PRISM: Profiling-Free Symbolic Memory-Driven Strategy Planner for Large DNN Model Training

The rapid growth of large-scale deep neural networks (DNNs) has introduced severe memory and performance bottlenecks during distributed training. Existing automated planners for parallelization strategies often rely heavily on profiling or empirical tuning, which significantly increases engineering cost and wastes large-scale cluster resources. In this work, we present PRISM, a profiling-free, symbolic memory-driven strategy planner for large DNN training. PRISM introduces a unified symbolic memory cost model that captures the layered structure of modern architectures and integrates with a communication model to evaluate trade-offs across data, tensor, pipeline, virtual pipeline, expert, and sequence parallelism, as well as activation recomputation and optimizer sharding. By formulating strategy selection as an optimization problem, PRISM identifies globally optimal parallel strategies under device memory budgets. Our evaluation across representative large models demonstrates that PRISM achieves accurate memory prediction and substantial improvements in Model FLOPs Utilization (MFU), reducing bubble and communication overheads without costly profiling.

1 Introduction

DNNs have progressed rapidly in recent years—improving accuracy across language, vision, and multimodal tasks—while simultaneously growing in parameter count, sequence length, and modality breadth. Training such models now requires large distributed high-performance computing (HPC) systems that compose multiple forms of parallelism to sustain throughput and fit within device memory, including *Data Parallelism (DP)* [6], *Tensor/Model Parallelism (TP)* [18], *Pipeline Parallelism (PP)* [8, 13], *Virtual Pipeline Parallelism (VPP)* [14], *Expert Parallelism (EP)* for Mixture-of-Experts [7, 11], *Sequence (SP)* [9, 10], *Optimizer Partitioning (OP, ZeRO)* [16], and *activation recomputation* (checkpointing) [5].

Memory is the primary limiter of DNN training throughput. Even when multiple parallelisms are composed, the instantaneous peak on each accelerator gates feasible micro-batch size, interleaving depth, and overlap. Simply "adding machines" raises the volume and cadence of collectives, often amplifying communication and eroding expected gains. As a rule of thumb: DP is memory-hungry and prone to OOM; TP is communication-intensive; SP reduces activation footprint by sharding along the sequence dimension but introduces additional collectives around attention; PP creates pipeline bubbles; EP is heavy due to token routing; and OP (ZeRO-style optimizer-state sharding) cuts memory at the cost of optimizer-step traffic. Concretely, DP keeps full parameters, gradients, and optimizer states on every rank, so per-rank peak scales with model size (not DP degree), and gradient all-reduce occurs after activations are materialized—DP alone rarely relieves peaks. TP shards tensors across ranks but forces frequent all-gather / reduce-scatter / all-reduce around matmuls and attention every forward/backward step; both collective count and payload grow with the shard dimension, making TP communication-dominant. SP computes attention via efficient communication, when sequence length and device count scale proportionally, the per-step communication volume remains roughly constant—enabling extreme long-sequence training without exploding memory. PP partitions the network into stages and uses micro-batches

Author's Contact Information:

Manuscript submitted to ACM

System	Techniques	Cost model	Planner
Megatron-LM [14, 18]	DP, SP, TP, PP, Recomp	Specific	Manual
ZeRO-DP/OP [16]	DP, OP	Specific	Manual
Galvatron [12]	DP, OP, TP, PP	Profiling-based	Tuning
vTrain [4]	DP, TP, PP	Profiling-based	Tuning
PRISM (this work)	DP, SP, TP, PP, VPP, EP, OP, Recomp	Symbolic	Algorithmic

Table 1. Representative systems for large-model training. "Specific" denotes system-specific cost accounting embedded in the implementation; "Profiling-based" denotes empirical cost estimation obtained via short trial runs; "Symbolic" denotes an implementation-agnostic closed-form model.

to fill and drain the pipeline, introducing warm-up/flush idle slots and sensitivity to stage imbalance. EP distributes experts but routes tokens via all-to-all at each expert layer in both directions; traffic scales with sequence length, hidden size, and capacity factor, while routing/padding and load skew inflate latency and bandwidth demand. OP shards optimizer states—and in higher stages, gradients and even parameters—across the DP group, reducing per-rank memory roughly with sharding degree; however, it introduces per-step reduce-scatter/all-gather of gradients and (ZeRO-3–like) parameter all-gathers that must be carefully prefetched and overlapped to avoid optimizer-step latency and background bandwidth pressure. Balancing these modes in practice means trading memory headroom against collective volume, bubble overhead, and overlappability.

Memory-Performance trade-offs are tightly coupled. Mechanisms that lower on-rank residency typically increase communication or pipeline idle time; mechanisms that reduce bubbles often widen activation liveness or add FLOPs. For a fixed model and global batch, useful compute per step is essentially invariant, so performance hinges on minimizing communication and bubble overhead while staying within per-accelerator memory limits and satisfying the schedule. The design space is discrete, non-separable, and topology dependent across DP, SP, TP, PP, VPP, EP, OP settings, recomputation policy, and micro-batches. Micro-batching co-determines both peak memory and pipeline bubbles and cannot be tuned independently. Practical planners must jointly choose strategy and micro-batching under the actual execution schedule, balancing memory feasibility with communication and bubble costs.

State of the art and limitations. Table 1 contrasts representative systems along three axes—mechanism coverage, cost modeling, and planning method. Megatron-LM [14, 18] exposes a rich set of mechanisms (DP/SP/TP/PP, recomputation) but relies on *system-specific* accounting embedded in implementation choices with a *manual* planner, which shifts the burden to practitioners and limits portability and reproducibility across clusters. ZeRO-DP/OP [16] offers precise, *specific* accounting for optimizer/parameter sharding and offload but remains a component rather than a unified planner; strategy composition across TP/PP/EP/VPP still requires manual coordination. Galvatron [12] and vTrain [4] enlarge the mechanism set to hybrid strategies and replace hard-coded accounting with *profiling-based* cost models coupled to *tuning* loops; this improves automation on a fixed stack but consumes accelerator hours for trial runs, lengthens queue time, and degrades when model shapes, schedules/interleaving variants, kernels, or interconnect/topology change. Moreover, empirical regressors provide limited visibility into where peak memory occurs and scale poorly as the DP/SP/TP/PP/VPP, EP/OP, and micro-batching dimensions interact combinatorially. In contrast, PRISM (this work) covers a superset of mechanisms—including VPP and EP—and replaces profiling with a *symbolic* model and an *algorithmic* planner. This yields portable, schedule-aware feasibility checks and transparent communication–bubble trade-offs without trial runs, enabling dependable planning across hardware and workloads.

