Proposition 1 is stated for an arbitrary finite index family and covers both.

Summary. Each of the five objectives is a maximum over a finite index family of sums of point-estimated costs: LAER-MoE, FCP, and HCMS are source-to-sink makespans of small task graphs, RoundPipe is the makespan of a time-expanded pipeline graph, and NEST is a bottleneck maximum over per-stage sums. Memory enters each method as a feasibility constraint, as in (3), and never through the duration vector. The overlap coefficients of FCP and the interference terms of HCMS are those methods' own modeling assumptions about $\hat{d}$; the representation does not depend on them. The claim covers these five methods and does not extend to schedulers that already model run-time variability or adapt at run time. □

A.2 Proof of Proposition 2 (Deterministic scheduling as the point-mass case)

Proof. If $\mathcal{D}_\phi(z) = \delta_{\hat{d}}$, the expectation in (2) equals $(T(z,\hat{d}) - \tau)_+$, so with $a_z := T(z,\hat{d})$,

$$F_\alpha(z,\tau) = \tau + \frac{1}{\alpha}(a_z - \tau)_+.$$

For $\tau \ge a_z$ this equals τ and is minimized at $\tau = a_z$ with value a_z. For $\tau < a_z$ it equals $a_z/\alpha + (1 - 1/\alpha)\tau$, which decreases in τ because $\alpha < 1$, so its infimum over $\tau < a_z$ is again a_z. Hence $\rho_\alpha(z) = T(z,\hat{d})$ for every $\alpha \in (0,1)$, attained at $\tau = T(z,\hat{d})$, and (3) becomes $\min_{z\in\mathcal{Z}_{\mathrm{feas}}} T(z,\hat{d})$. By Proposition 1 this is the problem the five deterministic schedulers solve.

For the limits, let $q_u = \inf\{t : \Pr(T(z,d) \le t) \ge u\}$ be the u-quantile of $T(z,d)$. CVaR at tail probability α is the average of the upper quantiles (Rockafellar & Uryasev, 2002),

$$\rho_\alpha(z) = \frac{1}{\alpha}\int_{1-\alpha}^{1} q_u \,\mathrm{d}u.$$

As $\alpha \uparrow 1$ the right-hand side tends to $\int_0^1 q_u \,\mathrm{d}u = \mathbb{E}[T(z,d)]$, finite by assumption. As $\alpha \downarrow 0$ it averages q_u over $u \in [1-\alpha, 1]$, and q_u is nondecreasing with $\lim_{u\uparrow 1} q_u = \operatorname{ess\,sup} T(z,d)$, so the average tends to $\operatorname{ess\,sup} T(z,d)$ (possibly $+\infty$). □

A.3 Proof of Proposition 3 (Jensen gap and CVaR gap)

Proof. Fix z with G_z acyclic, let $\mathcal{P}_z$ be the set of source-to-sink paths of G_z, and let $N = |\mathcal{P}_z|$. For $p \in \mathcal{P}_z$ let $L_p(d) = \sum_{v \in p} d_v$, so $T(z,d) = \max_p L_p(d)$ by (1), and write $L_p(d) = L_p(\mathbb{E}d) + \xi_p$ with $\xi_p = \sum_{v \in p}(d_v - \mathbb{E}d_v)$; integrability of d gives $\mathbb{E}\xi_p = 0$ and $\mathbb{E}[L_p(d)] = L_p(\mathbb{E}d)$, and $T(z, \mathbb{E}d) = \max_p L_p(\mathbb{E}d)$.

Lower bound. The map $d \mapsto T(z,d)$ is a maximum of affine functions, hence convex, and Jensen's inequality gives $T(z, \mathbb{E}d) \le \mathbb{E}[T(z,d)]$.

Upper bound on the mean. Let $M = \max_p \xi_p$. Since $\max_p(a_p + b_p) \le \max_p a_p + \max_p b_p$,

$$T(z,d) \le T(z, \mathbb{E}d) + M \qquad \text{for every realization } d, \tag{15}$$

so $\mathbb{E}[T(z,d)] - T(z,\mathbb{E}d) \le \mathbb{E}M$. Assume each ξ_p is sub-Gaussian with parameter σ, that is, $\mathbb{E}e^{\lambda \xi_p} \le e^{\lambda^2\sigma^2/2}$ for all $\lambda \in \mathbb{R}$; independence across paths is not assumed. For $\lambda > 0$, Jensen's inequality and $e^{\lambda M} \le \sum_p e^{\lambda \xi_p}$ give

$$\mathbb{E}e^{\lambda M} \le N e^{\lambda^2\sigma^2/2}, \qquad \mathbb{E}M \le \frac{1}{\lambda} \ln \mathbb{E}e^{\lambda M} \le \frac{\ln N}{\lambda} + \frac{\lambda\sigma^2}{2}. \tag{16}$$

Minimizing over $\lambda > 0$, at $\lambda^\star = \sqrt{2\ln N}/\sigma$ when $N > 1$, yields $\mathbb{E}M \le \sigma\sqrt{2\ln N}$, the maximal inequality for sub-Gaussian families (Boucheron et al., 2013); for $N = 1$, $M = \xi_1$ and $\mathbb{E}M = 0$.

