PROMA: PROGRESSIVE TRAINING EFFICIENCY OPTIMIZATION VIA MULTI-AGENT PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimizing large-scale training efficiency remains heavily reliant on human expertise. Existing automated methods, whether relying on profiling, prior knowledge, or iterative search, fail to remove this burden. They still incur the full tuning cost at every new setting, whenever the model, cluster, or scale changes, making large-scale training optimization prohibitively expensive. We propose PROMA, a PRogressive training efficiency Optimization framework via Multi-Agent planning. PROMA begins with a small-scale pilot study, where a team of agents identifies the experience that remains valid across settings. In the target large-scale setting, PROMA guides an iterative search with this experience, refining proposals as training results accumulate. Two estimators filter out infeasible and clearly inferior proposals. Across different model architectures, training scales from 8 to 256 GPUs, and multiple hardware platforms, PROMA improves MFU by 25.9% over SOTA agentic search methods and 17.4% over expert-designed strategies, while reducing tuning GPU hours by 34.5% on average. The source code is available at `https://anonymous.4open.science/r/PROMA`.

1 INTRODUCTION

Model scaling improves performance across downstream tasks, and large dense and mixture-of-experts (MoE) models now contain hundreds of billions of parameters (Fedus et al., 2022; Jiang et al., 2024). A single device cannot store the full model together with its optimizer states, so training requires clusters of hundreds to thousands of accelerators (Narayanan et al., 2021b). At this scale, **hardware utilization is highly sensitive to the training configuration**. Small changes in parallelism, batch size, or activation checkpointing can shift the bottleneck between compute, memory, and communication (Narayanan et al., 2021b). A poorly chosen configuration therefore leaves much of the cluster idle, and default or manually tuned settings rarely close the gap. Automating the search for high-utilization configurations is therefore a central practical problem.

However, the search itself is expensive. The space of valid configurations is combinatorially large, and each trial consumes GPU hours that cannot be recovered. Under a finite budget, the goal is to reach a high MFU before the budget is exhausted. Existing automated search pipelines optimize for a single fixed setting and cannot reuse knowledge when the model architecture or cluster layout changes (Chen et al., 2026; Ding et al., 2026). Cost-model-based methods fit a scenario-specific model that becomes invalid at a new scale (Lin et al., 2025). LLM and multi-agent methods search iteratively within one setting and treat every new configuration as an independent task (Chen et al., 2026; Ding et al., 2026). In both cases, every scale transition restarts from scratch, incurring substantial computational overhead for cross-scenario adaptation.

We observe that the configuration search space has structural properties that remain stable across cluster sizes. These properties fall into three categories. First, *infeasible regions*. Many failures observed at small scale persist at larger scales, such as unsupported kernels or communication patterns. Detecting it at small scale avoids repeating the same failure at larger scales. Second, *high-priority search axes*. Some dimensions, such as activation checkpointing or expert parallelism, influence MFU far more than others, and their elative importance follows from the model architecture and the hardware properties, not from the cluster size. Third, *estimator bias*. A memory or throughput estimator that over-predicts on a certain layout will do so at any scale, because the bias comes from the analytical model, not from the number of accelerators. These properties are

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.

determined by the model and the hardware, not by the scale. Measuring them at small scale requires far fewer GPU hours than discovering them at the target scale, and avoids repeating the same expensive trials.

We present PROMA, **a progressive multi-agent framework** that exploits this observation. A pilot study runs short adaptive trials at reduced scale to distill the scale-invariant properties, and a target-scale search uses them to constrain candidate generation. Only promising candidates reach real training, and the measured MFU feeds back to update the proposal strategy.

In summary, our contributions are as follows:

- We propose **PROMA**, a multi-agent framework for progressively optimizing distributed training efficiency. PROMA orchestrates a team of agents that plan, propose, validate, and diagnose across iterative rounds, refining the search toward the target setting.
- We introduce a **pilot study** and experience transfer mechanism. PROMA extracts properties from small-scale pilots that remain valid across settings, and uses them to prune the target-scale search space rather than re-exploring from scratch.
- We design two estimators, **MemEst** and **MfuEst**, with substantially higher accuracy than existing analytical tools. They filter out infeasible and clearly inferior proposals before real training, reducing wasted trials and mitigating the effect of LLM hallucination.
- We conduct extensive experiments across model architectures, training scales (8 to 256 GPUs), and hardware platforms. PROMA improves MFU by **25.9%** over SOTA agentic search methods and **17.4%** over expert-designed strategies, while reducing GPU hours by **34.5%** on average.

2 RELATED WORK

Cost-Model-Based Auto-Parallelism. Automatic parallelization frameworks search for optimized partitioning strategies by estimating computation, memory, and communication overheads (Zheng et al., 2022; Miao et al., 2022; Liu et al., 2024). AutoHAAP (Wang et al., 2026) extends this paradigm to heterogeneous hardware with memory-aware initialization and load-balancing estimation. These methods build scenario-specific cost models to evaluate candidate strategies. Any change to model architecture, GPU count, or hardware platform invalidates the cost model, forcing recalibration and a fresh search.

LLM and Multi-Agent-Based Configuration Optimization. LLM-driven tuning replaces handcrafted cost models with agent workflows and profiling tools. PROMPTS (Ding et al., 2026) produces candidate configurations via knowledge retrieval, OptiCo (Chen et al., 2026) iteratively improves configurations with multi-agent search, and ASAP (Mankowitz et al., 2016) integrates profiling and roofline analysis into agent optimization loops. However, these methods treat distinct training setups as independent tasks and preserve no transferable tuning knowledge across settings.

Underlying Estimators. The performance of these search strategies is limited by the accuracy of the estimator used to predict memory and runtime. Existing analytical estimators such as llm-analysis (Li, 2023) and Calculon (Isaev et al., 2023) assume ideal FLOPs or coarse memory accounting, and do not model MoE all-to-all, MLA, or hardware-specific stream behavior.

Existing methods optimize for a fixed setting and cannot reuse knowledge across settings. PROMA addresses this gap via progressive experience accumulation. Cost-model-based methods rely on a fitted scenario model to guide search and select optimal configurations. In contrast, PROMA uses lightweight estimators to identify clearly infeasible or inferior candidates, and final decisions come from real training. Agent-based methods generate configurations for new setups via iterative search; PROMA instead validates candidates through small-scale pilot runs and extracts scale-invariant experience, rather than re-searching from scratch.

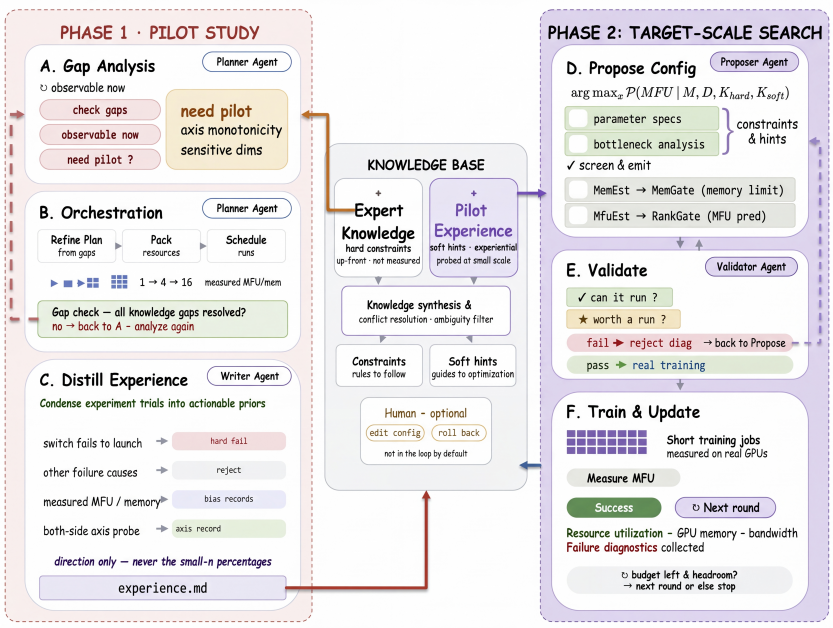


Figure 1: Overview of PROMA.