Scalability of profiling and tuning. As the mechanism set grows (DP/TP/PP/VPP, SP, EP, OP, recomputation, micro-batch size,schedules), the strategy space expands *combinatorially*. Let $\mathcal{S} = \mathcal{D} \times \mathcal{T} \times \mathcal{P} \times \mathcal{V} \times \mathcal{S} \times \mathcal{E} \times \mathcal{O} \times \mathcal{R} \times \mathcal{B} \times \mathsf{Sched}$

denote choices for DP degree, TP degree, PP stages, VPP interleaving degree, EP degree, OP degree, recompute policy, micro-batch size, and schedule. Even with modest per-axis cardinalities, $|\mathcal{S}|$ quickly reaches 10^4–10^6 per model–hardware setting. Profiling-based estimation then costs roughly $|\mathcal{S}| \cdot \tau_{\text{trial}}$ and must be repeated whenever shapes/schedules/topology change, while tuning on a discrete, non-separable surface suffers superlinear complexity due to cross-terms and schedule-dependent peaks. As dimensions proliferate, the profiling and tuning burdens explode, making the approaches in Table 1 increasingly hard to sustain at scale.

Our position. Memory is the binding constraint for large-model training. A profiling-free, symbolic view of the model and schedule is sufficient to (i) bound per-stage peak memory and (ii) expose the trade-off between pipeline bubbles and communication. Hence the choice of DP/TP/PP/VPP, SP, EP, OP, recomputation, micro-batch size,schedule should be solved jointly under a memory budget.

In this paper, we introduce PRISM, a profiling-free planner that analytically predicts peak memory and communication costs, then searches for a configuration that minimizes both bubble and communication overhead subject to memory limits. PRISM supports dense and MoE models and outputs deployable strategies.

Contributions.

(1) Profiling-free symbolic memory model. A grammar-driven, closed-form model that computes stage-wise peak memory without profiling, covering DP/TP/PP/VPP/SP/EP/OP, recomputation, micro-batch size, schedule.
(2) Unified memory+communication model with a joint solver. We define a unified cost $J = \mathcal{B} + T_{\text{comm}}$ combining pipeline bubbles and communication, and propose an algorithmic planner (PRISM-Search) that jointly selects $(dp, tp, pp, vpp, sp, ep, op, recompute, b, schedule)$ under a memory budget.
(3) Experimental results. On an Ascend-910 cluster (up to 1,024 devices) across DeepSeek, LLaMA, TextHawk, and MoE workloads, PRISM delivers up to 1.93× MFU speedup and achieves tight memory prediction with median absolute error within ≈7%.

2 Approach

2.1 Notation

This subsection catalogs the symbols used by our memory and performance models. We group notation into six categories: *Model hyperparameters*, *Data types and byte widths*, *Buckets*, *Sharding divisors and indicators*, *Schedule-aware terms*, and *Communication timing*.

Model hyperparameters. h (hidden width), h_{ff} (*Feed-Forward Network, FFN* inner width), v (vocabulary size), n_h (number of attention query heads), d_h (per-head projection width), n_{kv} (number of key–value heads under *Grouped-Query Attention, GQA* or *Multi-Query Attention, MQA*; typically $n_{kv} \le n_h$), s (sequence length), and L (number of layers). *Batching:* b is the per-rank micro-batch size and m is the number of micro-batches per iteration; the global batch is $G = dp \cdot b \cdot m$. A *strategy* specifies parallel degrees $(dp, tp, pp, vpp, sp, op, ep)$, where *Data Parallelism (DP)* replicates parameters across ranks; *Tensor Parallelism (TP)* shards tensors along model dimensions; *Pipeline Parallelism (PP)* partitions layers into stages; *Virtual Pipeline Parallelism (VPP)* interleaves multiple virtual chunks per stage to reduce bubbles; *Sequence Parallelism (SP)* shards tokens along the time/sequence axis; and *Expert Parallelism (EP)* distributes experts in *Mixture-of-Experts (MoE)* layers across ranks. We also specify a schedule $\texttt{sched} \in \{\texttt{1F1B}, \texttt{SeqPipe}, \texttt{DualPipeV}\}$, an optional recomputation policy $\mathcal{R}$ (*recomputation*), and the$op \le dp$ for *Optimizer State parallel (OP).*

Data types and byte widths. Bytes per element are denoted by B_p^{FP} (parameters), B_{grad}^{FP} (gradients), B_{act}^{FP} (activations), and B_{os}^{FP} (optimizer states), typically chosen from $\{1, 2, 4\}$ for FP8/FP16/FP32.

Buckets. For each layer ℓ we track element counts $\mathrm{Param}(\ell)$, $\mathrm{Grad}(\ell)$, and activation buckets $\mathrm{Act}_A(\ell)$ with $A \in \{\text{Attn:QKV, Attn:Score, Attn:Proj, FFN, MoE:router, MoE:routed, MoE:shared, MoE:concat, Norm/Util}\}$, where *MoE* (Mixture of Experts) uses a router with top-k gating and a capacity factor. Byte sizes follow by multiplication: $\mathrm{ParamBytes}(\ell) = \mathrm{Param}(\ell)\, B_p^{FP}$, $\mathrm{GradBytes}(\ell) = \mathrm{Param}(\ell)\, B_{grad}^{FP}$, and $\mathrm{ActBytes}_A(\ell) = \mathrm{Act}_A(\ell)\, B_{act}^{FP}$. *Peak device memory* aggregates (i) *static* residency (parameters, gradients, optimizer states) and (ii) *dynamic* terms.

