LM-SIMBENCH: A UNIFIED FRAMEWORK, DATASET, AND BENCHMARK FOR LARGE LANGUAGE MODEL TRAINING SIMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Selecting parallelization strategies for large-scale language-model training is prohibitively expensive because evaluating each candidate requires a distributed run. Performance simulators can reduce this cost, but existing tools model different stages of the training stack and are evaluated on incompatible workloads, precluding controlled comparison. We introduce LM-SimBench, a unified framework, profiling dataset[1], and benchmark[2] for evaluating LLM training simulators. The framework decomposes simulation into an end-to-end pipeline from model compilation to network-level estimation and characterizes methods by workflow granularity, cost estimators, and modeling focus. Following this framework, we profile 23 model settings from 10 dense and Mixture-of-Experts (MoE) LLM families on an Ascend Neural Processing Unit (NPU) cluster. The resulting dataset contains 348 feasible configurations that jointly vary five parallelism dimensions and activation recomputation, with six views spanning configuration, rank, operator, event, and kernel granularity. A matched V100/Megatron-LM supplement covers two workloads. The benchmark evaluates system-level time and memory prediction, component-level operator and communication modeling, and parallel-strategy ranking, thereby measuring both numerical accuracy and configuration-selection utility. Baseline experiments illustrate why this multi-level evaluation is necessary: Calculon identifies the fastest compatible configuration, yet rejects 7 of 20 measured-feasible configurations and underestimates a multi-node iteration time by 34%. Across platforms, matrix-problem difficulty transfers well, whereas absolute step time requires target-platform calibration.

1 INTRODUCTION

The progress of large language and foundation models is closely tied to scale: model quality improves predictably with model size, data, and compute, while recent frontier models extend this trend through larger parameter counts, longer contexts, and larger pre-training corpora Kaplan et al. (2020); Hoffmann et al. (2022); Grattafiori et al. (2024). Training at this scale requires distributed execution and coordinated decisions about model placement, communication, and parallelism. System designers must therefore answer two coupled questions: whether a model can be trained within the available memory, and which model–parallelism–hardware configuration trains it most efficiently.

Answering these questions by trial and error is impractical. Publicly reported runs span PaLM on 6,144 TPU v4 chips Chowdhery et al. (2023), Megatron-LM on 3,072 GPUs Narayanan et al. (2021), and Llama 3 405B on up to 16K H100 GPUs Grattafiori et al. (2024). At such scales, adding devices does not guarantee efficiency: performance depends on coupled choices in data, tensor, pipeline, context, and expert parallelism; activation recomputation; memory placement; scheduling; and network topology. An ostensibly feasible configuration may still be slowed by pipeline imbalance, synchronization, or communication, whereas exhaustive search consumes substantial accelerator time and engineering effort.

[1] https://huggingface.co/datasets/anonymous-ljasd/LM-SimBench
[2] https://anonymous.4open.science/r/test-xmllll1653

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

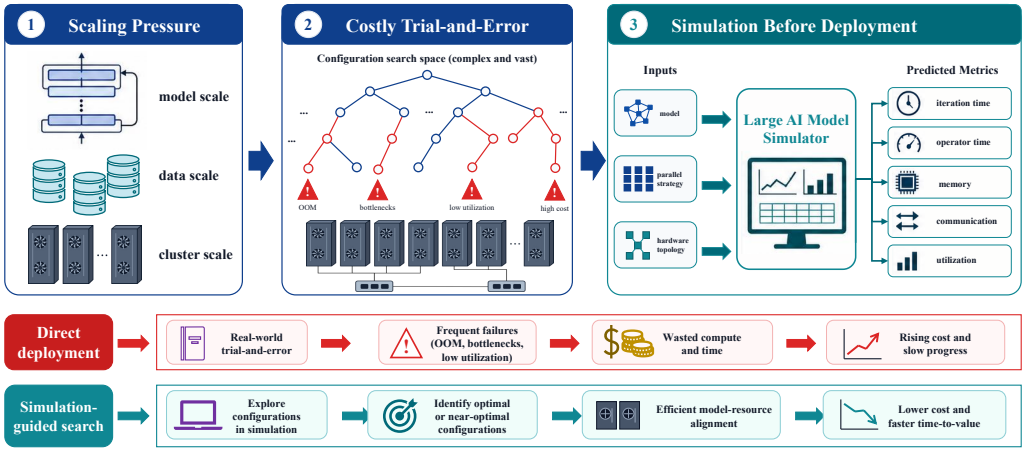


Figure 1: Simulation for model–resource alignment. Effective alignment improves utilization and reduces cost; poor alignment creates bottlenecks and resource waste. Simulation estimates these effects before deployment, reducing the need for repeated full-scale trials.

Performance simulation offers a lower-cost alternative. Given a model, hardware platform, and training strategy, a simulator can estimate iteration time, peak memory, communication overhead, and resource utilization before an expensive run. These estimates support configuration search, capacity planning, hardware–software co-design, and cost-aware deployment; Figure 1 illustrates this role. Existing systems have advanced distributed-training and hardware-exploration simulation Bang et al. (2024); Rashidi et al. (2020); Isaev et al. (2023); Wang et al. (2025b). Their evaluations, however, differ in execution scope, abstraction level, hardware assumptions, profiling requirements, and reported metrics. Consequently, it remains difficult to distinguish broad predictive accuracy from calibration to a narrow workload or platform.

The underlying obstacle is the lack of public, structured profiling data and standardized evaluation protocols. Many simulators are validated with private traces, limited workloads, or incompatible metrics, hindering reproducibility and fair comparison. A useful benchmark must specify both the target claim— for example, iteration-time prediction, memory-feasibility screening, communication modeling, operator-latency estimation, or configuration ranking—and the conditions under which it is evaluated, including model families, parallel strategies, hardware stack, and profiling granularity. Existing public artifacts address only subsets of these requirements. To our knowledge, none provides per-rank, multi-granularity profiles over a controlled parallelism sweep that includes both dense and Mixture-of-Experts (MoE) LLMs on Neural Processing Unit (NPU) hardware (Section 2).

We address this gap with LM-SimBench, a unified framework, dataset, and benchmark for large-model training simulation. The framework describes the end-to-end simulation pipeline, while the dataset instantiates this view for training-time simulation on the MindSpeed-LLM Ascend Community (2024) and Ascend NPU Liao et al. (2021) stack. The released artifacts connect the framework, data schema, and benchmark tasks, making simulator evaluation more reproducible, interpretable, and comparable while explicitly delimiting the claims supported by the current hardware and workload coverage.

Our contributions are:

- **A unified framework.** We formalize training simulation as $\mathcal{S}_\theta(M, P, N) \to (\widehat{T}, \widehat{\mathbf{V}}, \widehat{C}, \widehat{z})$ and decompose it into five stages: model compilation, device binding, computation-process modeling, operator-level estimation, and network-level estimation. This decomposition organizes representative simulators by workflow, estimator family, and modeling focus.
- **A multi-granularity profiling dataset.** We release, to our knowledge, the first public training-profiling dataset on the Ascend NPU stack. It contains 23 model settings and 348 feasible configurations from 10 dense and MoE LLM families, jointly varying data, tensor, pipeline, context, and expert parallelism (DP, TP, PP, CP, and EP) and activation

recomputation. Six profiling views expose iteration time, peak memory, time breakdowns, operator latency, communication events, and matrix kernels. The release also includes a matched V100/Megatron-LM instance to test the protocol beyond one stack (Section 6.1).

- **A three-level benchmark.** We define system-level tasks (iteration time and memory feasibility), component-level tasks (operator-category, matrix-operator, and communication latency), and a strategy-level task (parallel-configuration ranking). The protocol separates end-to-end accuracy from agreement with diagnostic signals and evaluates both prediction error and configuration-selection utility.
- **Baseline and cross-platform evaluations.** Calculon correctly identifies the fastest compatible configuration and preserves the ordering of dominant matrix hotspots, but its memory model rejects 7 of 20 measured-feasible configurations and its compatibility-mode estimate understates multi-node iteration time by 34%. Across six common test configurations, comparison with vTrain exposes an accuracy–cost trade-off between analytical and profiling-based simulation. Matched NPU/GPU measurements further identify which benchmark signals transfer across stacks and which require target-platform calibration.

2 Related Work

Large-model training simulators. Distributed-training simulators span analytical methods (Calculon Isaev et al. (2023); Shoeybi et al. (2019) and SimAI Wang et al. (2025b)), graph- or task-based methods (ASTRA-sim Rashidi et al. (2020); Won et al. (2023) and Optimus Kundu et al. (2024)), and profiling-based methods (vTrain Bang et al. (2024), LLMEmu Yang et al. (2025), Habitat Yu et al. (2021), and ScaleDL Wang et al. (2025a)); practical systems may combine these approaches Guo et al. (2024); Svedas et al. (2025). Evaluations are often tied to private traces, restricted workloads, or platform-specific metrics, which limits controlled comparison across tools. LM-SimBench complements these systems with a common profiling artifact and task protocol. Appendix A discusses earlier operator predictors and serving-oriented simulators, which fall outside our training-focused scope.

Benchmarks and profiling artifacts. System benchmarks such as MLPerf Mattson et al. (2020); Reddi et al. (2020) standardize end-to-end runtime reporting but do not provide the multi-granularity traces needed to validate a simulator's intermediate predictions. Existing public artifacts each cover part of this need. NeuSight Lee et al. (2025) provides operator and end-to-end latency measurements for primarily dense models over a restricted parallelism space, but not per-rank memory, communication, or recomputation profiles. Cluster-scale traces such as AcmeTrace Hu et al. (2024) and the Alibaba GPU traces Alibaba (2023) expose scheduler-level job records rather than per-rank signals from controlled strategy sweeps. Chakra Sridharan et al. (2023), Lumos Liang et al. (2025), and AICB Alibaba (2024) standardize trace representation, replay, or generation, but do not ship a curated measurement corpus with the same coverage. Appendix A.5 compares these artifacts by workload, parallelism dimensions, released signals, and hardware platform.

3 A Unified Framework for Large-Model Training Simulation

Large-model training simulators differ in the execution phases they represent, their workflow abstractions and profiling requirements, and the metrics they predict. A reported accuracy can therefore describe iteration-time prediction, operator-level agreement, communication-event replay, or calibration on a small profile set—quantities that are not directly comparable. We instead characterize a simulator by its three inputs (M, P, N), its outputs, and the pipeline stages that produce those outputs. Appendix B expands this formulation and maps representative simulators to the framework in Table 5.

3.1 Pipeline and Formal Problem

Let a simulation instance be

$$x = (M, P, N), \qquad \widehat{y} = \mathcal{S}_\theta(x) = (\widehat{T}, \widehat{\mathbf{V}}, \widehat{C}, \widehat{z}), \tag{1}$$

where M specifies the model and workload, P the training configuration, and N the system. The outputs comprise predicted iteration time $\widehat{T}$, per-device peak memory $\widehat{\mathbf{V}} = (\widehat{V}_d)_{d \in D(N)}$, a task-

specific cost $\widehat{C}$, and optional diagnostic signals $\widehat{z}$, such as operator latencies, communication events, or rank-level time breakdowns. When a scalar memory metric is required, we use $\widehat{V} = \max_{d \in D(N)} \widehat{V}_d$. A configuration is feasible only if its strategy is valid for the target system and its predicted memory fits on every device:

$$I_{\text{feas}}(M, P, N) = \mathbf{1}\{P \in \mathcal{K}(M, N)\} \prod_{d \in D(N)} \mathbf{1}\{\widehat{V}_d \leq \text{cap}_d\}, \tag{2}$$

where $\mathcal{K}(M, N)$ is the valid strategy space, $D(N)$ the device set, and cap_d the memory capacity of device d. During search or ranking, infeasible configurations can be assigned $\widehat{T} = +\infty$.

Figure 2 decomposes $\mathcal{S}_\theta$ into five stages. Stage A compiles the model under the selected strategy; Stage B binds the compiled model to devices and parallel groups; and Stage C constructs an execution representation:

$$\bar{M} = \mathcal{C}(M, P), \qquad \Pi = \mathcal{B}(\bar{M}, P, N), \qquad R = \mathcal{R}(\bar{M}, \Pi, P, N), \tag{3}$$

Here, $\bar{M}$ is the compiled model, Π records device placement and communication groups, and R may be an analytical phase decomposition, an operator-level directed acyclic graph (DAG), a profiled trace, or a hybrid schedule.

Stage D estimates the costs of the computation and communication units in R. Let $\mathcal{U}(R) = \mathcal{U}_{\text{op}}(R) \cup \mathcal{U}_{\text{comm}}(R)$ denote the units represented by the simulator. Their predicted latencies are

$$\widehat{t}_u = \begin{cases} \mathcal{F}_{\text{op}}(\phi_u, \Pi, N), & u \in \mathcal{U}_{\text{op}}(R), \\ \mathcal{F}_{\text{comm}}(\phi_u, \Pi, N), & u \in \mathcal{U}_{\text{comm}}(R), \end{cases} \tag{4}$$

where ϕ_u contains features such as tensor shape, floating-point operations (FLOPs), bytes transferred, primitive type, group size, and placement. The estimator may use analytical formulas, structure-aware models, profiling data, or learned predictors.

Stage E aggregates these unit costs over the distributed schedule, subject to the ordering, placement, and resource constraints encoded by R and Π:

$$\widehat{T} = \mathcal{A}\big(R, \Pi, \{\widehat{t}_u\}_{u \in \mathcal{U}(R)}, N\big), \qquad \widehat{\mathbf{V}} = \mathcal{V}(R, \Pi, N), \tag{5}$$

The simulator then derives $\widehat{C}$ and $\widehat{z}$ from the same representation as required by the task. The memory estimator $\mathcal{V}$ uses the layout encoded by R and Π together with the hardware characteristics in N; it does not require the unit-latency estimates $\{\widehat{t}_u\}$. In Figure 2, profiler data may calibrate Stage D or supply traces to Stage C, while the optimizer consumes the predicted outputs to search over P or N.

Implementations of $\mathcal{A}$ range from closed-form phase rules, through constrained-makespan scheduling on operator graphs $G = (\mathcal{O}, \mathcal{E})$, to trace replay under a fixed pipeline schedule. This choice determines whether the simulator can represent overlap, scheduling, and stream-level effects (Appendix B).

3.2 Modeling Choices and Evaluation Claims

The tuple (M, P, N) encodes workload, training, and system attributes. M records Transformer dimensions $(L, h, n_{\text{head}}, d_{\text{ff}}, s, b)$, extended with routing or modality parameters when needed. P records DP/TP/PP/CP/EP, micro-batch size, recomputation, precision, fusion, and overlap; N records compute and memory capabilities, bandwidth, latency, topology, and efficiency. Process models range from analytical phase decompositions to operator-level DAGs and trace-driven hybrids, while computation and communication estimators may be analytical, structure-aware, profiling-based, or learned. These choices trade simulation cost for fidelity and determine which predictions can be validated against profiling data. Hardware and software assumptions likewise bound the scope of benchmark results: transfer to another accelerator family or runtime requires additional validation. Appendix B lists the configuration dimensions and estimator families in detail.

LM-SimBench follows the same input–output structure:

$$\mathcal{D} = \{(x_i, y_i^\star, z_i^\star)\}_{i=1}^n, \tag{6}$$

where $x_i = (M_i, P_i, N_i)$, $y_i^\star$ contains configuration-level labels (iteration time and per-device peak memory), and $z_i^\star$ contains intermediate profiling views (rank-level time breakdowns, operator

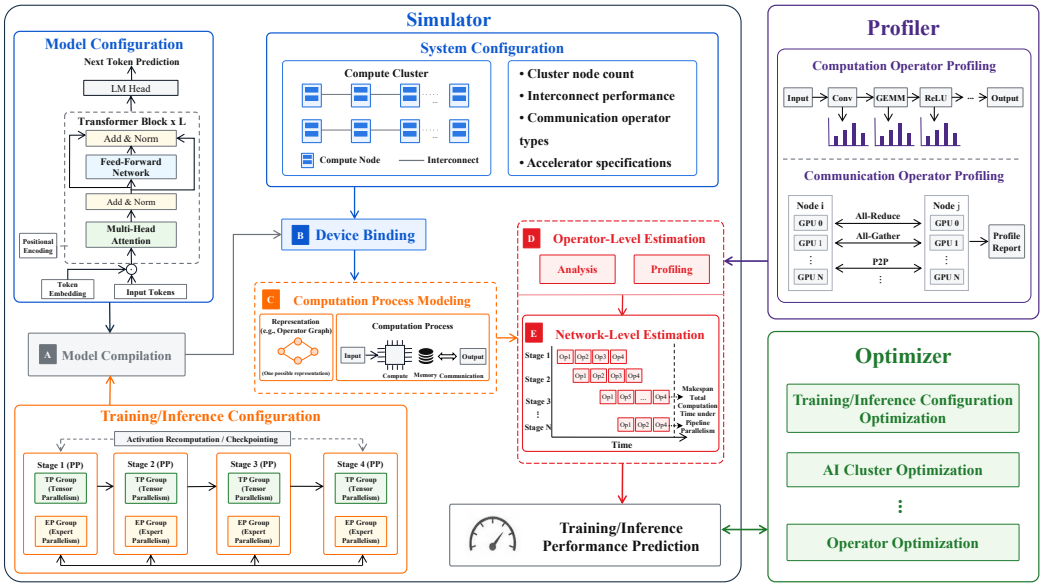


Figure 2: Unified pipeline for large-model training simulation. Model, training, and system configurations pass through model compilation, device binding, computation-process modeling, operator-level estimation, and network-level estimation. Profiled data can calibrate the pipeline, and an optimizer can use the predictions for configuration search.

latencies, communication events, and matrix-kernel counters). This schema separately evaluates end-to-end accuracy, ranking quality, memory feasibility, and agreement with intermediate signals on the MindSpeed-LLM/Ascend stack. Claims about unmeasured model families, serving workloads, or hardware topologies require further validation.