3 METHODOLOGY

3.1 PROMA FRAMEWORK

Improving the efficiency of large-scale distributed training requires finding a configuration that achieves high MFU. With a limited search budget, exhaustively evaluating all possible configurations is infeasible, and every trial consumes expensive GPU hours. PROMA addresses this with a progressive design in which cheap signals constrain and guide the search before expensive trials.

Figure 1 gives an overview of PROMA. PROMA has two stages. First, a pilot study runs short trials at small scales, choosing each scale adaptively according to what needs to be verified, and distills the results into transferable knowledge. Second, a search stage applies this knowledge at the target scale.

The search iterates between proposal and measurement. The proposal agent (A_{p}) generates configurations that balance prior knowledge with measured evidence, conditioned on model architecture and device specification. It uses two estimators to keep the proposal focused on feasible and promising candidates. The memory estimator (MemEst) removes configurations that would exceed the memory limit, and the MFU estimator (MfuEst) drops those whose predicted MFU is clearly worse than the rest. The validator agent (A_{v}) acts as a filter between proposal and measurement, rejecting any configuration that cannot run or is unlikely to improve on the best measured MFU. A short training job measures the true MFU, which feeds back into the next proposal. Successful runs return MFU feedback, while failed runs return diagnosticfeedback through an error analysis module.

Optionally, a human can intervene at any point during the search. The human can edit the configuration for the next training run, roll back to any earlier round, and either let the proposal agent regenerate a new configuration or specify one manually. It does not change how knowledge is acquired or applied. It only lets a human inject judgment when the search stalls or when a human expert wants to adjust the search direction.

3.2 PILOT STUDY

The pilot study explores scale-invariant properties of the search space, including switch feasibility, switch preference, estimator bias, axis monotonicity, and search-sensitive dimensions. It takes external expert knowledge as an initial prior and refines it through small-scale trials. The planner agent (A_{plan}) identifies coverage gaps and schedules experiments, and the writer agent (A_{w}) turns

raw measurements into transferable priors. For each gap, A_{w} records which configurations failed and why, whether the estimators over- or under-predicted on the tested layout class, where an integer axis changes behavior, and which dimensions deserve more trials at the target scale. Each record is encoded as a hard constraint or a soft hint so that the target-scale search can consume it directly.

The pilot study checks whether each property has enough evidence. A property with no measurement or with ambiguous evidence is treated as a gap. It prioritizes gaps that are most likely to change the search behavior at the target scale, and plans measurements to fill them. Different properties become observable at different scales. A property such as switch feasibility can often be tested on a single node, while communication overlap only shows its effect across multiple nodes. The pilot study therefore starts from the smallest scale that can validate a given property and increases the scale only when the current one cannot, with larger models raising the starting point.

The resulting knowledge enters the search as hard constraints enforced by A_{v} and soft hints that guide A_{p}. Incorrect entries are corrected by the measured MFU or filtered out by A_{v} and the estimators, so the pilot study reduces the cost of finding good regions.

3.3 Memory and MFU Estimators

PROMA uses two estimators to evaluate proposals. MemEst predicts peak GPU memory and rejects configurations that would run out of memory. MfuEst predicts throughput and rejects configurations with low MFU. Both compute memory and MFU from hardware and architecture parameters rather than fitting a single coefficient per setting, so the estimates remain valid when the configuration changes. Error analysis is in Section 4.4, Appendix C, and Appendix D.

MemEst predicts peak GPU memory without profiling the training timeline, since OOM is triggered by instantaneous peaks. For a configuration θ, peak memory decomposes as

$$M_{\text{total}}(\theta) = M_{\text{param}} + M_{\text{opt}} + M_{\text{act}} + M_{\text{runtime}}, \tag{1}$$

where M_{param} is weight memory, M_{opt} is optimizer-state memory, M_{act} is activation memory, and M_{runtime} is runtime overhead. The first two terms shard across GPUs according to the distributed optimizer, and the last covers CUDA context and NCCL scratch. These follow established models (Appendix A.1 and Appendix A.4).

Activation memory M_{act} dominates estimation error. Prior estimators treat it as a single term, but tensors within a layer have different lifetimes. We decompose by lifetime:

$$M_{\text{act},s} = \underbrace{u \cdot S \cdot L_s \cdot I_s}_{\text{saved}} + \underbrace{u \cdot W}_{\text{work}} + \underbrace{q \cdot X \cdot L_s}_{\text{width}}, \tag{2}$$

where $u = \text{seq}_{\text{local}} \times \text{mbs} \times h$ is the per-layer token count, $\text{seq}_{\text{local}}$ is the local sequence length, mbs is the micro-batch size, and h is the hidden size. L_s is the number of layers of type s, and I_s is the number of in-flight micro-batches on those layers. S, W, and X are dimensionless coefficients, and q is the per-token width scale in the same units as u. Saved tensors are retained for backward and scale with both L_s and I_s. Work tensors are layer-local and counted once. Width tensors track tensor-core workspace and scale with L_s but not I_s. Total activation memory is the sum over layer types,

$$M_{\text{act}} = \sum_s M_{\text{act},s}. \tag{3}$$

The coefficients depend on layer type and are detailed in Appendix A.2.

We further account for the asymmetric MoE all-to-all, where send buffers carry the full top-k volume while receive buffers scale with $1/\text{EP}$, and separate MLA and DeepEP persistent buffers as their own term (Appendix A.2). Checkpointing shifts tensors from saved to work without changing the total. Selective recomputation follows module-level rules derived in Appendix A.3.

MfuEst computes MFU from theoretical FLOPs and predicted step time:

$$\widehat{\text{MFU}}(\theta) = \frac{\text{FLOPs}_{\text{theory}}(\theta)}{t_{\text{step}}(\theta) \cdot N \cdot P_{\text{peak}}}, \tag{4}$$

where N is the number of GPUs, P_{peak} is the per-GPU peak throughput, and $\text{FLOPs}_{\text{theory}}$ counts forward, backward, and weight-gradient passes of one step. Step time decomposes into the forward

pass, recomputation, and step-level overheads:

$$t_{\text{step}} = n_{\text{mb}} \left(3t_{\text{fwd}} + t_{\text{recompute}}\right) + t_{\text{overhead}}, \tag{5}$$

where n_{mb} is the number of microbatches, t_{fwd} is the forward time of one microbatch, $t_{\text{recompute}}$ is the recomputation time under activation checkpointing, and t_{overhead} includes pipeline fill (Narayanan et al., 2021a), data-parallel communication, and host-side latency. The factor 3 reflects one forward and two backward passes under standard backpropagation. Checkpointing and grouped expert GEMMs change the backward cost, and their effect is captured by $t_{\text{recompute}}$. Forward time sums over layers on the critical pipeline stage as

$$t_{\text{fwd}} = \sum_{\ell} \left(t^{(\ell)}_{\text{comp}} + t^{(\ell)}_{\text{exposed}}\right), \tag{6}$$

where $t^{(\ell)}_{\text{comp}}$ is the compute time and $t^{(\ell)}_{\text{exposed}}$ is the exposed communication time of layer ℓ. Exposed communication is the part that cannot be hidden behind compute. It is counted once per layer. Overlap across layers and pipeline stages is already reflected in this exposed portion.