Sharding divisors and indicators. Materialization under sharding is captured by $\phi_{tp}^{\text{in/out}}(\ell) \in \{tp, 1\}$ and $\psi_{\text{seq}}^{\text{in/out}}(\ell) \in \{sp, 1\}$, marking whether inputs/outputs are in sharded or global view for TP, SP. Communication needs are indicated by $\chi_\ell^{\text{in-gather}}, \chi_\ell^{\text{out-red}}$ (TP all-gather / all-reduce) and $\chi_\ell^{\text{seq-gather}}, \chi_\ell^{\text{seq-red}}$ (SP gather / reduce-scatter). For MoE, $\phi_{\text{route}}(\ell) \in [0, 1]$ is the fraction of routed activations that leave a rank under $(ep, top\_k, \texttt{capacity})$. Pipeline boundaries incident to stage i are enumerated by $\mathcal{U}_i$, with an effective shard divisor $\varphi_{tp,sp} \in \{tp, sp, tp \cdot sp, 1\}$ on boundary tensors.

Schedule-aware terms. Activation concurrency on stage i is $\mathrm{PPFactor}_i^{\texttt{sched}} = f_{\text{pp}}(i;\, m, pp, vpp, \texttt{sched})$ (Eq. (5)); $\mathrm{BackwardOverhead}_i^{\texttt{sched}}$ accounts for warm-up/flush tails and recomputation. The bubble model uses $\Theta_{\text{mb}}(b)$ (per-micro-batch compute time), $g_{\texttt{sched}}(pp, vpp, m, \Delta_{\text{imb}})$ (schedule factor for imbalance Δ_{imb}), and $c_{\text{recomp}}(\mathcal{R})$ (recompute overhead) as in Eq. (37).

Communication timing (for the performance objective). Collectives follow the latency–bandwidth (α–β) model $\tau_{\text{coll}}(b; g, \text{pat}) = \alpha_{\text{pat}}(g) + b/\mathcal{B}_{\text{pat}}(g)$ for $\text{pat} \in \{\text{allreduce, allgather, reducescatter, alltoall, point-to-point (p2p)}\}$ (Eq. (24)); multiplicities per step follow Eq. (35). Per-step communication time $T_{\text{step}}^{\text{comm}}$ (Eq. (36)) is combined with the bubble term to form the overall objective $J(s, b, m)$ (Eq. (39)), while *memory feasibility* of (s, b, m) is checked solely against Eq. (2) (excluding transient comm buffers).

2.2 Model abstraction

Any DNN model can be abstracted as a sequence of building blocks, of which the coarsest-grained are called layers. In particular, all LLMs encountered so far respect a specific architecture represented by the following grammar:

$$\begin{array}{rcl} \text{DNN} & \rightarrow & \text{embedding} \cdot \text{Layer}^{+} \cdot \text{MTP}^{*} \cdot \text{Output} \\ \text{Layer} & \rightarrow & (\text{Norm} \cdot \text{Attention})^{*} \cdot \text{Norm} \cdot \text{FFN} \\ \text{Norm} & \rightarrow & \text{layerNorm} \mid \text{rmsNorm} \mid \ldots \\ \text{Attention} & \rightarrow & \text{selfAttention} \mid \text{crossAttention} \mid \ldots \\ \text{FFn} & \rightarrow & \text{feedForward} \mid \text{routedExpert} \mid \ldots \\ \text{MTP} & \rightarrow & \text{Norm} \cdot \text{Norm} \cdot \text{linear} \cdot \text{Layer} \\ \text{Output} & \rightarrow & \text{Norm} \cdot \text{linear} \end{array} \tag{1}$$

Also, a complex multimodal model is easily represented as: MultiModal $\rightarrow$ DNN^{+}. Once a network is expressed in the language represented by this grammar, it can be easily parsed and analyzed to compute the memory it needs. Hence, rather than analyzing the liveness of each data structure in a complex dataflow graph as a low-level approach would, this only requires memory functions for each symbol of the grammar. The terminal symbols of the grammar, or the most basic building blocks (represented with a lowercase first letter), are detailed:

- *Embedding/Adapters* map raw inputs (tokens, patches) and positional information to width h;
- *Self-Attention* operates on a single stream with n_h heads of width d_h, optionally using n_{kv} for GQA/MQA [3, 17];
- *Cross-Attention* allows queries from one stream to attend to keys/values from another, which is how multimodal stacks couple modalities;
- *Feed-Forward/MLP* applies a position-wise projection with expansion h_{ff};
- *Mixture-of-Experts (MoE)* replaces an MLP by a gated expert bundle parameterized by n_{ex}^r, n_{ex}^s, and top_k [7, 11, 15];
- light-weight utilities such as the normalizations or linear layers.

2.3 Memory cost model

Goal. Given the layer–stage/chunk mapping from Section 2.2, the per-device peak memory is the stage-wise maximum of *static* and *dynamic* terms plus a small schedule tail:

$$\text{Peak}_{\text{mem}} = \max_{i \in \text{Stages}} \left[\sum_{c_{ij} \in \text{Chunks}_i} \left(\sum_{\ell \in \mathcal{L}_{c_{ij}}} \left(\text{Mem}_{\text{stat}}(\ell) + \text{Mem}_{\text{dyn}}(\ell) \right) \right) + \text{BackwardOverhead}_i^{\text{sched}} \right]. \tag{2}$$

Equation (2) computes the peak memory usage across all pipeline stages. Inside the max$[\cdot]$, for each stage i we sum over all chunks c_{ij} on that stage, and within each chunk we sum over all layers ℓ in that chunk the layer's static memory $\text{Mem}_{\text{stat}}(\ell)$ plus dynamic memory $\text{Mem}_{\text{dyn}}(\ell)$. We then add BackwardOverhead$^{\text{sched}}i$, an extra memory term accounting for the pipeline's tail end (the warm-up/flush overhead on stage i due to the schedule and any recomputation). Taking the maximum over i yields the peak memory usage of the most memory-demanding stage. In other words, a strategy is feasible only if each device has at least Peak_{mem} memory available.

