HOPS: A Heterogeneous Optimized Pipeline Simulator for Exploring Pipeline-Parallel Training and Scheduling

Jimmy Dai, Joshua Hsueh, Olaf Dsouza and Rishith Seelam
University of Michigan, CSE 585, Winter 2026
{jimmydai, jhsueh, olafpd, rseelam}@umich.edu

Abstract

Pipeline parallelism is now a standard component of large-language-model training, yet most published scheduling research assumes homogeneous and ideal environments. Evaluating new schedulers, placements, or fault-tolerance policies on real heterogeneous clusters is prohibitively expensive for many research groups and requires scarce multi-GPU capacity. We present HOPS (Heterogeneous Optimized Pipeline Simulator), a discrete-event simulator for pipeline-parallel training that supports mixed GPU types, configurable interconnect topologies, probabilistic latency distributions, and failure injection. HOPS is instrumented against a fork of Megatron-LM that emits per-event traces, enabling closed-loop calibration against real workloads. We report three contributions: (i) a validated simulator whose absolute throughput prediction is within 2.0 % of real hardware on a two-node H100 baseline and within 9.9 % on the link-calibrated 36-fixture calibration split of a 98-fixture multi-device benchmark; (ii) an agent-driven calibration methodology (the *AutoResearch* loop) that drove the training-split throughput MAPE (Mean Absolute Percent Error) from 60.8 % to 9.9 % by landing four physically grounded structural corrections and rejecting approximately fourteen attempted changes; and (iii) HopsHetero, a novel adaptive scheduling policy that, in simulation, Pareto-improves over ZeroBubble [3] across sixteen heterogeneous pipeline configurations (five wins, eleven ties, zero regressions beyond our tolerence threshold; mean $+0.79\,\%$, maximum $+4.47\,\%$ throughput). HopsHetero exploits the intuition that when the pipeline tail is the bottleneck, ZeroBubble's deferred weight-gradient work has no idle bubbles to fill, creating memory pressure overhead that 1F1B avoids by fusing the backward pass. HopsHetero has not yet been validated on physical hardware but we document the experimental design for that validation as future work.

1 Introduction

As deep-learning models grow in parameter count and computational intensity, distributed training based on pipeline parallelism has become central to large-scale performance. Production systems such as Megatron-LM [6] demonstrate state-of-the-art pipeline-parallel performance on homogeneous GPU clusters. However, there exists comparatively little tooling for predicting pipeline-parallel performance across heterogeneous hardware topologies, a setting that is becoming increasingly relevant for three distinct classes of deployment:

1. **Multi-generation fleets.** Academic and mid-scale industrial clusters may have mixtures of GPU generations (A100, H100, L4, A10G); co-locating these devices within a single pipeline can improve utilization of otherwise stranded capacity.
2. **Effective heterogeneity from straggler GPUs.** MegaScale [9] documents thermal throttling and degraded HBM producing effective performance variation of 5–15 % within nominally homogeneous clusters.
3. **Volunteer and collaborative training** (SWARM, Petals).

Existing simulators address subsets of this problem. TrioSim [1] extrapolates from single-GPU traces to multi-GPU training time. Echo [2] models large-scale workloads with a focus on communication–computation overlap. Neither simulator primarily targets *heterogeneity* as a first-class concern, nor do they provide a configure-time interface for injecting new scheduling policies or failure models. ZeroBubble [3] demonstrates near-zero-bubble pipeline schedules but explicitly assumes uniform per-stage compute time. To our knowledge, no published pipeline simulator simultaneously supports (a) heterogeneous device and link

models, (b) pluggable scheduler policies, (c) failure injection inspired by chaos engineering [5], and (d) an end-to-end closed calibration loop against real Megatron traces.