Four choices distinguish MfuEst from prior estimators. Compute time follows a roofline model (Williams et al., 2009), but we derive the compute limit from tensor-core occupancy instead of a model-level efficiency constant, so the estimate responds to each GEMM individually. This matters for MoE experts with varying per-rank token counts. Communication time follows the α-β model (Thakur et al., 2005) with a separate α and β per collective type, since they differ in message size and peer topology (Appendix B.1). Overlap is determined from the platform's stream configuration rather than a global factor (Appendix B.2). For MoE layers, we add two terms that dense-only estimators miss, the asymmetric all-to-all and the reduced SM count on platforms that reserve capacity for communication (Appendix B.3).

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We conduct experiments on three types of GPU, denoted as GPU-1, GPU-2, and GPU-3, with 64 GB, 80 GB, and 96 GB memory per device, respectively. The experiments span cluster sizes from 8 to 256 GPUs, covering dense models from 7B to 70B and MoE models from 6B to 236B. We use a patched Megatron-LM (Shoeybi et al., 2019) as the training framework, and compare against expert-tuned configurations and OptiCo (Chen et al., 2026). PROMA and the OptiCo baseline run on the same patched Megatron-LM version. Both methods search over the same candidate space. All LLM based agents use DeepSeek-V4-Flash for inference (DeepSeek-AI, 2026).

4.2 MAIN RESULTS

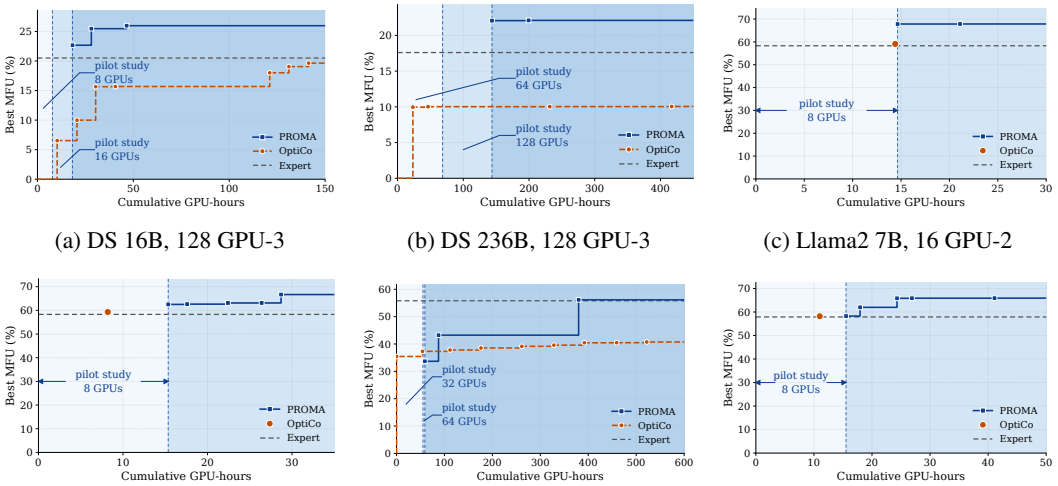
PROMA reaches a higher MFU than both baselines on every configuration, as reported in Table 1. PROMA improves over OptiCo by 25.9% in MFU (6.3 pp) and over the expert-tuned configuration by 17.4% (4.7 pp). The gain is larger on MoE models, where the search space is more sensitive to parallel layout. PROMA also outperforms both baselines on every platform and architecture, demonstrating generalization across hardware and model types. The GPU hours reported for PROMA in Table 1 include the pilot study and platform calibration.

In terms of search cost, PROMA spends 56.7 GPU-hours on average, compared with 86.5 GPU-hours for OptiCo, saving 29.8 GPU-hours. OptiCo underperforms the expert baseline on large-scale configurations because its search space grows combinatorially with cluster size, whereas PROMA transfers scale invariant constraints from experiments at small scale to prune that search space before the first trial. The two methods converge to different MFU, so these costs are not directly comparable. A method that reaches a higher MFU may spend more time. The controlled comparison under a fixed GPU-hour budget is shown in Figure 2.

Figure 2 plots the search progress in terms of best MFU against cumulative GPU-hours. At the same GPU-hour budget, PROMA has found a higher MFU than OptiCo. On the large models, PROMA reaches a near-final MFU within the first few GPU-hours after the pilot study, while OptiCo improves slowly. For example, in Figure 2(b), PROMA jumps to 22.1% MFU at around 200 GPU-hours,

Table 1: MFU (%) and search cost across models and cluster sizes. †OptiCo numbers are from Chen et al. (2026), and all others are from our experiments. DS = DeepSeek-V2. The GPU hours for PROMA include the pilot study. Values in parentheses for OptiCo and PROMA show the relative change over the Expert baseline. Values in parentheses for PROMA GPU-h show the absolute change in GPU hours over OptiCo. Red indicates an increase and green indicates a decrease.

Model	Platform	GPUs	Expert	OptiCo		PROMA	
				MFU (%)	GPU-h	MFU (%)	GPU-h
DS 16B	GPU-1	256	29.2	29.3 (+0.3%)	59.7	**32.3** (+10.6%)	25.8 (-33.9)
	GPU-1	128	40.2	29.9 (-25.6%)	10.3	**40.9** (+1.7%)	56.8 (+46.5)
	GPU-3	128	20.5	19.6 (-4.4%)	141.4	**26.0** (+26.8%)	46.5 (-94.9)
	GPU-2	16	20.1	20.4 (+1.5%)	24.0	**27.7** (+37.8%)	12.8 (-11.2)
DS 236B	GPU-3	128	17.6	10.1 (-42.6%)	417.0	**22.1** (+25.6%)	199.4 (-217.6)
	GPU-3	64	19.1	18.8 (-1.6%)	150.8	**20.8** (+8.9%)	24.7 (-126.1)
Llama2 7B†	GPU-2	16	58.3	59.1 (+1.4%)	14.4	**67.8** (+16.3%)	21.1 (+6.7)
	GPU-2	8	67.5	67.7 (+0.3%)	27.8	**68.4** (+1.3%)	14.6 (-13.2)
Llama3 8B†	GPU-2	16	58.3	59.3 (+1.7%)	8.2	**66.7** (+14.4%)	28.7 (+20.5)
	GPU-2	8	67.9	68.2 (+0.4%)	26.0	**69.2** (+1.9%)	26.4 (+0.4)
Llama3-70B	GPU-3	128	55.8	40.7 (-27.1%)	522.0	**56.1** (+0.5%)	379.9 (-142.1)
Qwen2.5 7B†	GPU-2	16	57.9	58.2 (+0.5%)	11.0	**65.9** (+13.8%)	41.1 (+30.1)
	GPU-2	8	67.3	67.9 (+0.9%)	13.7	**68.3** (+1.5%)	30.2 (+16.5)
Qwen3 6B†	GPU-2	16	12.6	13.7 (+8.7%)	12.3	**25.2** (+100.0%)	14.3 (+2.0)
	GPU-2	8	19.2	19.5 (+1.6%)	6.3	**22.1** (+15.1%)	7.5 (+1.2)
GPT3 7B†	GPU-2	16	56.1	57.2 (+2.0%)	3.2	**66.3** (+18.2%)	24.3 (+21.1)
	GPU-2	8	64.1	64.8 (+1.1%)	21.7	**65.2** (+1.7%)	9.1 (-12.6)