Upper bound on CVaR. For any random variable X with $\mathbb{E}e^{\lambda X} < \infty$ and any $\lambda > 0$,

$$\mathrm{CVaR}_\alpha(X) \le \frac{\ln \mathbb{E}e^{\lambda X} + \ln(1/\alpha)}{\lambda}. \tag{17}$$

To see this, note $(x)_+ \le e^{\lambda x}/(e\lambda)$ for all $x \in \mathbb{R}$, because $x\,e^{-\lambda x} \le 1/(e\lambda)$. Hence by the variational form (2), for every τ,

$$\mathrm{CVaR}_\alpha(X) \le \tau + \frac{1}{\alpha}\mathbb{E}[(X-\tau)_+] \le \tau + \frac{e^{-\lambda\tau}\,\mathbb{E}e^{\lambda X}}{\alpha e \lambda}.$$

The right-hand side is minimized at $\tau^\star = \lambda^{-1}\big(\ln \mathbb{E}e^{\lambda X} - \ln\alpha - 1\big)$, where it equals $\tau^\star + 1/\lambda$, which is (17). Applying (17) to M with the moment bound in (16),

$$\mathrm{CVaR}_\alpha(M) \le \inf_{\lambda > 0}\Big[\frac{\ln N + \ln(1/\alpha)}{\lambda} + \frac{\lambda\sigma^2}{2}\Big] = \sigma\sqrt{2\ln(N/\alpha)},$$

the infimum being attained at $\lambda^\star = \sqrt{2\ln(N/\alpha)}/\sigma$; here $N/\alpha > 1$ because $\alpha < 1$. CVaR is monotone and translation-equivariant (Rockafellar & Uryasev, 2002), so (15) gives

$$\rho_\alpha(z) = \mathrm{CVaR}_\alpha\big(T(z,d)\big) \le T(z,\mathbb{E}d) + \mathrm{CVaR}_\alpha(M) \le T(z,\mathbb{E}d) + \sigma\sqrt{2\ln(N/\alpha)}.$$

When $\sigma = 0$ every ξ_p vanishes, $M = 0$, and both gaps are zero, matching the bounds in the limit $\lambda \to \infty$. □

A.4 Proof of Theorem 1 (Smoothing error)

Proof. For $m \ge 1$ reals $x_1, \dots, x_m$ and $\varepsilon > 0$,

$$\max_j x_j \le \mathrm{smax}_\varepsilon(x_1,\dots,x_m) \le \max_j x_j + \varepsilon \ln m, \tag{18}$$

because $e^{\max_j x_j/\varepsilon} \le \sum_j e^{x_j/\varepsilon} \le m\, e^{\max_j x_j/\varepsilon}$; for $m = 0$ both max and $\mathrm{smax}_\varepsilon$ equal 0 by convention. Let ℓ_v be the number of edges on the longest path of G_z ending at v, so $\ell_v \le L$, and let $\Delta = \max_v |P_z(v)|$. We show by induction on ℓ_v that

$$0 \le \widetilde{C}^\varepsilon_v - C_v \le \varepsilon\, \ell_v \ln \Delta. \tag{19}$$

If $\ell_v = 0$ then $P_z(v) = \varnothing$ and $\widetilde{C}^\varepsilon_v = C_v = d_v$. If $\ell_v = k \ge 1$, every $u \in P_z(v)$ has $\ell_u \le k-1$, so (19) for the predecessors, monotonicity of $\mathrm{smax}_\varepsilon$ in each argument, and (18) with $m = |P_z(v)| \le \Delta$ give

$$\begin{aligned} C_v = d_v + \max_{u \in P_z(v)} C_u \le \widetilde{C}^\varepsilon_v &\le d_v + \max_{u \in P_z(v)} \widetilde{C}^\varepsilon_u + \varepsilon \ln \Delta \\ &\le d_v + \max_{u \in P_z(v)} C_u + \varepsilon(k-1)\ln\Delta + \varepsilon\ln\Delta = C_v + \varepsilon k \ln \Delta. \end{aligned}$$

