FROM ESTIMATION TO EXPERIENCE: BUILDING REUSABLE DEPLOYMENT KNOWLEDGE FOR LLM PARALLEL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training large language models (LLMs) in data center networks (DCNs) requires exploring a large space of parallelization and resource configurations. Yet real-system observations are costly to obtain and therefore sparse, making deployment experience scarce and slow to accumulate. A natural alternative is to use performance estimators for low-cost exploration, but existing estimators typically produce scenario-specific solutions rather than reusable deployment knowledge. We therefore ask whether low-cost estimation can be converted into reusable deployment knowledge, and whether accumulated experience can in turn guide subsequent estimation. To this end, we propose `Est2Exp` (Estimation to Experience), a unified framework that couples a performance estimator with an experience bank through bidirectional feedback. The estimator uses a hybrid-granularity directed acyclic graph (DAG) to capture computation–communication dependencies, and LLM-assisted calibration with limited real-system measurements to refine latency functions, including irregular behaviors that are difficult to capture with conventional physical models. The experience bank distills estimator outputs into reusable deployment knowledge, which is reused to warm-start subsequent exploration and jointly refined with the estimator as new real-system measurements become available. Experimental results show that `Est2Exp` achieves 99.3–99.9% DAG-level latency accuracy across evaluated configurations with limited real-system measurements. Reusing accumulated deployment experience further improves cross-scale search success from 73.3% to 98.3% while substantially reducing search overhead.

1 INTRODUCTION

Training large language models (LLMs) involves massive computation, so the training workload is distributed across a GPU cluster. Distributed training employs parallelization strategies such as data, tensor, and pipeline parallelism. These strategies determine how computation is partitioned across GPUs and what communication is induced over network links, so the choice of parallelization strategy strongly affects training performance. Exploring these deployment choices through repeated physical training runs is both time-consuming and expensive. For example, GPT-4.5 is estimated to have used 40,000–100,000 H100 GPUs for 90–165 days, with a final training-run cost of approximately $386 million (You, 2025). Even the post-training stage of DeepSeek-R1 required 512 H800 GPUs for about 80 hours (DeepSeek-AI et al., 2025).

To reduce this cost, performance estimators for distributed LLM training can estimate training performance and evaluate system configurations before physical deployment (Wang et al., 2025; Gui et al., 2025; Yang et al., 2025). For example, `vTrain` searches for cost-effective LLM parallelization strategies (Bang et al., 2024), while `HyGra` estimates inter-node communication latency and effective link throughput under different parallelization strategies (Wang et al., 2026). An estimator typically produces a solution for a specific deployment scenario, but such a solution is one-off and cannot be directly reused. When the model, cluster, or deployment conditions change, the solution usually needs to be regenerated through repeated estimation.

An experience bank can organize such deployment solutions for reuse by subsequent tasks. For example, `AscendOptimizer` stores reusable operator-optimization experience (Wu et al., 2026),

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.

while `Kernel Foundry` accumulates reusable GPU-kernel optimization knowledge (Huang et al., 2026). These experience banks are built from real-hardware measurements. Such measurements are affordable for relatively lightweight operator and kernel optimization tasks, but applying the same approach to LLM training would require costly and time-consuming physical training runs.

Existing work targeting large-scale LLM training therefore tends to treat the two approaches separately: performance estimators generate scenario-specific deployment results, while experience-bank approaches rely on a small number of real-hardware evaluations to accumulate limited reusable experience. However, combining them makes the two complementary: the estimator can provide low-cost evidence to validate and expand the experience bank, while the experience bank can reuse prior deployment knowledge to prioritize promising configurations and guide subsequent estimation. We therefore combine performance estimation and the experience bank in a unified framework.

Realizing this design presents three challenges:

1) *How can the estimator and the experience bank mutually reinforce each other?* Estimation results must be distilled into reusable deployment experience, while accumulated experience should in turn guide subsequent estimation and exploration. Establishing this bidirectional feedback across deployment settings is difficult.
2) *How can we develop an accurate estimator from sparse real-system measurements?* This requires capturing execution dependencies at an appropriate granularity and accurately estimating computation and communication performance, including irregular small-message communication. Sparse and costly real-system measurements further complicate reliable estimation.
3) *How can estimator results be distilled into reliable and reusable deployment experience?* Deployment knowledge may arise from either individual factors or interactions among multiple factors. Meanwhile, each extracted finding must have clear applicability conditions and sufficient evidence to support reliable reuse. Distilling such results into reusable experience is therefore difficult.

To address these challenges, we develop `Est2Exp`, which integrates performance estimation with reusable deployment experience. Our main contributions are fourfold.

- **A bidirectional estimator–experience framework.** We tightly couple a performance estimator with an experience bank through three feedback loops for estimator calibration, experience generation and reuse, and conflict resolution. These loops allow estimation and accumulated experience to continuously reinforce each other. To the best of our knowledge, `Est2Exp` is the first framework to couple performance estimation with the construction and reuse of deployment experience for distributed LLM training in in data center networks (DCNs). (Section 3)
- **A hybrid-granularity estimator with LLM-assisted calibration.** We construct a hybrid-granularity computation–communication DAG to capture execution dependencies under different parallel strategies while balancing estimation accuracy and efficiency. With limited real-system measurements, LLM-assisted calibration improves system latency estimation, including latency behaviors that are difficult to capture with conventional models. (Section 4)
- **Estimator-driven experience distillation and reuse.** We distill estimator results into reusable deployment experience, using direct extraction for single-factor findings and interpretable policy search for multi-factor interactions. Rather than relying on manual experience construction, an LLM-powered agent automatically organizes and consolidates these findings, enabling efficient accumulation and reuse of deployment knowledge across settings. (Section 5)
- **Real-system evaluation on Ascend NPUs.** We evaluate `Est2Exp` on Ascend NPUs in a real DCN environment, where accumulated deployment experience is relatively limited compared with NVIDIA GPU platforms. The results validate the proposed estimator and experience bank, and demonstrate the framework's effectiveness in identifying useful deployment experience for distributed LLM training. (Section 6)

2 Related Work

2.1 Performance Estimation for Distributed LLM Training

Existing performance estimators for distributed LLM training generally follow three modeling paradigms: physics-based modeling, data-driven modeling, and hybrid modeling. *Physics-based*

methods estimate performance using analytical models of computation, hardware, and communication, but without support from real observations, their accuracy can be affected when model assumptions or parameters deviate from the actual system (Won et al., 2023; Moolchandani et al., 2023; Isaev et al., 2023; Kundu et al., 2024; Qi et al., 2017). *Data-driven methods* rely on profiling measurements or execution traces and can reflect actual system performance, but collecting sufficient data is expensive, and the resulting observations usually cover only a limited set of configurations (Lu et al., 2023; Liang et al., 2025; Hu et al., 2022). *Hybrid methods* combine system models with empirical measurements to overcome the limitations of relying on either source alone. Measurements ground performance estimates in actual system behavior, while models extend evaluation to unmeasured configurations, thereby balancing estimation accuracy and configuration coverage with controlled measurement overhead (Wang et al., 2025; Feng et al., 2024; Duan et al., 2023; Xie et al., 2024; Choi et al., 2020). Together, these advances have steadily improved estimation accuracy.

Limitations of existing estimators. First, many existing performance estimators use general physical models to characterize computation and communication. These models estimate latency primarily from message size and effective bandwidth and are well suited to bulk communication, including gradient synchronization in data parallelism (DP), intermediate-result exchange in tensor parallelism (TP), and activation transfer in pipeline parallelism (PP). However, not all communication follows this pattern: control, synchronization, and runtime activities may exchange very little data, sometimes only a single scalar, yet exhibit disproportionately high latency. Applying the same size-bandwidth relationship to such activities may therefore underestimate the measured latency. Second, existing performance estimators typically retain configuration-search outcomes as scenario-specific solutions rather than structured deployment experience. Although a previous solution may serve as an initial point for a new search, it does not capture the underlying performance trend or applicability conditions, limiting its ability to guide deployment across changing scenarios.