(a) DS 16B, 128 GPU-3 (b) DS 236B, 128 GPU-3 (c) Llama2 7B, 16 GPU-2

(d) Llama3 8B, 16 GPU-2 (e) Llama3 70B, 128 GPU-3 (f) Qwen2.5 7B, 16 GPU-2

Figure 2: Search progress in terms of best MFU against cumulative GPU-hours. OptiCo curves shown as single markers are from Chen et al. (2026), which reports only the final value.

whereas OptiCo takes over 400 GPU-hours to reach 10.1% MFU. In Figures 2(c) and 2(d), PROMA surpasses the expert baseline within 30 GPU-hours, while OptiCo improves only slightly over it. On the 7B and8B dense models at 16 GPUs, PROMA also reaches a higher MFU than OptiCo and the expert baseline, although the gap is narrower because the feasible configuration space is smaller at this scale. Remaining configurations are reported in Appendix E.

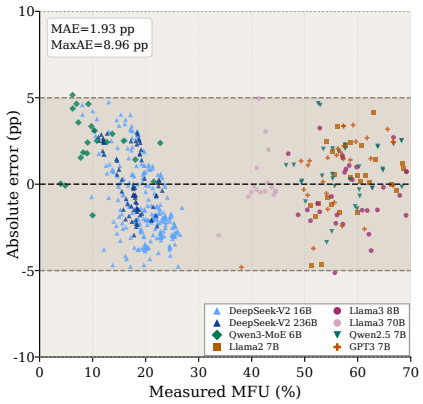


Figure 3: MfuEst absolute error.

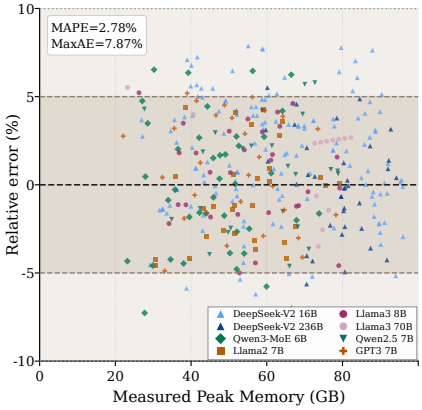


Figure 4: MemEst relative error.

4.3 Pilot Study Analysis

In our experiments, the pilot study for DeepSeek 16B starts on 8 GPUs. On 8 GPUs, it measures estimator bias, the enabled and disabled states of the fused kernel, and both sides of an integer axis split such as expert parallelism. It scales to 16 GPUs only for properties that a single node cannot measure, such as communication overlap and cross-node all-to-all. For DeepSeek 236B, the agent selects 64 GPUs as the smallest scale that can hold the model, since smaller configurations run out of memory and cannot yield valid measurements.

We compare PROMA against two alternatives that reuse small-scale experiments in different forms. The first seeds the search with the best small-scale configuration. The second feeds the small-scale trial log as history. Both are compared against an empty history search, which uses the two estimators but no pilot study knowledge. We conduct this comparison on GPU-3 at 128 GPUs.

Seeding with the best configuration performs worst. It reaches 21.8% MFU on DeepSeek 16B and 15.0% MFU on DeepSeek 236B, below the empty history search at 24.9% and 20.2%, and well below PROMA's 26.0% and 22.1% (Table 1). The configuration is tuned for the memory and network conditions at small scale, which differ from those at the target scale.

Feeding the trial log performs better but still trails PROMA. It reaches 23.8% and 16.6%, also below the result of empty history. The log records what happened at small scale, not why, so it cannot indicate which regions are feasible or which axes are worth exploring at the target scale.

The object of transfer matters, not just the fact of transfer, and the difference is amplified at scale. In a training run that takes weeks, a small MFU gain saves hundreds of GPU hours.

4.4 Estimator Evaluation

We evaluate the prediction accuracy of both estimators and the decision quality of MemEst at the OOM gate. Configurations span dense and MoE models, a range of parallel layouts, micro-batch sizes, and all checkpointing granularities. Figure 3 and Figure 4 show the error distribution across all configurations. Over the full set, 99.4% of MFU predictions fall within ± 5 pp and 87.3% of memory predictions within $\pm 5\%$. Table 2 reports the comparison against existing tools.

On dense and MoE models, MfuEst and MemEst achieve the lowest error and the lowest standard deviation in every setting, as reported in Table 2. Llm-analysis supports MoE but not context parallelism, and Calculon supports neither, so the dense comparison includes all three tools and the MoE comparison includes only llm-analysis.

We evaluate each baseline under the default setting and a tuned setting suggested by the corresponding official documentation. Tuning improves MFU accuracy but leaves memory error unchanged, since the tuned parameters only affect the timing path and not the peak-memory formula. Even with tuning, the baselines remain far less accurate than MfuEst and MemEst. For example, tuned llm-analysis still has an MFU MAE of 12.8 pp and a memory MAPE of 31.8% on dense models.

We also report a structural version that keeps only coefficients derived from model architecture and hardware parameters. It reaches 7.9 pp and 16.0% on dense models, and 7.1 pp and 13.5% on MoE models, again with lower standard deviation than the baselines (Table 2). The remaining gap comes from effects that architecture and hardware alone cannot quantify, such as DeepEP dispatch and host-side launch, captured by platform-level coefficients. The full group analysis is in Appendix D.

Table 2: Estimator prediction error on dense and MoE models. MFU error is absolute error in percentage points (pp). Memory error is relative error in percent (%). MAE denotes mean absolute error, and MAPE denotes mean absolute percentage error. Std.denotes the sample standard deviation of the prediction errors. For MfuEst it is the standard deviation of absolute errors in pp, and for MemEst it is the standard deviation of relative errors in %.

Model type	Calibration	Estimator	MFU Error		Memory Error	
			MAE (pp)	Std.	MAPE (%)	Std.
Dense	Without	llm-analysis	32.6	23.5	31.8	30.4
		Calculon	14.1	7.9	122.3	132.7
		Ours (structural)	7.9	4.6	16.0	11.8
	With	llm-analysis (tuned)	12.8	13.4	31.8	30.4
		Calculon (tuned)	9.9	10.3	121.5	132.6
		Ours	**1.8**	**2.3**	**2.5**	**2.9**
MoE	Without	llm-analysis	11.5	13.1	206.4	118.6
		Ours (structural)	7.1	4.5	13.5	9.1
	With	llm-analysis (tuned)	7.7	9.1	206.4	118.6
		Ours	**2.1**	**2.4**	**2.9**	**3.4**

Accuracy alone does not determine whether MemEst is useful as a search gate. A false positive predicts a runnable configuration as OOM, lowering its priority and potentially missing a high-throughput configuration. A false negative predicts an OOM configuration as runnable, triggering a failed run that wastes GPU time. We treat OOM prediction as binary classification with OOM as the positive class. Table 3 reports precision, recall, and F1. Recall is high for all tools, as most OOM configurations are far from the memory limit and easy to detect. The key difference is precision. MemEst achieves high precision, so nearly all configurations rejected by MemEst are truly OOM. The baselines have very low precision. Most configurations rejected by the baselines are actually runnable. Since rejected configurations are deprioritized and may never be tried, this low precision causes the search to miss high-MFU configurations that use memory aggressively. In contrast, low recall only wastes GPU time on a failed run. Calculon covers a smaller subset because Calculon skips MoE and context-parallel configurations.