4 LM-SimBench Dataset

LM-SimBench is a structured profiling dataset collected from distributed LLM training runs. It combines end-to-end labels with intermediate signals across model families, scales, context lengths, parallel strategies, and recomputation settings. Within the profiled hardware and software stack, these records support evaluation of time and memory prediction, communication and operator modeling, simulator calibration, and parallel-strategy selection.

4.1 Evaluation Scope and Dataset Design

The primary release is built on MindSpeed-LLM Ascend Community (2024) and Ascend NPUs Liao et al. (2021). It converts raw training logs and profiler outputs into standardized, searchable tables, preserving signals from configuration to kernel granularity that cannot be recovered from analytical formulas or isolated microbenchmarks alone. Processed comma-separated-value (CSV) records support direct modeling, while raw profiler dumps enable trace-level inspection and reprocessing. The release covers dense and MoE LLMs at multiple scales and context lengths. Its scope is training-time simulation on the profiled stack; inference serving, other accelerator families, and unmeasured model classes require separate validation. Appendix C documents the platform, workload coverage, and file schemas.

Cross-platform supplement. To test whether the schema and tasks remain applicable beyond the primary stack, we add matched measurements from 8×NVIDIA V100 (32 GB) with Megatron-LM and 8×Ascend 910B3 (64 GB) with MindSpeed-LLM. Both stacks cover `qwen3_0.6b_4k` and `qwen3_1.7b_4k` under the same 18 CP=1 DP/TP/PP/recomputation candidates. The manifest marks incomplete GPU profiles explicitly rather than treating them as out-of-memory runs. Appendix E reports the platform metadata and matched comparisons.

4.2 Provenance, Processing, and Quality Control

The raw inputs comprise MindSpeed-LLM logs and per-rank Ascend Profiler outputs, each tagged with a model setting and a feasible parallel configuration. The processing pipeline extracts target-step iteration time, parses DP/TP/PP/CP/EP and recomputation settings, and converts rank-level memory, time breakdown, communication, operator, and matrix-kernel records to standardized CSV files (Appendix C.2). During conversion, we validate configuration names, ranks, units, schemas, and manifest coverage. A root manifest then links model-, configuration-, and rank-level records. Together, these checks support both black-box iteration-time prediction and diagnostic evaluation against measured intermediate signals.

4.3 Profiling Views and Supported Tasks

LM-SimBench mirrors the hierarchy of distributed execution. Each top-level entry identifies a model setting, which contains multiple feasible parallel configurations; each configuration, in turn, contains rank-level records. This organization supports two complementary forms of evaluation: predicting end-to-end performance and explaining variation in computation, memory, communication, and kernel execution.

Table 1 links the data views to their benchmark uses. Appendix C.4 and the dataset repository provide detailed schemas and representative fields; Appendix D visualizes representative profiles.

Table 1: LM-SimBench data views and their benchmark uses. The first six rows are measured profiling views; the final two are a derived ranking view and the dataset index.

Data view	**Granularity**	**Evaluation use**
Iteration time	Configuration	Iteration-time prediction and configuration ranking
Peak memory	Rank	Memory modeling and feasibility analysis
Time breakdown	Rank	Runtime decomposition and bottleneck diagnosis
Operator latency	Operator / rank	Operator modeling and module-level analysis
Communication records	Event / rank	Communication modeling and parallel-strategy analysis
Matrix kernels	Kernel / rank	General matrix multiplication (GEMM) modeling and hardware-aware analysis
Strategy ranking	Model / configuration set	Parallel-strategy ranking and configuration selection
Dataset manifest	Dataset	Indexing, integrity checking, and coverage analysis

Together, these views support iteration-time prediction, configuration ranking, memory-feasibility modeling, communication and operator analysis, bottleneck diagnosis, and simulator calibration. They also enable evaluation of generalization, error attribution, and diagnostic consistency in addition to aggregate accuracy.

4.4 Scale, Access, and Responsible Use

The processed release contains 23 model settings and 348 feasible parallel configurations from 10 dense and MoE LLM families. The configurations jointly vary DP/TP/PP/CP/EP and recomputation; coverage differs by model because of memory, sequence-length, and architectural constraints (Appendix C.3). The processed CSVs (25 GB) are available at LM-SimBench, and the raw profiler dumps (550 GB) at LM-SimBench_raw. Because the processed release exceeds 4 GB, we also provide a schema-preserving sample and Croissant metadata for both artifacts. The data is licensed under CC BY 4.0 and the accompanying code under Apache-2.0; Appendix C.6 gives the official repository pointers and maintenance plan. The records contain system telemetry rather than user or web-scraped content and no intentional personal information. Appendix C.5 documents intended and discouraged uses, coverage and platform biases, licensing, and out-of-scope claims such as cross-accelerator transfer, production serving, and language-model safety/fairness.

5 BENCHMARK DESIGN

LM-SimBench organizes the quantities required for training simulation into three task levels. *System-level* tasks evaluate end-to-end iteration time and memory feasibility. *Component-level* tasks test whether a method captures the internal cost structure of an iteration, including operator and communication latency. The *strategy-level* task evaluates whether these predictions identify useful parallel configurations. This hierarchy measures both numerical accuracy and downstream selection utility.

5.1 SYSTEM-LEVEL PREDICTION TASKS

Iteration time prediction. Given a model and parallel configuration, this task predicts the time for one training iteration. Labels come from the configuration-level iteration-time view. Inputs include the model family, scale, context length, dense/MoE type, DP/TP/PP/CP/EP degrees, and recomputation setting (the `RC` or `noRC` suffix in configuration names). We report mean absolute error (MAE) and symmetric mean absolute percentage error (sMAPE), which measure absolute and scale-normalized error, respectively.

Memory feasibility analysis. This task predicts whether a model–configuration pair fits in device memory. The configuration-level target is the maximum peak memory over all ranks in the rank-level memory view. We report absolute peak-memory error and derive a binary feasibility decision by comparing measured or predicted memory with device capacity.

5.2 COMPONENT-LEVEL PREDICTION TASKS

Component-level tasks assess whether a simulator explains where time is spent within an iteration. LM-SimBench considers operator-category latency, matrix-operator latency, and communication time.

Operator-category latency prediction. For a given model, configuration, and rank, this task predicts latency aggregated into categories such as *matmul*, *attention*, *normalization*, *activation*, *elementwise*, and *communication*. Targets come from the per-rank operator-latency view.

Matrix-operator latency prediction. This task predicts the latency of general and batched matrix-multiplication kernels from their operator type, tensor shape, data type, and dimensions (m, n, k). The targets come from the matrix-kernel view. Because these kernels account for a substantial fraction of Transformer computation, their accuracy directly affects iteration-time estimates.

Communication-time prediction. Using per-rank communication events and configuration-level summaries, this task predicts the time spent in collective communication, pipeline synchronization, and cross-node transfers. Predictions can be evaluated by communication primitive or message-size bucket.

For all component-level tasks, we report MAE, sMAPE, and R^2 to quantify absolute error, scale-normalized error, and explained latency variance.

5.3 PARALLEL STRATEGY RANKING

Given a model and hardware setting, this task ranks candidate configurations using predicted iteration time, memory feasibility, and communication overhead; predicted-infeasible configurations are excluded. We report Spearman correlation, Top-k hit rate, and Top-k regret. These metrics capture, respectively, global rank agreement, whether the measured optimum appears in the predicted shortlist, and the iteration-time penalty of the recommended configuration relative to the measured optimum.

5.4 CROSS-PLATFORM PROTOCOL EVALUATION

The cross-platform supplement in Section 4 tests whether the protocol remains meaningful after changing both accelerator and runtime. Each statistic uses only same-named configurations completed on both stacks and is reported with its intersection size. We compare configuration orderings using

Spearman ρ, Kendall τ_b, Top-1 agreement, and Top-k set overlap, both pooled and stratified by recomputation. Matrix operators are aligned by the hardware-neutral tuple $(\text{batch}, m, n, k, \text{dtype})$. Operator-category and communication-primitive behavior is compared through time-share vectors using Spearman correlation and Jensen–Shannon divergence (JSD; lower is more similar). We omit cross-platform MAE on absolute latency because the two stacks need not share a common time scale.

6 RESULTS

Table 2: Calculon results on `qwen15_1.8b_8k`. Lower values are better for MAE and sMAPE; higher values are better for R^2, Spearman ρ, and Top-1 hit.

Task	Level	MAE	sMAPE	R^2
Iteration time prediction	System	1.97 s	15.96%	0.358
Peak memory prediction	System	6.92 GB	39.34%	-3.087
Memory feasibility	System	Agreement 65.0%		
Matrix-operator latency	Component	0.109 ms	43.34%	0.817
Iteration time ranking	Strategy	Spearman ρ 0.753; Top-1 hit 1		

We evaluate Calculon on the LM-SimBench tasks that it supports natively: iteration time, peak memory, memory feasibility, matrix-operator latency, and configuration ranking. We omit operator-category and communication-event prediction because the compatibility setup does not emit these outputs. Calculon supports neither context parallelism (CP) nor grouped-query attention, so we use the CP=1 `qwen15_1.8b_8k` workload as the primary test case. Of its 20 measured-feasible configurations, Calculon produces 13 valid predictions and classifies the remaining 7 as exceeding the 64 GB device capacity. Table 2 summarizes the results.

At the system level, Calculon attains an iteration-time MAE of 1.97 s and an sMAPE of 15.96%, but its R^2 of 0.358 indicates that substantial variation across configurations remains unexplained (Appendix Fig. 6). Peak-memory prediction is weaker, with 6.92 GB MAE, 39.34% sMAPE, and negative R^2 (Appendix Fig. 7). The largest errors occur for noRC configurations, several of which are overestimated by 13–29 GB. This recomputation-sensitive bias causes seven measured-feasible configurations to be classified as out of budget.

At the component level, the matrix-operator module captures latency variation across shapes and configurations ($R^2 = 0.817$), although its 43.34% sMAPE shows that relative error remains high. The discrepancy is concentrated among operators with very small absolute latencies, for which percentage errors are amplified; the module therefore represents relative trends more reliably than absolute per-operator latency. Among the 13 configurations retained by the memory model, the predicted ranking has Spearman $\rho = 0.753$ and identifies the measured fastest configuration. Selection over the full candidate set, however, remains constrained by memory-feasibility errors.

On the larger multi-node `qwen2.5-32B` workload, Calculon predicts 73.76 s per iteration versus the measured 112.22 s, a 34.3% underestimate. We therefore treat this result as a compatibility-mode stress test, not a faithful reproduction of the measured stack. The simulator preserves the ordering of the dominant matrix shapes and recovers the principal `MatMulV3` hotspots, but underestimates their cumulative cost by 27–35% (Appendix Fig. 12 and Appendix D.2).

6.1 CROSS-PLATFORM PROTOCOL VALIDATION

We next evaluate the protocol on the matched V100/Megatron-LM supplement for `qwen3_0.6b_4k` and `qwen3_1.7b_4k` (Section 4). The two workloads provide 15 and 11 configurations completed on both stacks, respectively (26 pairs in total). Transfer is signal-dependent (Appendix Table 16). The three lowest-memory configurations are identical on both platforms, and recomputation increases time while reducing memory in every matched pair. Matrix-problem difficulty also transfers strongly ($\rho = 0.983$ and 0.987), despite median absolute-latency ratios of 2.79 and 2.84 between platforms.

By contrast, pooled iteration-time ordering transfers only moderately ($\rho = 0.718$ and 0.664), and operator-category composition has weaker agreement ($\rho = 0.503$ and 0.531). Non-overlapped

communication also accounts for larger shares on 910B3 (28.5% and 30.0%) than on V100 (17.0% and 10.9%). A single-factor NPU-to-GPU diagnostic yields 22.4% and 22.9% sMAPE with negative R^2, demonstrating that absolute iteration time requires target-platform calibration even when the benchmark protocol transfers. A calibrated Calculon instance reaches 7.97%–11.22% MAPE and ranking $\rho = 0.853$–0.950; Appendix E.7 compares it with vTrain.

Overall assessment. Calculon provides a useful analytical baseline: it captures relative matrix-operator trends and identifies the fastest configuration among those its memory model accepts. Its peak-memory errors restrict the searchable space, however, and its multi-node compatibility-mode result understates absolute runtime. These failures show why simulator evaluation should jointly measure end-to-end accuracy, component-level agreement, feasibility, and ranking—and why target-stack profiling data remain necessary for calibration.

7 Conclusion and Limitations

We introduced LM-SimBench, a unified framework, profiling dataset, and benchmark for large-model training simulation. The framework relates a simulator's inputs and outputs to five pipeline stages. The dataset provides configuration-, rank-, operator-, event-, and kernel-level measurements for 23 model settings and 348 feasible parallel configurations on the MindSpeed-LLM/Ascend stack. The benchmark then links these views to system-, component-, and strategy-level tasks, allowing an aggregate prediction error to be traced to feasibility, computation, communication, or ranking behavior. Our baseline and cross-platform results demonstrate the value of this decomposition: relative trends can remain useful even when absolute latency or memory estimates fail to transfer.

The current release has four main limitations; Appendix C.5 provides the full responsible-use discussion. (1) **Training only.** It excludes inference serving and therefore does not support claims about request-level latency, KV-cache behavior, dynamic batching, or serving schedules. (2) **Platform coverage.** Absolute latency, communication overlap, and fused-operator composition depend on the stack. The matched V100/Megatron-LM supplement shows transfer in peak-memory ranking, recomputation direction, and matrix-problem difficulty, but not in absolute iteration time, operator composition, or exposed communication share. A single NPU-to-GPU scale factor is insufficient, and TPUs and other NPU generations remain untested (Appendix E). (3) **Configuration coverage.** The grid is not exhaustive, and incomplete GPU profiles lack failure logs and are not labeled as out-of-memory. (4) **Baseline coverage.** The comparison between two simulator families uses six common configurations, and the multi-node Calculon result is a compatibility-mode stress test. The evaluation should therefore be interpreted as a protocol demonstration rather than a comprehensive ranking of simulators.

AI Use Statement

We used generative AI tools to assist with the writing and presentation of this manuscript: improving its organization, clarity, and academic English, and checking internal consistency and layout. Generative AI tools were not used to formulate the research hypotheses, to design the methodology or the experiments, to implement the released code or the evaluated simulators, to generate or process the measured profiling data, or to interpret the results. All AI-assisted text was reviewed by the authors, who verified every technical claim and reported value and take responsibility for the final content of this work.

Ethics Statement

LM-SimBench contains system telemetry from controlled LLM training runs, not user-generated or personal data. Its principal risks are coverage bias and extrapolation beyond the profiled hardware and software stacks. The cross-platform supplement bounds rather than eliminates these risks: it identifies signals that transfer between two stacks and those that require target-platform calibration, without establishing behavior for other accelerators or unbounded workload classes. Section 4 and Appendix C.5 document intended and discouraged uses, licensing, release terms, and scope limitations. No human subjects were involved.

Reproducibility Statement

Section 4 describes the processed and raw profiling artifacts. Appendix C reports the hardware and software stack, processing pipeline, configuration coverage, file layout, and schemas. Appendix E documents the second platform, matched and stack-specific factors, common configurations, and treatment of incomplete profiles. Section 5 defines the metrics, and Section 6 reports the baseline evaluation. The manuscript provides anonymous data and code links for reproduction during review.

References

Alibaba. Alibaba cluster trace program: GPU cluster traces, 2023. URL https://github.com/alibaba/clusterdata. Production GPU cluster traces (cluster-trace-gpu-v2020 and cluster-trace-gpu-v2023). Accessed: April 27, 2026.

Alibaba. AICB: An open-source AI communication benchmark suite, 2024. URL https://github.com/aliyun/aicb. Open-source software repository. Accessed: April 27, 2026.