2.2 Experience Banks for System Optimization

Unlike performance estimators, experience banks accumulate and organize knowledge obtained from previous tasks, enabling such knowledge to be reused to guide subsequent optimization. Existing experience-bank studies related to distributed LLM training have mainly focused on scenarios where experience can be collected from real hardware at relatively low cost. A representative example is operator optimization for AI accelerators, including NVIDIA GPUs and Ascend NPUs. For example, existing studies record candidate code, optimization actions, correctness results, and performance gains to guide subsequent compute-operator optimization, while `CUCo` further accumulates experience from the joint optimization of compute and communication operators (Wu et al., 2026; Li et al., 2026; Dong et al., 2026; Huang et al., 2026; Zhang et al., 2026; 2025; Sun et al., 2026; Zheng et al., 2026; Hu et al., 2026).

Limitations of existing experience banks. Experience acquisition in these scenarios is relatively inexpensive: compute operators can typically be evaluated on a single accelerator, while communication operators require only a small group of accelerators. In contrast, the scenario considered in this work, distributed LLM training in DCNs, requires large-scale accelerator clusters and involves an extensive deployment space. Consequently, collecting deployment experience directly from real-system executions is substantially more costly and difficult to scale.

Summary. Performance estimators provide low-cost evaluation of distributed LLM training configurations, while experience banks enable optimization knowledge to be accumulated and reused. Yet these two capabilities have largely been developed separately, and their integration for distributed LLM training in DCNs remains underexplored. Our work addresses this gap by coupling a performance estimator with an experience bank through bidirectional feedback, allowing the two to continuously reinforce each other.

3 Est2Exp Framework

3.1 Deployment Difficulties for LLM Training in DCNs

Distributed LLM training in DCNs faces two major deployment difficulties:

- **Large deployment space.** Distributed LLM training involves multiple deployment dimensions, including parallelization strategies, resource allocation, and device placement. Their combinations form a large candidate space, while training performance can vary substantially across

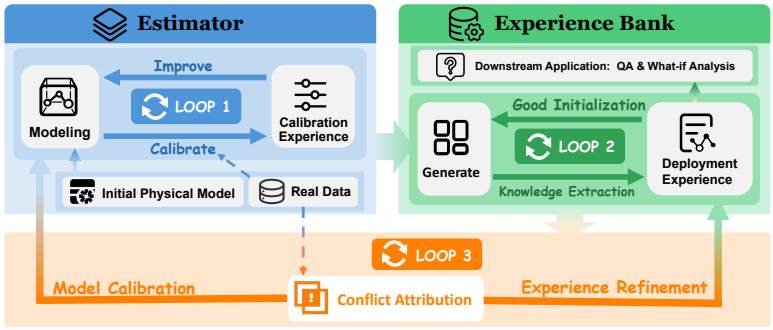

Figure 1: Est2Exp Framework.

configurations, making efficient deployment selection difficult (Liu et al., 2024; Zhu et al., 2025; Lin et al., 2025; Shi et al., 2025).

- **Sparse and expensive real-system observations.** LLM training in DCNs may involve hundreds, thousands, or even tens of thousands of accelerators, making each real-system evaluation costly in both time and computational resources. Exhaustively evaluating the large deployment space is therefore impractical, leaving real-system observations sparse and covering only a small fraction of configurations under different task and resource conditions.

3.2 Unified Estimator-Experience Framework

As shown in Fig. 1, Est2Exp consists of two core components: a performance estimator and a deployment experience bank.

- **Estimator.** The estimator combines an initial physical model with limited real-system measurements to estimate training performance under a given task, resource setting, and deployment configuration. It provides low-cost quantitative results for comparing candidate configurations and exploring deployment experience.
- **Experience bank.** The experience bank distills estimator results into reusable deployment knowledge and organizes accumulated experience for subsequent tasks. Such experience guides future deployment exploration, reducing repeated search and improving knowledge reuse.

The two components are connected through three feedback loops:

1) **Estimator calibration loop.** Real-system measurements are compared with estimator outputs to calibrate its latency functions. Validated calibration results are further stored as reusable calibration experience, allowing similar cases to benefit from previous corrections. This helps the estimator reflect real-system behavior even with sparse measurements.
2) **Experience generation and reuse loop.** Estimator results are distilled into reusable deployment experience and accumulated in the experience bank. Existing experience is then reused to warm-start new exploration and prioritize promising configurations. Through this loop, deployment experience can be continuously explored, accumulated, and refined at low cost.
3) **Conflict resolution and joint refinement loop.** As new real-system measurements are collected and the estimator is updated, newly derived deployment results may conflict with existing experience. When such conflicts arise, their source is verified using available measurements: estimator errors trigger recalibration, while inaccurate or overly generalized experience is refined in the experience bank. This joint correction improves the reliability of both components as new evidence accumulates.

The estimator provides low-cost performance estimates, while the experience bank converts accumulated results into reusable deployment knowledge. The three feedback loops couple these two components into a unified framework, enabling mutual reinforcement and co-evolution.

4 Estimator

We start by constructing an execution model for LLM training, then refine its latency functions through LLM-guided calibration against real profiling measurements, and finally organize validated calibration results into a reusable experience bank, yielding an accurate latency estimator.

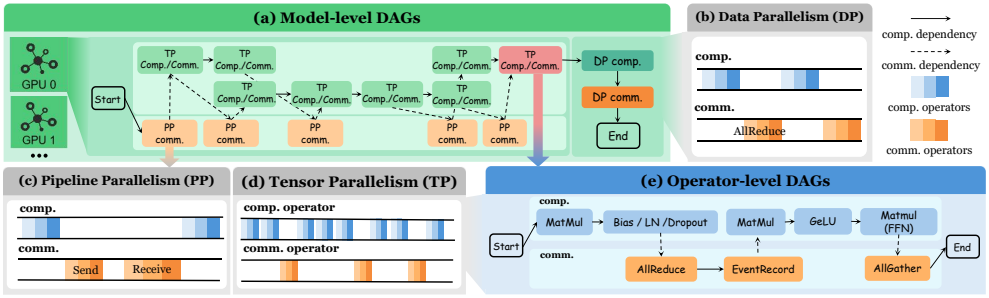

Figure 2: Hybrid-granularity DAG representation for distributed LLM training.

4.1 Hybrid-Granularity DAG for Computation–Communication Dependencies

LLM training workloads exhibit task dependencies and computation–communication overlap that cannot be captured by modeling network transmission alone. We therefore represent each training workload as an execution DAG. As illustrated in Fig. 2(a), each node denotes an execution unit, corresponding to either a computation or communication operator, or a group of operators, while each edge represents an execution dependency. The latency of each execution unit is assigned to its outgoing dependency edges and counted only once along each execution path.

Granularity design. As illustrated in Fig. 2(b) and 2(c), DP and PP operate over large execution domains, where computation and communication form clearly separated execution regions. This separation allows operators of the same type to be aggregated, making a *model-level DAG* sufficient to capture the overall computation–communication dependencies. In contrast, as illustrated in Fig. 2(d), frequent synchronization divides TP execution into short computation–communication segments, in which the output of a computation operator directly feeds the subsequent communication operator. We therefore represent TP using an *operator-level DAG* to capture the fine-grained computation–communication dependencies within each model layer.