For the iteration time, apply (18) once more with $m = |V|$ and use (19):

$$T(z,d) = \max_v C_v \le \widetilde{T}^{\varepsilon}(z,d) \le \max_v \widetilde{C}_v^{\varepsilon} + \varepsilon \ln |V| \le T(z,d) + \varepsilon\big(L \ln \Delta + \ln |V|\big).$$

Both inequalities hold for every realization d, which is the one-sided property used in Section 4.1. □

B Detailed Experiment Settings

B.1 Hardware

All experiments run on one PCIe server with eight NVIDIA RTX 4090 GPUs (24 GB each). The machine has two NUMA domains: GPUs 0–3 and GPUs 4–7 each share a domain, and traffic between the two domains crosses the CPU interconnect. The host has 120 vCPUs (Intel Xeon Platinum 8474C) and 640 GB of system memory. GPUs communicate over PCIe 4.0 with a peak bandwidth of 32 GB/s. Figure 3 shows the topology.

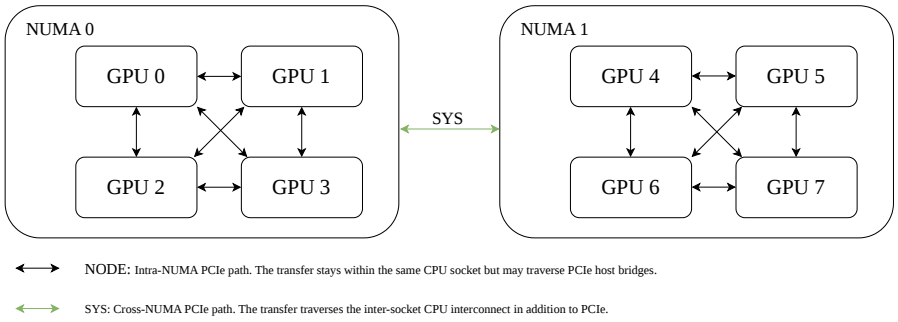


Figure 3: GPU topology of the eight-GPU PCIe server. GPUs 0–3 and 4–7 share a NUMA domain; traffic between the domains crosses the CPU interconnect.

B.2 Baselines

RoundPipe is the primary runtime baseline because it lets us change the scheduling objective while holding the execution stack fixed. Comparing against schedulers implemented in other systems would confound scheduling quality with differences in offloading, recomputation, kernel launching, tracing, and pipeline execution. RoundPipe also fits the setting: it performs pipeline scheduling with computation–communication overlap for multi-GPU training with CPU offloading. RP+Var gives RoundPipe the fitted duration distribution of Section 4.3 while keeping its deterministic objective, which separates the value of modeling run-time variability from the value of optimizing a tail-risk objective.

B.3 Training Parameter Settings

For each model we tune the global batch size, the number of micro-batches, and the recomputation granularity for the highest stable throughput within the 24 GB GPU memory budget. Larger models need smaller global batches or finer recomputation to avoid out-of-memory failures; smaller models use larger batches and stage-level recomputation. Warm-up iterations stabilize kernel execution, memory allocation, and pipeline scheduling and are excluded from all statistics.

All Qwen3 dense experiments use full-parameter training with CPU offloading: parameters, gradients, and optimizer states reside in host memory when not in use, and GPU memory holds only the tensors the current schedule needs. The reported peak GPU memory is therefore the maximum resident GPU memory at run time, which is smaller than the training state. All methods share the offloading, batch, recomputation, and hardware settings; only the scheduling policy differs.

B.4 GRAPH ENCODING

We encode one training iteration as a typed directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ represents an executable operation, such as attention, FFN, layer normalization, gate computation, expert computation, P2P transfer, or collective communication. Each edge $(u, v) \in \mathcal{E}$ represents either a data dependency or a resource-ordering relation. Data-dependency edges specify that operation v cannot start until the output of operation u is available, while resource-ordering edges represent serialization constraints among operations that share the same non-preemptive resource.

Each node is assigned a node type t_v, and each edge is assigned an edge type e_{uv}. For compute nodes, the feature vector contains operation-level attributes such as estimated FLOPs, output tensor size, memory footprint, and resource usage indicators. For communication nodes, the feature vector contains transferred bytes, communication type, and the associated link or collective resource. For MoE nodes, we additionally include routing-related statistics, such as expected token load and expert identity. We denote the resulting local node feature by x_v.