Alibaba Group. InferSim: A lightweight LLM inference performance simulator, Sep. 2025. URL https://github.com/alibaba/InferSim. Open-source software repository. Accessed: April 27, 2026.

Ascend Community. MindSpeed-LLM: A distributed training suite for large language models on Ascend NPUs, 2024. URL https://github.com/Ascend/MindSpeed-LLM. Open-source software repository. Accessed: April 27, 2026.

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, 2024. URL https://arxiv.org/abs/2312.12391.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL http://jmlr.org/papers/v13/bergstra12a.html.

Bernd Bischl et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, 2023. doi: 10.1002/widm.1484. URL https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1484.

Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. NeuralPower: Predict and deploy energy-efficient convolutional neural networks. In *Proceedings of the Ninth Asian Conference on Machine Learning (ACML)*, volume 77 of *Proceedings of Machine Learning Research*, pp. 622–637. PMLR, 2017. URL https://proceedings.mlr.press/v77/cai17a.html.

Jaehong Cho, Minsu Kim, Hyunmin Choi, Guseul Heo, and Jongse Park. LLMServingSim: A HW/SW co-simulation infrastructure for LLM inference serving at scale. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 15–29. IEEE, Sep. 2024. doi: 10.1109/IISWC63097.2024.00012. URL http://dx.doi.org/10.1109/IISWC63097.2024.00012.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1437–1446. PMLR, 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

Aaron Grattafiori et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Zhenhua Guo et al. A survey on performance modeling and prediction for distributed DNN training. *IEEE Transactions on Parallel and Distributed Systems*, 35(12):2463–2478, Dec. 2024. doi: 10.1109/TPDS.2024.3476390. URL https://doi.org/10.1109/TPDS.2024.3476390.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Qinghao Hu et al. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24)*, pp. 709–729, 2024. URL https://www.usenix.org/conference/nsdi24/presentation/hu.

Mikhail Isaev, Nic McDonald, Larry Dennison, and Richard W. Vuduc. Calculon: A methodology and tool for high-level co-design of systems and large language models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23)*. ACM, 2023. doi: 10.1145/3581784.3607102. URL https://doi.org/10.1145/3581784.3607102.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Mahmoud Khairy, Zhensheng Shen, Tor M. Aamodt, and Timothy G. Rogers. Accel-Sim: An extensible simulation framework for validated GPU modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 473–486, May 2020. doi: 10.1109/ISCA45697.2020.00047.

Joyjit Kundu et al. Performance modeling and workload analysis of distributed large language model training and inference. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 57–67, Sep. 2024. doi: 10.1109/IISWC63097.2024.00015. URL https://doi.org/10.1109/IISWC63097.2024.00015.

Seonho Lee, Amar Phanishayee, and Divya Mahajan. Forecasting GPU performance for deep learning training and inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '25)*, 2025. doi: 10.1145/3669940.3707265. URL https://arxiv.org/abs/2407.13853.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL https://jmlr.org/papers/v18/16-558.html.

Mingyu Liang et al. Lumos: Efficient performance modeling and estimation for large-scale LLM training. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025. URL https://arxiv.org/abs/2504.09307.

Heng Liao et al. Ascend: A scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 789–801, 2021. doi: 10.1109/HPCA51647.2021.00071.

Peter Mattson et al. MLPerf training benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pp. 336–349, 2020. URL https://proceedings.mlsys.org/paper_files/paper/2020/hash/411e39b117e885341f25efb8912945f7-Abstract.html.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=SyVVJ85lg.

Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. ASTRA-SIM: Enabling SW/HW co-design exploration for distributed DL training platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 81–92. IEEE, 2020. doi: 10.1109/ISPASS48437.2020.00018.

Vijay Janapa Reddi et al. MLPerf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459, 2020. doi: 10.1109/ISCA45697.2020.00045.

Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. SCALE-Sim: Systolic CNN accelerator simulator, 2019. URL https://arxiv.org/abs/1811.02883.

Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul N. Whatmough, Matthew Mattina, and Tushar Krishna. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 58–68, Aug. 2020. doi: 10.1109/ISPASS48437.2020.00016.

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

Srinivas Sridharan et al. Chakra: Advancing performance benchmarking and co-design using standardized execution traces. *arXiv preprint arXiv:2305.14516*, 2023. URL https://arxiv.org/abs/2305.14516.

Jonas Svedas et al. A survey of end-to-end modeling for distributed DNN training: Workloads, simulators, and TCO. *arXiv preprint arXiv:2506.09275*, Jun. 2025. URL https://arxiv.org/abs/2506.09275.

Xiaokai Wang, Shaoyuan Huang, Yuting Li, and Xiaofei Wang. ScaleDL: Towards scalable and efficient runtime prediction for distributed deep learning workloads. *arXiv preprint arXiv:2511.04162*, Nov. 2025a. URL https://arxiv.org/abs/2511.04162.

Xizheng Wang et al. SimAI: Unifying architecture design and performance tuning for Large-Scale large language model training with scalability and precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '25)*, pp. 541–558, Apr. 2025b. ISBN 978-1-939133-46-5. URL https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, Apr. 2009. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

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. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 283–294. IEEE, Apr. 2023. doi: 10.1109/ISPASS57527.2023.00035. URL http://dx.doi.org/10.1109/ISPASS57527.2023.00035.

Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, Nov. 2020. ISSN 0925-2312. doi: 10.1016/j.neucom.2020.07.061. URL http://dx.doi.org/10.1016/j.neucom.2020.07.061.

Siyuan Yang, Enda Yu, Pingjing Lu, and Dezun Dong. LLMEmu: A lightweight performance emulator for high-fidelity distributed LLM training. *Performance Evaluation*, 170:102526, Nov. 2025. doi: 10.1016/j.peva.2025.102526.

Yang You, Aydın Buluç, and James Demmel. Scaling deep learning on GPU and Knights Landing clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*, pp. 1–12. ACM, Nov. 2017. doi: 10.1145/3126908.3126912. URL http://dx.doi.org/10.1145/3126908.3126912.

Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A runtime-based computational performance predictor for deep neural network training. In *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC '21)*, pp. 503–521. USENIX Association, Jul. 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/conference/atc21/presentation/yu.

Hengrui Zhang, August Ning, Rohan Prabhakar, and David Wentzlaff. A hardware evaluation framework for large language model inference, 2023. URL https://arxiv.org/abs/2312.03134.

A Related Work

This appendix positions LM-SimBench relative to configuration search, neural-network performance prediction, large-model training simulation, and system benchmarks. Rather than attempting a comprehensive survey, we identify which prior artifacts support comparable evaluation claims and which gaps remain.

A.1 Configuration Search and Performance-Aware Optimization

Classical hyperparameter optimization (HPO) selects parameters such as learning rate, regularization strength, model depth, or batch size under a limited evaluation budget Yang & Shami (2020); Bischl et al. (2023). Grid search, random search, multi-fidelity methods, and Bayesian optimization reduce the number or cost of trials when each candidate must be trained at least partially Bergstra & Bengio (2012); Li et al. (2018); Falkner et al. (2018). These methods also provide general search strategies for expensive systems configuration spaces.

Large-model training, however, introduces a distinct set of systems decisions. Tensor, pipeline, data, context, and expert parallelism; micro-batch scheduling; activation recomputation; communication overlap; and placement directly determine memory feasibility, iteration time, utilization, and cost. The objective therefore concerns not only model quality but also whether a workload executes efficiently on a specific hardware and software stack. Simulation-guided search complements HPO by replacing many physical trials with lower-cost performance estimates. Evaluating such simulators requires memory, communication, operator, and kernel signals in addition to final runtimes.

A.2 Neural-Network Performance Prediction

Earlier work predicts the runtime, throughput, power, or energy of neural networks from model and hardware features. Analytical approaches decompose a model into layer or operator costs and estimate execution time from operation counts, memory traffic, and hardware characteristics. The Roofline model provides interpretable bounds based on arithmetic intensity and memory bandwidth Williams et al. (2009); SCALE-Sim and Accel-Sim model behavior at the systolic-array or GPU level Samajdar et al. (2019); Khairy et al. (2020). These tools support hardware analysis but generally do not represent the complete distributed execution of large-model training.

Profiling-based predictors instead measure representative operators on target hardware and aggregate the measurements into workload-level estimates. Paleo combines analytical modeling with calibrated measurements for deep neural network training Qi et al. (2017), while NeuralPower estimates runtime, power, and energy for convolutional-network layers Cai et al. (2017). Subsequent work develops operator- and workload-level predictors for different accelerators and distributed settings You et al. (2017); Samajdar et al. (2020); Yu et al. (2021). Most of these methods target smaller models, single-node settings, or restricted parallel patterns. LM-SimBench extends this empirical approach to distributed LLM training and exposes intermediate measurements for diagnosing aggregate runtime errors.

A.3 Large-Model System Simulators

Recent simulators model large-model execution under particular workload, parallelism, and hardware assumptions. Their dominant approaches are analytical, graph-based, and profiling- or measurement-based, although practical systems often combine them Guo et al. (2024); Svedas et al. (2025).

Analytical simulators estimate computation, communication, and memory costs from closed-form models or structured performance equations. Calculon predicts batch execution time and memory consumption for large-model training under Megatron-style parallelism Isaev et al. (2023); Shoeybi et al. (2019). SimAI builds a full-stack simulation infrastructure for large-scale LLM training across framework and network layers Wang et al. (2025b). Serving-oriented systems such as LLMCompass address related hardware–software co-design questions for LLM deployment, but they target different metrics and workload dynamics Zhang et al. (2023). These tools are efficient and interpretable, but their claims depend strongly on the validity of the assumed cost formulas and hardware abstractions.

Graph-based and task-based simulators represent execution as operators, tasks, dependencies, or collective-communication events. ASTRA-sim models distributed training with explicit computation and communication behavior, with particular attention to hierarchical networks and collective communication Rashidi et al. (2020); Won et al. (2023). Optimus uses operator graphs for large-model workload and accelerator analysis Kundu et al. (2024). Such representations can expose scheduling effects, pipeline bubbles, communication dependencies, and topology-sensitive behavior, but they require enough trace or graph information to justify fine-grained diagnostic claims.

Profiling-based and emulation-based simulators use measurements from real systems to calibrate or replace parts of the cost model. vTrain represents large-model training as a graph while deriving operator and communication costs from real measurements Bang et al. (2024). LLMEmu profiles GPU kernels and emulates distributed LLM training on central processing unit (CPU) clusters Yang et al. (2025). Habitat and ScaleDL further illustrate the data-driven direction by learning performance predictors from empirical observations across hardware or distributed environments Yu et al. (2021); Wang et al. (2025a). Serving simulators such as LLMServingSim and InferSim model request dynamics, throughput, and latency, which are important but outside the current LM-SimBench data release Cho et al. (2024); Alibaba Group (2025). These systems demonstrate strong progress, but their evaluations are often tied to private traces, narrow workload sets, tool-specific metrics, or particular hardware stacks.

Appendix B maps representative simulators onto our unified simulation pipeline. That mapping avoids duplicating a flat list of tools here and instead clarifies which stage each simulator models explicitly, which signals are observable, and which claims are comparable.

A.4 Benchmarks and Datasets for System Evaluation

System benchmarks such as MLPerf Training and MLPerf Inference define executable workloads, rules, quality targets, and reporting procedures for comparing machine-learning systems Mattson et al. (2020); Reddi et al. (2020). They make end-to-end hardware and software comparisons more reproducible, but their target is the performance of complete systems on specified workloads rather than the validation of simulators against reusable, multi-granularity profiles.

Large-model simulators need a different kind of evaluation artifact. A simulator may be useful for end-to-end iteration-time prediction, configuration ranking, memory-feasibility screening, communication analysis, operator-latency modeling, calibration, or bottleneck diagnosis. These claims require labels and intermediate evidence at different granularities. Public benchmark data for this purpose remain limited, especially for modern LLM training with multi-dimensional parallelism and measured rank-, event-, operator-, and kernel-level traces.

A.5 Profiling Datasets and Trace Releases

Several recent artifacts overlap partially with LM-SimBench. We group them by their role in the evaluation pipeline and compare workload coverage, the parallelism dimensions swept in the release, signal granularity, hardware, and artifact type (Table 3).

Performance-prediction datasets for deep learning. NeuSight Lee et al. (2025) releases operator-level and end-to-end latencies for six Transformer models (BERT-Large, GPT-2 Large, GPT-3 XL/2.7B, OPT-1.3B, and Switch Transformer) measured across seven NVIDIA and AMD GPUs, and is explicitly intended to support the development of training-performance simulators. It is the closest artifact to LM-SimBench in purpose. Compared with LM-SimBench, NeuSight covers only one MoE model (Switch), does not sweep expert parallelism, context parallelism, or recomputation, does not release per-rank peak-memory or communication-event traces, and targets GPUs rather than NPUs.

Cluster-scale operational traces. AcmeTrace Hu et al. (2024) releases six months of operational traces from Shanghai AI Lab's 4,704-GPU LLM-development clusters (Seren and Kalos), including scheduler logs, infrastructure monitoring, and failure events. The Alibaba GPU cluster traces Alibaba (2023) release multi-month production-GPU job traces from large datacenters. Both are widely used for cluster scheduling, capacity planning, and fault-tolerance research. They differ from LM-SimBench in two essential ways: their granularity is job-level scheduler data rather than per-rank operator/communication/kernel traces, and they were not collected as a controlled sweep of parallel configurations for the same workload.

Trace-format specifications and methodology toolkits. Chakra Sridharan et al. (2023) defines a standardized graph schema for execution traces and offers tooling to collect, generate, and replay them across simulators and emulators. Lumos Liang et al. (2025) is a trace-driven performance modeling toolkit evaluated on up to 512 NVIDIA H100s; the paper is primarily a methodology and does not release a curated dataset. AICB Alibaba (2024) is the open-source workload-description generator companion to SimAI, providing parameterized LLM workload specifications across DP/TP/PP/SP/EP and Megatron/DeepSpeed, but it ships specifications rather than measured traces. These artifacts complement LM-SimBench: a curated profiling corpus such as LM-SimBench can in principle be re-expressed in a Chakra-style format or consumed by tools like Lumos.

Inference-side serving traces. Several recent releases share the trace-collection methodology but target inference serving, including public request-arrival traces and evaluation suites for MoE inference. We mention them only to delimit scope: LM-SimBench is the training-side counterpart; serving-trace evaluation is out of scope for the current release.

Table 3: Comparison between LM-SimBench and related public artifacts. "Per-rank signals" lists the per-run profiling categories actually released. "Parallelism swept" lists the parallelism dimensions exposed as part of a configuration grid in the released artifact, not just supported in the underlying tool. DP/TP/PP/CP/EP denote data, tensor, pipeline, context, and expert parallelism. Recomp denotes activation recomputation.

Artifact	Type	Workloads	Per-rank signals released	Parallelism swept	Hardware
NeuSight Lee et al. (2025)	Profiling dataset	5 dense + 1 MoE (Switch) Transformer	Operator latency, end-to-end latency	DP/TP/PP (limited)	GPU (NV+AMD)
AcmeTrace Hu et al. (2024)	Cluster job trace	Production LLM development jobs	Scheduler logs, GPU/CPU/network monitoring, failures	Not applicable (job-level)	GPU (A100)
Alibaba GPU cluster traces Alibaba (2023)	Cluster job trace	Production GPU jobs	Job-level resource and timing	Not applicable (job-level)	GPU
Chakra Sridharan et al. (2023)	Trace format spec	N/A (schema)	DAG schema; example traces only	N/A	Hardware-agnostic
Lumos Liang et al. (2025)	Tool / methodology	GPT-3 family	Kineto / execution-trace tooling (not released as dataset)	Methodology-level	GPU (H100)
AICB Alibaba (2024)	Workload generator	LLaMA, GPT, Mistral-MoE, DeepSeek, Qwen3	Workload specifications only	Specs across DP/TP/PP/SP/EP	GPU-agnostic
LM-SimBench (this work)	Profiling dataset	23 settings, 10 LLM families, dense and MoE	Iteration time, peak memory, time breakdown, operator latency, communication events, GEMM HW counters	DP/TP/PP/CP/EP and recomputation (348 feasible configurations)	NPU (Ascend)

Summary of differences. LM-SimBench fills three gaps not jointly addressed by the artifacts above. First, to our knowledge, it is the first public training-profiling dataset on the Ascend NPU stack, complementing the GPU-centric coverage of NeuSight and cluster traces. Second, it provides six profiling views from configuration to kernel granularity for the same runs. These views support diagnostic evaluation—for example, distinguishing iteration-time error caused by operator modeling from error caused by communication estimation—rather than end-to-end scoring alone. Third, its configuration grid jointly varies DP/TP/PP/CP/EP and recomputation across dense and MoE models from 10 LLM families. This evaluation setting is instantiated for training-time simulation on MindSpeed-LLM and Ascend NPUs; inference serving, other accelerators or runtimes, and unmeasured model classes remain outside its validated scope.