DAG construction. We construct the hybrid-granularity DAG in a top-down manner. For the model-level DAG (Fig. 2(a)), dependencies are derived from layer execution order and coordination patterns imposed by each parallel strategy, such as pipeline scheduling in PP and gradient synchronization in DP. When finer-grained modeling is required, as in TP, such high-level information is insufficient to recover operator ordering and synchronization dependencies. We therefore follow `Maya` (Yarlagadda et al., 2026) and intercept CPU-side API calls and synchronization events to construct the operator-level DAG (Fig. 2(e)).

This hybrid-granularity DAG provides a basis for performance estimation and analysis, including:

1) *End-to-end performance estimation:* estimating training latency through critical-path analysis.
2) *Critical-path bottleneck analysis:* identifying execution regions that dominate total latency.
3) *Computation–communication overlap analysis:* using execution dependencies to identify opportunities for overlapping communication with computation.

4.2 LLM-Guided Automatic Function Calibration

The hybrid-granularity DAG specifies the execution structure that must be preserved, but not the latency of each execution unit. This raises the next question: *how can these latencies be estimated reliably from sparse real-system measurements?* We illustrate our calibration method using DP communication as a representative example.

A DP execution unit contains both regular and irregular communication behaviors, whose latency can be decomposed as

$$T_{\mathrm{DP}} = T_{\mathrm{regular}} + T_{\mathrm{irregular}}. \tag{1}$$

Here, T_{regular} captures communication whose latency mainly follows data volume and bandwidth, such as DP AllReduce, while $T_{\mathrm{irregular}}$ captures synchronization and runtime overheads that deviate from this relationship. These characteristics motivate two calibration procedures.

Regular communication calibration. For regular communication, the function form is known and can be expressed as

$$T_{\mathrm{regular}} = \sum\nolimits_s V_s/(\eta_s B_s) + \delta_s, \tag{2}$$

where V_s, B_s, η_s, and δ_s denote data size, bandwidth, bandwidth efficiency, and startup latency for communication step s, respectively. Since the function form is known, the LLM only uses sparse real-system measurements to identify suitable parameter ranges.

Irregular communication calibration. The data-volume-dominated function becomes inaccurate when synchronization and runtime overheads dominate. For example, synchronizing a 32-bit scalar in DP takes 55.8 ms in our profiling, whereas the regular function predicts only about 0.6 ms. We therefore use the LLM to search for an interpretable function form, such as

$$T_{\text{irregular}} = \alpha + \beta \log_2 P + \gamma I_{\text{inter-node}}. \tag{3}$$

Here, α represents the latency of one synchronization round trip, $\beta \log_2 P$ captures the increasing synchronization overhead as the number of participating accelerators P grows, while $\gamma I_{\text{inter-node}}$ captures the additional overhead when communication spans multiple nodes. Once the function form is identified, its parameters are fitted from real-system measurements as in regular calibration.

Computation calibration. The same calibration principle applies to computation units: known latency functions are calibrated directly from sparse profiling measurements, while poorly modeled behaviors are handled by LLM-guided function search and measurement-based parameter fitting.

4.3 Calibration Experience Bank

Repeatedly rediscovering calibration functions and parameters for similar configurations incurs unnecessary profiling and search costs. We therefore store validated calibration results in a reusable experience bank, consisting of *function-selection experience* and *coefficient-validation experience*. For regular communication, where the function form is known, prior experience narrows the search for suitable coefficients; for irregular communication, it further guides candidate function selection before parameter fitting and validation.[1]

5 Estimator-Driven Deployment Experience Bank

This section describes how `Est2Exp` transforms estimator results into reusable deployment experience and how accumulated experience guides subsequent estimation and exploration.

5.1 Estimator-Driven Experience Construction

Experience representation. For distributed LLM training in DCNs, reusable experience must specify where it applies and how it is supported. We therefore represent each experience entry as $\mathcal{E} = \langle C, F, V \rangle$, where C specifies its applicability conditions, F the deployment finding, and V its supporting evidence and validation status. For example, C may describe the model, cluster, and hardware conditions, while F specifies a preferred relation among PP, TP, and micro-batch count (MB). Given this structure, an LLM-powered agent automatically summarizes estimator results into experience entries, reducing manual effort.

Deployment findings vary in complexity. Some can be isolated by varying a single factor, whereas others emerge from interactions among multiple deployment dimensions. We therefore use two complementary mechanisms: direct evidence extraction and interpretable policy search.

1) **Direct evidence extraction.** When one factor can be isolated, we fix other conditions, vary it, and compare estimator outputs. For example, varying TP reveals when cross-node communication degrades performance, while perturbing DAG-node latencies identifies performance-critical execution units.
2) **Interpretable policy search.** When multiple deployment dimensions interact, their joint effect is difficult to extract directly. We therefore adapt FunSearch (Romera-Paredes et al., 2024) to search for interpretable scoring programs. Each candidate program ranks parallel configurations; the estimator evaluates the top-ranked configuration, and programs yielding lower latency are retained and iteratively refined. The following example illustrates one searched policy. Its coefficients and thresholds are specific to the corresponding deployment scenario.

[1] The complete calibration experience bank is provided in Appendix A.

```
# N=128 GPUs, E=192 experts, C=24 per EP rank
def score(ctx):
    score = 0.0
    if ctx.pp_degree >= 4.0:
        score += 20.0  # Favor deeper PP
    if ctx.expert_capacity_pressure > 0.5:
        score -= 10.0  # Avoid overloaded EP ranks
    if ctx.microbatch_count >= 2.0 * ctx.pp_degree:
        score += 15.0  # Favor larger MB to fill the pipeline
    if ctx.tp_degree <= 2.0:
        score += 5.0   # Limit TP communication
    return score
```

The retained policy is summarized as `PP >= 4, MB >= 2PP, TP <= 2, E/C<= EP <2E/C DP = N/(PP*TP)`, with scenario-specific thresholds.

Experience consolidation. Different scenarios may produce policies with similar deployment preferences. To consolidate them, we first identify the deployment dimensions that most clearly distinguish these preferences and then group policies with similar patterns into higher-level experience. For example, across the 128-GPU slow-device scenarios, PP, TP, and MB reveal two recurring patterns: localized cases tend toward deeper PP, smaller TP, and larger MB, whereas broader or heavier slow-device cases tend toward shallow PP, moderate TP, and smaller MB.

5.2 Experience Reuse and Joint Refinement

We next describe how solution-based and policy-based experience support subsequent deployment exploration in Loop 2, and how newly collected real-system evidence is used in Loop 3 to refine stored experience when inconsistencies arise.

Experience-guided deployment exploration. The two forms of experience support deployment exploration in complementary ways and can be reused in three modes:

1) *Solution-based experience only.* Known local relationships narrow the search space. For example, on a cluster with 8 NPUs per node, if cross-node TP degrades performance, TP can be restricted to node-local values before configuration search.
2) *Policy-based experience only.* Prior policy experience provides a warm start by prioritizing candidate configurations. For example, a policy from a similar slow-device scenario allows the estimator to evaluate promising configurations first.
3) *Both forms of experience.* Solution-based experience constrains the search space, while policy-based experience prioritizes the remaining candidates.

Conflict-driven experience refinement. For example, new 256-NPU measurements may yield experience that differs from previous 128-NPU scenarios. Loop 3 uses these new measurements to determine whether the discrepancy arises from estimator error or limited applicability of the stored experience. Estimator errors trigger recalibration, while experience-side conflicts update (F, V) or narrow the applicability conditions C.

6 Evaluation

6.1 Experimental Setup

Real-system environment: Our experiments use Ascend 910B/910C NPUs deployed across multiple servers. The NPUs are organized into affinity groups, where devices within the same group are connected through high-bandwidth HCCS links (1,400 Gbps aggregate bandwidth), while communication across groups uses 200-Gbps RoCE. CANN and HCCL provide device execution and collective communication support.