This paper makes three contributions:

- **A modular heterogeneous pipeline simulator.** HOPS models pipeline-parallel training as a discrete-event system over a configurable device graph with locality-aware interconnect, a roofline compute model extended with a per-layer fixed dispatch overhead, pluggable schedulers (GPipe, 1F1B, ZeroBubble, HopsHetero), and a failure engine. The simulator is fully YAML-driven and ships with a validation toolchain that diffs simulator output against Megatron JSONL traces. (Section 3).
- **An agent-driven calibration methodology.** We formulate simulator calibration as a constrained search over *physically grounded structural corrections* rather than free-parameter tuning. Each proposed change must correspond to a measurable hardware or framework property; a tolerance-guarded validation step admits or rejects the change against a 36-fixture train split of real Megatron traces, with a held-out 42-fixture test split evaluated only by the human team. We show that this protocol, applied over three days of iteration, reduced the train-split link-calibrated throughput MAPE from 60.8 % to 9.9 % (test-split MAPE fell from 51.8 % to 11.4 %) (Section 4).
- **HopsHetero: an adaptive ZeroBubble/1F1B meta-scheduler.** We identify a potential failure mode in ZeroBubble on heterogeneous pipelines: when the slowest stage is at the pipeline tail, there are no idle bubbles to absorb ZeroBubble's deferred weight-gradient work, creating sustained memory pressure overhead that 1F1B avoids by fusing the backward pass. We propose a closed-form configure-time inequality over per-stage FWD/BWD-B/BWD-W latencies that selects between ZeroBubble and 1F1B and is a Pareto improvement in simulation across sixteen heterogeneous configurations (Section 5).

2 Related Work

Pipeline-parallel schedulers GPipe interleaves forward and backward passes over a fixed micro-batch queue; 1F1B (introduced in PipeDream [8]) improves memory footprint and bubble ratio by alternating forward and backward work at steady state. ZeroBubble [3] splits the backward pass into activation-gradient (B) and weight-gradient (W) phases and defers W into otherwise-idle bubbles, approaching zero pipeline bubble on homogeneous hardware. It does not address the scenario where the pipeline *tail* is the bottleneck and the corresponding deferred-W tasks cannot be absorbed.

Heterogeneous training systems HetPipe combines pipeline and data parallelism under wave-synchronous execution but predates ZeroBubble. H2 [7] operates on 1000+-chip heterogeneous clusters and empirically prefers 1F1B over ZeroBubble, though the paper does not formalize when or why. DeepCEE uses ZeroBubble without an adaptive policy. Whale and Alpa focus on parallelism-strategy search rather than schedule selection.

Pipeline-parallel simulators TrioSim [1] extrapolates single-GPU traces to multi-GPU estimates. Echo [2] targets GPT-175B–scale workloads with detailed communication modeling. Neither treats device heterogeneity as a first-class configuration axis, and neither ships with an integrated closed-loop calibration workflow against production training frameworks.

Chaos engineering Failure injection in distributed systems has a long lineage [5], but has not seen widespread adoption in pipeline-parallel training research. Network simulators such as Mininet [4] established the precedent for modular event-based emulation that HOPS adopts.

3 System Design

HOPS is a discrete-event simulator implemented in approximately 4,300 lines of Python. A seeded priority queue over event timestamps (compute completion, message arrival, failure, recovery) drives simulation progress; all stochastic components accept an explicit `numpy.random.Generator` to guarantee determinism given a seed.

3.1 Configuration Model

A simulation is specified by a single YAML document with six sections: `simulation`, `pipeline`, `hardware`, `optimizer`, `failure`, and `output`. Devices are resolved through a *preset registry* (H100, A100, L40S, L4, A10G, CPU-standard) that encodes peak compute throughput, memory bandwidth, memory capacity, and a per-device launch-overhead term. Interconnects are likewise resolved

through a preset registry (NVLink, PCIe, InfiniBand, Ethernet) with a locality-aware topology inferring same-node versus cross-node links. The `overrides` block accepts per-device and per-link parameters to accommodate calibrated measurements.