To condition the graph on runtime uncertainty and workload context, we concatenate global features to every node. Specifically, ϕ summarizes the fitted duration distribution $\mathcal{D}_\phi$, $\psi(h)$ summarizes the available routing context, and α specifies the risk preference. The encoded input for node v is

$$\bar{x}_v = [x_v \,\|\, \phi \,\|\, \psi(h) \,\|\, \alpha] \,,$$

and the initial hidden representation is

$$h_v^{(0)} = W_{\text{in}} \bar{x}_v .$$

This graph encoding is shared across Dense and MoE workloads: Dense models produce mostly chain-like graphs with P2P communication edges, while MoE models produce branching graphs with gate, All-to-All dispatch, parallel expert, and All-to-All combine nodes.

B.5 NETWORK ARCHITECTURE AND OUTPUT HEADS

The scheduler π_θ is a heterogeneous graph attention network (HAN) over the encoded graph, with initial node states $h_v^{(0)}$ from the previous section.

At layer ℓ, we use type-specific query, key, and value projections for each node type. For an edge (u, v) with edge type e_{uv}, the attention score is

$$a_{uv}^{(\ell)} = \frac{\left(W_{t_v}^{Q} h_v^{(\ell)}\right)^{\top} \left(W_{t_u}^{K} h_u^{(\ell)}\right)}{\sqrt{d_h}} + b_{e_{uv}} ,$$

where $b_{e_{uv}}$ is a learnable edge-type bias. The normalized attention weight over incoming neighbors of v is

$$\alpha_{uv}^{(\ell)} = \frac{\exp(a_{uv}^{(\ell)})}{\sum_{u' \in \mathcal{N}(v)} \exp(a_{u'v}^{(\ell)})} .$$

The node representation is updated by aggregating type-specific value messages:

$$m_v^{(\ell)} = \sum_{u \in \mathcal{N}(v)} \alpha_{uv}^{(\ell)} W_{t_u}^{V} h_u^{(\ell)}, \qquad h_v^{(\ell+1)} = \text{LN}\left(h_v^{(\ell)} + W_O m_v^{(\ell)}\right) .$$

After L_{gnn} message-passing layers, the final node embeddings are used by several task-specific heads. Discrete heads output logits for stage assignment, device placement, and window packing:

$$\ell_v^{\text{stage}} = W_{\text{stage}} h_v^{(L)}, \quad \ell_v^{\text{dev}} = W_{\text{dev}} h_v^{(L)}, \quad \ell_v^{\text{win}} = W_{\text{win}} h_v^{(L)} .$$

During training, these logits are converted into differentiable discrete decisions using the straight-through Gumbel-Softmax estimator. A priority head outputs a scalar priority score

$$s_v = W_{\text{prio}} h_v^{(L)},$$

which is used to construct the soft priority matrix for resource-ordering relaxation. Given soft device assignments A_v, we define

$$[\Pi_z]_{vu} = \left(A_v^{\top} A_u\right) \cdot \text{sigm}\left(\frac{s_u - s_v}{\tau_p}\right) ,$$

where $A_v^\top A_u$ softly indicates whether u and v are assigned to the same resource and τ_p is the priority temperature.

For continuous scheduling decisions, we first compute a graph-level embedding by mean pooling:

$$h_G = \frac{1}{|V|} \sum_{v \in V} h_v^{(L)}.$$

The continuous head predicts Gaussian parameters

$$\mu_z = W_\mu h_G, \qquad \log \sigma_z = W_\sigma h_G,$$

and samples the continuous scheduling variables as

$$z^{\mathrm{c}} = \mu_z + \sigma_z \odot \xi, \qquad \xi \sim \mathcal{N}(0, I).$$

Finally, a separate head predicts the CVaR threshold parameter

$$\tau_\theta = \mathrm{softplus}(W_\tau h_G).$$

One forward pass therefore yields the discrete decisions, the continuous knobs, the priority scores, and the CVaR threshold.