A.6 Positioning of This Work

Existing surveys organize simulator methods and the system-level performance-prediction literature Guo et al. (2024); Svedas et al. (2025); simulator papers contribute concrete tools, and system benchmarks compare complete training or serving systems. LM-SimBench is complementary: it provides a common framework for expressing simulator assumptions and outputs, a dataset containing both final labels and intermediate evidence, and protocols for testing which claims a simulator supports. The resulting artifact makes explicit what is measured, under which assumptions the measurement is meaningful, and which conclusions exceed the dataset's scope.

B Unified Simulation Framework

This appendix expands the unified framework introduced in Section 3. The framework is an abstraction for organizing large-model training simulators, not a simulator implementation. It describes how a model, system, and training strategy are transformed into an execution representation and how computation, communication, and memory costs are estimated. The same abstraction determines how LM-SimBench organizes configurations, profiling traces, operator records, and communication events for reproducible evaluation.

B.1 Simulation Scope and Formalization

Large-model training simulation approximates a workload's execution without running every candidate configuration on a physical cluster. Direct measurement offers high fidelity at high cost, whereas abstraction improves efficiency but may omit important effects. The framework makes this fidelity–cost trade-off explicit for performance prediction and strategy comparison.

Simulation inputs. We describe the target scenario with three inputs. The model configuration M records architectural and workload parameters such as layer types, hidden dimensions, sequence length, batch size, and parameter count. The system configuration N specifies the target hardware, including compute capability, memory capacity, inter-device bandwidth, and topology. The training configuration P specifies choices such as data, tensor, and pipeline parallelism; recomputation; offloading; numerical precision; operator fusion; and computation–communication overlap.

Performance prediction problem. Given (M, P, N), the simulator $\mathcal{S}_\theta$ predicts the performance of one training iteration. Following Eq. 1, its outputs are iteration time $\widehat{T}$, per-device peak memory $\widehat{\mathbf{V}} = (\widehat{V}_d)_{d \in D(N)}$, a task-specific cost $\widehat{C}$, and optional diagnostics $\widehat{z}$. Below, $\widehat{T}$ is the representative metric, and $\widehat{V} = \max_{d \in D(N)} \widehat{V}_d$ denotes scalar peak memory when required.

For a training configuration P, model compilation transforms the abstract model into

$$\bar{M} = \mathcal{C}(M, P), \tag{7}$$

where $\mathcal{C}(\cdot)$ partitions or replicates components according to the selected parallel strategy. Tensor parallelism partitions parameters within operators, pipeline parallelism assigns blocks to stages, and data parallelism replicates the model across groups. Device binding then assigns each component of $\bar{M}$ to devices and communication groups:

$$\Pi = \mathcal{B}(\bar{M}, P, N), \tag{8}$$

where Π records placement and parallel-group membership, allowing the simulator to distinguish intra-node from inter-node communication.

The bound model is then transformed into an execution representation:

$$R = \mathcal{R}(\bar{M}, \Pi, P, N), \tag{9}$$

where R may be an analytical phase decomposition, an operator-level dependency graph, a profiled trace, or a hybrid schedule. It captures the execution order, synchronization constraints, and computation–communication structure induced by pipeline scheduling, tensor partitioning, recomputation, and overlap.

Unit-level and representation-level estimation. The simulator next estimates the latency of each modeled computation or communication unit in $\mathcal{U}(R) = \mathcal{U}_{\text{op}}(R) \cup \mathcal{U}_{\text{comm}}(R)$ and aggregates these

estimates under the ordering and resource constraints in R. As in Eq. 4, the predicted latency of unit u is

$$\widehat{t}_u = \begin{cases} \mathcal{F}_{\text{op}}(\phi_u, \Pi, N), & u \in \mathcal{U}_{\text{op}}(R), \\ \mathcal{F}_{\text{comm}}(\phi_u, \Pi, N), & u \in \mathcal{U}_{\text{comm}}(R), \end{cases} \tag{10}$$

where ϕ_u records features such as tensor shape, FLOPs, bytes transferred, primitive type, group size, and placement; Π provides placement and group membership; and N provides the required hardware parameters. The estimator may be analytical, profiling-based, trace-based, or data-driven, depending on the desired fidelity and cost.

At the representation level, the simulator aggregates unit estimates subject to scheduling and feasibility constraints:

$$\widehat{T} = \begin{cases} \mathcal{A}\big(R, \Pi, \{\widehat{t}_u\}_{u \in \mathcal{U}(R)}, N\big), & I_{\text{feas}}(M, P, N) = 1, \\ +\infty, & I_{\text{feas}}(M, P, N) = 0, \end{cases} \tag{11}$$

The per-device peak memory is aggregated analogously, mirroring Eq. 5:

$$\widehat{\mathbf{V}} = \mathcal{V}(R, \Pi, N), \tag{12}$$

Here, $\mathcal{A}(\cdot)$ aggregates time over the execution representation, and $\mathcal{V}(\cdot)$ aggregates peak memory from the placement and system characteristics in R, Π, and N. The indicator $I_{\text{feas}}(M, P, N)$ from Eq. 2 checks both that P is valid for the target cluster and that every $\widehat{V}_d$ is at most cap_d. A violation makes the configuration infeasible.

Strategy search objective. The same framework supports strategy search. Given a feasible space $\mathcal{K}(M, N)$, the predicted optimum is

$$\min_{P \in \mathcal{K}(M,N)} \widehat{C}, \tag{13}$$

where $\widehat{C}$ may be iteration time, resource cost, or a weighted objective combining time, memory, and hardware utilization. When P only includes tensor parallelism, pipeline parallelism, data parallelism, and recomputation, the feasible space can be constrained as

$$P_{tp} \cdot P_{pp} \cdot P_{dp} \leq N_{\text{proc}}, \tag{14}$$

$$P_{rc} \in \{0, 1\}, \tag{15}$$

where P_{tp}, P_{pp}, and P_{dp} are the tensor-, pipeline-, and data-parallel degrees; P_{rc} is the recomputation switch; and N_{proc} is the number of available processors. Sequence and context parallelism, expert parallelism, micro-batch size, offloading, and overlap can be incorporated through additional decision variables and constraints.

B.2 Core Component Modeling

The core components below determine simulation granularity, fidelity, and cost. Each admits multiple levels of detail: finer modeling can capture more runtime effects but incurs greater simulation and profiling overhead. The framework therefore retains the abstractions most relevant to performance while allowing each stage to be instantiated at an appropriate fidelity.

Large-model abstraction. Large AI models include language models, vision Transformers, multimodal foundation models, and hybrid architectures. Many are built from repeated attention and feedforward blocks. We use decoder-only Transformers as the representative abstraction because they are common in large-scale training and are the focus of LM-SimBench.

A Transformer block contains multi-head self-attention and a position-wise feedforward network. Its computation, memory, and communication are strongly influenced by the hidden dimension h, sequence length s, number of attention heads n_{head}, and number of blocks L. A compact abstraction is therefore

$$(h, s, n_{\text{head}}, L). \tag{16}$$

This tuple is a subset of M in Section 3, which also records d_{ff}, batch size b, and routing or modality parameters when applicable. The reduced tuple omits low-level detail while retaining major structural determinants of computation and memory. Table 4 gives representative examples.

The abstraction extends to other model families by adding the parameters that affect their execution. Encoder–decoder Transformers require separate encoder and decoder stacks; vision and multimodal

Table 4: Simplified abstraction of representative large language models.

Model	h	s	n_{head}	L
GPT-3-13B	5140	2048	40	40
GPT-3-175B	12288	2048	96	96
Megatron-1T	25600	2048	160	128
PaLM-540B	18432	2048	48	118
Turing-530B	20480	2048	128	105
Llama-3-405B	16384	131072	128	126
Qwen3-32B	5120	32768	64	64

Transformers require modality- and tokenization-specific inputs; and Mixture-of-Experts (MoE) models require routing and sparsity parameters. All retain a block-level representation that can feed the same simulation pipeline.

Cluster and hardware abstraction. Cluster modeling reduces the physical environment to three performance-relevant components: compute, memory, and network (Figure 3). This abstraction captures dominant training costs without requiring every runtime or operating-system detail.

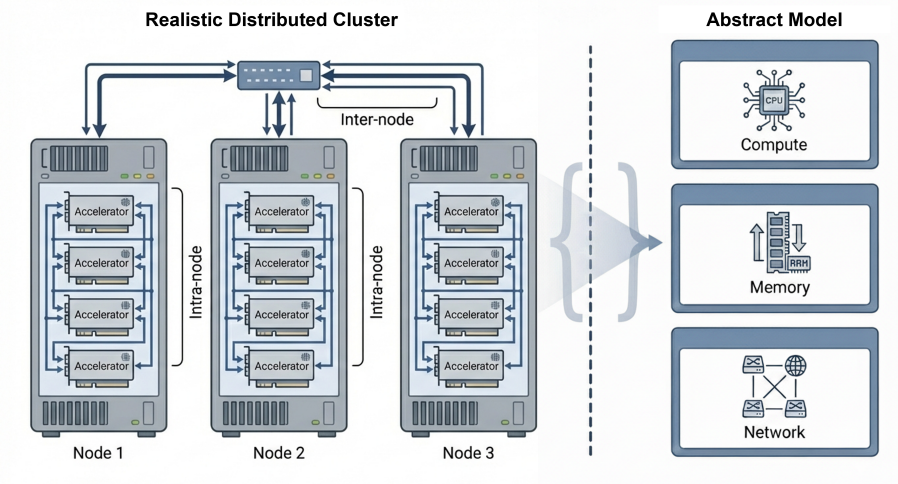


Figure 3: Simplified cluster abstraction. The physical cluster is represented by compute, memory, and network components.

Compute is characterized by data type, peak FLOPs, and empirical efficiency; memory by capacity, bandwidth, and efficiency; and the network by bandwidth, latency, topology, and communication efficiency. The simulator combines peak capabilities with efficiency factors to obtain effective performance.

The abstraction omits effects such as operating-system scheduling, low-level protocol behavior, and device-specific kernel implementations. This choice improves scalability and portability across cluster descriptions; when profiles are available, measured operator or communication latency can replace selected analytical parameters.

Training workflow modeling. Workflow modeling specifies the temporal order of computation and communication after compilation and device binding. In an operator-level graph, nodes represent computation or communication and edges encode data dependencies. A training graph typically includes the forward pass, backward pass, gradient synchronization, and parameter update.

Workflow granularity determines which execution effects can be represented. Graph-based models construct an operator-level DAG for fine-grained timing and dependency analysis. Analytical models reduce execution to phases such as forward, backward, communication, and optimizer update;

they are faster but may miss overlap and synchronization. Hybrid models represent critical or communication-heavy segments explicitly and estimate the remainder analytically.

Computation cost modeling. Computation models estimate the latency of operators or phases using analytical formulas, structure-aware simulation, profiles, learned predictors, or combinations thereof. Roofline-style models estimate latency from FLOPs, throughput, and efficiency. Structure-aware models additionally represent hardware hierarchy, tiling, memory movement, and scheduling. Profiling-based methods reuse measured operator latency. Analytical models can thus provide broad coverage while profiles calibrate important or hardware-specific kernels.

Communication cost modeling. Communication models estimate the latency introduced by distributed parallelism. Data, tensor, pipeline, context, and expert parallelism induce collective or point-to-point operations such as All-Reduce, All-Gather, Reduce-Scatter, Send, and Receive. Their cost depends on message size, primitive, group size, bandwidth, latency, topology, and overlap with computation.

Mechanistic models estimate collectives from topology, bandwidth, latency, and algorithmic decomposition. Trace-driven models replay measured events and preserve their dependencies; packet-level simulators represent congestion and routing at higher computational cost; and data-driven predictors learn latency from message, topology, and configuration features. The appropriate choice depends on whether the goal is rapid search, workload-specific optimization, or high-fidelity architectural analysis.

Together, the input abstractions, workflow representation, and cost estimators determine simulator fidelity and scalability.

B.3 Mapping Representative Simulators

Table 5 maps representative training simulators to the workflow, computation, and communication stages. It records each system's dominant abstraction to clarify comparability, not to rank quality.

B.4 Simulation Outputs and Strategy Search

The pipeline produces task-specific outputs for evaluating candidate configurations: execution time, memory footprint, cost, and optional diagnostics. Time measures predicted efficiency, while memory determines whether a configuration fits on the target devices.

Execution time. The primary output is the estimated iteration time $\widehat{T}$. The aggregator $\mathcal{A}$ in Eq. 11 combines forward, backward, parameter-update, and communication costs under the modeled schedule.

Memory footprint. The pipeline estimates per-device peak memory $\widehat{V}_d$ for every $d \in D(N)$ and reports $\widehat{V} = \max_{d \in D(N)} \widehat{V}_d$ when a scalar is required. The principal components are parameters, activations, gradients, and optimizer states:

$$\widehat{V}_d = \widehat{V}_{\mathrm{param},d} + \widehat{V}_{\mathrm{act},d} + \widehat{V}_{\mathrm{grad},d} + \widehat{V}_{\mathrm{opt},d}, \tag{17}$$

where $\widehat{V}_{\mathrm{param},d}$, $\widehat{V}_{\mathrm{act},d}$, $\widehat{V}_{\mathrm{grad},d}$, and $\widehat{V}_{\mathrm{opt},d}$ denote the memory used by parameters, activations, gradients, and optimizer states on device d, respectively. The value of each term is affected by the model architecture, sequence length, batch size, numerical precision, parallel strategy, and memory optimization methods such as recomputation or offloading.

Cost function. Iteration time and per-device memory can be combined in a task-specific cost function. Consistent with the mapping $\mathcal{S}_\theta(M, P, N) \to (\widehat{T}, \widehat{\mathbf{V}}, \widehat{C}, \widehat{z})$, we write

$$\widehat{C} = C(\widehat{T}, \widehat{\mathbf{V}}, N), \tag{18}$$

where N contains the system characteristics. Depending on the use case, $\widehat{C}$ may represent iteration time, total training time, resource cost, memory pressure, or a weighted combination of objectives.

Strategy search. Strategy selection then becomes the optimization in Eq. (13). For the restricted P above, the feasible space also satisfies Eqs. (14) and (15). More complete searches may add micro-

Table 5: Mapping representative large-model training simulators onto stages C–E of the unified pipeline. *Workflow*: G = graph-based, A = analytical, H = hybrid. *Computation*: An = analytical, St = structure-aware, Pr = profiling-based, Data = data-driven. *Communication*: Mech = white-box mechanistic, Trace = trace-driven, Pkt = packet-level, Data = data-driven. *Open*: publicly released or open-source artifact.

Simulator	Workflow	Comp.	Comm.	Open	Primary focus
Calculon Isaev et al. (2023)	A	An	Mech	✓	Analytical iteration-time and memory estimation for LLM training.
SimAI Wang et al. (2025b)	A	An	Mech	✓	Full-stack large-scale training simulation across framework and network layers.
Optimus Kundu et al. (2024)	G	An/St	Mech	×	Operator-graph modeling for workload and accelerator microarchitecture exploration.
ASTRA-sim 1.0 Rashidi et al. (2020)	H	An	Mech/Pkt	✓	Network-topology and collective-communication simulation.
ASTRA-sim 2.0 Won et al. (2023)	G	An	Mech/Pkt	✓	Detailed workload graphs combined with network-level simulation.
vTrain Bang et al. (2024)	G	Pr	Trace	✓	Graph-based training simulation calibrated by real-system measurements.
LLMEmu Yang et al. (2025)	H	Pr	Trace	✓	Kernel-profiled GPU emulation of distributed LLM training on CPU clusters.
Habitat Yu et al. (2021)	A	Pr/Data	Data	✓	Machine learning (ML)-based runtime prediction across unseen GPU generations.
ScaleDL Wang et al. (2025a)	A	Data	Data	×	Data-driven performance prediction for distributed training.
Paleo Qi et al. (2017)	A	An	Mech	✓	Early CNN training-time estimator for GPU clusters.

and global-batch size, gradient accumulation, sequence length, context and expert parallelism, mixed precision, offloading, communication overlap, and distributed-optimizer settings.

Summary. The framework converts a model, system, and training configuration into comparable predictions of time, memory, cost, and diagnostic signals under explicit assumptions.

C Additional Details of LM-SimBench Dataset

This appendix documents the profiling platform, processing pipeline, workload coverage, and file schema of LM-SimBench. It also explains how to join the profiling views for benchmark construction and diagnostic analysis.

Table 6: Common configuration dimensions considered in large-model training strategy search.