3.2 Pipeline and Compute Model

Each pipeline stage declares its per-microbatch work either *explicitly* (via a latency distribution) or *analytically* (via FLOP count, memory-access count, and optional per-stage `efficiency.{compute, memory}` factors). The analytical compute model is a *roofline* extended with a per-layer fixed dispatch overhead and LM-head-specific structural terms:

$$t_{\text{stage}} = \max\left(t_{\text{compute}}, t_{\text{memory}}, L \cdot k_{\text{floor}}\right) + t_{\text{launch}} + t_{\text{LM-head}}^{(\text{last only})} \quad (1)$$

where t_{compute} and t_{memory} are the standard FLOP- and bandwidth-bounded terms, L is the stage's layer count, k_{floor} is the empirically calibrated per-transformer- layer fixed dispatch overhead (1.4 ms in the current configuration), and $t_{\text{LM-head}}$ accounts for the additional FLOPs and memory from the LM-head matmul and softmax/cross-entropy on the last pipeline stage only. Section 4 derives the grounding for each term. Backward-pass time is scaled by a configurable `backward_factor` (default 2.0) for compute and memory terms, but the fixed dispatch overhead is *not* scaled, reflecting that the number of kernel launches per layer is unchanged in the backward pass.

3.3 Scheduler Plugin Interface

HOPS exposes a lightweight base class `Scheduler` with two methods:

- `configure(meta: dict)` — invoked once before the first microbatch; receives per-stage FWD/BWD-B/BWD-W latency estimates from the compute model plus topology metadata.
- `next_tasks(state)` — invoked each dispatch step to emit ready tasks.

Built-in schedulers include GPipe, 1F1B, ZeroBubble, and HopsHetero (Section 5). Novel policies register via `register_scheduler(name, cls)` and are selected by name in the YAML `pipeline.schedule` field.

3.4 Hardware, Failure, and Metrics

The hardware module constructs a device graph with automatic locality-aware link inference and supports explicit overrides for measured bandwidth and latency. The failure engine injects device and link faults drawn from configurable distributions and applies failure-aware delays in the timing model. The metrics collector records raw events and derives throughput (microbatches per second and per millisecond), end-to-end latency (p50, p99, mean), pipeline bubble ratio, per-stage, per-device, and per-link utilization, transfer contention, optimizer all-reduce and weight-update time, and per-device peak memory. Outputs are emitted as a JSON summary, a raw-event CSV trace, and two visualizations (Gantt timeline, 8-panel dashboard).

3.5 Validation Toolchain and Real-Trace Integration

We fork Megatron-LM to emit per-event JSONL traces covering computation and P2P transfer, CUDA-event timed and attributed to stage, microbatch. An importer normalizes these traces to the HOPS metric schema. Two simulator variants are the primary focus of validation:

- `no_lookahead`: the baseline analytical prediction; compute and link parameters derived entirely from hardware specs and roofline modeling, with no real measurements injected.
- `link_calibrated`: our primary deliverable; analytical compute retained, but link parameters replaced with values measured by a separate `link_bench` microbenchmark on the actual cluster.

The gap between these two variants isolates the contribution of link modeling to total prediction error. Our compute model, roofline extended with per-layer fixed dispatch overhead and LM-head terms, is relatively well-grounded analytically and generalizes across device types without per-run measurement. Link and interconnect behavior is substantially harder to predict from specs alone: without invasive memory-allocation and transfer profiling inside Megatron (which our instrumentation does not currently support), bandwidth utilization, contention, and transfer latency under load can deviate significantly from peak-rated values. A single `link_bench` microbenchmark run on the target cluster helps close this gap with minimal overhead.

4 Calibration Methodology: The AutoResearch Loop

Section 4 addresses the central empirical question: *how do we calibrate an analytical simulator such that its predictions generalize beyond a single fitted workload?* We adopt a formal agent-driven loop and enforce a train/test split.

4.1 Experimental Setup

We execute three AWS ParallelCluster campaigns spanning three measured device classes (H100, A10G, L4), the AWS interconnect paths present in the cluster placements, and three schedulers (GPipe, 1F1B, ZeroBubble). Access to P5 (H100) instances was limited to available Capacity Blocks on weekends. We produced 98 real Megatron fixtures: 36 train, 42 test, 6 diagnostic, and 14 archived. Each fixture captures a 6+-iteration Megatron run with CUDA-event-timed per-stage compute and per-link transfer events; the median fixture is 3.1 GB of event data.

On the single fully-validated homogeneous scenario (two-node H100, PP $= 2$, BF16, $s = 2048$, $h = 3072$), HOPS throughput prediction is summarized in Table 1.