4.5 Ablation Study

Table 4 reports the ablation results. All three components are necessary. We report MFU improvement over OptiCo and the expert baseline, OOM rate and average over-limit memory, the number of rounds to the first successful run, and the rounds and GPU hours needed to reach the best MFU.

Effect of MfuEst. MfuEst has the largest impact on peak MFU. Without it, the improvement over the expert-tuned baseline drops from 17.4% to 3.32%. This confirms that MfuEst is essential for steering the search toward high-throughput configurations. Without MfuEst, the search retreats to a conservative region, lowering the OOM rate to 0.14 but also reducing peak MFU. It also takes 16 rounds and 87.9 GPU hours to reach the best MFU, compared with 8.3 rounds for PROMA.

Effect of MemEst. Without MemEst, the average OOM over-limit grows from 4.91 GB to 21.73 GB, and the first successful run requires 3.5 rounds instead of one. MemEst filters out configurations that exceed the memory limit, improving search efficiency. The best MFU is reached after 14.3 rounds and 82.8 GPU hours, but at a much lower MFU level.

Effect of pilot study. Removing the pilot study degrades both effectiveness and efficiency. The improvement over the expert baseline falls to 8.57%, the first successful run takes 2.25 rounds, and the best MFU is reached after 15.4 rounds and 84.9 GPU hours. The pilot study transfers constraints from small scale to prune the search space at the target scale, accelerating convergence.

Table 3: OOM gate detection metrics. OOM is the positive class. TP denotes OOM predicted as OOM. FP denotes runnable predicted as OOM. FN denotes OOM predicted as runnable. TN denotes runnable predicted as runnable. Precision is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$. Recall is $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$. F1 is the harmonic mean of precision and recall.

Estimator	TP	FN	FP	TN	Precision (%)	Recall (%)	F1
llm-analysis	61	4	189	157	24.4	93.8	38.7
Calculon	3	0	55	69	5.2	100.0	9.8
MemEst	**65**	**0**	**3**	**343**	**95.6**	**100.0**	**97.7**

Table 4: Ablation study. MFU improvement is relative to OptiCo and the expert baseline. OOM rate is the proportion of training runs that fail due to out of memory, and over limit is the average memory beyond the limit in GB. First successful run reports the number of rounds until the first successful run. Best MFU round and GPU hours report when the best MFU is reached. Search behavior statistics are averaged over all evaluated cases.

Variant	MFU improvement		OOM		First successful run	Best MFU round	Best MFU GPU-h
	vs. OptiCo	vs. Expert	rate	over limit (GB)			
PROMA	**25.9%**	**17.4%**	0.18	**4.91**	**1**	**8.3**	**56.7**
w/o pilot study	17.96%	8.57%	0.22	12.92	2.25	15.4	84.9
w/o MemEst	15.52%	8.12%	0.24	21.73	3.5	14.3	82.8
w/o MfuEst	13.72%	3.32%	**0.14**	12.26	1.5	16.0	87.9

Overall, our components are complementary. MfuEst provides the search direction, MemEst ensures feasibility, and the pilot study accelerates convergence.

4.6 Cost Benefit Analysis

On average, PROMA spends 56.7 GPU hours on tuning and consumes about 0.4M LLM tokens. With DeepSeek-V4-Flash pricing at \$0.14 per million input tokens and \$0.28 per million output tokens, the token cost is under \$0.1.At \$1.50 per A100 GPU hour, tuning costs about \$85. For a training job at the largest scale in our experiments, 256 A100 GPUs for three months, the 17.4% MFU improvement over the expert baseline reduces training time by 14.8%, saving about 13 days and 81,838 GPU hours. This corresponds to about \$122K in saved training cost. The tuning cost is negligible compared with the savings.

5 Limitations

Our approach has two main limitations. First, extracting invariant properties **relies on prior knowledge** about which properties to measure and how to define coverage gaps. Without such knowledge, the pilot study may be less effective. Second, PROMA **searches over existing configuration options** and does not optimize at the operator or feature level. Its achievable MFU is therefore bounded by the configuration space exposed by the training framework and the hardware. These limitations do not affect the main finding that transferring invariant properties improves search efficiency, but they **limit the upper bound of the current method**. We leave end-to-end agentic optimization of the full AI infrastructure stack for future work.

6 Conclusion

Large scale training is expensive, and the training configuration determines how much of that cost is wasted. PROMA reduces this waste by extracting **scale invariant** properties from small scale experiments and using them to guide the target scale search. On average, PROMA improves MFU by **25.9%** over OptiCo and **17.4%** over the expert baseline, while using fewer GPU hours. We evaluate PROMA on dense and MoE models from 6B to 236B parameters, cluster sizes from 8 to 256 GPUs, and three hardware platforms. The improvement holds consistently across all settings. These results suggest that transferring invariant properties, rather than configurations or raw histories, is a practical path to efficient large scale tuning.

AI USE STATEMENT

In this work, we used generative AI tools for writing assistance such as polishing, grammar checking, improving conciseness, and translating drafts between languages. We also used AI to retrieve and discover related work and to assist with brainstorming. In terms of research execution, we used large language models as the inference engine for the agents in PROMA. And we used AI tools to help write code and debug. We did not use generative AI tools for synthetic data generation, theoretical models, mathematical claims and proof assistance, dataset cleaning, or qualitative analysis and interpreting results. We have reviewed all AI-assisted work. We manually verified AI-polished and AI-translated text against our original results, checked AI-suggested references through a literature survey, and tested AI-assisted code. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or sensitive information. All experiments use publicly available models and datasets on standard computing clusters, and we foresee no ethical concerns or conflicts of interest.

REPRODUCIBILITY STATEMENT

We provide an anonymous implementation at `https://anonymous.4open.science/r/PROMA`. The experimental setup, model configurations, and hyperparameters are described in Section 4.1. The full formulation of MemEst and MfuEst is provided in Appendix A and Appendix B.

REFERENCES

Sheng Chen, Tang Zhe, Weixing Zhang, Fei Yang, Yuanyuan Wang, Tianlin Li, and Yang Liu. OptiCo: Adaptive distributed training optimization via collaborative agent reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27847–27867, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1283. URL https://aclanthology.org/2026.acl-long.1283/.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Yuran Ding, Ruobing Han, Xiaofan Zhang, and Xinwei Chen. PROMPTS: Performance optimization via multi-agent planning for LLM training and serving. In *Ninth Conference on Machine Learning and Systems*, 2026. URL https://openreview.net/forum?id=FTOfgVHcZn.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), January 2022. ISSN 1532-4435.

Mikhail Isaev, Nic Mcdonald, Larry Dennison, and Richard 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, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607102. URL https://doi.org/10.1145/3581784.3607102.

Ting Jiang, Shaohan Huang, Zhongzhi Luan, Deqing Wang, and Fuzhen Zhuang. Scaling sentence embeddings with large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 3182–3196, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.181. URL https://aclanthology.org/2024.findings-emnlp.181/.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 341–353. Curran, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/80083951326cf5b35e5100260d64ed81-Paper-mlsys2023.pdf.

Cheng Li. Llm-analysis: Latency and memory analysis of transformer models for training and inference. https://github.com/cli99/llm-analysis, 2023.