Configuration category	Strategy or parameter
Distributed parallelization	Data parallelism, tensor parallelism, pipeline parallelism, sequence parallelism, context parallelism, expert parallelism
Batch configuration	Micro-batch size, global batch size, gradient accumulation steps
Model architecture	Number of layers, hidden size, feedforward hidden size, number of attention heads
Sequence configuration	Sequence length, maximum context length
Memory optimization	Activation recomputation, gradient checkpointing, offloading, optimizer-state partitioning
Numerical configuration	Mixed-precision training, such as FP16 or BF16
Communication/execution optimization	Communication overlap, gradient synchronization, fused kernels, efficient attention kernels, distributed optimizer
System configuration	Number of accelerators, number of nodes, interconnect bandwidth, network topology

C.1 Profiling Platform and Software Stack

LM-SimBench is constructed from runs in a fixed MindSpeed-LLM/Ascend NPU environment. Because system profiles depend on both hardware and software, Table 7 records the environment used for data collection. These metadata provide context for reproduction, hardware-aware modeling, calibration, and interpretation.

The platform uses an aarch64 host with Kunpeng-920 CPUs and Ascend 910B3 NPUs. Each NPU has 65,536 MB of high-bandwidth memory (HBM), and Huawei Cache Coherent System (HCCS) connects the NPUs. The software stack comprises Compute Architecture for Neural Networks (CANN) 8.0.0, PyTorch 2.1.0, torch-npu 2.1.0.post3, MindSpeed 0.8.0, and the MindSpeed-LLM 2.1.0 branch.

Table 7: Hardware and software environment used to collect the primary LM-SimBench profiles.

Category	**Item**	**Value**
Host processor	CPU architecture	aarch64
	CPU model	Kunpeng-920
	CPU cores	192
Accelerator	NPU model	Ascend 910B3
	HBM per NPU	65536 MB
	NPU interconnect	HCCS
Ascend software stack	CANN toolkit	8.0.0
	CANN kernels	910B, 8.0.0
	CANN Neural Network Acceleration Library (NNAL)	8.0.0
Deep learning runtime	Python	3.9.11
	PyTorch	2.1.0
	torch-npu	2.1.0.post3
	Transformers	4.43.2
Training framework	MindSpeed	0.8.0
	MindSpeed-LLM	branch 2.1.0, commit `bf1e61f4`

C.2 Processing Pipeline and Quality Control

The processed release is derived from MindSpeed-LLM logs and Ascend Profiler outputs. Raw records are indexed by model setting, parallel configuration, and device rank; conversion exposes them through a consistent tabular schema. Algorithm 1 summarizes how the parser extracts each distributed configuration, maps profiler directories to ranks, obtains the target-step iteration time, and aggregates rank-level views into CSV files.

Algorithm 1 Data processing pipeline for LM-SimBench

Require: Raw training log directory L, raw profiler directory $\mathcal{P}_{\mathrm{raw}}$
Ensure: Structured dataset directory D
Initialize the structured output directory D
for each model m in $\mathcal{P}_{\mathrm{raw}}$ **do**
  Identify feasible parallel configurations of m
  for each configuration c of m **do**
    Parse DP, TP, PP, CP, EP, and recomputation settings from c
    for each rank directory r under c **do**
      Read profiler metadata and obtain the rank identifier
      Extract memory, operator, time-breakdown, and communication records
      Extract matrix-kernel features from kernel-level profiler files
    end for
    Parse training logs and extract target-step iteration time
    Aggregate rank-level records into structured CSV files
  end for
end for
Normalize field names, rank identifiers, units, and directory layout
Generate manifest records for coverage and size checks
return Structured dataset directory D

Conversion includes five quality-control steps. It parses configuration names into explicit DP/TP/PP/CP/EP and recomputation fields; normalizes rank identifiers across profiling views; standardizes columns and units across models; removes redundant profiler fields; and records file coverage and size in the root manifest. These checks do not remove measurement noise, but they make the release auditable and reduce ambiguity during benchmark construction.

C.3 Workload and Configuration Coverage

Table 8 summarizes the release, and Table 9 gives the per-model coverage. For each model setting, the latter lists the number of feasible configurations and the observed degree of each parallelism dimension.

Table 8: Scale and coverage of the processed LM-SimBench release. A model setting is a model–context pair; a parallel configuration is a feasible combination of DP/TP/PP/CP/EP and recomputation. Size excludes raw profiler dumps.

Item	Value
Model settings	23
Parallel configurations	348
Data format	CSV
Data modality	Tabular profiling records
Model coverage	Dense and MoE models from 10 LLM families
Configuration dimensions	DP, TP, PP, CP, EP, recomputation
Profiling signals	Time, memory, operators, communication, matrix kernels
Processed dataset size	25 GB

Table 9: Parallel configuration coverage of the sampled large-model training workloads in LM-SimBench. Each row corresponds to one model setting in the processed release. **Configs** denotes the number of feasible configurations sampled for that model. **DP**, **TP**, **PP**, **CP**, and **EP** denote data, tensor, pipeline, context, and expert parallelism, respectively. **Recompute** indicates whether activation recomputation is enabled. A cell such as "1/2/4/8" means that the corresponding dimension covers these distinct degrees across the sampled configurations, while "–" indicates that the dimension is not applicable to the model.

Model	Configs	DP	TP	PP	CP	EP	Recompute
gemma2_27b_8k	3	1/2	4/8	2	1/2	–	RC
gemma2_9b_8k	5	1/2	2/4	2	1/2	–	RC/noRC
glm4_9b_8k	6	1/2	2	2/4	1/2	–	RC/noRC
internlm25_1point8b_32k	16	1/2/4	1/2/4	1/2/4	2/4/8	–	RC/noRC
internlm25_20b_32k_16p	9	1/2	2/4/8	2/4/8	1/2/4	–	RC
internlm25_7b_32k	5	1/2	2/4	2/4	1/2	–	RC/noRC
internlm3_8b_8k	7	1/2	1/2	2/4	2/4	–	RC/noRC
mistral_7b_32k	11	1/2/4	1/2/4/8	1/2/4/8	1	–	RC/noRC
phi35_mini_4k	38	1/2/4/8	1/2/4/8	1/2/4/8	1/2/4	–	RC/noRC
phi35_moe_4k	10	1	1/2/4/8	1/2/4/8	1	1/2/4/8	RC/noRC
qwen15_1.8b_8k	40	1/2/4/8	1/2/4/8	1/2/4/8	1/2/4/8	–	RC/noRC
qwen15_4b_8k	40	1/2/4/8	1/2/4/8	1/2/4/8	1/2/4/8	–	RC/noRC
qwen25_0point5_32k	6	1/2	2	2/4	1/2	–	RC/noRC
qwen25_14b_8k	1	1	4	8	1	–	RC
qwen25_32b_32k	4	1/2	4/8	4	1/2	–	RC/noRC
qwen25_3b_32k	6	1	1/2	1/2	2/4	–	RC/noRC
qwen25_72b_8k	3	1	4/8	4/8	1	–	RC/noRC
qwen25_7b_8k	6	1/2	1/2	4	1/2	–	RC/noRC
qwen2_1.5b_4k	26	1/2/4/8	1/2	1/2/4	1/2/4	–	RC/noRC
qwen3_0.6b_4k	38	1/2/4/8	1/2/4/8	1/2/4	1/2/4/8	–	RC/noRC
qwen3_1.7b_4k	38	1/2/4/8	1/2/4/8	1/2/4	1/2/4/8	–	RC/noRC
qwen3_30b_a3b_4k	2	1	4	2	2	1	RC/noRC
qwen3_8b_4k	28	1/2/4	1/2/4/8	1/2/4	1/2/4	–	RC/noRC

Feasible coverage varies with architecture, parameter count, context length, memory capacity, and cluster size. Smaller settings therefore admit broader DP/TP/PP/CP combinations than some larger or more constrained settings. For MoE workloads, we additionally report expert parallelism because it changes both computation placement and communication.

A matched V100/Megatron-LM supplement covers a CP=1 subset of the released configurations for two Qwen3 model settings; its platform metadata, alignment accounting, and per-configuration comparisons are reported in Appendix E.

C.4 File Schema and Field Definitions

The processed release stores structured CSV files in a hierarchy that mirrors distributed execution. A global manifest indexes model settings; each model directory contains feasible configurations; and each configuration contains rank-level records. Users can therefore begin with end-to-end labels and drill down to memory, communication, operator, and matrix-kernel measurements.

The layout below places the global index in `data_manifest.csv`. Each `<model_name>` directory contains configuration- and rank-level summaries of iteration time, peak memory, time breakdowns, operator latency, and communication. Each `<parallel_config>` directory stores event- and kernel-level files under `rank_<id>/`.

```
LM-SimBench/
  data_manifest.csv
  <model_name>/
    step_time_summary.csv
```

```
peak_memory.csv
step_trace_time_summary.csv
operator_latency_by_category.csv
comm_summary.csv
<parallel_config>/
  rank_<id>/
    comm_time.csv
    matrix_operator_latency.csv
```

Table 10 describes each file's location, granularity, and benchmark role. Table 11 lists representative join keys, targets, and features. The dataset repository provides complete column definitions.

The primary join key is `optimization_config`, which identifies the distributed configuration of a profiling run. Dense-model names generally follow `dp<dp>-tp<tp>-pp<pp>-cp<cp>-<RC|noRC>`; MoE names may additionally contain `ep<ep>`. The suffixes `RC` and `noRC` indicate whether activation recomputation is enabled.

For end-to-end prediction, `step_time_summary.csv` supplies labels that can be joined with configuration-derived features. For feasibility modeling, `peak_memory.csv` supplies rank-level targets. The step-trace, operator, and communication summaries support attribution to computation, communication, memory pressure, or pipeline effects. Finally, `comm_time.csv` and `matrix_operator_latency.csv` expose event- and kernel-level records for hardware-aware analysis.

The schema thus supports both black-box prediction and interpretable error analysis: users can score final iteration time and determine whether residual errors agree with observed memory, communication, operator, or kernel behavior.

C.5 Dataset Documentation and Responsible Use

This section summarizes the dataset-documentation elements used to accompany the release. The release documentation and Croissant metadata provide the canonical machine-readable record, while the paper summarizes the main information needed to interpret the dataset responsibly. Table 12 lists the key documentation topics.

C.6 Dataset Availability, License, and Maintenance

Dataset availability. LM-SimBench consists of processed profiling tables, raw profiling traces, benchmark examples, and supporting documentation for large-model training simulation. For anonymous review, all resources are provided through anonymized repositories. Upon acceptance, the repositories will be de-anonymized and publicly released. The raw profiling traces will be hosted on Hugging Face, while the benchmark code, preprocessing scripts, and usage examples will be hosted in the accompanying code repository.

License. The LM-SimBench datasets, including processed profiling tables, raw profiling traces, metadata, dataset cards, and documentation, are released under the CC BY 4.0 license, which permits sharing and adaptation with appropriate attribution. The accompanying code, including preprocessing scripts, benchmark construction scripts, and evaluation examples, is released under the Apache-2.0 license. LM-SimBench contains system-level profiling records collected from controlled distributed training runs and does not include personal information, user-generated text, or private user data.

Documentation. The released repositories provide dataset documentation, field descriptions, usage examples, and datasheet-style documentation. These materials describe the dataset motivation, composition, collection process, preprocessing pipeline, file organization, intended uses, limitations, and ethical considerations. The repository also includes examples for constructing common benchmark tasks, including iteration time prediction, memory usage analysis, communication overhead modeling, operator-level latency modeling, and parallel-configuration selection.

Maintenance plan. LM-SimBench will be maintained as a versioned public dataset and benchmark. Released versions will be preserved to support reproducibility, and major updates will be documented

Table 10: Overview of the file organization in the processed LM-SimBench release. The table reports the storage location, profiling granularity, and benchmark role of each major CSV file. Model-level files provide configuration- and rank-level summary views, while configuration/rank-level files expose event-level communication behavior and matrix-kernel execution details.

File	Location	Granularity	Benchmark role
`data_manifest.csv`	Dataset root	Dataset / model level	Provides the global index of model directories, configuration counts, rank-directory counts, file coverage, and dataset-scale statistics.
`step_time_summary.csv`	Model directory	Configuration level	Provides the primary end-to-end runtime label for iteration-time prediction and configuration ranking.
`peak_memory.csv`	Model directory	Rank level	Provides rank-level peak memory labels for memory modeling, feasibility analysis, and imbalance analysis across ranks.
`step_trace_time_summary.csv`	Model directory	Rank level	Decomposes iteration time into computation, communication, idle time, and pipeline-related components for bottleneck diagnosis.
`operator_latency_by_category.csv`	Model directory	Operator-category / rank level	Aggregates profiler operators into semantic categories for operator-latency modeling and module-level performance analysis.
`comm_summary.csv`	Model directory	Communication-group / rank level	Provides aggregated communication statistics for comparing collective and point-to-point communication costs across configurations.
`comm_time.csv`	Configuration / rank directory	Communication-event / rank level	Preserves event-level communication behavior for analyzing waiting time, transfer size, link bandwidth, and communication bottlenecks.
`matrix_operator_latency.csv`	Configuration / rank directory	Matrix-kernel / rank level	Provides matrix-operator shapes, latency, data types, and hardware-counter features for GEMM modeling and hardware-aware analysis.

in a changelog. Future updates may include additional model families, parallel configurations, profiling traces, processed tables, and benchmark scripts. Issues related to missing files, inconsistent field names, preprocessing errors, or data-loading problems will be tracked through the public code repository. Major updates will be released as new dataset versions rather than overwriting previous releases.

Official repositories. In the camera-ready version, the official repositories will be released as follows:

Table 11: Representative fields of the main CSV files in LM-SimBench. The table lists key columns used for joining records and constructing benchmark targets or features. `optimization_config` is the primary key for configuration-level records, while `rank_id` identifies rank-level profiling records.

File	Representative fields
`data_manifest.csv`	`model_name`, `parallel_config_count`, `rank_directory_count`, `size_bytes`, `size_human`.
`step_time_summary.csv`	`optimization_config`, `iteration_time_ms`.
`peak_memory.csv`	`optimization_config`, `rank_id`, `peak_memory_mb`.
`step_trace_time_summary.csv`	`optimization_config`, `rank_id`, `computing`, `communication`, `communication_not_overlapped`, `overlapped`, `free`, `bubble`.
`operator_latency_by_category.csv`	`optimization_config`, `rank_id`, `op_category`, `op_type`, `core_type`, `count`, `total_time_us`, `avg_time_us`, `ratio_percent`.
`comm_summary.csv`	`comm_scope`, `comm_type`, `comm_group`, `op_count`, `elapsed_time_sum_ms`, `wait_time_sum_ms`, `link_bandwidth`.
`comm_time.csv`	`comm_scope`, `comm_type`, `comm_group`, `op_name`, `elapsed_time_ms`, `wait_time_ms`, `link_type`, `link_transit_size_mb`, `link_bandwidth_gbps`.
`matrix_operator_latency.csv`	`type`, `duration_us`, `input_shapes`, `output_shapes`, `a_shape`, `b_shape`, `c_shape`, `batch`, `m`, `n`, `k`, `aic_mac_ratio`, `cube_utilization_percent`.

- Processed profiling tables: `https://huggingface.co/datasets/anonymous-ljasd/LM-SimBench`
- Raw profiling traces: `https://huggingface.co/datasets/anonymous-ljasd/LM-SimBench_raw`
- Benchmark code and examples: `https://anonymous.4open.science/r/test-xmllll1653`

D Representative Profiling Cases in LM-SimBench

This appendix illustrates the granularity of LM-SimBench through two case studies. Rather than comparing every workload, we visualize configuration-level iteration time, rank-level memory and time breakdowns, communication events, and matrix-operator latency.

D.1 Case Study I: Single-node Small Model

The first case fixes `qwen15_1.8b_8k` and varies DP, TP, PP, CP, and recomputation to expose within-workload differences in iteration time and memory.

Figure 4 shows substantial variation in iteration time across parallel configurations. Figure 5 reports rank-level peak memory, illustrating the effects of parallel partitioning and recomputation.

D.2 Case Study II: Multi-node Dense Model

The second case uses the multi-node dense workload `qwen25_32b_32k` to examine rank-level memory, iteration-time components, and communication. We inspect the lower-level records for rank 0 under `dp1-tp8-pp4-cp1-RC`. Figure 10 aggregates communication time by primitive;

Table 12: Dataset documentation and responsible-use summary for LM-SimBench. The entries summarize the information provided in the dataset card, Croissant metadata, and paper appendix.