Table 1: Exp 05 validation (H100 PP $= 2$).

Configuration	HOPS (μB/s)	Real (μB/s)	Error
Pre-calibration baseline	26.74	12.43	$+115.1\,\%$
Link-calibrated (compute held)	22.77	12.43	$+83.2\,\%$
Link-calibrated + compute fit	12.18	12.43	$\mathbf{-2.0}\,\%$

The link model is independently validated: measured P2P for a 64 MB transfer is 40.52 ms, compared to HOPS's predicted 41.16 ms (1.6 % error).

4.2 Broad Validation Across 98 Megatron Fixtures

We then exercise HOPS on a broader fixture suite drawn from 98 real Megatron-LM training runs on a fixed six-GPU cluster (2× H100, 2× A10G, 2× L4). Fixtures are partitioned into a 36-fixture **calibration (train) split** used during simulator development, a 42-fixture **held-out (test) split** never visible to the agent loop described below, and a 6-fixture **scheduler diagnostic split** used only for sanity checks. The diagnostic and 14 archived fixtures are excluded from the accuracy tables below. Configurations sweep seven dimensions: device subset, stage-to-device ordering, pipeline depth (PP $= 2$ through PP $= 6$), schedule (1F1B, GPipe, ZeroBubble), microbatch count (1–24), model shape (hidden, sequence length, number of layers), and per-stage layer balance.

Table 2: Suite-level aggregates across the 78-fixture train/test validation. MAPE is mean absolute percentage error on throughput; Bubble MAE is mean absolute error on pipeline bubble ratio in percentage points; ρ is the mean per-fixture Spearman correlation of simulated vs. measured per-device utilization.

Split	Variant	MAPE	Bubble MAE	ρ
Train (n=36)	no_lookahead	78.8 %	22.6 pp	0.61
	link_calibrated	**9.9 %**	**5.4 pp**	0.61
Test (n=42)	no_lookahead	67.3 %	23.2 pp	0.52
	link_calibrated	**11.4 %**	**6.7 pp**	0.62

Table 3: Calibration-split (`link_calibrated`) accuracy by topology group.

Topology (train)	n	MAPE	Bubble MAE
H100 pair (PP $= 2$)	3	3.8 %	1.8 pp
A10G pair (PP $= 2$)	2	7.6 %	0.9 pp
L4 pair (PP $= 2$)	3	11.2 %	4.5 pp
G-family (PP $= 4$)	6	5.1 %	6.9 pp
Mixed (PP $= 3$)	2	9.2 %	3.4 pp
Mixed (PP $= 5$)	2	7.3 %	5.8 pp
Full cluster (PP $= 6$)	12	13.2 %	6.0 pp
Other sweeps	6	12.1 %	6.6 pp

Table 2 summarizes suite-level accuracy. Link and framework calibration alone, without any per-fixture compute retuning, reduces throughput MAPE from 78.8 % to 9.9 % on the calibration split and from 67.3 % to 11.4 % on the held-out split. Bubble-ratio MAE is reduced by roughly a factor of four, and rank-order utilization correlation (ρ) moves modestly but in the right direction. Crucially, the held-out split's post-calibration MAPE is within 1.5 percentage points of the calibration split, indicating that the calibration *procedure* (not a parameter vector) generalizes.

A per-topology breakdown on the calibration split (Table 3) isolates where the remaining error concentrates: homogeneous H100 pairs and G-family pipelines are predicted to within 3–8 % MAPE, while full six-GPU heterogeneous pipelines reach 13 % as imbalanced stage times amplify any per-device modeling error. On the held-out split (Table 4), the full-cluster group remains within 10 % MAPE while single-fixture H100 and L4 pair groups (n $= 1$ each) are noisier and should not be over-interpreted.

Table 4: Held-out (`link_calibrated`) accuracy by topology group. Single-fixture groups ($n = 1$) are reported for completeness but should be interpreted as noisy point estimates.