Algorithm 1 TROPOS training and deployment

Require: workload distribution over (G, ϕ); tail probability α; sample budget K; tilt η; cosine schedules for ε, the Gumbel temperature τ_g, and τ_p

1: **for** each training step **do**
2: sample (G, ϕ); run π_θ to obtain decision logits, priority scores s_v, Gaussian parameters, and τ_θ; form the relaxed schedule z (straight-through Gumbel-Softmax, Sinkhorn projection, Π_z from (10), reparameterized continuous knobs)
3: draw $d^{(1)}, \ldots, d^{(K)}$ from q_η in (12) through the duration model (11); compute self-normalized weights ω_k; evaluate $\widetilde{T}^{\varepsilon}(z, d^{(k)})$ by the recursion (8)
4: take a gradient step on (14); advance the temperature schedules
5: **end for**
6: **Deployment:** one forward pass of π_θ; decode categorical heads by argmax and resource orders by sorting s_v; execute the schedule

B.6 Training Hyperparameters

Table 4 lists the training hyperparameters. The policy is a 3-layer HAN with hidden dimension 128, and the CVaR estimator uses $K = 32$ duration samples per step. The smoothing temperature ε, the Gumbel temperature τ_g, and the priority temperature τ_p follow a cosine schedule from 1.0 to 0.03, so training starts from smooth decisions and ends at schedules that decode to hard, executable ones. For the MoE transfer experiment we adapt the dense-trained policy for 50 gradient steps on MoE traces.

Table 4: Training hyperparameters of the TROPOS policy.

Parameter	Value
Number of GNN layers	3
Hidden dimension	128
Monte Carlo samples K	32
CVaR smoothing ε'	0.05
Max smoothing ε	cosine annealed, $1.0 \to 0.03$
Gumbel temperature τ_g	cosine annealed, $1.0 \to 0.03$
Priority temperature τ_p	cosine annealed, $1.0 \to 0.03$
MoE adaptation steps	50

C Additional Experiment Results

C.1 Component Ablation

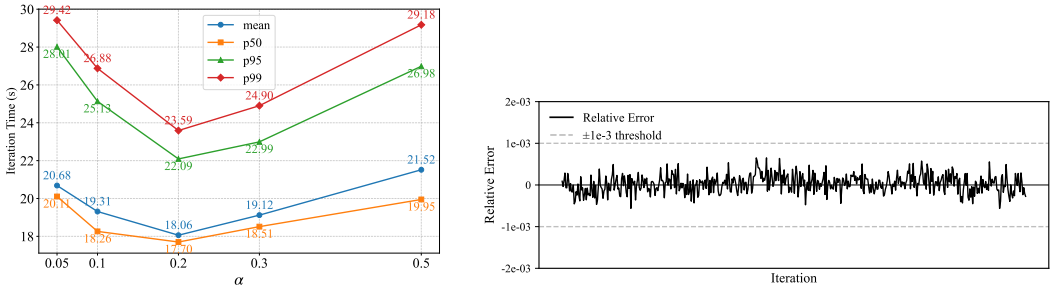
Table 5 changes one component of TROPOS at a time on Qwen3-1.7B. The full system achieves 18.060 s mean and 23.590 s P99.

Table 5: **Component ablation on Qwen3-1.7B.** Each row changes one component of TROPOS; the last row is the full system.

Component	Variant	Mean (s)	P99 (s)
Duration model (Sec. 4.3)	point estimates	21.000	29.862
	no contention term	18.610	24.780
	no residual tail	18.490	25.126
Critical path (Sec. 4.1)	hard max (no convergence)	20.410	30.126
Discrete relaxation (Sec. 4.2)	greedy argmax	20.226	28.740
	Gumbel-Softmax w/o straight-through	18.732	25.604
Resource ordering (Sec. 4.2)	no priority modeling	19.386	27.912
	hard priority order	18.904	26.735
Tail sampling (Sec. 4.4)	naive Monte Carlo, $K = 32$	–	24.692
Solver (Sec. 4.5)	MLP on pooled graph features	19.742	28.136
	HAN without edge types	18.821	25.744
	iterative CVaR solver (286 ms)	17.930	23.214
TROPOS (full, 1.24 ms)		18.060	23.590

C.2 Risk Level and Numerical Equivalence

Figure 4 reports the α sweep for the risk level and the check that TROPOS reproduces RoundPipe's model outputs.



(a) (b)

Figure 4: (a) **Latency versus tail probability α on Qwen3-1.7B.** All four statistics are minimized at $\alpha = 0.2$; larger α ignores the tail and smaller α makes the estimator noisy. (b) **Relative difference between TROPOS and RoundPipe model outputs** stays within $\pm 10^{-3}$ across iterations, so the gains come from scheduling alone.

C.3 Tail-Latency Stability Analysis

Table 6 reports the mean and standard deviation of TROPOS over three end-to-end runs per model.

C.4 Ablation Study

All ablations run on Qwen3-1.7B with the hardware and runtime configuration of Appendix B. Unless stated otherwise, variants share the fitted duration distribution $\mathcal{D}_\phi$, sequence length, global batch size, and measurement protocol, and differ only in the ablated component. Each table reports

Table 6: Run-to-run variation of TROPOS latency (mean $\pm$ standard deviation over three runs).