Topic	Summary
Intended use	Evaluation and development of large-model training simulators, including iteration-time prediction, configuration ranking, memory-feasibility modeling, communication analysis, operator modeling, and simulator calibration.
Evaluation scope	Claims are scoped to the profiled MindSpeed-LLM and Ascend NPU training environment, the sampled dense and MoE LLM workloads, and the released parallel configurations. Transfer to other hardware, runtime stacks, inference serving, or unprofiled model classes requires additional validation.
Provenance	Records are derived from controlled distributed training runs using MindSpeed-LLM logs and Ascend Profiler outputs. The processed release is a normalized tabular derivative of the raw profiler dumps.
Personal or sensitive information	The dataset contains system profiling records rather than user prompts, personal records, or web-scraped text. It does not intentionally contain personal or sensitive information.
Biases and limitations	The primary biases are coverage and platform biases: workloads, feasible configurations, operator implementations, profiler behavior, and communication patterns are tied to the sampled models and the Ascend/MindSpeed software-hardware stack.
Synthetic data	The release does not use synthetic samples or human annotations. The records are measured profiling outputs generated by controlled training executions.
Access and metadata	The processed and raw releases are hosted separately. The submission package includes Croissant metadata for both released artifacts and a compact sample of the processed schema for reviewer inspection because the processed release exceeds 4 GB. License and access terms are specified in the release metadata.
Not recommended uses	The dataset should not be used as evidence for universal simulator accuracy, production-serving behavior, cross-accelerator performance, or safety/fairness properties of language models without additional experiments.

`batchSendRecv` dominates the total. Figure 11 ranks matrix shapes by cumulative latency, showing that a small subset accounts for most matrix-operator time.

We also report the operator-level predictions used in the Calculon baseline. For compatibility, this multi-node test substitutes multi-head attention for grouped-query attention and enables activation recomputation because Calculon natively supports neither grouped-query attention nor context parallelism. Calculon predicts 73.76 s per iteration versus the measured 112.22 s, a 34.3% underestimate; the result is therefore a compatibility-mode stress test rather than a faithful reproduction. Figure 12 compares measured and predicted cumulative matrix-operator latency. Calculon preserves the ordering of dominant shapes and recovers the main `MatMulV3` hotspots, but underestimates their cumulative cost by 27–35%.

Figure 8 compares rank-level peak memory under four configurations; the segmented curves reflect parallel partitioning and pipeline-stage assignment. Figure 9 decomposes iteration time by rank and reports non-overlapped communication, revealing nonuniform computation and communication across the distributed execution.

Table 13: Matrix-operator prediction metrics grouped by parallel configuration for `qwen15_1.8b_8k`.

Parallel configuration	MAE (ms)	sMAPE (%)	R^2
dp1-tp1-pp1-cp8-RC	0.048	40.505	0.779

Continued on next page

Table 13: Matrix-operator prediction metrics grouped by parallel configuration for `qwen15_1.8b_8k` (continued).

Parallel configuration	MAE (ms)	sMAPE (%)	R^2
dp1-tp1-pp1-cp8-noRC	0.053	42.190	0.778
dp1-tp1-pp2-cp4-RC	0.072	36.794	0.865
dp1-tp1-pp2-cp4-noRC	0.076	38.126	0.874
dp1-tp1-pp4-cp2-RC	0.119	30.313	0.889
dp1-tp1-pp4-cp2-noRC	0.128	31.596	0.895
dp1-tp1-pp8-cp1-RC	0.251	31.436	0.875
dp1-tp1-pp8-cp1-noRC	0.267	32.491	0.881
dp1-tp2-pp1-cp4-RC	0.056	42.201	0.725
dp1-tp2-pp1-cp4-noRC	0.061	45.811	0.741
dp1-tp2-pp2-cp2-RC	0.093	43.991	0.792
dp1-tp2-pp2-cp2-noRC	0.113	49.147	0.769
dp1-tp2-pp4-cp1-RC	0.203	42.833	0.741
dp1-tp2-pp4-cp1-noRC	0.245	48.783	0.721
dp1-tp4-pp1-cp2-RC	0.060	39.834	0.731
dp1-tp4-pp1-cp2-noRC	0.069	43.331	0.731
dp1-tp4-pp2-cp1-RC	0.112	41.730	0.700
dp1-tp4-pp2-cp1-noRC	0.144	48.204	0.690
dp1-tp8-pp1-cp1-RC	0.107	60.527	0.608
dp1-tp8-pp1-cp1-noRC	0.109	61.202	0.655
dp2-tp1-pp1-cp4-RC	0.072	36.826	0.864
dp2-tp1-pp1-cp4-noRC	0.076	38.311	0.873
dp2-tp1-pp2-cp2-RC	0.120	30.403	0.890
dp2-tp1-pp2-cp2-noRC	0.127	31.581	0.897
dp2-tp1-pp4-cp1-RC	0.252	31.580	0.874
dp2-tp1-pp4-cp1-noRC	0.268	32.662	0.880
dp2-tp2-pp1-cp2-RC	0.101	45.635	0.758
dp2-tp2-pp1-cp2-noRC	0.118	50.418	0.748
dp2-tp2-pp2-cp1-RC	0.203	42.782	0.741
dp2-tp2-pp2-cp1-noRC	0.245	48.749	0.722
dp2-tp4-pp1-cp1-RC	0.117	43.081	0.696
dp2-tp4-pp1-cp1-noRC	0.141	47.530	0.690
dp4-tp1-pp1-cp2-RC	0.120	30.453	0.888
dp4-tp1-pp1-cp2-noRC	0.127	31.512	0.894
dp4-tp1-pp2-cp1-RC	0.262	32.662	0.869
dp4-tp1-pp2-cp1-noRC	0.271	32.963	0.879
dp4-tp2-pp1-cp1-RC	0.210	43.702	0.727
dp4-tp2-pp1-cp1-noRC	0.245	48.707	0.721
dp8-tp1-pp1-cp1-RC	0.279	34.472	0.859
dp8-tp1-pp1-cp1-noRC	0.292	34.973	0.859

Table 14: Matrix-operator prediction metrics grouped by matrix dimensions for `qwen15_1.8b_8k`.

m	n	k	MAE (ms)	sMAPE (%)	R^2
1024	2048	2048	0.011	24.151	-23.785
1024	2048	5504	0.056	55.184	-17.086
1024	2048	6144	0.068	58.875	-1485.887
1024	2048	11008	0.089	46.131	-11.877
1024	5504	2048	0.046	47.035	-7.170
1024	6144	2048	0.048	45.422	-48.490
1024	11008	2048	0.064	35.649	-17.001
1024	151936	2048	0.968	40.604	-8433.476
2048	1024	2048	0.009	19.450	-54.752
2048	2048	1024	0.002	4.102	-1.917
2048	2048	2048	0.031	44.121	-192.209
2048	2048	2752	0.077	68.929	-119.713
2048	2048	3072	0.082	63.776	-3.877

Continued on next page

Table 14: Matrix-operator prediction metrics grouped by matrix dimensions for `qwen15_1.8b_8k` (continued).

m	n	k	MAE (ms)	sMAPE (%)	R^2
2048	2048	5504	0.096	48.069	-5.655
2048	2048	6144	0.075	37.723	-465.362
2048	2048	11008	0.153	41.648	-1665.730
2048	2752	2048	0.062	59.668	-59.700
2048	3072	2048	0.050	47.278	-42.370
2048	5504	2048	0.070	38.430	-26.830
2048	6144	2048	0.073	36.576	-336.359
2048	11008	2048	0.093	27.610	-75.468
2048	75968	2048	0.736	32.456	-19609.963
2048	151936	2048	1.269	28.616	-20609.887
4096	512	2048	0.009	20.546	-39.561
4096	1024	2048	0.035	48.722	-190.596
4096	1376	2048	0.066	62.631	-160.490
4096	1536	2048	0.047	44.944	-69.564
4096	2048	512	0.005	12.478	-7.420
4096	2048	1024	0.039	51.339	-7.258
4096	2048	1376	0.065	61.595	-134.887
4096	2048	1536	0.038	37.719	-7.578
4096	2048	2048	0.044	33.438	-41.673
4096	2048	2752	0.099	50.468	-16.146
4096	2048	3072	0.192	68.259	-3.183
4096	2048	5504	0.170	43.264	-3.610
4096	2048	6144	0.079	21.620	-626.595
4096	2048	11008	0.263	38.596	-6264.952
4096	2048	37984	2.144	72.136	-3379.506
4096	2752	2048	0.081	43.180	-27.359
4096	3072	2048	0.078	38.374	-16.436
4096	5504	2048	0.114	32.488	-12.658
4096	6144	2048	0.108	28.405	-31.871
4096	11008	2048	0.193	29.637	-27.133
4096	37984	2048	1.271	48.828	-7.287
4096	75968	2048	1.350	30.161	-42459.596
4096	151936	2048	2.326	26.538	-1863.017
8192	256	2048	0.018	36.910	-912.448
8192	512	2048	0.032	45.419	-246.820
8192	688	2048	0.191	113.572	-2936.190
8192	768	2048	0.078	64.092	-15.424
8192	1024	2048	0.042	32.793	-101.502
8192	1376	2048	0.091	47.508	-38.599
8192	1536	2048	0.078	38.603	-21.473
8192	2048	256	0.009	20.407	-18.166
8192	2048	512	0.030	43.394	-19.203
8192	2048	688	0.234	120.810	-13.080
8192	2048	768	0.032	33.151	-405.936
8192	2048	1024	0.062	42.996	-4.525
8192	2048	1376	0.092	47.680	-33.115
8192	2048	1536	0.121	48.883	-1.636
8192	2048	2048	0.074	28.819	-12.340
8192	2048	2752	0.177	45.856	-8.363
8192	2048	3072	0.545	88.956	-13.979
8192	2048	5504	0.435	54.585	-5.358
8192	2048	6144	0.189	26.503	-15.315
8192	2048	11008	0.387	29.873	-81.126
8192	2048	18992	1.663	60.863	-5194.037
8192	2048	37984	5.344	82.560	-10632.637
8192	2752	2048	0.139	38.224	-18.318
8192	3072	2048	0.137	34.575	-13.285
8192	5504	2048	0.268	38.426	-8.547
8192	6144	2048	0.259	34.041	-6.929

Continued on next page

Table 14: Matrix-operator prediction metrics grouped by matrix dimensions for `qwen15_1.8b_8k` (continued).

m	n	k	MAE (ms)	sMAPE (%)	R^2
8192	11008	2048	0.440	32.862	-9.871
8192	18992	2048	1.423	53.131	-8.056
8192	37984	2048	2.795	52.109	-6.426
8192	75968	2048	2.778	30.899	-8019.203
8192	151936	2048	5.433	30.293	-136.711

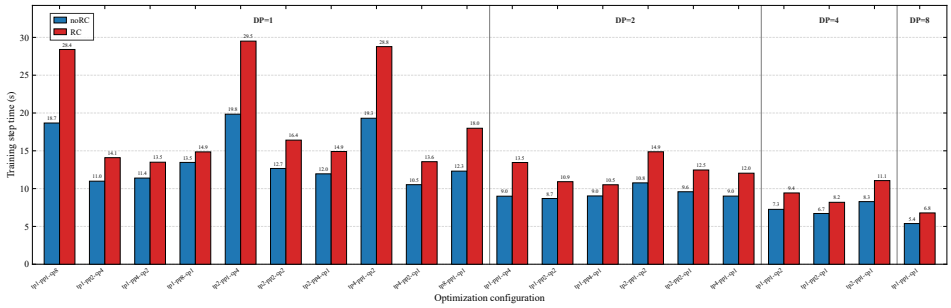


Figure 4: **Iteration time across parallel configurations.** Mean iteration time of `qwen15_1.8b_8k` under different DP/TP/PP/CP and recomputation settings. Colors indicate recomputation, and shaded regions group configurations by DP degree.

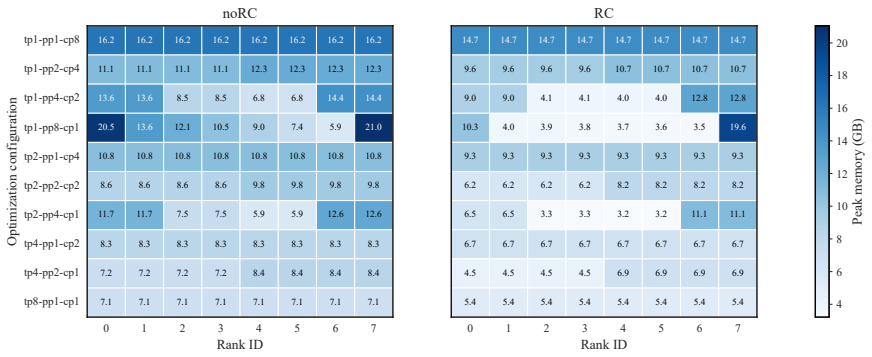


Figure 5: **Rank-level peak memory under DP=1 configurations.** Peak memory of `qwen15_1.8b_8k` by rank across TP/PP/CP configurations. The panels show noRC and RC configurations, respectively.

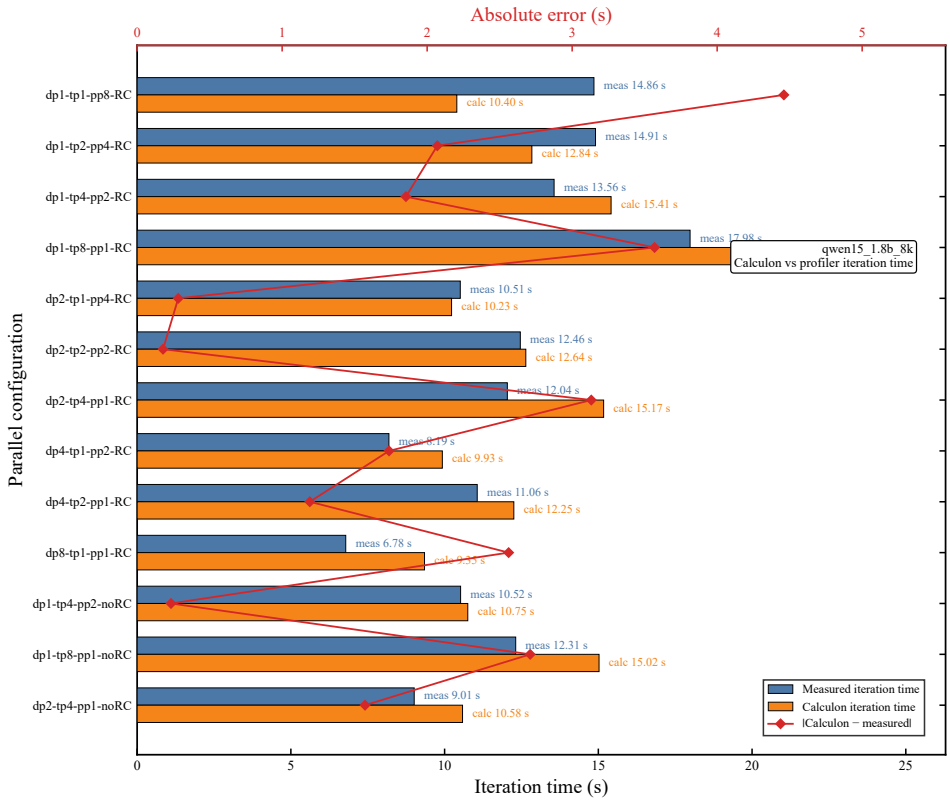


Figure 6: **Calculon iteration-time prediction on `qwen15_1.8b_8k`.** Measured and Calculon-predicted iteration time for the 13 configurations accepted by Calculon's memory model. The right axis shows absolute error.

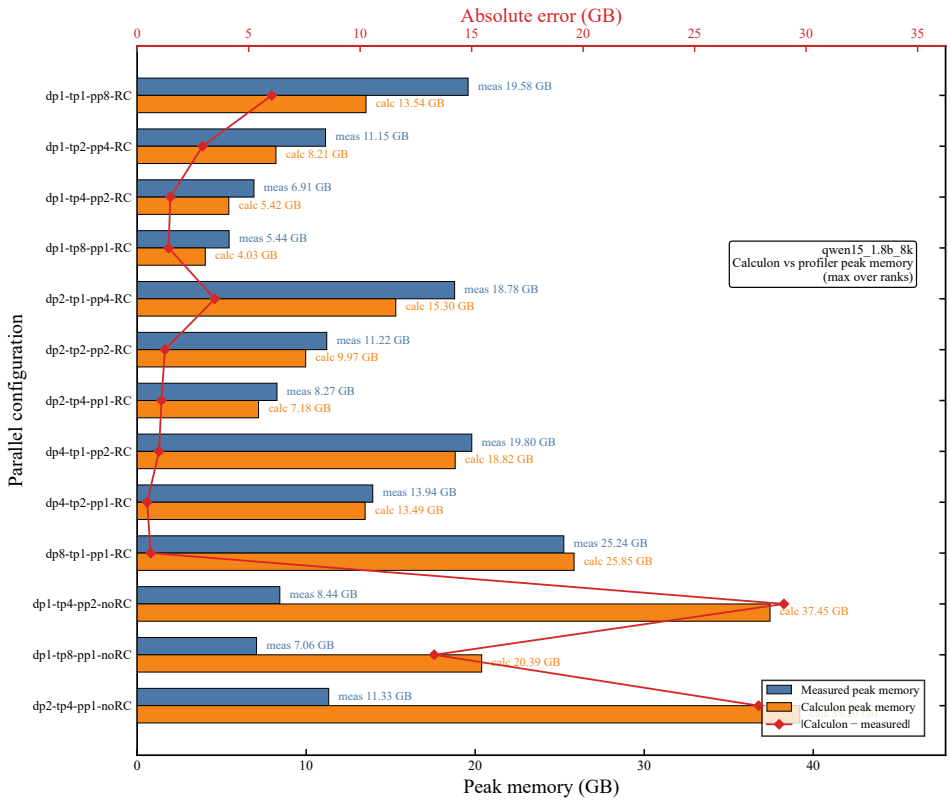


Figure 7: **Calculon peak-memory prediction on `qwen15_1.8b_8k`.** Measured maximum rank-level peak memory and Calculon predictions for 13 configurations. The right axis shows absolute error.