Profiling data: Using MindStudio Profiler, we collect real-system training traces spanning model sizes from 30B to 505B parameters and cluster scales from 128 to 16K NPUs. Specifically, the traces cover 30B models on 128, 256, and 2K NPUs, a 90B model on 6K NPUs, and a 505B model on 16K NPUs. Because large-scale real-system profiling is costly, the available measurements are inherently sparse. As shown later, `Est2Exp` still achieves accurate estimation with sparse measurements.

Table 1: Execution-unit and DAG longest-path latency accuracy before and after calibration under the 30B-128 configuration.

DP execution unit†	$\mathrm{Acc}_{\mathrm{unit}}$ (%) ↑		DP execution unit‡	$\mathrm{Acc}_{\mathrm{total}}$ (%) ↑	
	Initial	+Profile		Initial	+Profile
Z3 G-RS DP-CP	4.1	**99.5**	DP Scalar AR	1.3	**98.2**
Z3 G-RS DP-EP	3.6	**99.3**	DP-CP expert load AR	50.58	**94.8**
Z3 MUON G-AG DP-CP	96.3	**99.5**	DP Scalar AR tail	**32.1**	31.2
Z3 next P-AG DP-CP	97.0	**99.3**	All GPU AR 0	63.2	**95.1**
Z3 MUON G-AG DP-EP	96.3	**99.7**	All GPU AR 1	**55.0**	42.2
Z3 next P-AG DP-EP	96.3	**99.7**	All GPU AG	35.6	**99.2**
DAG longest-path latency accuracy ($\mathrm{Acc}_{\mathrm{total}}$, %) ↑				93.6	**99.3**

† Regular execution units; ‡ irregular execution units.

Table 2: Execution-unit and DAG-level latency accuracy across configurations after calibration.

Scenario	Execution	$\mathrm{Acc}_{\mathrm{unit}}$ (%) ↑		$\mathrm{Acc}_{\mathrm{total}}$ (%) ↑	
		Initial	+Profile	Initial	+Profile
30B-128	DP+TP	68.27	**94.09**	98.62	**99.30**
30B-256	DP+TP	30.99	**86.51**	98.59	**99.83**
30B-2K	DP+TP	40.80	**92.24**	99.11	**99.87**
90B-6K	DP+TP	9.77	**78.73**	**99.74**	99.70
505B-16K	DP+TP+PP	81.26	**94.04**	99.55	**99.61**

Metrics: We evaluate `Est2Exp` fidelity using execution-unit latency accuracy ($\mathrm{Acc}_{\mathrm{unit}}$) and the accuracy of the execution-DAG longest-path latency($\mathrm{Acc}_{\mathrm{total}}$), which corresponds to the total training latency. We evaluate deployment experience reuse using target-reaching success rate (Succ.), iterations to target (Iter.), and evaluations to target (Eval.).

6.2 Estimator Accuracy

Calibration accuracy. Table 1 compares the latency accuracy before and after calibration based on profiling results from the 30B model running on 128 NPUs. We uses DP as a representative case, covering 12 regular and irregular execution-unit categories under CP, EP, ZeRO-3, and the MUON optimizer. The calibration loop improves the accuracy of most execution units. In particular, Z3 G-RS DP-CP and Z3 G-RS DP-EP improve from 4.1% and 3.6% to 99.5% and 99.3% after calibration captures previously unmodeled communication behavior. The DAG longest-path latency accuracy consequently increases from 93.6% to 99.3%. Some irregular units, such as DP Scalar AR tail and All GPU AR 1, remain inaccurate because the available traces provide insufficient evidence, indicating the need for more targeted profiling rather than simple parameter fitting.

Cross-scale extrapolation. Table 2 evaluates the calibrated estimator across model and cluster scales. The 30B-2K, 90B-6K, and 505B-16K scenarios are not used for calibration and serve only as extrapolation tests. Their execution-unit accuracies improve from 40.80%, 9.77%, and 81.26% to 92.24%, 78.73%, and 94.04%, respectively, while DAG-level accuracy remains above 99.6%. These results show that the calibrated estimator maintains high accuracy on unseen configurations, supporting extrapolation to substantially larger LLM training scales.

6.3 Deployment Experience Construction and Reuse

Experience construction. We construct a heterogeneous 128-GPU topology with fast and slow GPUs and select four representative scenarios to evaluate whether `Est2Exp` can extract meaningful deployment experience.[2] Table 3 summarizes the experience extracted through the two mechanisms. Solution-based analysis provides coarse-grained guidance from controlled comparisons, whereas policy-based analysis produces scenario-conditioned constraints on PP, TP, DP, EP, and MB across broader deployment settings. The former guides specific deployment choices, while the latter provides structured constraints for narrowing the deployment search space.

Experience-guided search. We further reuse the 128-GPU experience bank to warm-start deployment search on 256 GPUs. Cold-start search follows the same procedure without prior experience and serves as the baseline. As shown in Ta-

Table 4: Cross-scale deployment search with policy-based warm start.

Method	Success	Iter.	Eval.
Cold start	73.3%	3	12
Policy warm start	**98.3%**	**1**	**4**

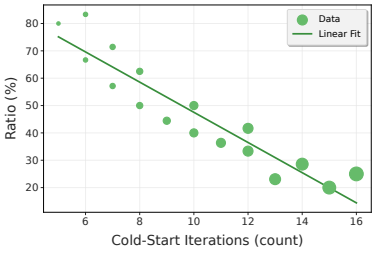

Figure 3: Warm-to-cold iteration ratio as cold-start search length increases.

[2]The complete deployment experience bank and its construction procedure are provided in Appendix B.

Table 3: Deployment experience obtained from solution-based and policy-based analysis in representative scenarios.

Scenario	Solution-based experience	Policy-based experience
PP-stage placement	Prefer aligning PP-stage boundaries with affinity-group boundaries. The appropriate PP depth is determined separately rather than fixed by the number of affinity groups.	For localized degradation, representative policies include $PP \geq 4, TP \leq PP/2, DP = N/(PP \times TP)$, and $MB \geq 2PP$.
TP locality	Prefer placing TP ranks within the same node. When inter-node placement is necessary, keeping the TP group within the same affinity group is preferred to crossing affinity groups.	TP is constrained jointly with other parallel dimensions. Representative policies include $PP = 1, TP = 4$, $DP = N/4, MB = 1$, or $TP \leq PP/2$ under localized degradation.
DP grouping	Separating fast and slow GPUs into homogeneous DP groups does not by itself improve performance. TP, EP, and PP synchronization exposure must also be considered.	DP is mainly constrained jointly with PP and TP through $DP = N/(PP \times TP)$. No independent policy requiring homogeneous fast/slow DP groups is consistently observed.
Slow-GPU distribution	Under a fixed deep-pipeline configuration, concentrating slow GPUs within fewer stages and synchronization domains is preferable to broadly dispersing the same number of slow GPUs.	Localized degradation commonly yields $PP \geq 4$, $TP \leq PP/2$, and $MB \geq 2PP$, whereas broadly affected scenarios frequently yield $PP = 1, TP = 4, DP = N/4$, and $MB = 1$.

ble 4, warm start achieves a target-reaching success rate of 98.3%, compared with 73.3% for cold start. Among successful runs, candidate evaluations decrease from 12 to 4. Figure 4 further shows that the warm-to-cold iteration ratio generally decreases as cold-start search becomes longer, indicating larger relative savings for harder search scenarios.