Model	Mean (s)	P99 (s)
Qwen3-1.7B	18.060 ± 0.471	23.590 ± 0.853
Qwen3-4B	39.410 ± 1.046	51.650 ± 1.847
Qwen3-8B	59.830 ± 1.664	78.230 ± 2.964
Qwen3-14B	49.700 ± 1.284	64.930 ± 2.331
Qwen3-32B	42.000 ± 1.104	55.480 ± 1.936

the metrics relevant to its component: mean and P99 iteration latency, $\text{CVaR}_{0.1}$, throughput, planner delay, or constraint violations.

Smoothing. Table 7 compares the hard maximum, log-sum-exp (LSE) with a fixed temperature, and LSE with cosine annealing from $\varepsilon = 1.0$ to 0.03. The hard maximum does not converge and has the worst tail latency: its gradient reaches only the operations on the current argmax path. A fixed temperature trades two errors against each other, since $\varepsilon = 1.0$ over-smooths the makespan and $\varepsilon = 0.03$ is close to the true objective but concentrates the gradient again. Annealing starts with dense gradients and ends near the hard makespan, and it gives the lowest P99 (23.590 s).

Table 7: Smoothing ablation.

Variant	Mean (s)	P99 (s)	$\text{CVaR}_{0.1}$ (s)	Converged
Hard max	20.410	30.126	28.772	No
LSE, fixed $\varepsilon = 1.0$	18.940	25.864	24.907	Yes
LSE, fixed $\varepsilon = 0.03$	18.520	24.906	24.018	Yes
LSE, cosine annealing	18.060	23.590	22.931	Yes

Discrete relaxation. Table 8 compares ways of handling the categorical decisions. Greedy argmax selects the largest logit and blocks gradient flow. Softmax relaxation is differentiable but trains on schedules that are not executable. Gumbel-Softmax samples the categorical choices; the straight-through (ST) estimator runs the forward pass on hard one-hot decisions while keeping the soft gradient; annealing the temperature closes the remaining gap to executable schedules. Constraint violation is the fraction of decoded schedules that violate a memory, placement, or resource-ordering constraint. Greedy argmax has the highest violation rate (3.8%) and the worst latency; each step toward the full relaxation lowers both, and the annealed straight-through estimator decodes without violations.

Table 8: Gumbel/discrete relaxation ablation.

Variant	Mean (s)	P99 (s)	Thr.	Constraint Viol.
Greedy argmax	20.226	28.740	50902	3.8%
Softmax relaxation	19.104	26.912	53418	2.1%
Gumbel-Softmax	18.732	25.604	54877	1.6%
Gumbel-Softmax + ST	18.285	24.418	56312	0.7%
Gumbel-Softmax + ST + annealing	18.060	23.590	57026	0.0%

Resource ordering. Conflict rate is the fraction of same-resource operation pairs whose decoded order still overlaps or is inconsistent. Without priority modeling the scheduler cannot learn the order of operations that compete for a resource (conflict rate 7.4%, P99 27.912 s). A hard priority order lowers both, but the order itself is not differentiable. The soft priority matrix (10) is, and annealing its temperature sharpens the learned order and gives the lowest conflict rate and P99 (Table 9).

Duration model. Table 10 compares duration models. Point estimate uses one profiled duration per operation, the assumption of the deterministic schedulers. LogNormal only fits a log-normal residual; LogNormal + Pareto adds the heavy-tail component; Contention-aware only uses the inflation term of (11) without a residual; the last row is the full model (11). Rel. Err. is the relative error of the predicted mean iteration time and P99 Err. that of the predicted P99, both against measurement; Mean and P99 are the measured latencies of the schedule trained on each model. Point estimates

Table 9: Soft priority matrix ablation.

Variant	Mean (s)	P99 (s)	$\text{CVaR}_{0.1}$ (s)	Conflict Rate
No priority modeling	19.386	27.912	26.404	7.4%
Hard priority order	18.904	26.735	25.216	4.9%
Soft priority, fixed temp.	18.428	24.772	23.930	2.3%
Soft priority, annealed temp.	18.060	23.590	22.931	1.8%

under-predict by 8.7% in the mean and 18.9% at P99. The log-normal residual removes about half of each error, the Pareto tail most of the remaining P99 error, and the contention term most of the remaining mean error. The full model predicts the mean to within 0.09% and the P99 to within 2.4%, and the schedule trained on it has the lowest measured latency.

Table 10: Duration modeling ablation.