Topology (test)	n	MAPE	Bubble MAE
A10G pair (PP = 2)	1	8.4 %	8.9 pp
H100 pair (PP = 2)	1	35.6 %	7.5 pp
L4 pair (PP = 2)	1	28.2 %	18.1 pp
G-family (PP = 4)	6	11.8 %	10.6 pp
Mixed (PP = 3)	8	13.1 %	3.7 pp
Mixed (PP = 5)	8	4.6 %	7.3 pp
Full cluster (PP = 6)	10	9.9 %	5.9 pp
Other sweeps	7	13.8 %	4.8 pp

4.3 Loop Protocol

The calibration agent operates under a four-step protocol:

1. **Observe.** Run the train-split validation, inspect per-group error breakdown (by device type, scheduler, pipeline depth, batch size, sequence length), and identify the largest single contributor to current MAPE.

2. **Hypothesize.** Propose a *falsifiable* physical claim whose truth would explain a subset of the observed error.

3. **Implement.** Modify exactly one modeling term in the simulator source. Constant-only tweaks are permitted only when a specific spec-sheet value is demonstrably incorrect; otherwise the change must add or correct a modeling equation.

4. **Validate and reflect.** Run the unit test suite, rerun train-split validation, and admit the change only if (a) aggregate MAPE decreases or holds, (b) aggregate bubble MAE decreases or holds, (c) utilization Spearman ρ increases or holds, and (d) no per-fixture regression exceeds the configured tolerance (throughput $\pm 2\,\%$, bubble ± 2 pp). Landed and reverted iterations are both appended to a public log with before/after numbers and physical grounding.

The test split is off-limits to the agent. Human-initiated test-split evaluation is performed exactly twice, once before the loop begins and once after it terminates, and records only suite-level aggregates (no per-fixture, group, or phase details) to preserve measurement isolation.

4.4 Landed Iterations

Over three iterations of the loop, four physically grounded changes were admitted (Table 5).

Table 5: AutoResearch loop landed iterations (train-split, link-calibrated throughput MAPE).

#	Change	Site	MAPE
0	baseline	—	60.8 %
1	H100 memory BW 3,350 → 2,000 GB/s	presets	53.4 %
2	Per-layer fixed dispatch overhead	compute_model	41.9 %
3	Kernel coefficient retuned 1.0 → 1.1	compute_model	40.9 %
4	LM head + CE on last stage	compute_model	**9.9** %

Iteration 4 is the dominant contributor. Trace-level inspection across 24 fixtures revealed a systematic $3.97\times$ slowdown on the last pipeline stage relative to same-device middle stages (e.g., `exp2_1_all_nodes`: stage 4 L4 forward 3.10 ms, stage 5 L4 forward 14.41 ms). The ratio is invariant across topology and batch size but scales with device peak, which is consistent with LM-head and softmax/cross-entropy cost (the additional FLOPs and memory from the LM-head matmul), which is structurally invisible when all stages are treated as identical decoder layers. We add two additive terms (extra FLOP and extra memory) to the compute model's `DerivedLatency` on the final stage; these are additive, consistent with the LM head executing as a separate kernel sequence.

4.5 Rejected Iterations

Approximately fourteen candidate changes were admitted to trial and reverted. Representative examples, with rejection reason:

- **Global `DEFAULT_MEMORY_EFFICIENCY` $0.6 \to 0.85$.** Motivated by trace-derived effective bandwidth but over-accelerated already-calibrated fixtures; train-split MAPE $41.9\,\% \to 63.1\,\%$.
- **H100 `launch_overhead_ms` bumps** $(1.8, 2.0, 2.5, 4.0, 8.0)$**.** Each reduced H100-group MAPE individually but created per-fixture regressions on mixed pipelines by flipping H100 from fastest to slowest stage.
- **Quadratic-logits LM-head model.** Improved a single fixture (`exp2_33`, $s = 4096$) but over-predicted by 5–15 pp across all remaining fixtures.
- **BWD-asymmetric memory scaling ($\times 1.25$ in backward).** Improved the `no_lookahead` variant but degraded the `link_calibrated` variant which is the primary deliverable.