2.3.1 Decomposition. We express each layer's memory as the sum of a *static* component and a *dynamic* component. $\text{Mem}_{\text{stat}}(\ell)$ covers the memory for layer ℓ's parameters, gradients, and optimizer states (accounting for any sharding by dp, tp, sp, ep, or op). $\text{Mem}_{\text{dyn}}(\ell)$ contains (i) live activations (intermediate activations that must be kept for backward) scaled by schedule-aware concurrency and (ii) transient communication buffers from collectives or point-to-point transfers. We formalize these in Eqs. (3)–(4).

$$\text{Mem}_{\text{dyn}}(\ell) = \underbrace{\text{PPFactor}_i^{\text{sched}} \ \text{ActBytes}(\ell) \ \frac{1}{tp \cdot sp}}_{\text{live activations}} + \underbrace{\text{CommTransBytes}(\ell)}_{\text{comm. transients}}, \tag{3}$$

$$\text{CommTransBytes}(\ell) = \max\left(\text{Comm}_{\text{DP/OP}}(\ell),\ \text{Comm}_{\text{TP}}(\ell),\ \text{Comm}_{\text{SP}}(\ell)\right) \ + \ \text{Comm}_{\text{EP}}(\ell). \tag{4}$$

In Eq. (3), the first term is the memory from live activations of layer ℓ on stage i, computed as $\text{ActBytes}(\ell)$ (the total bytes of activations produced by layer ℓ for one micro-batch) multiplied by PPFactor$^{\text{sched}}i$ (the number of micro-batches concurrently alive on stage i under the pipeline schedule) and divided by $tp \cdot sp$ (since if tensor or sequence parallelism is applied, each rank only holds a $1/(tp \cdot sp)$ fraction of those activations). The third term $\text{CommTransBytes}(\ell)$ represents the bytes of transient communication buffers. Equation (4) defines $\text{CommTransBytes}(\ell)$ more concretely: we take the maximum among the required communication buffers for data parallel / optimizer partitioning (CommDP/OP), tensor parallel (CommTP), and sequence parallel (CommSP) for layer ℓ (since typically only one of these largest communications would be active at a time), and we add $\text{Comm}_{EP}(\ell)$, the MoE expert-parallel communication (which usually occurs separately as an all-to-all). This formulation assumes that different types of collective communications do not peak simultaneously, so the largest one dominates the transient memory.

2.3.2 Pipeline factor and tail. Activation concurrency on stage i:

$$\text{PPFactor}_i^{\text{sched}} = f_{\text{pp}}(i;\ m,\ pp,\ vpp,\ \texttt{sched} \in \{\text{1F1B}, \text{SeqPipe}, \text{DualPipeV}\}), \tag{5}$$

and BackwardOverhead$^{\text{sched}}i$ models the pipeline warm-up/flush tail and any recomputation effects for stage i. In other words, $f$pp gives the number of micro-batches simultaneously active on stage i for a given schedule sched (with pp physical stages, vpp virtual stages per physical, and m micro-batches), and BackwardOverhead quantifies how much extra memory is used at stage i at the end of an iteration (e.g., the last activation that remains until its backward pass,

or duplicated activations due to recomputation). We will derive closed-form expressions for these schedule-dependent terms below.

2.3.3 *Embedding layer.*

$$\mathrm{Mem}_{\mathrm{stat}}(\mathrm{Embed}) = \mathrm{Param}(\mathrm{Embed})\,\left(B_p^{FP} + B_{\mathrm{grad}}^{FP} + 2B_{\mathrm{os}}^{FP}\right)\,\frac{1}{\mathrm{shard}_{\mathrm{emb}}\cdot sp}, \tag{6}$$

$$\mathrm{Mem}_{\mathrm{dyn}}(\mathrm{Embed}) = \mathrm{PPFactor}_i^{\mathsf{sched}}\;\frac{\mathrm{ActBytes}(\mathrm{Embed})}{tp\cdot sp} + \mathrm{CommTransBytes}(\mathrm{Embed}), \tag{7}$$

with Transformer-based shapes

$$\mathrm{Param}(\mathrm{Embed}) = h\,v, \qquad \mathrm{ActBytes}(\mathrm{Embed}) = B_{act}^{FP}\,s\,b\,h.$$

A convenient transient bound consistent with the comm formulas is

$$\mathrm{CommTransBytes}(\mathrm{Embed}) = \max\left(\underbrace{\mathrm{comm}_{dp}\;\frac{\mathrm{ParamBytes}(\mathrm{Embed})}{\mathrm{shard}_{\mathrm{emb}}\cdot sp}}_{\text{DP/OP}}, \underbrace{\mathrm{comm}_{tp}\;B_{act}^{FP}\,s\cdot b\;\cdot h\;\frac{tp-1}{tp\cdot sp}}_{\text{TP gather}}\right), \tag{8}$$

where t=tp, $\mathrm{shard}_{\mathrm{emb}} \in \{1, dp, dp\cdot tp\}$, and $\mathrm{comm}_{dp}, \mathrm{comm}_{tp} \in \{0,1,2,3\}$ denote mode/masks.

Equation (6) calculates the static memory for the embedding layer. It multiplies the number of embedding parameters Param(Embed) by ($B^{FP}p + B^{FP}$grad + $2B^{FP}$os) (bytes per parameter + gradient + two optimizer states). This formula covers storing the embedding weight, its gradient, and two optimizer moments per parameter, consistent with a typical Adam optimizer memory breakdown [16]. For a Transformer embedding layer of hidden dimension h and vocabulary size v, Param(Embed) = $h \cdot v$.