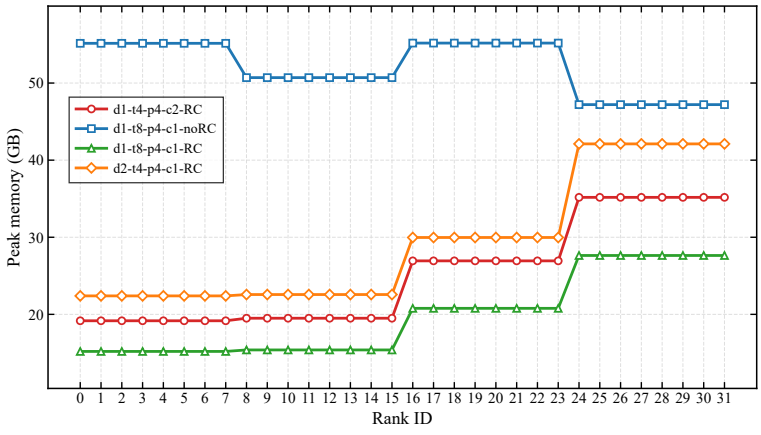
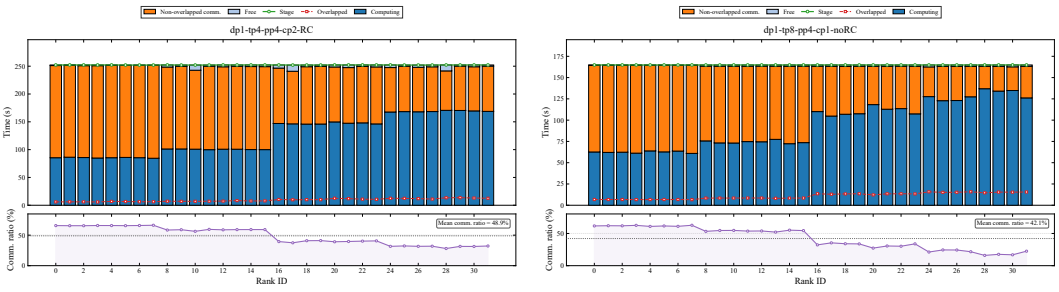


Figure 8: **Rank-level peak memory across configurations.** Peak memory by rank for `qwen25_32b_32k` under four parallel configurations; each line denotes one configuration.



(a) `dp1-tp4-pp4-cp2-RC`. (b) `dp1-tp8-pp4-cp1-noRC`.

(c) `dp1-tp8-pp4-cp1-RC`. (d) `dp2-tp4-pp4-cp1-RC`.

Figure 9: **Rank-level iteration-time breakdowns.** Each panel decomposes `qwen25_32b_32k` into computation, non-overlapped communication, free, stage, and overlapped time and reports the communication ratio for one configuration.

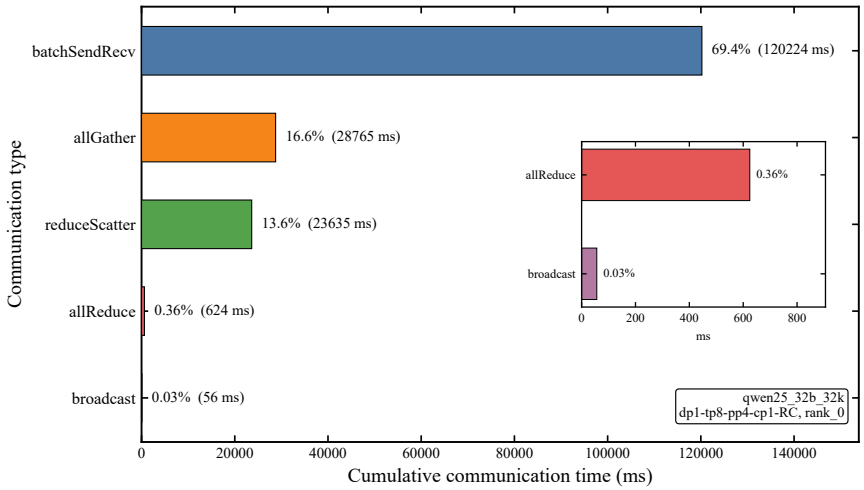


Figure 10: **Communication time breakdown.** Cumulative time by communication primitive for rank 0 of `qwen25_32b_32k` under `dp1-tp8-pp4-cp1-RC`. The inset enlarges primitives with small shares.

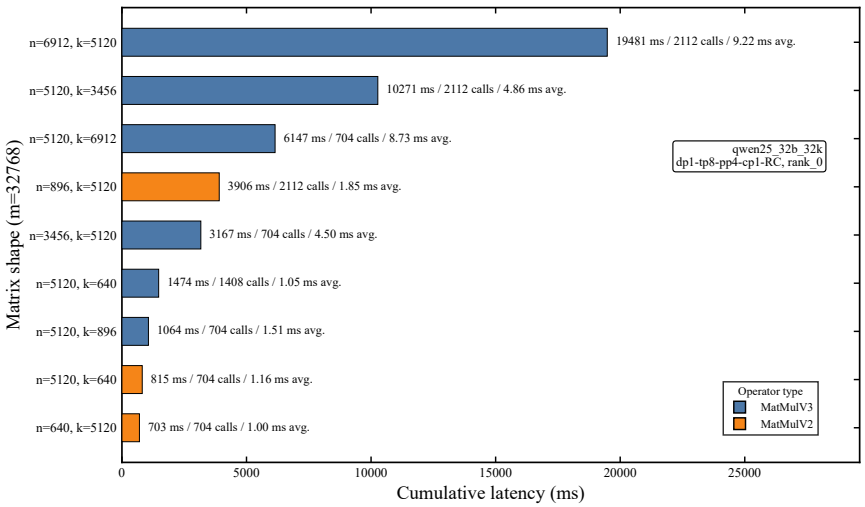
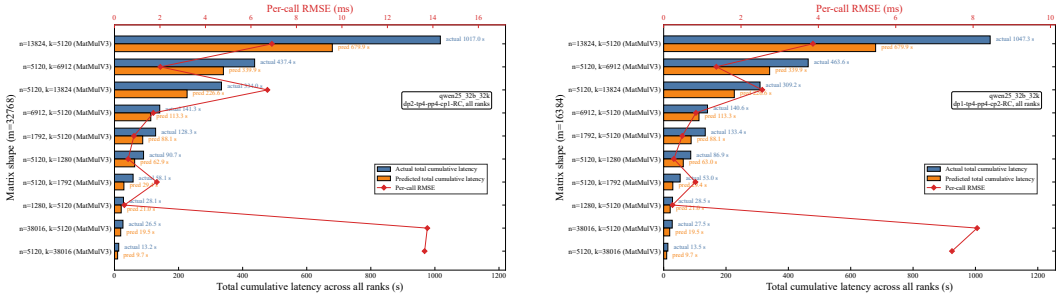
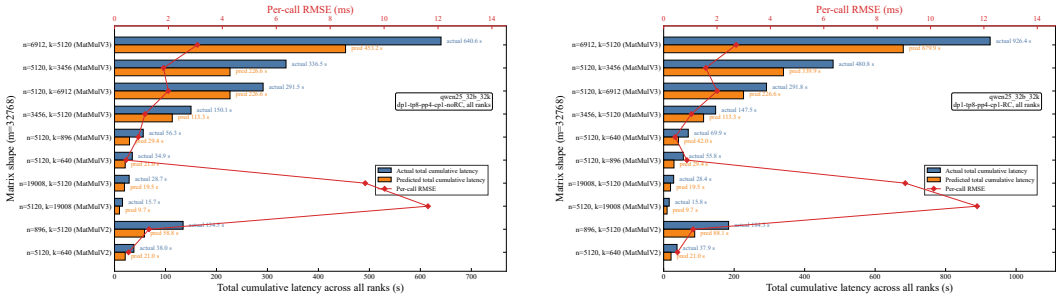


Figure 11: **Top matrix-operator shapes by cumulative latency.** Matrix operators on rank 0 are grouped by (n, k) with $m = 32768$ for `qwen25_32b_32k` under `dp1-tp8-pp4-cp1-RC`. Each bar reports cumulative latency, call count, and mean latency.



(a) `dp2-tp4-pp4-cp1-RC`.

(b) `dp1-tp4-pp4-cp2-RC`.



(c) `dp1-tp8-pp4-cp1-noRC`.

(d) `dp1-tp8-pp4-cp1-RC`.

Figure 12: Matrix-latency predictions for `qwen25_32b_32k` under four parallel configurations. Bars compare measured and predicted cumulative latency across ranks for dominant shapes; the red curve shows per-call RMSE for each shape.

E Cross-Platform Supplement: NVIDIA V100/Megatron-LM

The primary LM-SimBench collection is an Ascend 910B3/MindSpeed-LLM instance, and its absolute measurements need not carry over to another accelerator. This appendix documents a matched supplement collected on one eight-device NVIDIA V100 node (32 GB per device) with Megatron-LM, profiled against the 8×Ascend 910B3 (64 GB) MindSpeed-LLM reference. The purpose is to test whether the data schema and the system/component/strategy task definitions can

Table 16: Cross-platform protocol validation on V100/Megatron-LM versus Ascend 910B3/MindSpeed-LLM. Correlations compare same-named configurations with complete profiles on both stacks ($n = 15$ and 11). "Direction" records whether recomputation changes time and memory in the same direction. *Only three paired configurations. Per-configuration tables are in Appendix E.

Coverage	Paired complete configurations	15	11
Strategy	Iteration rank ρ / τ_b, pooled	0.718 / 0.562	0.664 / 0.491
	Iteration rank ρ, noRC / RC	0.714 / 0.683	1.000* / 0.690
System	Peak-memory rank ρ / τ_b	0.982 / 0.924	0.991 / 0.964
	Lowest-memory Top-3 overlap	3/3	3/3
	RC time/memory direction agreement	6 / 6	3 / 3
Component	Matrix median-latency ρ / τ_b	0.983 / 0.892	0.987 / 0.909
	Slowest matrix-problem Top-10 overlap	8/10	10/10
	Operator-category median ρ / JSD	0.503 / 0.436	0.531 / 0.356
	Communication-primitive median ρ / JSD	0.949 / 0.019	0.872 / 0.035
	Non-overlapped comm. share, GPU / NPU	17.0% / 28.5%	10.9% / 30.0%

be instantiated on a mainstream accelerator/runtime stack, and to quantify which measured signals survive a stack change and which require target-platform calibration. We deliberately do not claim that Ascend measurements can be reused numerically on GPUs.

E.1 Scope and Alignment

Both stacks profile `qwen3_0.6b_4k` and `qwen3_1.7b_4k` at a 4096-token sequence length with micro/global batch sizes of 1/32, eight devices, context parallelism (CP) of 1, a distributed optimizer, and a fixed seed, using the same 18 DP/TP/PP/recomputation candidate names per model. The profiled iteration is the sixth step after five warm-up steps on both stacks. This 18-candidate grid is the CP=1 subset of the 38 configurations released for each of these two model settings; the remaining CP>1 configurations exist only on the NPU side.

The comparison is a protocol-transfer test rather than a hardware-only microbenchmark, because several platform-native choices are not controlled. Table 15 separates matched factors from stack differences. Precision (FP16 on V100, BF16 on 910B3), fused-kernel availability, sequence-parallel activation, and the set of profiler-visible counters differ by design. Ascend-specific `AICore`/`MTE`/`cube` and `HCCS`/`RDMA`/`SDMA` fields have no GPU equivalent and are left empty rather than synthesized. Complete environment metadata, including driver, CUDA/CANN, PyTorch, collective-library, and framework revisions, is released with the artifact.

Table 15: Matched factors and platform-native differences in the cross-platform supplement. Entries marked as stack-specific are part of the compared hardware/software stacks rather than nuisance parameters of a single-variable experiment.

Aspect	**Setting**
Matched	Model architecture, layer counts, hidden size, attention heads, and grouped-query attention; sequence length 4096; micro/global batch 1/32; eight devices; CP=1; candidate configuration names; optimizer hyperparameters and seed; profiled iteration index.
Stack-specific	Data type (FP16 on V100 vs. BF16 on 910B3); fused kernels (FlashAttention and fused RoPE/SwiGLU/RMSNorm disabled in the GPU sweep, enabled on the Ascend stack); sequence parallelism (enabled on the NPU side); profiler-visible hardware counters and communication-link fields; input data path.

Table 17: Candidates that are not complete on both stacks. Incomplete profiles are recorded rather than silently dropped, and are excluded from paired statistics.

Model	Configuration	GPU status	NPU status
Qwen3-0.6B	dp2-tp1-pp4-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-0.6B	dp4-tp1-pp2-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-0.6B	dp8-tp1-pp1-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-1.7B	dp1-tp2-pp4-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-1.7B	dp2-tp1-pp4-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-1.7B	dp2-tp2-pp2-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-1.7B	dp4-tp1-pp2-cp1-RC	Complete	Missing
Qwen3-1.7B	dp4-tp1-pp2-cp1-noRC	Missing/incomplete (empty directory)	Missing
Qwen3-1.7B	dp4-tp2-pp1-cp1-noRC	Missing/incomplete (empty directory)	Complete
Qwen3-1.7B	dp8-tp1-pp1-cp1-noRC	Missing/incomplete (empty directory)	Complete

Table 18: Iteration-time rank agreement within recomputation strata. "Top-1 same" records whether both stacks select the same fastest configuration. The qwen3_1.7b_4k noRC stratum contains only three paired configurations, and its pooled row mixes the two strata.

Model	Stratum	n	ρ	τ_b	Top-3	Top-1 same
Qwen3-0.6B	all	15	0.718	0.562	2/3	yes
Qwen3-0.6B	noRC	6	0.714	0.600	2/3	yes
Qwen3-0.6B	RC	9	0.683	0.556	2/3	yes
Qwen3-1.7B	all	11	0.664	0.491	2/3	**no**
Qwen3-1.7B	noRC	3	1.000	1.000	3/3	yes
Qwen3-1.7B	RC	8	0.690	0.571	2/3	yes

E.2 Coverage and Incomplete Profiles

Complete GPU profiles are available for 15 of 18 candidates for qwen3_0.6b_4k and 12 of 18 for qwen3_1.7b_4k; the NPU side is complete for 18 and 16 candidates, respectively. Paired statistics therefore use 15 and 11 same-named configurations with complete profiles on both stacks, 26 pairs in total. Table 17 lists every candidate that is not complete on both stacks. Incomplete GPU directories are retained in the manifest as missing_or_incomplete with an empty_profile_directory reason; because no failure log accompanies them, we do not attribute them to out-of-memory.

E.3 Strategy-Level Comparison

Pooled over both recomputation settings, iteration-time rank agreement between the two stacks is moderate ($\rho = 0.718$ and 0.664, $\tau_b = 0.562$ and 0.491 for the two models). Pooling RC and noRC is itself a confound, since enabling recomputation changes the time and memory ordering. Table 18 therefore repeats the comparison within strata. Within every model×recomputation stratum the two stacks select the same Top-1 configuration; the pooled disagreement for qwen3_1.7b_4k reflects a cross-stratum ordering flip rather than a failure to identify the best candidate within a comparable set. Stratum sample sizes are small, so the corresponding coefficients should be read as descriptive.

Table 19 reports the per-configuration iteration times. A single NPU-to-GPU scale factor is not sufficient to transfer absolute iteration time between stacks: a leave-one-configuration-out diagnostic that fits one median scale factor reaches 22.4% and 22.9% sMAPE with $R^2 = -8.33$ and -4.74. This is a useful negative result, since it shows that the gap is not a constant hardware ratio.

E.4 Memory and Recomputation

Peak-memory ordering transfers strongly: rank correlation is $\rho = 0.982$ and 0.991 ($\tau_b = 0.924$ and 0.964), and the three lowest-memory configurations coincide on both stacks for both models.