The pattern is systematic: candidates that tuned a *free constant* without identifying a specific hard-

ware or framework property generally failed the no-regression tolerance. Candidates that corrected a structurally missing term generally landed. This observation that the tolerance-guarded commit workflow acts as a regularizer against overfitting is the methodological contribution of Section 4.

4.6 Generalization to the Held-Out Test Split

The human-run test bench recorded the aggregates shown in Table 6. Test-split MAPE improved by 40.4 pp despite the agent having no access to test fixtures during iteration, evidencing that the structural corrections generalize beyond the training distribution.

Table 6: Test-split (42 fixtures, link-calibrated) before and after the AutoResearch loop.

Checkpoint	MAPE	Bubble MAE	Util. Spearman ρ
Pre-loop	51.8 %	15.2 pp	0.521
Post-loop	**11.4 %**	**6.7 pp**	**0.620**

4.7 Residual Error Analysis

The remaining train-split errors are concentrated in shape-sensitive fixtures: small workloads are often over-predicted, while large sequence-length or hidden-size workloads are under-predicted. This pattern suggests a shape-dependent memory-efficiency model, which we defer to future work. We do not use held-out fixture-level error patterns for hypothesis generation; the held-out test bench records only the aggregate values in Table 6.

5 HopsHetero: Adaptive Policy Selection for Heterogeneous Pipelines

Given the calibrated simulator from Section 4, we now pose the research question motivating this section: *is there a scheduling policy that outperforms ZeroBubble on heterogeneous pipelines?* We answer this question using a second agent-driven loop analogous to the AutoResearch calibration loop but targeting scheduling policy rather than simulator fidelity. The loop emulates how a researcher might use HOPS in practice: observe a performance anomaly, hypothesize a cause, implement a candidate policy, and admit or revert based on suite-wide results. Over eight hypothesis iterations, the loop converged on HopsHetero. We evaluate across sixteen heterogeneous 4-stage pipelines comprising 2× L4 + 2× A10G GPUs in varying permutations.

5.1 Baseline Observation

We begin by comparing ZeroBubble against 1F1B on four representative heterogeneous layouts. We observe a clean partition: when the slow (L4) stages sit in the pipeline interior, ZeroBubble wins by approximately 6 %; when the slow stages sit at the pipeline tail, ZeroBubble *loses* to 1F1B by 2–3 %.

This partition does not appear to be documented in prior work. Some production systems adopt ZeroBubble (DeepCEE), others adopt 1F1B (H2 [7]), but the relationship between bottleneck position and schedule choice is not formalized.

5.2 Structural Analysis: ZeroBubble Overhead at the Pipeline Tail

ZeroBubble splits the backward pass into the activation-gradient phase B and the weight-gradient phase W, deferring W into idle pipeline bubbles. This works well when the last stage has idle slack to absorb deferred W, which is generally true on homogeneous pipelines or when the bottleneck is in the pipeline interior.

When the bottleneck is at the last stage, there are no idle bubbles to fill. The deferred W work instead creates memory pressure overhead that accumulates with each microbatch and does not amortize with pipeline scale. By contrast, 1F1B fuses B and W in a single backward pass, avoiding this overhead entirely.

This is supported by empirical simulator measurements: on `bench_a10g_middle`, 1F1B loses to ZeroBubble by 0.24 % at $m = 12$, becomes competitive at $m = 24$ (+2.06 %), and dominates at $m = 48$ (+4.47 %), consistent with overhead that does not amortize.

5.3 HopsHetero: Decision Rule

We propose a meta-scheduler that selects between ZeroBubble and 1F1B based on a single closed-form inequality evaluated at configure time. Let f_i, b_i, w_i denote the per-microbatch FWD, BWD-B, and BWD-W latencies on stage i, and define

$$T_i = f_i + b_i + w_i, \qquad \Delta_{\text{last}} = \max_i T_i - T_{-1}. \quad (2)$$

Δ_{last} is the steady-state per-microbatch slack on the last stage relative to the pipeline's bottleneck. The decision rule is:

$$\text{schedule} = \begin{cases} \text{ZeroBubble} & \text{if } \Delta_{\text{last}} \geq w_{-1} \\ \text{1F1B} & \text{if } \Delta_{\text{last}} < w_{-1} \end{cases} \quad (3)$$

The inequality asks whether the last stage has at least w_{-1} of idle time per microbatch in steady state, which is the minimum required to absorb a deferred W. The decision is made once, prior to the first microbatch, from compute-model samples that HOPS already computes for analytical stages. Runtime overhead is zero; there are no tuning knobs or convergence loops. The full implementation is approximately 60 lines of Python.