Equation (7) gives the dynamic memory of the embedding layer. The first term $\mathrm{PPFactor}^{\mathsf{sched}}{}_i \cdot \frac{\mathrm{ActBytes(Embed)}}{tp\cdot sp}$ is the memory used by live embedding activations on stage i, scaled by the pipeline concurrency factor. We divide by tp and sp because if the embedding's output is partitioned across tp model-parallel ranks or sp sequence-parallel ranks, each rank holds only a fraction of the activations. We have ActBytes(Embed) = $B^{FP}act \cdot s \cdot b \cdot h$, since the embedding produces an h-dimensional activation for each of s tokens in each of b micro-batch samples. The second term CommTransBytes(Embed) accounts for any transient communication buffer needed for the embedding layer (for example, if using ZeRO partitioning, gathering the embedding weights, or if using model parallelism, all-gathering the token embeddings).

The expression above for CommTransBytes(Embed) takes the maximum of two possible communication contributions: (i) a data-parallel or optimizer-partitioning related buffer (labeled "DP/OP"), and (ii) a tensor-parallel all-gather of activations (labeled "TP gather"). The DP/OP term $\mathrm{comm}dp \cdot \frac{\mathrm{ParamBytes(Embed)}}{\mathrm{shard}_{\mathrm{emb}}\cdot sp}$ would be nonzero if, for example, ZeRO-3 partitioning is used for the embedding weight (so each rank initially has only $1/dp$ of the embedding and must all-gather it, or similarly must all-reduce the gradients) [16]. The TP term $\mathrm{comm}tp \cdot B^{FP}act, s, b, h \cdot \frac{tp-1}{tp\cdot sp}$ represents model-parallel communication: if the embedding output (of size $s \cdot b \cdot h$ per rank globally) is split across tp ranks, an all-gather of those activations might be needed so that each rank obtains the full h-dimensional embedding for each token. The factor $(tp-1)/(tp\cdot sp)$ indicates that each rank sends $(tp-1)/tp$ of the activation (and receives an equivalent amount) across the sp sequence groups (if $sp > 1$, multiple ranks share each portion). We take the maximum of these two under the assumption that the largest of these communication buffers will determine the transient memory.

Here $\text{comm}_{dp}, \text{comm}_{tp}$ are mode flags: for example, $\text{comm}_{dp} = 2$ or 3 might signal ZeRO-3 is in use, requiring gather and scatter operations, while $\text{comm}_{tp} = 1$ might signal that model parallel all-gather is needed.

2.3.4 *Output / LM head (with optional MTP).*

$$\text{Mem}_{\text{stat}}(\text{Output}) = \frac{\big(\text{Param}(\text{Output}) + n_{\text{MTP}}\text{Param}_{\text{MTP}}\big)\big(B_p^{\text{FP}} + B_{\text{grad}}^{\text{FP}} + 2B_{\text{os}}^{\text{FP}}\big)}{op \cdot sp}, \tag{9}$$

$$\text{Mem}_{\text{dyn}}(\text{Output}) = \text{PPFactor}_i^{\text{sched}} \cdot \frac{\text{ActBytes}(\text{Output}) + n_{\text{MTP}} \cdot \text{ActBytes}_{\text{MTP}}}{\text{shard}_{\text{out}}} + \text{CommTransBytes}(\text{Output}) + n_{\text{MTP}} \cdot \text{CommTransBytes}_{\text{MTP}}. \tag{10}$$

with Transformer-based

$$\text{Param}(\text{Output}) = h \cdot v + v, \qquad \text{ActBytes}(\text{Output}) = B_{act}^{FP} \cdot s \cdot b\,(v + h),$$

and DP/OP transients

$$\text{CommTransBytes}(\text{Output}) = \text{comm}_{dp}\ \frac{\text{ParamBytes}(\text{Output})}{tp \cdot sp}.$$

For multi-token prediction (MTP),

$$\text{Param}_{\text{MTP}} = 2h^2{+}4h{+}\text{Param}(\text{Embed}){+}\text{Param}(\text{Output}), \quad \text{ActBytes}_{\text{MTP}} = B_{act}^{FP}\ 3sbh{+}\text{ActBytes}(\text{Embed}){+}\text{ActBytes}(\text{Output}),$$

$$\text{CommTransBytes}_{\text{MTP}} = \text{comm}_{dp}\ \frac{\text{ParamBytes}_{\text{MTP}}}{tp \cdot sp}, \qquad \text{shard}_{\text{out}} \in \{1, tp\}.$$

Equation (9) is the static memory for the output layer plus an optional multi-token prediction module. $\text{Param}(\text{Output}) + n_{\text{MTP}}\text{Param}_{\text{MTP}}$ represents the total number of parameters in the output softmax/linear layer and the MTP block (if $n\text{MTP} > 0$ MTP blocks are present). This is multiplied by $(B^{FP}p + B^{FP}grad + 2B^{FP}os)$ and divided by $op \cdot sp$. Here op (with $op \le dp$) is the number of partitions for optimizer state. Thus, if $op > 1$, each rank holds only a $1/op$ fraction of the output (and MTP) parameters and states. The factor sp in the denominator reflects that if sequence parallelism is splitting the batch by sequences, the output weights might be further effectively replicated across sp groups. In a typical Transformer language model head, $\text{Param}(\text{Output}) = h \cdot v + v$, the weight matrix of shape $h \times v$ plus a bias or embedding tying vector of length v.

Equation (10) gives the dynamic memory for the output layer (and MTP). The first term scaled by $\text{PPFactor}^{\text{sched}}i.$ accounts for live activations in the output and MTP sections, scaled by pipeline concurrency. $\text{shardout} \in 1, tp$ indicates whether the output activations are replicated across all tp ranks or partitioned (e.g., if the output linear layer is model-parallel, then each rank only has $1/tp$ of the output activations and $\text{shard}out = tp$). The remaining terms $\text{CommTransBytes}(\text{Output}) + n_{\text{MTP}}\text{CommTransBytesMTP}$ are the transient communication buffers associated with the output and MTP parameters. For instance, $\text{CommTransBytes}(\text{Output}) = \text{comm}_{dp}\frac{\text{ParamBytes}(\text{Output})}{tp \cdot sp}$ corresponds to the data-parallel all-reduce of output gradients (since each rank has only $1/(tp \cdot sp)$ of the output parameters if model/context parallelism is applied, they must be synchronized across dp ranks after backward). Similarly, $\text{CommTransBytes}_{\text{MTP}}$ is the analogous term for the MTP parameters.