Table 19: Per-configuration iteration time on both stacks, ordered by the GPU rank within each model. Ranks are computed over the paired configurations, with RC and noRC pooled.

Configuration	GPU (s)	NPU (s)	GPU/NPU	GPU rank	NPU rank
Qwen3-0.6B					
dp4-tp2-pp1-cp1-noRC	6.40	3.93	1.63	1	1
dp2-tp4-pp1-cp1-noRC	6.69	6.11	1.09	2	7
dp2-tp2-pp2-cp1-noRC	7.20	4.48	1.61	3	2
dp1-tp4-pp2-cp1-noRC	7.33	6.74	1.09	4	10
dp8-tp1-pp1-cp1-RC	7.53	4.69	1.61	5	3
dp1-tp2-pp4-cp1-noRC	8.28	5.10	1.62	6	4
dp4-tp2-pp1-cp1-RC	8.60	5.63	1.53	7	6
dp2-tp4-pp1-cp1-RC	8.93	9.48	0.94	8	12
dp4-tp1-pp2-cp1-RC	8.95	5.50	1.63	9	5
dp1-tp8-pp1-cp1-noRC	9.06	11.39	0.80	10	14
dp2-tp2-pp2-cp1-RC	9.46	6.27	1.51	11	9
dp1-tp4-pp2-cp1-RC	9.69	9.75	0.99	12	13
dp2-tp1-pp4-cp1-RC	10.25	6.25	1.64	13	8
dp1-tp2-pp4-cp1-RC	10.65	6.98	1.53	14	11
dp1-tp8-pp1-cp1-RC	12.86	18.36	0.70	15	15
Qwen3-1.7B					
dp2-tp4-pp1-cp1-noRC	8.85	6.42	1.38	1	2
dp8-tp1-pp1-cp1-RC	9.46	5.49	1.73	2	1
dp1-tp4-pp2-cp1-noRC	9.82	7.43	1.32	3	4
dp1-tp8-pp1-cp1-noRC	10.75	12.09	0.89	4	10
dp4-tp2-pp1-cp1-RC	11.20	7.04	1.59	5	3
dp2-tp4-pp1-cp1-RC	11.74	9.26	1.27	6	8
dp2-tp2-pp2-cp1-RC	12.34	7.83	1.58	7	6
dp1-tp4-pp2-cp1-RC	12.82	10.14	1.26	8	9
dp2-tp1-pp4-cp1-RC	13.28	7.46	1.78	9	5
dp1-tp2-pp4-cp1-RC	14.10	8.69	1.62	10	7
dp1-tp8-pp1-cp1-RC	14.28	16.78	0.85	11	11

Absolute memory levels do not coincide (median GPU/NPU ratio 0.77 and 0.70), which is why we compare ordering rather than absolute agreement. Table 20 lists the per-configuration peak memory.

Recomputation behaves consistently across stacks. Across six matched base configurations for `qwen3_0.6b_4k` and three for `qwen3_1.7b_4k`, enabling recomputation increases iteration time and decreases peak memory in the same direction in every pair (100% directional agreement). The median time overhead is 32.9% (GPU) versus 44.0% (NPU) for the smaller model and 32.6% versus 38.8% for the larger one; the median memory saving is 85.6% versus 77.2% and 77.5% versus 69.7%, respectively. The magnitudes differ by platform, but the trade-off direction does not.

E.5 Component-Level Comparison

Matrix operators are aligned by the hardware-neutral mathematical problem $(\mathrm{batch}, m, n, k, \mathrm{dtype})$ instead of platform kernel names. Each model has 27 shared problem types, and those types account for 72.8% and 76.7% of GPU matrix calls. Ranking the shared problems by median latency gives $\rho = 0.983$ and 0.987 ($\tau_b = 0.892$ and 0.909), with slowest-shape Top-10 overlap of $8/10$ and $10/10$. Relative matrix difficulty therefore transfers, whereas absolute speed differs by a median factor of 2.79 and 2.84 (Table 21).

Operator-category composition, in contrast, agrees only moderately (median $\rho = 0.503$ and 0.531; median Jensen–Shannon divergence 0.436 and 0.356), which is consistent with different fusion and memory-movement implementations. Communication primitives are far more consistent in their ordering (median $\rho = 0.949$ and 0.872; median JSD 0.019 and 0.035), and the dominant primitive matches in 86.7% and 72.7% of configurations. Table 22 reports the time-share vectors. The residual differences concentrate in platform-specific categories: on 910B3 the *other* and *shape_transform* categories absorb a large share of device time, and *attention* is attributed to a separate category only on the GPU stack.

Table 20: Peak memory at the configuration level (maximum over ranks) on both stacks, ordered by the GPU rank within each model.

Configuration	GPU (GiB)	NPU (GiB)	GPU/NPU	GPU rank	NPU rank
Qwen3-0.6B					
dp1-tp8-pp1-cp1-RC	1.47	2.18	0.67	1	1
dp1-tp4-pp2-cp1-RC	2.18	3.16	0.69	2	2
dp2-tp4-pp1-cp1-RC	2.37	3.25	0.73	3	3
dp1-tp2-pp4-cp1-RC	3.64	5.26	0.69	4	5
dp2-tp2-pp2-cp1-RC	3.67	5.09	0.72	5	4
dp4-tp2-pp1-cp1-RC	4.14	5.37	0.77	6	6
dp2-tp1-pp4-cp1-RC	6.44	8.91	0.72	7	7
dp4-tp1-pp2-cp1-RC	6.68	8.94	0.75	8	9
dp8-tp1-pp1-cp1-RC	7.70	9.63	0.80	9	10
dp1-tp8-pp1-cp1-noRC	8.55	8.92	0.96	10	8
dp1-tp4-pp2-cp1-noRC	14.99	13.23	1.13	11	11
dp2-tp4-pp1-cp1-noRC	15.41	14.25	1.08	12	12
dp1-tp2-pp4-cp1-noRC	28.11	22.97	1.22	13	14
dp2-tp2-pp2-cp1-noRC	28.13	22.74	1.24	14	13
dp4-tp2-pp1-cp1-noRC	29.03	24.91	1.17	15	15
Qwen3-1.7B					
dp1-tp8-pp1-cp1-RC	2.81	4.54	0.62	1	1
dp1-tp4-pp2-cp1-RC	3.61	5.72	0.63	2	2
dp2-tp4-pp1-cp1-RC	4.26	6.13	0.69	3	3
dp1-tp2-pp4-cp1-RC	5.38	8.34	0.64	4	5
dp2-tp2-pp2-cp1-RC	5.71	8.24	0.69	5	4
dp4-tp2-pp1-cp1-RC	7.16	9.31	0.77	6	6
dp2-tp1-pp4-cp1-RC	8.92	12.73	0.70	7	7
dp1-tp8-pp1-cp1-noRC	11.24	14.02	0.80	8	8
dp8-tp1-pp1-cp1-RC	12.96	15.66	0.83	9	9
dp1-tp4-pp2-cp1-noRC	18.10	18.85	0.96	10	10
dp2-tp4-pp1-cp1-noRC	18.96	20.20	0.94	11	11

Communication overlap is the clearest stack-dependent signal: the median non-overlapped communication share is 17.0% and 10.9% on V100 versus 28.5% and 30.0% on 910B3, and the per-configuration correlation of this share across stacks is only $\rho = 0.179$ and 0.364. How much communication is exposed therefore cannot be transferred, even though which primitive dominates largely can.

E.6 Platform Calibration of Calculon

The supplement also tests whether one simulator workflow can be calibrated to each stack rather than rewritten for it. We completed Calculon's Qwen3 model description and memory accounting (grouped-query attention, SwiGLU, RMSNorm, Q/K normalization, vocabulary, embedding and output projections, and tied embeddings) and added kernel launch latency, per-pipeline-stage execution, non-interleaved 1F1B activation residency, edge stages, and rank-level memory output. V100 and 910B3 each use one independent parameter set, calibrated for segmented effective compute efficiency, launch latency, interconnect communication, and fixed runtime memory; the same set is applied to both models and the full candidate grid. Table 23 reports the resulting accuracy. After calibration, iteration-time MAPE is 7.97%–11.22% and time-ranking ρ is 0.853–0.950; rank-level peak-memory MAPE is 12.55%–15.99% and configuration-level peak-memory ranking ρ is 0.986–0.994. Across the four model×recomputation settings, GPU measurement, NPU measurement, Calculon-GPU, and Calculon-NPU select the same Top-1 configuration (4/4). These are calibration-set fit statistics on the profiles used for parameter fitting; generalization to unseen configurations would require a held-out split.

Table 21: Representative shared matrix problems matched by $(\mathrm{batch}, m, n, k, \mathrm{dtype})$, with median per-call latency on each stack aggregated over all paired configurations and all eight ranks. The $\mathrm{batch}:mnk$ key is shown as reported by the profiler.

Model	Matrix problem	GPU (µs)	NPU (µs)	GPU/NPU
Qwen3-0.6B	`b8: 4096x4096 @ 4096x128`	654.5	388.4	1.69
Qwen3-0.6B	`b8: 4096x128 @ 128x4096`	693.3	234.0	2.96
Qwen3-0.6B	`b16: 4096x4096 @ 4096x128`	1187.3	697.2	1.70
Qwen3-0.6B	`b4: 4096x4096 @ 4096x128`	278.4	133.5	2.09
Qwen3-0.6B	`b4: 4096x128 @ 128x4096`	352.4	97.0	3.63
Qwen3-0.6B	`b16: 4096x128 @ 128x4096`	1371.0	484.1	2.83
Qwen3-0.6B	`b1: 4096x1024 @ 1024x1024`	125.5	41.9	2.99
Qwen3-0.6B	`b1: 4096x1024 @ 1024x3072`	320.2	111.4	2.87
Qwen3-0.6B	`b2: 4096x4096 @ 4096x128`	177.6	53.7	3.31
Qwen3-0.6B	`b1: 4096x1024 @ 1024x1536`	168.8	63.7	2.65
Qwen3-1.7B	`b1: 4096x2048 @ 2048x3072`	603.4	212.4	2.84
Qwen3-1.7B	`b1: 4096x2048 @ 2048x6144`	1191.1	402.7	2.96
Qwen3-1.7B	`b1: 4096x3072 @ 3072x2048`	617.2	208.8	2.96
Qwen3-1.7B	`b8: 4096x4096 @ 4096x128`	710.1	387.5	1.83
Qwen3-1.7B	`b1: 4096x2048 @ 2048x1536`	309.2	118.0	2.62
Qwen3-1.7B	`b1: 4096x1536 @ 1536x2048`	374.8	111.9	3.35
Qwen3-1.7B	`b4: 4096x4096 @ 4096x128`	284.5	134.9	2.11
Qwen3-1.7B	`b1: 4096x2048 @ 2048x12288`	2395.5	795.2	3.01
Qwen3-1.7B	`b1: 4096x6144 @ 6144x2048`	1111.7	412.1	2.70
Qwen3-1.7B	`b1: 4096x2048 @ 2048x2048`	470.8	147.4	3.19

Table 22: Operator-category and communication-primitive time shares, normalized within each model and stack. "op" and "comm" mark the two component kinds. Category definitions follow the profiler's own classification on each stack, so identical labels need not denote identical kernel sets.

Component	0.6B GPU	0.6B NPU	1.7B GPU	1.7B NPU
op: matmul	28.6%	14.9%	38.9%	22.0%
op: memory_movement	20.7%	4.2%	16.2%	3.3%
op: elementwise	18.6%	17.2%	19.1%	19.9%
op: attention	14.7%	0.0%	11.2%	0.0%
op: memory_init	13.0%	0.2%	10.1%	0.2%
op: other	2.3%	40.3%	2.2%	34.1%
op: shape_transform	1.6%	22.8%	1.4%	20.0%
op: activation	0.5%	0.2%	0.7%	0.3%
op: embedding	0.0%	0.0%	0.0%	0.0%
op: optimizer	0.0%	0.1%	0.1%	0.1%
op: indexing_or_embedding	0.0%	0.0%	0.0%	0.0%
op: normalization	0.0%	0.0%	0.0%	0.0%
comm: all_reduce	54.4%	82.9%	54.3%	84.1%
comm: point_to_point	42.3%	15.3%	41.3%	14.0%
comm: broadcast	1.8%	0.9%	1.8%	0.7%
comm: reduce_scatter	1.5%	0.5%	2.7%	0.6%
comm: all_gather	0.0%	0.4%	0.0%	0.5%

Table 23: Calculon accuracy after platform-specific calibration. Entries are computed on the configurations used for fitting and are not held-out estimates.

Model	Platform	Time MAPE (%)	Time ρ	Rank-mem. MAPE (%)	Config-mem. ρ
Qwen3-0.6B	V100	10.28	0.861	15.99	0.993
Qwen3-0.6B	910B3	10.19	0.950	13.20	0.986
Qwen3-1.7B	V100	11.22	0.853	12.56	0.993
Qwen3-1.7B	910B3	7.97	0.935	12.55	0.994

E.7 Comparison with a Second Simulator Family

To test whether the benchmark separates simulators with different modeling choices, we add vTrain Bang et al. (2024) to the `qwen15_1.8b_8k` case study. Under the classification of Appendix B, Calculon is analytical/mechanistic ($A/An/Mech$) while vTrain is graph/profiling/trace-based ($G/Pr/Trace$). Ten configurations are used for vTrain calibration and ten disjoint configurations only for testing; the adapter consumes LM-SimBench profiles in place of vTrain's native profiler input. Table 24 reports the comparison on the six test configurations for which both simulators return valid predictions.

Table 24: Simulator-family comparison on `qwen15_1.8b_8k`, supplementing Table 2. Iteration-time accuracy, ranking, and simulation speed are evaluated on the six test configurations for which both simulators return valid predictions; this subset is smaller than the 13 valid configurations used for Calculon in Table 2, so ranking coefficients are not directly comparable across the two tables. N/A denotes a signal that a simulator does not emit, not a surrogate score. Simulation time measures simulator software time, not predicted training time.

Level	Task and metric	Calculon	vTrain
System	Iteration time, MAE / sMAPE / R^2	1.059 s / 8.88% / 0.630	**0.207 s / 1.86% / 0.990**
	Peak memory, MAE / sMAPE / R^2	6.92 GB / 39.34% / -3.087	N/A
	Memory feasibility, agreement	65.0% (13/20)	N/A
Component	Matrix-operator latency, MAE / sMAPE / R^2	0.109 ms / 43.34% / 0.817	N/A
	Stage computation, MAE / sMAPE / R^2	N/A	716.10 ms / 11.60% / 0.462
	Recomputation stages, MAE / sMAPE / R^2	N/A	**37.99 ms / 2.75% / 0.801**
	Non-overlapped communication, MAE / sMAPE / R^2	N/A	8.710 s / 136.20% / -0.643
Strategy	Iteration-time ranking, Spearman ρ	0.943	**1.000**
	Iteration-time ranking, Kendall τ	0.867	**1.000**
	Top-1 hit / selection regret	1 / 0%	1 / 0%
	Top-3 set overlap	3/3	3/3
Efficiency	Median wall-clock per configuration	**1.78 ms**	25.46 ms

The two simulators expose a clear accuracy/cost trade-off. vTrain has lower absolute iteration-time error on five of six configurations, with a 7.02-percentage-point lower sMAPE, while Calculon is $14.3\times$ faster to execute. The component results localize the residual error: vTrain models recomputation stages accurately but does not reproduce the overlap observed in the profiler, consistent with the stack-dependent overlap reported in Section E.5. On the same eight test configurations, ten profiles spanning TP/DP/PP give 2.08% sMAPE versus 11.77% for six TP-endpoint profiles, a 9.69-percentage-point difference, which illustrates how calibration coverage affects reported accuracy.

E.8 Interpretation and Boundaries

The supplement supports the following reading. The schema, the system/component/strategy task definitions, and the simulator calibration workflow all transfer to a CUDA/Megatron-LM stack. Peak-memory ordering, the direction of the recomputation time–memory trade-off, and relative matrix-problem difficulty are preserved, so the diagnostic signals the benchmark relies on remain meaningful on both stacks. In contrast, absolute iteration time, absolute memory, operator-category composition, and the exposed fraction of communication do not transfer, and a single global scale factor is demonstrably insufficient.

We do not claim the following. The supplement covers two Qwen3 workloads on one 8-device V100 node with CP=1 and does not establish behavior for larger models, multi-node topologies, or newer devices. Precision, fusion, and sequence-parallel settings differ between stacks, so the comparison should not be read as a hardware-only speedup measurement. Incomplete GPU profiles are not interpreted as out-of-memory runs. Finally, real GPU profiles provide labels and calibration targets; they do not by themselves establish GPU prediction accuracy for any simulator other than the calibrated instance reported in Section E.6.