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

Guodong Liu, Youshan Miao, Zhiqi Lin, Xiaoxiang Shi, Saeed Maleki, Fan Yang, Yungang Bao, and Sa Wang. Aceso: Efficient parallel dnn training through iterative bottleneck alleviation. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, pp. 163–181, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704376. doi: 10.1145/3627703.3629554. URL https://doi.org/10.1145/3627703.3629554.

Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Adaptive skills adaptive partitions (asap). In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/98d6f58ab0dafbb86b083a001561bb34-Paper.pdf.

Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *Proc. VLDB Endow.*, 16(3):470–479, November 2022. ISSN 2150-8097. doi: 10.14778/3570690.3570697. URL https://doi.org/10.14778/3570690.3570697.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 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*, SC '21, New York, NY, USA, 2021a. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021b.

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

Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, February 2005. ISSN 1094-3420. doi: 10.1177/1094342005051521. URL https://doi.org/10.1177/1094342005051521.

Yuanyuan Wang, Nana Tang, Yuyang Wang, Shu Pan, Dingding Yu, Zeyue Wang, Mou Sun, Kejie Fu, Fangyu Wang, Yunchuan Chen, Ning Sun, and Fei Yang. Autohaap: Automated heterogeneity-aware asymmetric partitioning for llm training. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–17, 2026. doi: 10.1109/HPCA68181.2026.11408533.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009. ISSN 0001-0782. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 559–578, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin.

A MemEst Details

This appendix provides the full accounting for MemEst. Section A.1 covers parameters and optimizer states, Section A.2 covers activation coefficients by layer type, Section A.3 covers checkpointing rules, and Section A.4 covers runtime overhead and boundary cases.

A.1 Parameters and Optimizer States

For a model with L layers, hidden size h, FFN hidden size h_{ffn}, padded vocabulary V, and N_e experts, the per-layer parameter counts are

$$P_{\text{dense}} = 2h \cdot (h_{\text{ffn}} \cdot g + 2) + P_{\text{attn}}, \tag{7}$$

$$P_{\text{moe,non}} = 2h \cdot (h_{\text{shared}} \cdot g + 2) + P_{\text{attn}}, \tag{8}$$

$$P_{\text{moe,exp}} = 2h \cdot h_{\text{moe}} \cdot N_e \cdot g, \tag{9}$$

where $g = 3/2$ for SwiGLU and $g = 1$ otherwise, and P_{attn} follows the standard dense or MLA formula. The per-parameter byte count is

$$b(\text{DP}) = \begin{cases} 18 & \text{without distributed optimizer,} \\ 6 + 12/\text{DP} & \text{with distributed optimizer.} \end{cases} \tag{10}$$

Non-expert parameters shard across TP and DP; expert parameters shard across EP and ETP. When TP $>$ ETP, expert replicas are reduced by the larger non-TP DP group, matching measured MoE+TP peaks. Here TP, DP, EP, and ETP denote tensor, data, expert, and expert tensor parallelism, respectively.

A.2 Activation Coefficients

The activation coefficients S, W, X in Eq. 2 depend on layer type and are given in Table 5.

Table 5: Activation coefficients in Eq. 2 by layer type. S, W, and X are the saved, work, and width coefficients defined in Section 3.3. A value of 0 means the corresponding term does not contribute for that layer type. Units are per-token activation elements; bytes are applied via bf16. MoE = mixture of experts, MLA = multi-head latent attention.

Layer type	Term	S (saved)	W (work)	X (width)
Dense	attention	13.5	4.5	0
	MLP body	$4h_{\text{ffn}}/h$	0	$12\max(0, h_{\text{ffn}}/h - 4)$
MoE	attention	13.5	4.5	0
	routed expert	$4 \cdot \text{topk} \cdot h_{\text{moe}} \cdot \rho/h$	0	0
MLA	persistent	0	0	X_{DeepEP}

The MLA persistent width term X_{DeepEP} is a function of sequence length and hidden size, as detailed in the MLA layers paragraph below.

Table 6: Checkpointing effect on the saved set, per recompute module. Values are multiplicative factors on the module's saved contribution. The module names follow the Megatron recompute configuration, where `core_attn` is core attention, `moe` is the routed expert layer, `moe_act` is the SwiGLU activation inside the expert, `layernorm` is layer normalization, and `shared_experts` is the shared expert MLP.

Recompute module	Saved factor	Notes
`core_attn`	≈ 1.0	FA already avoids score matrix
`moe`	0.3–0.5	removes dispatch and expert GEMM
`moe_act`	0.8–0.95	discards SwiGLU output only
`layernorm`	≈ 1.0	pre-norm activations are small
`shared_experts`	0.9–0.95	shared expert MLP only
`mlp` (dense)	0.3–0.5	dense MLP recompute

Dense layers. Attention and MLP bodies follow the standard accounting of Korthikanti et al. (2023). Attention contributes 18 coefficient units (plus 2 for MHA over GQA), and the MLP body contributes $4h_{\text{ffn}}/h$. The width term adds $12\max(0, h_{\text{ffn}}/h - 4)$ for wide FFNs, plus a $1/h$ term for small hidden sizes where absolute workspace does not shrink proportionally. Attention intermediates split into saved and work with the work fraction fixed at 0.25, reflecting FlashAttention's avoidance of the full score matrix.

MoE layers. Routed expert GEMM activations scale with the number of tokens assigned to each rank. Under force load balancing this is exactly $1/\text{EP}$ of the expanded token count; otherwise the hot-rank ratio is bounded by a balls-and-bins estimate. For narrow experts (width ≤ 512), grouped-GEMM fuses the SwiGLU backward set, so the activation multiplier drops from 4 to 2. The all-to-all dispatch and combine path contributes 7 live tensor copies, with the send side at full top-k volume and the receive side scaled by the EP-sharded token count.

MLA layers. MLA compresses KV through low-rank projections, so the KV stream scales with the compressed rank rather than the full head dimension. DeepEP persistent buffers are pre-allocated per sequence and hidden dimension, not per micro-batch, and are modeled as a separate term that does not scale with I_s.

A.3 Checkpointing Rules

Checkpointing reclassifies tensors between saved and work without changing the total. Table 6 lists the module-level rules.

Full recomputation. Saved reduces to the layer input plus a small retain that accounts for unrecomputed workspace. The remainder becomes work. Under MBS > 1, the retain does not fully scale, reflecting allocator reuse across micro-batches.

Selective recomputation. The effect depends on which modules are checkpointed. Recomputing `core_attn` barely reduces the saved set because FlashAttention already avoids materializing the score matrix. Recomputing `moe` removes both dispatched token buffers and expert GEMM activations from the saved set. Recomputing `layernorm` and `shared_experts` produces smaller reductions, modeled as module-specific scale factors on the saved set.

A.4 Runtime Overhead and Boundary Cases

Runtime overhead. CUDA context is 256 MB, reduced to 128 MB for small hidden sizes. NCCL scratch scales with local sequence length and hidden dimension, reflecting the symmetric buffer sizing of collective operations. CUDA and NCCL denote the GPU compute platform and the NVIDIA collective communication library, respectively.

Context parallelism. Under CP > 1, each rank holds the KV stream of the full sequence through ring exchange, adding a residue proportional to seq $\times\, h \times \text{CP}^{-1}$. CP denotes context parallelism.

Cross-entropy. Unfused cross-entropy materializes logits, softmax working state, and gradient buffers, contributing 6 B per (token, vocab) element plus up to 4 B extra when vocab tensors dominate layer activations. Fused cross-entropy reduces this to a single HBM pass. HBM denotes high bandwidth memory.