The displayed formulas for the MTP component provide an example of how we compute its parameter and activation sizes. Considering the MTP's extra Transformer layer as part of the number of layers L, the formulas only counts the rest of the component. In a multi-token prediction head that generates, say, k tokens per sequence at once, $\text{Param}_{\text{MTP}}$ includes additional linear and normalization operations. The formula $\text{Param}_{\text{MTP}} = 2h^2 + 4h + \text{Param}(\text{Embed}) + \text{Param}(\text{Output})$ is

a possible configuration counting linear and normalization weight matrices, plus reuse of embedding/output parameters; $\text{ActBytes}_{\text{MTP}} = B_{act}^{FP}(3s \cdot b \cdot h) + \text{ActBytes(Embed)} + \text{ActBytes(Output)}$ suggests the MTP might produce about 3× the usual activation volume (for example, if it involves an extra forward pass of similar complexity to the main network) in addition to reusing embedding/output activations. These specifics depend on the MTP design and are included for completeness. Their communication transient $\text{CommTransBytes}_{\text{MTP}}$ is defined similarly to the output layer's. In our model, we treat n_{MTP} as the number of extra output layers of this form; if none, $n_{\text{MTP}} = 0$ and these terms drop out. Also note $\text{shard}_{\text{out}} \in 1, tp$ indicates that the output of the MTP might or might not be model-parallel sharded (often it is not, so $\text{shard}_{out} = 1$ for the final logits).

2.3.5 *Transformer block (regular / recomputation).*

Regular (no recomputation).

$$Mem_{\text{stat}}(\text{NotRecLayer}) = \left(\frac{Param_{\text{Attn}} + Param_{\text{Norm}}}{op \cdot sp} + \frac{Param_{\text{FFn}}}{gcd(n^{rex}, op \cdot sp)}\right) \cdot (B_p^{FP} + B_{grad}^{FP} + 2B_{os}^{FP}), \tag{11}$$

$$\begin{aligned} Mem_{\text{dyn}}(\text{NotRecLayer}) = PPFactor_i^{sched}\Bigg(&\frac{Act_{Attn}^{QKV} + Act_{Attn}^{Score} + Act_{Attn}^{Proj}}{tp \cdot sp} \\ &+ \frac{Act_{router}^{MOE} + Act_{routed}^{MOE} + Act_{shared}^{MOE} + Act_{concat}^{MOE}}{tp \cdot sp} + \frac{Act_{\text{Norm}}}{tp \cdot sp}\Bigg) + CommTransB_i. \end{aligned} \tag{12}$$

with the transient bound in (4):

$$\text{CommTransBytes(NotRecLayer)} = \max\big(\text{Comm}_{\text{DP/OP}},\ \text{Comm}_{\text{TP}},\ \text{Comm}_{\text{SP}}\big) + \text{Comm}_{\text{EP}}.$$

Equation (11) gives the static memory for a standard Transformer layer without recomputation (denoted `NotRecLayer`). Inside the parentheses, $\frac{\text{ParamAttn+ParamNorm}}{op \cdot sp}$ represents the fraction of the attention and normalization parameters stored per rank (since ZeRO optimizer partitioning and context parallelism can shard those parameters across op and sp ranks respectively), and $\frac{\text{ParamFFN}}{\text{gcd}(n^{rex}, op \cdot sp)}$ represents the per-rank FFN parameter count. The $\text{gcd}(n^{rex}, op \cdot sp)$ in the denominator handles MoE scenarios: if $n^r ex$ experts are local to each rank and $op \cdot sp$ shards also partition those experts, the greatest common divisor gives the replication factor of FFN parameters per rank (for example, if $n^r ex = 4$ and $op \cdot sp = 2$, each rank holds FFN parameters for $4/\text{gcd}(4, 2) = 2$ experts). This sum is multiplied by $(B^{FP}p + B^{FP}grad + 2B_{os}^{FP})$ to convert to bytes (one copy each for weight and gradient, two for optimizer states).

Equation (12) gives the dynamic memory for the same layer. The term inside the large parentheses is the total activation bytes produced by this layer, broken into contributions from different sub-components: (1) the attention mechanism outputs (queries/keys/values, attention scores, and output projections), (2) the MoE expert outputs (router output, routed tokens to experts, any shared expert outputs, and the concatenated expert outputs), and (3) the normalization/utility outputs. Each of these sums is divided by $tp \cdot sp$ because we assume that if tensor or sequence parallelism is used, each rank only holds a portion of those activations. This total is then multiplied by $\text{PPFactor}^{\text{sched}}i$, reflecting that $f$pp micro-batches worth of these activations can be alive concurrently on stage i. Finally, CommTransB_i (a shorthand for CommTransBytes on that layer at stage i) is added, representing the transient communication buffer for this layer. As defined by Eq. (4), CommTransBytes(NotRecLayer) takes the maximum of the DP/OP, TP, and SP communication needs and adds any EP communication. Essentially, for each layer, we budget memory for whichever collective communication (gradient all-reduce, activation all-gather, etc.) is largest.

Selective recomputation (`SelRecLayer`*).* Keep (11); in (12) apply a mask $\mathrm{Rec}(\cdot) \in \{0,1\}$ to the activation buckets to drop kept-not-needed parts; typical recomputable ops include (batch) matmul, dropout, softmax, norm, gather, activation, and cast.