Variant	Rel. Err.	P99 Err.	Mean (s)	P99 (s)
Point estimate	8.7%	18.9%	21.000	29.862
LogNormal only	4.3%	10.6%	19.120	26.440
LogNormal + Pareto	2.1%	5.8%	18.610	24.780
Contention-aware only	2.8%	7.1%	18.490	25.126
Contention-aware + residual tail	0.09%	2.4%	18.060	23.590

Tail sampling. Table 11 compares naive Monte Carlo draws from $\mathcal{D}_\phi$ with the tilted proposal (12) at $K \in \{8, 16, 32\}$; CVaR Var. and Grad. Var. are the variances of the CVaR estimate and of its gradient. Under naive sampling few draws fall in the upper tail, so both variances are high at small K. The tilted proposal lowers both variances by a factor of 1.8 to 2.7 at every K and lowers P99 for the same budget; with $K = 8$ it reaches a lower P99 than naive sampling with $K = 16$.

Table 11: Importance sampling ablation.

Variant	K	CVaR Var.	Grad. Var.	P99 (s)
Naive MC	8	1.84	2.31	26.804
Naive MC	16	1.12	1.67	25.410
Naive MC	32	0.71	1.03	24.692
Importance sampling	8	0.93	1.28	25.118
Importance sampling	16	0.48	0.76	24.031
Importance sampling	32	0.26	0.41	23.590

Policy architecture. Table 12 compares policy architectures. MLP summary pools hand-crafted graph features and ignores topology; GCN uses connectivity with uniform message passing; GAT adds attention over neighbors without node or edge types; the two HAN ablations remove the edge-type biases or the node-type-specific projections. HAN has the lowest latency. Removing either type of information costs 1.7–2.2 s of P99, and the MLP costs 4.5 s. Planner delay grows from 0.38 ms (MLP) to 1.24 ms (HAN), small against an iteration time of 18 s.

Table 12: Solver architecture ablation.

Variant	Mean (s)	P99 (s)	Thr.	Planner Delay (ms)
MLP summary	19.742	28.136	52104	0.38
GCN	19.016	26.502	53966	0.74
GAT	18.534	24.980	55612	1.11
HAN w/o edge types	18.821	25.744	54830	1.03
HAN w/o node types	18.693	25.316	55194	1.06
HAN	18.060	23.590	57026	1.24

Amortization. Table 13 compares planners at run time. The CPU greedy planner makes local decisions from profiled durations; the iterative CVaR solver runs gradient-based search on the objective

for each new graph; the amortized HAN solver is TROPOS. The iterative solver reaches a P99 lower by 0.38 s but needs 286 ms per plan, 230 times the amortized policy. The greedy planner needs 41.7 ms per plan and matches RoundPipe's latency.

Table 13: Amortization ablation.

Planner	Planner Delay (ms)	Mean (s)	P99 (s)	Thr.
CPU greedy planner	41.7	21.000	29.862	50132
Iterative CVaR solver	286.4	17.930	23.214	57412
Amortized HAN solver	1.24	18.060	23.590	57026

C.5 Algorithmic Complexity

One smoothed max-plus forward and backward pass costs $O(|E|)$, the same order as the hard-max recursion, and returns gradients with respect to every duration and every relaxed decision. Estimating the CVaR objective with K duration samples costs $O(K|E|)$ per training step; the $K = 32$ samples are evaluated in parallel on the GPU, and this cost is paid offline. At run time the policy performs one HAN forward pass of cost $O(L_{\text{gnn}}|E|)$ and sorts priority scores per resource, 1.24 ms in total on Qwen3-1.7B (Table 13).

C.6 MFU

Table 14 reports model FLOPs utilization (MFU). MFU rises with model size up to 14B because larger models expose more compute per scheduling decision and amortize fixed run-time overheads. TROPOS raises MFU over RP and RP+Var on every model, by 4.4 points on Qwen3-1.7B and 17.8 points on Qwen3-14B, with the larger gains on the larger models where deterministic schedules lose more to contention and near-critical-path variability. Qwen3-32B has lower MFU than Qwen3-14B under all three methods, consistent with heavier CPU offload and cross-NUMA traffic at that size on this server.

Table 14: Model FLOPs utilization (MFU) across model sizes.