Small models. For hidden sizes below 1536, absolute workspace does not shrink proportionally, and we add a $1/h$ correction to both attention and width terms. The CUDA context is halved for these models to avoid over-penalizing their small peaks.

B MfuEst Details

This appendix provides the full accounting for MfuEst. Section B.1 covers the compute and communication models, Section B.2 covers the overlap rules, and Section B.3 covers the MoE-specific terms. In this section, TP, PP, CP, and EP denote tensor, pipeline, context, and expert parallelism. SM denotes streaming multiprocessor.

B.1 Compute and Communication Models

Compute. A GEMM with shape (M, N, K) is partitioned into tiles of size $128 \times 128 \times 32$ and distributed across the available SMs. The tile count is

$$\text{tiles} = \lceil M/128 \rceil \cdot \lceil N/128 \rceil \cdot \lceil K/32 \rceil. \tag{11}$$

The occupancy is the fraction of SM capacity filled across waves,

$$\eta_{\text{occ}} = \frac{\text{tiles}}{\lceil \text{tiles}/N_{\text{SM}} \rceil \cdot N_{\text{SM}}} \cdot e_M \cdot e_N \cdot e_K, \tag{12}$$

where e_M, e_N, e_K account for partial edge tiles. The compute limit is $\text{FLOPs}/(P_{\text{peak}}\eta_{\text{occ}})$.

Split-K and K-drain. When the $M \times N$ tile grid is smaller than the SM count, split-K mints additional CTAs along the contraction dimension to fill the wave. When the grid already fills all SMs and K is short, the tensor pipe drains between tiles, and we apply a drain factor proportional to $\lceil K/32 \rceil$. The drain applies only to wide-N GEMMs, since skinny-N tiles are already edge-bound.

MoE expert GEMM. For grouped expert GEMMs, each expert has its own tile grid, and the occupancy is a blend of the per-expert grid and the packed grid. The packed token count m_{pack} amortizes expert switching, while the per-expert token count m_{one} determines the per-group fill. Attention, dense MLP, MoE expert GEMM, router, and logits each supply their own (M, N, K) derived from the model dimensions and the parallel layout.

Communication. Each collective follows the α-β model, $t = \alpha + \beta \cdot D_{\text{msg}}$, where α is the fixed launch latency and β is the inverse bandwidth. Both are calibrated per platform and per collective type. Ring allreduce and all-to-all carry different α intercepts, since their per-message overheads differ. The all-to-all payload splits into a send side that carries the full top-k volume and a receive side that scales with $1/\text{EP}$ under load balancing.

B.2 Overlap Rules

Whether communication overlaps with compute follows from the platform's stream configuration, which the estimator reads from two fields. `max_cuda_connections` records how many CUDA connections are available to the process. `singleton_dist_stream` records whether all distributed operations are serialized on a single stream. NCCL denotes the NVIDIA collective communication library. DP and SP denote data and sequence parallelism. P2P denotes point-to-point communication.

When only one CUDA connection is available, NCCL shares the compute stream and every collective is exposed, so $t_{\text{exposed}} = t_{\text{comm}}$. When multiple connections are available, NCCL runs on a separate stream and only the portion exceeding compute is exposed, so $t_{\text{exposed}} = \max(0, t_{\text{comm}} - t_{\text{comp}})$.

Table 7: Mean absolute error by model family (dense only). Memory error is relative (%).

Model	llm-analysis	Calculon	MemEst
GPT3-7B	28.2	91.0	2.9
Llama2-7B	29.6	92.6	2.3
Llama3-8B	25.1	85.2	2.4
Qwen2.5-7B	29.4	49.4	2.7
Llama2-0.5B	40.4	257	2.2
Llama3-0.5B	39.2	236	2.0
Qwen2.5-0.5B	36.1	124	3.4

Several cases modify this baseline. An intra-node TP allreduce on a multi-node job shares the compute stream with inter-node DP traffic, so a fraction of the raw TP time stays exposed even when `tp_comm_overlap` is enabled. Sequence parallelism issues separate reduce-scatter and all-gather, which cannot fuse on a single connection. Pipeline point-to-point activations expose a fraction of their time under single-stream scheduling. MoE combine overlaps the next layer's attention, while dispatch stays blocking.

B.3 MoE-Specific Terms

Two terms are specific to MoE layers. NVLink and RDMA denote intra-node and inter-node interconnect technologies. EP denotes expert parallelism, and SM denotes streaming multiprocessor. DeepEP is the expert-parallel communication library used on GPU-3.

Asymmetric all-to-all. The dispatch and combine path carries different volumes on each side. Each rank sends its full top-k tokens, while it receives only the $1/\text{EP}$ share assigned to its local experts. The all-to-all time is therefore modeled with the send-side volume for latency and the receive-side volume for the throughput-bound portion. When the EP group spans both NVLink and RDMA domains, the payload is split by peer fraction, and each portion is charged at its own bandwidth.

SM reservation. On platforms that enable DeepEP, a fixed pool of SMs is reserved for communication, reducing the SM count available to the expert GEMM. The fraction of compute covered by dispatch depends on the packed token count, since fat expert tiles hide more of the dispatch behind compute. The reservation is also reflected in the tile count used for occupancy.

C Estimator Evaluation

This appendix reports the accuracy of both estimators. The overall comparison with existing estimators is in Section 4.4. Section D provides per-model breakdowns, error distributions, coefficient ablation, and OOM gate confusion matrices.

C.1 Setup

We evaluate both estimators on configurations spanning dense and MoE models, parallel layouts from TP=1,PP=1 to TP=8,PP=8,EP=32, and micro-batch sizes from 1 to 16. Ground truth is the maximum memory reported by the training framework's device allocator for MemEst, and the MFU reported by the training framework over a full step for MfuEst. Configurations are split by model family, with entire families held out to test generalization.

C.2 Memory Error Breakdown

Table 7 reports error by model family, and Table 8 reports error by parallel layout. MemEst's error is stable across model sizes, whereas llm-analysis underestimates consistently and Calculon overestimates severely on small models.

Table 8: Mean absolute error by parallel layout (dense only). Memory error is relative (%). TP, PP, CP, and EP denote tensor, pipeline, context, and expert parallelism, respectively.

Layout	llm-analysis	Calculon	MemEst
TP=1, PP=1	33.8	127	2.5
TP$\geq$2, PP=1	33.5	100	2.7
PP$\geq$2	25.9	152	2.4
TP$\geq$2, PP$\geq$2	26.3	66.9	2.6

Table 9: Mean absolute error by model family (dense only). MFU error is in percentage points (pp).

Model	llm-analysis	Calculon	MfuEst
GPT3-7B	31.4	6.5	2.0
Llama2-7B	28.9	9.2	1.6
Llama3-8B	30.7	9.5	1.7
Qwen2.5-7B	28.8	6.5	1.5
Llama2-0.5B	39.4	19.7	2.3
Llama3-0.5B	34.0	20.5	1.7
Qwen2.5-0.5B	38.0	15.2	2.0

C.3 MFU Error Breakdown

Table 9 reports error by model family, and Table 10 reports error by parallel layout. MfuEst stays below 2.3 pp across all dense families, while llm-analysis overestimates consistently due to its peak-FLOPs assumption and Calculon underestimates due to its no-overlap model.

D Estimator Error Breakdown

This appendix reports the per-model error breakdown, the error distribution within fixed bounds, the coefficient group ablation, and the OOM gate confusion matrices.

D.1 Per-Model Error Breakdown

Table 11 reports MFU MAE in percentage points and memory MAE in relative terms for each model. Calculon does not support MoE or context parallelism, so its column is empty for MoE rows.