6.4 Real-System Feedback for Experience Refinement

Conflict diagnosis and refinement. We further evaluate whether `Est2Exp` can determine whether a conflict between the estimator and stored experience arises from estimation error or outdated experience. We use the DP-CP and DP-EP AllGather communication units in the 30B-2K setting as a controlled case. Experience derived from the 128- and 256-GPU configurations indicates that DP-CP AllGather is faster than DP-EP AllGather, whereas the estimator under the 2K configuration predicts the opposite ordering. The 2K profiling measurements confirm the estimator's ordering, with DP-CP and DP-EP latencies of 95.219 ms and 88.148 ms, respectively. `Est2Exp` therefore identifies the conflict as experience-side, restricts the previous experience to its supported configurations, and records the 2K observation as a counterexample. The estimator remains unchanged. This case demonstrates that `Est2Exp` can use real profiling feedback to determine which side of an estimator–experience conflict should be updated and refine the experience bank accordingly.

Experience-bank application. Finally, as an application showcase, we organize the constructed deployment experience into a DeepWiki-based knowledge base for distributed training. It supports intelligent Q&A on parallelization strategies and deployment choices across training environments, as well as what-if analysis under changes in system conditions or deployment configurations[3].

7 Discussion

Evolving capability frontier. As described in Sections 4 and 5, LLM-powered agents automate function calibration and experience construction. As the reasoning and search capabilities of LLMs and agents continue to advance, these automated processes can become increasingly effective, further expanding the capability frontier of the proposed framework. In particular, stronger agents may identify more suitable latency functions from sparse measurements and extract more generalizable deployment experience from complex search results. This allows `Est2Exp` to benefit directly from future advances in agent reasoning without redesigning its overall framework.

Broader applicability. Although this work focuses on distributed LLM training in DCNs, the framework can also extend to other AI system optimization scenarios, such as concurrent LLM inference, edge-cloud deployment, and accelerator-system co-design. By combining low-cost estimation with experience reuse, these domains can more efficiently explore deployment strategies and accumulate reusable system knowledge. The specific estimator and deployment variables may differ across systems, but the underlying workflow remains similar: estimate system behavior, distill reusable experience, and use accumulated experience to guide subsequent exploration. This makes the estimator-experience paradigm applicable beyond the training setting studied in this work.

[3] A complete demonstration is provided in Appendix C.

AI USE STATEMENT

In this work, we did not use generative AI tools for research tasks requiring disclosure under the ICLR AI policy; the research problem, methodology, system design, experiments, and core contributions were developed by the authors. Additionally, we used generative AI tools for literature retrieval and discovery, including identifying potentially relevant related work, and for language polishing, grammar correction, and improving the clarity, readability, and positioning of the manuscript's contributions. We independently reviewed and verified all AI-assisted work. In particular, all retrieved references were manually checked for relevance and correctness, and all AI-assisted text and suggestions were reviewed and revised by the authors. We take responsibility for the final content of this work, including any text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the methodological and experimental details needed to reproduce the main results throughout the paper and appendices. The Estimator section specifies the hybrid-granularity DAG construction, latency functions, and LLM-guided calibration procedure, while the Estimator-Driven Deployment Experience Bank section describes the experience representation, construction, reuse, and conflict-resolution mechanisms. The Evaluation section reports the hardware and software environment, profiling workloads, evaluation metrics, experimental configurations, and comparison protocols. The appendices provide the complete calibration and deployment-experience records, the corresponding construction procedure, and demonstrations of deployment question answering and evidence-based what-if analysis. Exact replication of the largest experiments requires access to comparable Ascend clusters; nevertheless, the reported topology, configurations, profiling measurements, and evaluation protocol support reimplementation and validation on available hardware.

REFERENCES

Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. vtrain: A simulation framework for evaluating cost-effective and compute-optimal large language model training. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 153–167, 2024.

Jaemin Choi, David Richards, Laxmikant V. Kale, and Abhinav Bhatele. End-to-end performance modeling of distributed GPU applications. In *ACM International Conference on Supercomputing (ICS)*, 2020.

DeepSeek-AI et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 2025.

Kris Shengjun Dong, Sahil Modi, Dima Nikiforov, Sana Damani, Edward Lin, Siva Kumar Sastry Hari, and Christos Kozyrakis. Kernelblaster: Continual cross-task CUDA optimization via memory-augmented in-context reinforcement learning. *arXiv preprint arXiv:2602.14293*, 2026.

Jiangfei Duan, Xiuhong Li, Ping Xu, Xingcheng Zhang, Shengen Yan, Yun Liang, and Dahua Lin. Proteus: Simulating the performance of distributed DNN training. *arXiv preprint arXiv:2306.02267*, 2023.

Yicheng Feng, Yuetao Chen, Kaiwen Chen, Jingzong Li, Tianyuan Wu, Peng Cheng, Chuan Wu, Wei Wang, Tsung-Yi Ho, and Hong Xu. Echo: Simulating distributed training at scale. *arXiv preprint arXiv:2412.12487*, 2024.

Fei Gui, Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Ran Zhang, Hongbing Yang, and Dian Xiong. Accelerating design space exploration for LLM training systems with multi-experiment parallel simulation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 473–488, 2025.

Bodun Hu, Yoga Sri Varshan, Saurabh Agarwal, and Aditya Akella. CUCo: An agentic framework for compute and communication co-design. *arXiv preprint arXiv:2603.02376*, 2026.

Hanpeng Hu, Chenyu Jiang, Yuchen Zhong, Yanghua Peng, Chuan Wu, Yibo Zhu, Haibin Lin, and Chuanxiong Guo. dPRO: A generic profiling and optimization system for expediting distributed DNN training. *arXiv preprint arXiv:2205.02473*, 2022.

Zixuan Huang, Da Chen, Kecheng Huang, Lihao Yin, Xing Li, Huiling Zhen, Mingxuan Yuan, and Zili Shao. Kernel foundry: A diagnosis-driven evolutionary kernel optimizer with multi-experts. *arXiv preprint arXiv:2605.30359*, 2026.

Mikhail Isaev, Nic McDonald, Larry Dennison, and Richard Vuduc. Calculon: A methodology and tool for high-level codesign of systems and large language models. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2023.

Joyjit Kundu, Wenzhe Guo, Ali BanaGozar, Udari De Alwis, Sourav Sengupta, Puneet Gupta, and Arindam Mallik. Performance modeling and workload analysis of distributed large language model training and inference. In *IEEE International Symposium on Workload Characterization (IISWC)*, pp. 57–67, 2024.

Dongfang Li, Xiaodong Luo, Ruoyu Sun, Xuhui Chen, et al. SLAI T-Rex: Full-parameter post-training of the DeepSeek-V4 family on ascend superpod. *arXiv preprint arXiv:2607.20145*, 2026.

Mingyu Liang, Hiwot Tadese Kassa, Wenyin Fu, Brian Coutinho, Louis Feng, and Christina Delimitrou. Lumos: Efficient performance modeling and estimation for large-scale LLM training. *arXiv preprint arXiv:2504.09307*, 2025.

Hao Lin, Ke Wu, Jie Li, Jun Li, and Wu-Jun Li. UniAP: Unifying inter- and intra-layer automatic parallelism by mixed integer quadratic programming. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20947–20957, 2025.

Guodong Liu, Youshan Miao, Zhiqi Lin, Xiaoxiang Shi, Saeed Maleki, Fan Yang, Yungang Bao, and Sa Wang. Aceso: Efficient parallel DNN training through iterative bottleneck alleviation. In *European Conference on Computer Systems (EuroSys)*, pp. 163–181, 2024.

Guandong Lu, Runzhe Chen, Yakai Wang, Yangjie Zhou, Rui Zhang, Zheng Hu, Yanming Miao, Zhifang Cai, Li Li, Jingwen Leng, and Minyi Guo. DistSim: A performance model of large-scale hybrid distributed DNN training. *arXiv preprint arXiv:2306.08423*, 2023.

Diksha Moolchandani, Joyjit Kundu, Frederik Ruelens, Peter Vranckx, Timon Evenblij, and Manu Perumkunnil. AMPeD: An analytical model for performance in distributed training of transformers. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023.

Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Ziji Shi, Le Jiang, Ang Wang, Jie Zhang, Chencan Wu, Yong Li, Xiaokui Xiao, Wei Lin, and Jialin Li. TAPAS: Fast and automatic derivation of tensor parallel strategies for large neural networks. In *International Conference on Parallel Processing (ICPP)*, pp. 804–815, 2025.

Qitong Sun, Jun Han, Tianlin Li, Zhe Tang, Sheng Chen, Fei Yang, Aishan Liu, Xianglong Liu, and Yang Liu. Kernelskill: A multi-agent framework for GPU kernel optimization. *arXiv preprint arXiv:2603.10085*, 2026.

Wenyi Wang, Zheng Wu, Yanmeng Wang, Haolin Mao, Lei Han, Gaogang Xie, and Fu Xiao. Hygra: Accelerating network-state simulation for LLM training in DCNs via adaptive packet-flow granularity. *arXiv preprint arXiv:2603.12671*, 2026.

Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. SimAI: Unifying architecture design and performance tuning for large-scale large language model training with scalability and precision. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 541–558, 2025.

William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. ASTRA-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. *arXiv preprint arXiv:2303.14006*, 2023.

Jiehao Wu, Zixiao Huang, Wenhao Li, Chuyun Shen, Junjie Sheng, and Xiangfeng Wang. AscendOptimizer: Episodic agent for Ascend NPU operator optimization. *arXiv preprint arXiv:2603.23566*, 2026.

Zhen Xie, Murali Emani, Xiaodong Yu, Dingwen Tao, Xin He, Pengfei Su, Keren Zhou, and Venkatram Vishwanath. Centimani: Enabling fast AI accelerator selection for DNN training with a novel performance predictor. In *USENIX Annual Technical Conference (USENIX ATC)*, pp. 1203–1221, 2024.

Siyuan Yang, Enda Yu, Pingjing Lu, and Dezun Dong. LLMEmu: A lightweight performance emulator for high-fidelity distributed LLM training. *Performance Evaluation*, 170:102526, 2025.

Srihas Yarlagadda, Amey Agrawal, Elton Pinto, Hakesh Darapaneni, Mitali Meratwal, Shivam Mittal, Pranavi Bajjuri, Srinivas Sridharan, and Alexey Tumanov. Maya: Optimizing deep learning training workloads using GPU runtime emulation. In *European Conference on Computer Systems (EuroSys)*, pp. 1738–1758, 2026.

Josh You. OpenAI's compute spend, 2025. Epoch AI.

Genghan Zhang, Shaowei Zhu, Anjiang Wei, Zhenyu Song, Allen Nie, Zhen Jia, Nandita Vijaykumar, Yida Wang, and Kunle Olukotun. Accelopt: A self-improving LLM agentic system for AI accelerator kernel optimization. *arXiv preprint arXiv:2511.15915*, 2025.

Yining Zhang, Mingyang Yi, Chen Wang, Xuwen Xiang, Tianhe Jia, Zedong Dan, Chengqing Zong, and Yue Wang. HTAM: Hierarchical transition-attended memory for operator optimization. *arXiv preprint arXiv:2605.29734*, 2026.

Yujie Zheng, Zhuo Li, Shengtao Zhang, Hanjing Wang, Junjie Sheng, Jiaqian Wang, Junchi Yan, Weinan Zhang, Ying Wen, Bo Tang, and Muning Wen. Towards cold-start drafting and continual refining: A value-driven memory approach with application to NPU kernel synthesis. In *International Conference on Machine Learning (ICML)*, 2026.

Zhanda Zhu, Christina Giannoula, Muralidhar Andoorveedu, Qidong Su, Karttikeya Mangalam, Bojian Zheng, and Gennady Pekhimenko. Mist: Efficient distributed training of large language models via memory-parallelism co-optimization. In *European Conference on Computer Systems (EuroSys)*, pp. 1298–1316, 2025.

APPENDICES

A REPRESENTATIVE CALIBRATION EXPERIENCE

Table 5: Historical correction experience for profiling-based calibration. Entries retain their applicability and evidence status; fitting accuracy alone does not establish generalization.

Experience	Target	Observed issue	Stored correction	Reuse conditions	Evidence status
Function-selection experience					
Payload and topology	DP large	Payload interpretation, bucket configuration, collective factors, or link hierarchy can cause systematic modeling errors.	Distinguish logical volume, local shard, and stage bytes. Explicitly account for bucket configuration and count collective factors once. Map intra-server, intra-affinity, and inter-affinity stages to their supported HCCS/RoCE paths.	Matching payload convention, profiling boundary, collective algorithm, and topology.	Retain only corrections supported by profiling evidence.
Small-message latency	DP/PP small	Wire-transfer time alone cannot explain the observed latency; similar available inputs may produce different durations.	Use $T_{\mathrm{wire}} + T_{\mathrm{control}}$ as a candidate correction. Model startup, coordination, or waiting only when supported by the available execution context. Keep DP and PP corrections separate without shared evidence.	Matching operation type and observable execution context.	Candidate correction. Unresolved behavior remains explicit rather than being fitted with configuration-specific offsets.
Chunk timing boundary	PP	Scalar intervals can be incorrectly interpreted as complete-chunk communication, leading to an incorrect transfer model.	Separate scalar and large-payload communication and retain the corresponding chunk-level profiling-boundary interpretation.	Matching chunk organization and profiling boundary. Preserve unresolved scalar-dtype assumptions.	Timing interpretation must be validated before reuse.
Coefficient-validation experience					
Shared link parameters	DP large	Nominal bandwidth efficiency and startup do not fully capture the observed communication latency.	Calibrate link-specific bandwidth efficiency and startup across compatible observations. Store the HCCS/RoCE path, calibrated coefficients, fixed parameters, calibration context, and validation results.	Matching communication path and operating context. Coefficients are not treated as universal constants.	Validated coefficients are reused only within their supported context.
Coefficient applicability	DP irregular	Low fitting error on sparse profiling samples does not establish a reusable correction.	Store fitted coefficients together with their supporting observations and applicable feature range. Do not promote configuration-specific offsets as reusable calibration experience.	Matching irregular behavior and supported feature range.	Insufficient profiling evidence prevents promotion to reusable experience and requires further behavioral analysis.
Bandwidth–startup separation	PP large	Available forward/backward observations may not reliably identify bandwidth efficiency and startup jointly.	Fix the configured startup and calibrate one shared forward/backward bandwidth-efficiency coefficient. Retain both the fixed startup and fitted coefficient as part of the calibration context.	Matching communication route, payload definition, and startup convention.	Reuse requires the same parameterization assumptions and profiling-based validation.

The calibration experience bank stores validated calibration results for reuse under similar configurations. It contains two categories: *function-selection experience*, which narrows candidate functions for irregular communication, and *coefficient-validation experience*, which reuses calibrated coefficients when the function form is known.

Function-selection experience. For irregular communication, the physical model may not fully explain the profiled latency. Once a correction is validated, the bank stores the correction together with its applicability conditions. For example, when small DP/PP communication exhibits latency far beyond its wire-transfer time, the bank considers $T_{\mathrm{wire}} + T_{\mathrm{control}}$ while keeping DP and PP corrections separate unless profiling supports their sharing.

The bank also records corrections to communication interpretation, including payload definition, bucket configuration, collective factors, and communication paths. For PP, profiling boundaries and chunk-level communication are likewise retained to avoid repeating the same modeling errors.

Importantly, lower fitting error alone is insufficient to promote a correction as reusable experience. Cases with insufficient profiling evidence remain unresolved; for example, DP Scalar AR tail and All GPU AR 1 require finer-grained analysis rather than configuration-specific fitting.