Model	RP	RP+Var	TROPOS
Qwen3-1.7B	31.2%	34.5%	35.6%
Qwen3-4B	34.8%	39.9%	42.6%
Qwen3-8B	38.5%	47.9%	52.0%
Qwen3-14B	41.7%	53.4%	59.5%
Qwen3-32B	39.4%	52.2%	58.5%

C.7 Peak GPU Memory Usage

Table 15 reports the peak resident GPU memory, maximized over the eight GPUs. Peak memory grows with model size up to 14B with the activation buffers, streamed parameter blocks, and temporary buffers. TROPOS adds 0.5–0.8 GB over RP, from more aggressive overlap and prefetching and from scheduling buffers, and stays within the 24 GB budget on every workload. Qwen3-32B uses less peak memory than Qwen3-14B because it runs with a smaller global batch and finer recomputation, which lowers per-micro-batch activation pressure.

Table 15: Peak GPU memory usage across model sizes.

Model	RP	RP+Var	TROPOS
Qwen3-1.7B	18.4 GB	18.7 GB	18.9 GB
Qwen3-4B	19.6 GB	20.1 GB	20.4 GB
Qwen3-8B	21.2 GB	21.7 GB	22.0 GB
Qwen3-14B	22.6 GB	23.0 GB	23.2 GB
Qwen3-32B	20.8 GB	21.1 GB	21.4 GB

D Error Analysis

We classify each candidate schedule on two axes: whether the surrogate predicts a P99 improvement over RP+Var, and whether the measured P99 improves. Most schedules fall in the predicted-improved/measured-improved cell. The dominant failure is predicted-improved/measured-regressed: the surrogate predicts a gain that the measurement does not show. The residual error is therefore a cost-model mismatch, concentrated on schedules with many stage boundaries; the search and the decoding return the schedules the surrogate prefers.

The representative failure partitions Qwen3-4B into 24 stages, against the 16–18 stages that the corrected objective selects. In the surrogate the finer partition exposes more overlap and shortens several near-critical compute segments. At run time each extra boundary adds activation movement, stream synchronization, and pipeline scheduling overhead; the cost per boundary is small, but 24 of them shift the measured critical path enough that the measured P99 regresses while the predicted P99 still improves on RP+Var.

We add a partition penalty to the loss (14),

$$\mathcal{L}(\theta) + \lambda_{\text{part}} \frac{N_{\text{stage}}(z)}{N_{\text{layer}}},$$

where $N_{\text{stage}}(z)$ is the number of stages under z and N_{layer} the number of Transformer layers. The penalty does not force coarse partitions; an additional stage has to lower the tail enough to pay for its boundary cost. Table 16 sweeps λ_{part}. Pred. Err. is the relative error of the surrogate's P99 prediction against measurement, and Fail. Rate is the fraction of schedules predicted to improve P99 over RP+Var whose measured P99 regressed. Without the penalty the average stage count is 24.6 and the failure rate 20.8%; at $\lambda_{\text{part}} = 0.03$ the average stage count is 16.8, the failure rate 8.3%, and both mean and P99 latency are lowest. Larger penalties lower the prediction error further but under-partition and raise latency.

The remaining failures are high-contention cases in which the boundary cost is not constant: it depends on activation size, stream synchronization, and concurrent communication pressure, which a fixed penalty averages over. The repair we see is a schedule-conditioned model of boundary overhead inside the duration model.

Table 16: Error analysis of over-partitioning on Qwen3-4B LoRA fine-tuning.

Variant	Avg. Stages	Mean (s)	P99 (s)	Pred. Err.	Fail. Rate
No penalty	24.6	42.36	56.18	10.7%	20.8%
$\lambda_{\text{part}} = 0.01$	20.2	41.18	53.72	6.8%	16.7%
$\lambda_{\text{part}} = 0.03$	16.8	39.84	50.63	3.1%	8.3%
$\lambda_{\text{part}} = 0.05$	14.7	40.26	51.08	2.4%	8.3%
$\lambda_{\text{part}} = 0.10$	10.9	42.05	54.91	1.1%	12.5%

E Discussion

Limitations. The best stage-partition granularity differs between training modes of the same model family. On Qwen3 LoRA fine-tuning the policy over-partitions more often than on full-parameter training, because fine-tuning shifts the balance between computation, activation movement, synchronization, and boundary overhead, and the duration model fits these mode-dependent costs less well. A cost model conditioned on training mode, together with the schedule-conditioned boundary term of Appendix D, would address this.

TROPOS treats SM quotas and kernel tile sizes as continuous scheduling variables, but the prototype does not rewrite vendor GEMM, attention, or NCCL kernels. It approximates these controls with runtime knobs: chunk granularity, stream priority, communication-window assignment, and launch concurrency. This keeps the integration with PyTorch and RoundPipe simple, and it leaves to a custom CUDA executor whatever gain direct kernel control would add. TROPOS models contention and schedules around it; it does not remove contention at the kernel level.