D.2 Error Distribution within Bounds

Table 12 reports the fraction of configurations whose prediction falls within ± 5 and ± 10 of the measured value. The fitted estimator keeps almost all configurations within ± 5 on MFU and 87.3% on memory. The structural version covers a smaller share but still outperforms both baselines on MAE.

D.3 Coefficient Group Ablation

We quantify how much each coefficient group contributes by restoring it one at a time on top of the structural version. Table 13 reports the result. A large gain indicates that the effect the group quantifies is important for accuracy.

On dense models, two groups account for about 90% of the MFU gap. Stream contention arises when NCCL shares the compute stream with GEMM on single-connection platforms. Host-side overhead covers launch latency and allocator cost per micro-batch. Both depend on runtime behavior of the framework and the collective library rather than on architecture and hardware parameters alone, which is why the structural model captures their direction but not their magnitude. The memory gap is dominated by runtime residuals corresponding to allocator fragmentation, pipeline in-flight activations, and MLA persistent buffers. On MoE, the DeepEP group reduces MFU MAE from 7.1 pp to 3.9 pp, confirming that MoE-specific modeling is essential for sparse architectures.

Table 10: Mean absolute error by parallel layout (dense only). MFU error is in percentage points (pp). TP, PP, CP, and EP denote tensor, pipeline, context, and expert parallelism, respectively.

Layout	llm-analysis	Calculon	MfuEst
TP=1, PP=1	31.0	15.8	2.0
TP≥2, PP=1	33.6	12.8	1.9
PP≥2	33.7	13.3	1.4
TP≥2, PP≥2	21.0	19.2	1.2

Table 11: MAE by model. MFU error is in percentage points (pp); memory error is relative (%). Ours (structural) is the estimator variant that keeps only coefficients derived from architecture and hardware parameters. llm-analysis and Calculon are existing analytical estimators.

Model	Ours	Ours (structural)	llm-analysis	Calculon
Dense, MFU (pp)				
GPT3-7B	2.0	9.7	31.4	6.5
Llama2-7B	1.6	9.2	28.9	9.2
Llama3-8B	1.7	9.6	30.7	9.5
Qwen2.5-7B	1.5	8.8	28.8	6.5
Llama3-70B	1.1	11.0	13.5	25.5
Llama2-0.5B	2.3	5.3	39.4	19.7
Llama3-0.5B	1.7	6.2	34.0	20.5
Qwen2.5-0.5B	2.0	7.8	38.0	15.2
MoE, MFU (pp)				
DeepSeek-V2 16B	2.2	5.9	12.1	—
DeepSeek-V2 236B	1.6	9.5	9.8	—
Qwen3-MoE 6B	2.5	11.1	11.7	—
Dense, Memory (%)				
GPT3-7B	2.9	9.3	28.2	91.0
Llama2-7B	2.3	10.6	29.6	92.6
Llama3-8B	2.4	9.8	25.1	85.2
Qwen2.5-7B	2.7	12.0	29.4	49.4
Llama3-70B	2.7	6.5	24.8	20.1
Llama2-0.5B	2.2	24.3	40.4	257
Llama3-0.5B	2.0	22.2	39.2	236
Qwen2.5-0.5B	3.4	33.8	36.1	124
MoE, Memory (%)				
DeepSeek-V2 16B	3.0	14.1	211	—
DeepSeek-V2 236B	2.3	15.4	246	—
Qwen3-MoE 6B	3.1	9.5	154	—

E Additional MFU Curves

We report the optimization progress of MFU on the remaining configurations in Figure 5. The format follows Figure 2: each panel shows the best MFU found so far at each cumulative GPU-hour budget.

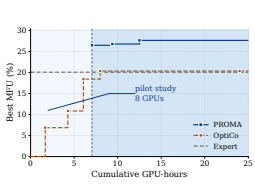
The same pattern holds across the remaining configurations. On the MoE models, PROMA reaches a near-final MFU within the first few GPU-hours and then refines. On DS-236B at 64 GPUs, PROMA reaches 20.8% at 24.7 GPU-hours, while OptiCo takes over 100 GPU-hours to approach 19%. On Qwen3-MoE at both 16 and 8 GPUs, PROMA reaches a higher MFU than OptiCo, though its GPU-hour cost is comparable. On the dense 7B models at 8 GPUs, the curves are closer since the search space is small, but PROMA still reaches a higher MFU at every budget.

Table 12: Fraction of configurations within an error bound. MFU error is in percentage points (pp); memory error is relative (%).

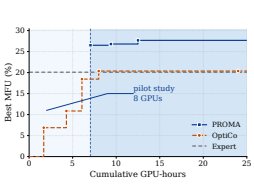
Version	Within ±5 (%)	Within ±10 (%)
MemEst	87.3	100.0
MemEst (structural)	19.4	42.8
MfuEst	99.4	100.0
MfuEst (structural)	30.5	72.3

Table 13: Accuracy gain from each coefficient group, restored one at a time on top of the structural version. Lower is better. MFU error is in percentage points (pp), and memory error is relative (%). Group A covers stream contention, E covers host launch and straggler, D covers L2 and HBM saturation, B covers communication overlap, C covers MoE and DeepEP residuals, and F covers memory residuals.

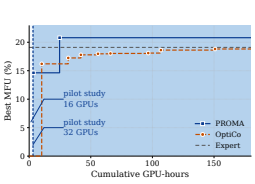
Group	Description	MFU Error (pp)	Mem Error (%)
—	Structural	8.0	16.0
A	Stream contention	5.0	—
E	Host / launch / straggler	5.4	—
D	L2 / HBM saturation	6.8	—
B	Communication overlap	8.0	—
C	MoE / DeepEP residuals	8.0	—
F	Memory residuals	—	3.5
—	Fitted	2.0	2.5



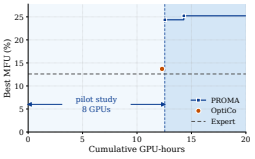
(a) DS 16B, 16 GPU-3



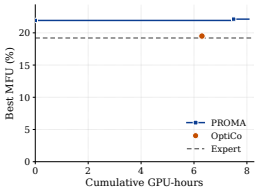
(b) DS 16B, 16 GPU-3



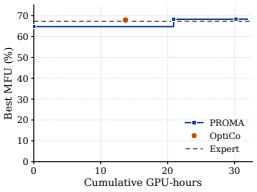
(c) DS 236B, 64 GPU-3



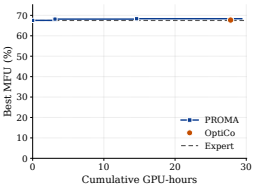
(d) Qwen3-MoE, 16 GPU-2



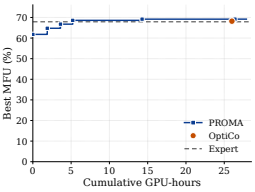
(e) Qwen3-MoE, 8 GPU-2



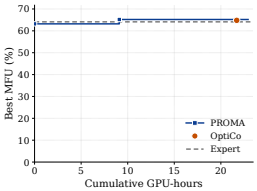
(f) Qwen2.5 7B, 8 GPU-2



(g) Llama2 7B, 8 GPU-2



(h) Llama3 8B, 8 GPU-2



(i) GPT3 7B, 8 GPU-2

Figure 5: Best MFU at each cumulative GPU-hour budget for the remaining configurations. DS = DeepSeek-V2. OptiCo results shown as single markers are from Chen et al. (2026), which reports only the final value.