Coefficient-validation experience. Once the function form is determined, validated historical coefficients and their applicable contexts are reused to narrow parameter search.

Scenario	Controlled Configuration	Retained Experience	Supporting Evidence
PP-stage placement	Fixed $TP = 4$, $EP = 8$, and $MB = 4$. Compare $PP = 4$ mappings with aligned and shifted stage boundaries, together with an aligned $PP = 2$ configuration.	Align PP-stage boundaries with affinity-group boundaries when possible, while treating PP depth as a separate deployment decision.	Aligned $PP = 4$: 1.900 s; misaligned $PP = 4$: 3.205 s; aligned $PP = 2$: 1.674 s.
TP locality	Fixed $PP = 2$, $TP = 4$, $DP = 16$, $EP = 8$, and $MB = 4$. The four TP ranks are placed within one node, across two nodes in the same affinity group, or across affinity groups.	Prefer intra-node TP placement. If TP must cross nodes, keep the group within the same affinity group rather than crossing affinity-group boundaries.	Intra-node: 1.674 s; inter-node/intra-affinity: 1.686 s; inter-affinity: 2.722 s.
DP grouping	Fixed $PP = 4$, $TP = 4$, $DP = 8$, $EP = 8$, and $MB = 8$. Eight slow GPUs are arranged either across mixed fast/slow DP groups or as one all-slow DP group.	Fast/slow homogeneity within DP groups alone is insufficient to determine deployment quality; the exposure of TP, EP, and PP synchronization domains must also be considered.	Mixed grouping: 2.040 s; homogeneous grouping: 2.040 s; all-normal baseline: 1.692 s.
Slow-GPU distribution	Fixed $PP = 4$, $TP = 2$, $DP = 16$, $EP = 8$, and $MB = 8$. Eight slow GPUs are either concentrated within one node or distributed across eight nodes.	Under the evaluated deep-pipeline configuration, limiting slow-GPU exposure to fewer stages and synchronization domains is preferable to broadly distributing the same number of slow GPUs.	Concentrated: 1.853 s; distributed: 2.337 s; all-normal baseline: 1.506 s.

For large DP communication, bandwidth efficiency and startup are calibrated only across compatible observations, and link-specific coefficients remain tied to their HCCS/RoCE paths. Profiling-identified settings, such as the bucket configuration for Z3 G-RS DP-CP and Z3 G-RS DP_EP, are also retained to avoid repeated configuration search.

For PP communication, available observations may not identify bandwidth efficiency and startup simultaneously. We therefore fix the configured startup, calibrate a shared forward/backward bandwidth-efficiency coefficient, and retain both as part of the calibration context.

Experience reuse and update. When new profiling becomes available, compatible entries are reused to narrow function or coefficient search and are then validated against the new measurements. Consistent observations extend their validation evidence; conflicts instead restrict or revise their applicability and trigger recalibration when necessary. The bank therefore treats historical results as reusable guidance rather than fixed rules, allowing new profiling to both reuse and refine accumulated calibration experience. Table 5 summarizes representative entries.

B Construction Procedure of Deployment Experience Bank

B.1 128-GPU Heterogeneous Deployment Experience Bank

We summarize the deployment experience constructed from the 128-GPU heterogeneous setting below. The resulting bank contains four solution-based entries obtained from controlled comparisons and a policy-based hierarchy constructed from searched deployment scenarios.

B.1.1 Solution-based Experience

The following table reports the controlled configurations, retained findings, and supporting measurements for the four solution-based experience entries.

B.1.2 Policy-based Experience

We construct the policy-based experience bank on a 128-GPU heterogeneous topology by varying both the number and spatial distribution of fast and slow GPUs across deployment scenarios. The resulting scenarios cover different slow-GPU loads and distribution patterns across nodes and affinity groups, providing diverse heterogeneous conditions for deployment search. After excluding one inconsistent policy formula during validation, 99 valid scenarios are retained. Their searched deployment policies are organized into two experience families and further associated with six scenario motifs.

Table 6: Policy-based experience families retained from the 128-GPU scenarios.

Family	Count	Deployment Characteristics	Scenario Profile
A	15	**Deeper-pipeline family.** PP tends to be medium or deep, TP remains relatively small, and MB increases with PP. DP is commonly derived as $DP = N/(PP \times TP)$, while EP remains around E/C and nearby values.	Slow GPUs are predominantly localized. Excluding the no-slow-GPU baseline, all 14 slow-GPU scenarios cover only one or two of the four affinity groups. For dual-affinity cases, strong intra-affinity imbalance is common.
B	84	**Shallow-pipeline family.** PP is commonly 1 or the minimum feasible value, TP remains moderate and is often around 4, and MB is commonly 1 or the minimum feasible value. DP is mainly derived from $N/(PP \times TP)$.	Slow GPUs are generally broadly distributed. Most scenarios cover all four affinity groups; partially covered cases are typically either relatively balanced within affinity groups or exhibit a high overall slow-GPU load.

Table 7: Scenario motifs and retained policy-based deployment experience.

Motif	Count	Scenario Characteristics	Retained Deployment Experience
A1	1	**No-slow-GPU baseline.** $\rho = 0$.	$PP \geq 4$; $TP \leq 2$; $MB \geq 2PP$; $DP = N/(PP \times TP)$; EP remains within its feasible range around E/C.
A2	7	**Single-affinity localization.** $C_A = 0.25$. Slow GPUs are confined to one affinity group and affect only one or two nodes in the current samples.	PP is commonly no smaller than 4, or TP is constrained relative to PP, e.g., $TP \leq PP/2$. MB is commonly at least 8 or scales with PP, such as $MB \geq 2PP$. EP remains around E/C to approximately $2E/C$.
A3	7	**Dual-affinity localization with imbalance.** $C_A = 0.50$; six of seven scenarios satisfy $D_{\text{intra}} \geq 0.50$.	TP is commonly constrained by $TP \leq PP/2$. PP is typically no smaller than 4 or satisfies $PP \geq 2TP$. MB is commonly no smaller than $2PP$ or 8. Scenario-specific EP bounds are retained.
B1	71	**Full-affinity coverage.** $C_A = 1$. At least half of the nodes are affected, while the slow-GPU rate varies widely across scenarios.	PP is typically 1 or the minimum feasible value. TP is commonly around 4 or determined by a TP/DP ratio constraint. MB is typically 1 or the minimum feasible value. EP remains around E/C.
B2	8	**Partial coverage with low intra-affinity imbalance.** $C_A = 0.50$–0.75 and $D_{\text{intra}} \leq 0.25$.	A shallow-pipeline configuration is dominant, with a strong tendency toward $PP = 1$ and $MB = 1$. TP is commonly around 4 or maintains a relatively low TP/DP ratio, while DP follows $N/(PP \times TP)$.
B3	5	**Partial coverage with high load or concentration.** $C_A = 0.50$–0.75, $D_{\text{intra}} = 0.375$–1.00, $\rho = 34.38\%$–61.72%, and $R_{\max} = 1$.	$PP = 1$ and $MB = 1$ are the dominant directions. TP is commonly around 4 or bounded by a scenario-specific range or ratio, while DP follows $N/(PP \times TP)$.

C Application of Q/A and what/if analysis

C.1 Deployment Question Answering

As an application showcase, we organize the constructed deployment experience into a DeepWiki-based knowledge base and use it to answer deployment questions under heterogeneous training environments. The following cases present representative responses generated from the knowledge base. The responses are condensed for presentation without changing the recommended candidate configurations. No new simulations are performed for these queries; therefore, the suggested configurations represent experience-based recommendations rather than latency predictions.