Full recomputation (`FullRecLayer`*).* Static unchanged; dynamic approximated by

$$\mathrm{Mem}_{\mathrm{dyn}}(\texttt{FullRecLayer}) = \mathrm{PPFactor}_i^{\mathrm{sched}} \ \frac{B_{act}^{FP} s \cdot b \cdot h}{\mathrm{shard}_{\mathrm{rec}}} + \mathrm{Comm}_{\mathrm{DP/OP}}, \qquad \mathrm{shard}_{\mathrm{rec}} \in \{1, t\}.$$

2.3.6 Attention buckets (Transformer-based). For multi-head, grouped-query, and multi-query attention:

$$\mathrm{Param}_{\mathrm{Attn}} = n_{\mathrm{MM}}^{\mathrm{Attn}} \cdot \tfrac{1}{2}\Big((h^2 + h) + (h \cdot d_h \cdot n_{kv} + h)\Big), \tag{13}$$

$$\mathrm{Act}_{\mathrm{Attn}}^{\mathrm{QKV}} = B_{act}^{FP}\, s \cdot b \left(\tfrac{1}{4}(n_{\mathrm{MM}}^{\mathrm{Attn}} + n_{p\mathrm{Cast}}^{\mathrm{Attn}})h + \tfrac{1}{2}(n_{\mathrm{MM}}^{\mathrm{Attn}} + n_{p\mathrm{Cast}}^{\mathrm{Attn}}) \cdot d_h n_{kv} + n_{\mathrm{BMM}}^{\mathrm{Attn}} \cdot d_h\right), \tag{14}$$

$$\mathrm{Act}_{\mathrm{Attn}}^{\mathrm{Score}} = B_{\mathrm{soft}}^{FP} \cdot n_{\mathrm{soft}} \cdot (s \cdot S_{FA} \cdot b \cdot n_h) \ + \ B_{\mathrm{drop}}^{FP} \cdot n_{\mathrm{drop}} \cdot (s \cdot S_{FA} \cdot b\, n_h), \tag{15}$$

$$\mathrm{Act}_{\mathrm{Attn}}^{\mathrm{Proj}} = B_{act}^{FP} \cdot s \cdot b \cdot h \cdot \tfrac{1}{4}(n_{\mathrm{MM}}^{\mathrm{Attn}} + n_{p\mathrm{Cast}}^{\mathrm{Attn}}) \ + \ B_{\mathrm{drop}}^{FP} \cdot n_{\mathrm{drop}}. \tag{16}$$

For multi-head latent attention (MLA), with compressed dims d_K^c, d_Q^c and RoPE head dim d_h^r:

$$\mathrm{Param}_{\mathrm{Attn}} = n_{\mathrm{MM}}^{\mathrm{Attn}}\Big(\tfrac{1}{2}(d_K^c(d_h \cdot n_{kv} + h)) + \tfrac{1}{4}(d_Q^c(n_h \cdot d_h + h + n_h \cdot d_h^r) + (h \cdot n_h \cdot d_h + h \cdot d_h^r))\Big), \tag{17}$$

$$\mathrm{Act}_{\mathrm{Attn}}^{\mathrm{QKV}} = B_{act}^{FP}\, s \cdot b \left(\tfrac{1}{4}(n_{\mathrm{MM}}^{\mathrm{Attn}} + n_{p\mathrm{Cast}}^{\mathrm{Attn}})\big((d_Q^c + 2n_h(d_h + d_h^r) + (d_h^r + n_{kv}(2d_h + d_h^r)) + (n_{kv} \cdot d_h + d_{kv}^c)\big) + n_{\mathrm{BMM}}^{\mathrm{Attn}} \cdot d_h\right). \tag{18}$$

The above formulas detail the memory contribution of various parts of the attention mechanism. In summary, $\mathrm{Param}_{\mathrm{Attn}}$ calculates the total number of attention parameters. For standard multi-head attention, it accounts for the Q, K, V projection matrices and the output projection matrix (each of dimension approximately $h \cdot h$ or $h \cdot d_h$). Factors like $\frac{1}{2}$ or $\frac{1}{4}$ depict the distribution of the activations across operators n_*^{Attn}. $\mathrm{Act}_{\mathrm{Attn}}^{QKV}$ is the total bytes of the Query, Key, and Value activations. It scales with s, b (tokens per micro-batch) and includes terms for each: the $\frac{1}{4}(n_{MM}^{Attn} + n^{Attn} pCast) \cdot h$ part corresponds to the Q (and similarly K) matrices, the $\frac{1}{2}(n_{MM}^{Attn} + n^{Attn} pCast) \cdot d_h n_{kv}$ corresponds to K/V which have $d_h \cdot n_{kv}$ elements per head, and $n_{BMM}^{Attn} \cdot d_h$ covers any batched matrix multiply outputs (like QK^T products) of width d_h. $\mathrm{Act}_{\mathrm{Attn}}^{Score}$ is the bytes of the attention score (softmax) and dropout outputs; it is proportional to $s \cdot S_{FA} \cdot b \cdot n_h$ (the size of the attention score matrix for n_h heads and the chunk of sequence S_{FA} (if tiling is applied on attention)) and multiplied by B_{soft}^{FP} or B_{drop}^{FP} and the number of such ops (n_{soft}, n_{drop}). Act^{Proj}Attn is the bytes of the output of the attention layer (projected back to hidden size h) plus any dropout applied to it; it scales with s, b, h and includes a proportion of 1/4 of $(n_{MM}^{Attn} + n_{pCast}^{Attn})$ for activations.

For multi-head latent attention (MLA), which involves compressed key/query representations and possibly Rotary Positional Embeddings (RoPE) of dimension d_h^r, the formulas adjust accordingly. The parameter count Param_{Attn} in MLA is lower because d_K^c and d_Q^c (compressed dimensions) replace h in some weight matrices, and there are additional terms for RoPE. The activation $\mathrm{Act}_{Attn}^{QKV}$ similarly becomes more complex, but essentially it shows that when keys/queries are compressed (smaller d_K^c, d_Q^c) and RoPE is applied, the activation size is reduced (fewer values per token to store) at the cost of slightly more complicated per-head computations (the formula inside the big parentheses captures that).

While these detailed formulas are used internally for precise accounting, a high-level understanding is that our model keeps track of each significant intermediate in the attention computation (Q, K, V, attention scores, outputs) and their contributions to memory. This allows it to adjust memory usage for variations like grouped-query attention

(where $n_{kv} < n_h$), or using FP32 for softmax (B^{FP}_{soft}) versus FP16 for other parts, etc., and to plug these into both the memory and communication cost estimates.