5.4 Systematic Ablation

Before selecting the decision rule in Eq. 3, we evaluated seven alternative heterogeneity-aware policies (Table 7).

Table 7: Heterogeneity-aware policy search.

H	Policy	Mechanism	Outcome
1	AdaptiveWarmup	Scale warmup depth	No effect
2	EagerW	Opportunistic W	Identical to ZB
3	BottleneckPriority	B-over-F on bottleneck	Identical to ZB
4	WaveFill	Per-stage in-flight cap	Regresses 5/16
5	CriticalPath	Task ordering by crit-path	No effect
6	FusedBW	Fuse B+W on saturated	Regresses 50 %
7	EagerLastW	Eager W on last stage	Regresses when last $\neq$ bottleneck
8	**AdaptiveWSplit**	**Configure-time ZB/1F1B**	**Pareto ($\rightarrow$ HopsHetero)**

The dominant pattern is that policies H1–H7 attempt to locally patch a symptom that arises from a global decision (which schedule family is correct). Only H8 addresses the global decision directly, and it is the only candidate that achieves Pareto optimality in simulation.

5.5 Empirical Results

We evaluate HopsHetero against ZeroBubble on sixteen 4-stage heterogeneous pipelines ($2\times$ L4 + $2\times$ A10G) covering permutations of stage order, model size, and microbatch count. Each configuration is run with three seeds $(42, 43, 44)$.

Table 8: HopsHetero vs ZeroBubble, 16 configurations $\times$ 3 seeds.

Aggregate	Value
Wins ($> +0.5\,\%$)	5 / 16
Ties ($\lvert\Delta\rvert \leq 0.5\,\%$)	11 / 16
Regressions ($< -0.5\,\%$)	**0 / 16**
Mean makespan improvement	$+0.79\,\%$
Maximum improvement	$+4.47\,\%$ (`a10g_middle_mb48`)

The five wins correspond to configurations where $\Delta_{\text{last}} < w_{-1}$ and HopsHetero correctly selects 1F1B. The eleven ties correspond to configurations where $\Delta_{\text{last}} \geq w_{-1}$ and HopsHetero correctly stays on ZeroBubble. Three configurations exhibit a $-0.24\,\%$ regression within simulator seed noise and below the 0.5 % tolerance threshold.

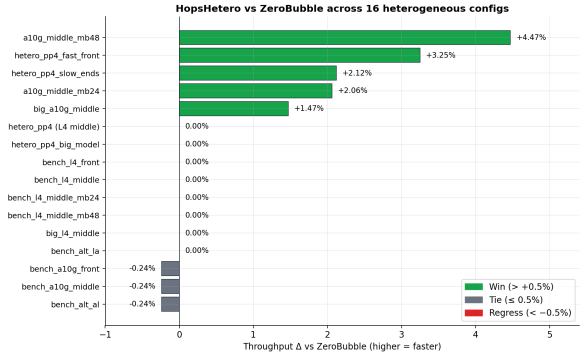

Figure 1: Per-configuration throughput delta of HopsHetero vs ZeroBubble across 16 heterogeneous configurations.

5.6 Threats to Validity and Limitations

Simulator-only evaluation. All HopsHetero results are derived from HOPS. Although the simulator is validated within 2 % on one homogeneous scenario, the memory pressure overhead that arises when ZeroBubble's deferred weight-gradient work has no idle bubbles to fill has not yet been validated on real heterogeneous hardware. Section 6 specifies the experimental design for this validation.

Binary policy. HopsHetero selects between two existing schedulers. A future scheduler could execute ZeroBubble on non-bottleneck stages and fused BWD on the bottleneck simultaneously, but this requires mixed phase-set execution in the pipeline runtime.