Case 1: Concentrated Slow GPUs. We consider a 128-GPU cluster with eight 16-GPU nodes and four two-node affinity groups. Among the 128 GPUs, 24 slow GPUs are concentrated on two nodes, with 12 slow GPUs on each node. Their compute speed is 50% of that of normal GPUs, while communication capability and memory capacity remain unchanged.

Question. Given this concentrated slow-GPU distribution, which parallelism configurations does the deployment experience base suggest for evaluation?

Answer. The experience base associates this scenario with the localized and imbalanced deployment pattern (A3), for which deeper PP, smaller TP, and larger MB commonly occur in the retained deployment experience. Accordingly, three representative candidates are suggested.

C1: PP4 / TP2 / DP16 / EP8 / MB8. This configuration follows the retrieved A3 experience most closely.

C2: PP4 / TP2 / DP16 / EP8 / MB4. This configuration retains the same PP/TP/DP/EP structure while varying MB.

C3: PP1 / TP4 / DP32 / EP8 / MB1. This configuration provides a contrasting shallow-pipeline structure.

Case 2: Dispersed Slow GPUs. We retain the same 128-GPU cluster and the same total number of 24 slow GPUs, but distribute them uniformly across all eight nodes, with three slow GPUs per node. The compute-speed reduction and other hardware assumptions remain unchanged.

Question. How does the suggested deployment change when the same number of slow GPUs is distributed across the entire cluster?

Answer. The experience base associates this distribution with the broad-coverage pattern (B1), in which shallow PP and moderate TP recur across related deployment scenarios. It therefore suggests three representative candidates.

C1: PP1 / TP4 / DP32 / EP8 / MB1. This configuration follows the retrieved B1 experience most closely.

C2: PP1 / TP8 / DP16 / EP8 / MB1. This configuration retains the shallow-pipeline structure while varying TP.

C3: PP4 / TP2 / DP16 / EP8 / MB8. This configuration provides a contrasting deeper-pipeline structure.

The two cases illustrate how the knowledge base adapts its recommendations to changes in hardware distribution. Although the number and compute capability of slow GPUs remain unchanged, changing their spatial distribution retrieves different deployment experience and therefore different candidate configurations. All candidates satisfy the basic parallelism constraints of the corresponding configuration, while their actual latency and memory feasibility must still be evaluated by the estimator before deployment.

C.2 Evidence-Based What-if Analysis

We examine how specified changes to GPU placement or microbatch count affect simulated training step latency. Tensor, pipeline, data, and expert parallelism are denoted by TP, PP, DP, and EP, respectively; MB denotes the number of microbatches per replica. Each comparison uses an explicit baseline rather than a newly optimized deployment.

The placement cases use 128 GPUs arranged in eight 16-GPU nodes and four two-node affinity groups, with no slow GPUs [E1]. The microbatch case uses the separate 256-GPU s006 scenario, containing 12 slow GPUs [E2]. Both use a global batch size of 128 and sequence length of 4096. Results are reported within each experiment; absolute latencies are not compared across the two scales.

These are analyses of existing simulator outputs, not new physical-cluster measurements. The configurations specify 1F1B scheduling, uniform expert routing, synchronized EP cohorts, and no communication–computation overlap. The 256-GPU model has not been independently calibrated. Memory figures are planner estimates.

```
=== Part 1:  What Happens When TP Spans Nodes?  ===
Question:  At fixed PP=2, TP=4, DP=16, EP=8, and MB=4, how does
changing TP placement affect simulated step latency?
Answer:  The baseline places each four-GPU TP group within one node
and yields 1.674481 seconds.  Placing each TP group across two nodes
within one affinity group yields 1.685863 seconds, an increase of
0.68%.  Placing TP groups across affinity groups yields 2.721524
seconds, an increase of 62.53% relative to the same baseline [E1].
```

Percentage changes are computed from the unrounded records.

These are net effects of complete rank remappings. In the within-affinity comparison, the recorded TP critical-path contribution remains 0.322090 seconds, while the DP component labeled `dp_cp` increases by 0.011381 seconds (from 0.081093 to 0.092475 seconds). This accounts for the latency difference. Thus, the small increase cannot be attributed to additional TP communication in this model. In the cross-affinity case, the TP contribution increases to 1.447463 seconds, while DP components also change.

The results distinguish within-affinity from cross-affinity placement under the tested configuration. They do not establish a universal penalty for cross-node TP or an isolated causal effect of a single communication operator.

==Part 2: What Happens When PP Stages Straddle Affinity Groups?==
Question: At fixed PP=4, TP=4, DP=8, EP=8, and MB=4, what happens when a rank remapping causes each pipeline stage to span two affinity groups instead of occupying one complete affinity group?
Answer: In the aligned baseline, simulated step latency is 1.900034 seconds. A cyclic shift of the physical mapping by 16 GPU positions makes each stage straddle two affinity groups and increases latency to 3.205436 seconds, a 68.70% increase [E1].
- The EP dispatch and combine contributions on the recorded critical path each increase from 0.228771 to 0.876712 seconds;
- The dp_cp contribution increases from 0.034050 to 0.043569 seconds;
- The PP communication contribution remains 0.011402 seconds, and the recorded compute and TP contributions are unchanged.

Consequently, this case demonstrates sensitivity to stage placement primarily through changes in EP communication, with a smaller DP contribution. It does not measure the effect of moving only one inter-stage transfer from an intra-affinity path to a cross-affinity path. Stage membership and inter-stage communication locality are distinct interventions. The latter question remains unquantified by this comparison.

=== Part 3: What Happens When MB Increases? ===
Question: With PP=4, TP=4, DP=16, EP=8, and global batch size $B = 128$ fixed, what happens when MB increases from 4 to 8?
Answer: Reported simulated latency decreases from 1.414608 to 1.351047 seconds, a reduction of 4.49%. Estimated memory usage decreases from 17.329 to 12.173 GiB, a reduction of 29.76% [E2]. At fixed B and DP, the implied microbatch size $b = B/(DP \times MB)$ decreases from two samples to one sample per replica.

This is therefore a change in batch partitioning: more microbatches are processed, but each contains fewer samples. The timing difference is compatible with improved pipeline utilization, but the candidate summary alone does not establish the contribution of pipeline bubbles or individual computation and communication events. Likewise, the estimated memory reduction should not be presented as a measured peak-memory reduction.

The two candidates share the listed parallelism degrees, scenario, and configured placement convention; no event-level attribution is made here. The simulator requires $(DP \times MB) \mid B$. Both tested values satisfy this constraint, whereas MB=16 is infeasible at DP=16 and $B = 128$. The two points do not establish monotonic improvement with increasing MB.

Scope of the findings:

a) The TP placement comparison yields a 0.68% within-affinity increase and a 62.53% cross-affinity increase in simulated step latency. Attribution requires examining all affected communication groups.

b) Misaligning PP stage membership with affinity groups yields a 68.70% increase, primarily associated with EP critical-path contributions rather than PP transfer time.

c) Increasing MB from 4 to 8 in s006 yields 4.49% lower simulated latency and 29.76% lower estimated memory usage, accompanied by a reduction in microbatch size.

These examples demonstrate evidence-backed impact analysis within the tested configurations; they do not establish real-cluster speedups, general sensitivity curves, or the accuracy of a question-answering system across unseen scenarios.

Evidence records. **[E1]** Archived 128-GPU placement experiment, 25 September 2026: `evidence.json`, cases `same_node`, `same_affinity_two_nodes`, `cross_affinity`, `aligned_pp4`, and `shifted_pp4`. The associated validation record reports that all 12 cases pass the listed structural and feasibility checks.

[E2] Archived 256-GPU s006 experiment, 22 September 2026: `simulation_top10.json`, MB=4 and MB=8 candidates at PP4 / TP4 / DP16 / EP8, with `resolved_config.json` specifying the scenario and simulator assumptions.