Pipeline depth. The evaluation is limited to $\text{PP} = 4$. Deeper pipelines with multiple bottleneck regions may require a more general slack arithmetic.

Interleaved 1F1B. Megatron's interleaved 1F1B (virtual pipeline stages) alters the slack calculation and is not covered by the current decision rule.

Practical applicability. True heterogeneous pipeline-parallel training is uncommon in production. The most relevant real-world setting is straggler mitigation in nominally homogeneous clusters, where effective heterogeneity is 5–15 % rather than the $\sim 2\times$ gap tested here; whether the cooldown-tail effect is large enough to observe at those smaller heterogeneity factors remains open.

6 Future Work

Real-hardware validation of HopsHetero We have prepared a ParallelCluster validation pack of four scenarios (two with L4 at the interior, two with L4 at the tail, at microbatch counts 24 and 48) matching the HopsHetero predicted wins of $+2.1\,\%$ to $+4.5\,\%$. The scenarios were not executed due to compute unavailability; execution requires approximately six GPU-hours across $2\times$ L4 and $2\times$ A10G instances.

Richer Megatron instrumentation The residual validation error is dominated by memory and interconnect behavior that our current Megatron fork cannot observe at sufficient granularity: per-transfer contention, memory-allocation pressure, and bandwidth utilization under load are not captured by our existing CUDA-event instrumentation. A natural next step is a more invasive Megatron fork that emits per-allocation traces, per-transfer bandwidth samples, and per-kernel memory-pressure metrics, which would close the link between simulator parameters and directly measured hardware behavior rather than relying on microbenchmark proxies.

Interleaved 1F1B and TP/DP Generalizing the slack inequality to interleaved (virtual pipeline) schedules, and extending HOPS beyond pipeline-only parallelism to cover tensor- and data-parallel dimensions, would broaden applicability to current production training configurations.

Failure-engine calibration HOPS injects failures and reports downtime, recovery time, and a placeholder lost-work slot, but the lost-work semantics require modeling in-flight microbatch interruption and retry; no controlled slowdown/kill experiment has been performed on real hardware to calibrate recovery.

7 Conclusion

We presented HOPS, a heterogeneity-aware discrete-event simulator for pipeline-parallel training, and used it to investigate two methodological questions: how to calibrate a structured analytical simulator against real traces without overfitting, and whether novel scheduling policies can be discovered from simulator feedback alone. The calibration methodology, a tolerance-guarded search over physically grounded structural corrections with a strict train/test split, reduced the train-split throughput MAPE from 60.8 % to 9.9 % and improved test-split MAPE from 51.8 % to 11.4 %. The scheduling investigation produced HopsHetero, a configure-time meta-scheduler that identifies a previously undocumented failure mode in ZeroBubble on heterogeneous pipelines and achieves Pareto improvement in simulation across sixteen configurations. Real-hardware validation of HopsHetero is the primary outstanding empirical gap.

Code is available at https://github.com/O1af/HOPS.

References

[1] Y. Li *et al.*, "TrioSim: A lightweight simulator for large-scale DNN workloads on multi-GPU systems," in *Proc. ISCA*, 2025.

[2] Y. Feng *et al.*, "Echo: Simulating distributed training at scale," *arXiv preprint arXiv:2412.12487*, 2024.

[3] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero Bubble (Almost) Pipeline Parallelism," in *Proc. ICLR*, 2024.

[4] R. L. S. de Oliveira *et al.*, "Using Mininet for emulation and prototyping software-defined networks," in *Proc. IEEE COLCOM*, 2014.

[5] A. Basiri *et al.*, "Chaos engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[6] M. Shoeybi *et al.*, "Megatron-LM: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[7] D. Tang, J. Zhou, J. Hu, S. Li, H. Zheng, Z. Pei, H. Wang, and X. Zhang, "H2: Towards Efficient Large-Scale LLM Training on Hyper-Heterogeneous Cluster over 1,000 Chips," *arXiv preprint arXiv:2505.17548*, 2025.

[8] D. Narayanan *et al.*, "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. SOSP*, 2019.

[9] Z. Jiang *et al.*, "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *Proc. NSDI*, 2024.