2.3.7 Feed-forward and Mistral of Experts (MoE).

Standard FFN (Transformer-based).

$$\text{Act}^{\text{MOE}}_{\text{router}} = \text{Act}^{\text{MOE}}_{\text{shared}} = \text{Act}^{\text{MOE}}_{\text{concat}} = 0, \quad \text{Act}^{\text{routed}}_{\text{expert}} = B^{FP}_{act} \cdot s \cdot b \cdot h_{ff} \left(\max(n^{\text{FFn}}_{\text{MM}}, n^{\text{FFn}}_{\text{BMM}}) + n^{\text{FFn}}_{p\text{Cast}} + 1 \right).$$

In a standard dense FFN (with no MoE), there are no MoE-specific activations. Thus $\text{Act}^{MOE}_{router}$, $\text{Act}^{MOE}_{shared}$, and $\text{Act}^{MOE}_{concat}$ are all zero. The only activation to consider from the FFN is the output of the hidden layer (which we treat as $\text{Act}^{routed}_{expert}$ for uniformity of notation). The formula above for $\text{Act}^{routed}_{\text{expert}}$ essentially calculates $B^{FP}_{act} \cdot s \cdot b \cdot h_{ff}$ (bytes for the h_{ff}-dimensional FFN hidden output for all $s \cdot b$ tokens) times a factor accounting for the operations that produce this output. The term $\max(n^{FFN}_{MM}, n^{FFN}_{BMM})$ corresponds to however many matrix multiplications are used in the FFN (e.g., one for a fused FFN or two for separate in/out projections), n^{FFN}_{pCast} covers any extra precision cast operations, and the +1 accounts for the activation function's output (ReLU, SwiGLU, ...). In simpler terms, we ensure that if the FFN has, say, two matmuls and one activation function, we count those intermediate outputs appropriately.

Mixture of Experts (MoE) FFN. Let $h_{ff} = h^{ex}_{ff}$.

$$\text{Param}_{\text{FFn}} = (n^r_{ex} + n^s_{ex}) \cdot \max(n^{\text{FFn}}_{\text{MM}}, n^{\text{FFn}}_{\text{BMM}}) \cdot (h \cdot h_{ff} + h_{ff}), \tag{19}$$

$$\text{Act}^{\text{MOE}}_{\text{router}} = B^{FP}_{act} \cdot s \cdot b \cdot (2n^r_{ex} + top_k), \tag{20}$$

$$\text{Act}^{\text{MOE}}_{\text{shared}} = B^{FP}_{act}\, n^s_{ex} \cdot s \cdot b \cdot h_{ff} \cdot \left(\max(n^{\text{FFn}}_{\text{MM}}, n^{\text{FFn}}_{\text{BMM}}) + n^{\text{FFn}}_{p\text{Cast}} + 1 \right), \tag{21}$$

$$\text{Act}^{\text{MOE}}_{\text{concat}} = B^{FP}_{act} \cdot s \cdot b \cdot h, \tag{22}$$

$$\text{Act}^{\text{MOE}}_{\text{routed}} = B^{FP}_{act} \cdot top_k \cdot s \cdot b \cdot h_{ff} \cdot \left(\max(n^{\text{FFn}}_{\text{MM}}, n^{\text{FFn}}_{\text{BMM}}) + n^{\text{FFn}}_{p\text{Cast}} + 1 \right). \tag{23}$$

In the MoE version of the FFN, Param_{FFN} multiplies the number of experts per rank (routed plus any shared experts) by the per-expert parameter count. Each expert is essentially an FFN of size $h \to hff \to h$, which has about $(h \cdot h_{ff} + h_{ff})$ parameters (weights plus biases for both layers). We multiply by $\max(n^{FFN}_{MM}, , n^{FFN}_{BMM})$ to account for whether one or two matrix multiplications define the FFN (similar reasoning as the dense case). Thus, for example, if each expert has two linear layers, this formula yields $(n^r_{ex} + n^s_{ex})$ times two sets of $(h \cdot h_{ff} + h_{ff})$ parameters.

The MoE activation terms are as follows. $\text{Act}^{MOE}_{router}$ is the gating output: for each of the $s \cdot b$ tokens, the router produces a probability or score for n^r_{ex} (the number of local experts) and perhaps additional top_k signals (like the indices or probabilities of the top-k selected experts). The factor $(2n^r_{ex} + top_k)$ is a proxy for the size of the router output per token (commonly, routers output n^r_{ex} probabilities and we might count the top-2 experts, so $2n^r_{ex}$ covers a one-hot or mask plus maybe two chosen indices).

$\text{Act}^{MOE}_{shared}$ is the activation output of any shared experts on that rank. There are n^s_{ex} shared experts (which each process all tokens, as opposed to routed experts where each token goes to only one). Each shared expert produces an h_{ff}-dimensional output per token. We multiply n^s_{ex}, s, b, h_{ff} by the same factor $(\max(n^{FFN}_{MM}, n^{FFN}_{BMM}) + n^{FFN}_{pCast} + 1)$ as in the dense case to account for intermediate outputs within the shared expert's computation.

$\text{Act}^{MOE}_{concat}$ is the output of the MoE layer after combining all expert outputs. After each token is processed by either one of n^r_{ex} experts or all n^s_{ex} shared experts, the results are summed/concatenated back into a single h-dimensional output per token. Thus we allocate $B^{FP}_{act} \cdot s \cdot b \cdot h$ bytes for this combined output.

$\text{Act}^{MOE}_{routed}$ is the activation output of the routed experts. Each token that is routed goes to top_k experts (often $top_k = 1$ or 2). Thus we consider $top_k \cdot s \cdot b$ "token-expert" combinations per micro-batch. Each such combination produces an hff-dimensional hidden activation (with the same factor $\max(n^{FFN}_{MM},, n^{FFN}_{BMM}) + n^{FFN}_{pCast} + 1$ accounting for the expert's internal layers). We multiply by B^{FP}_{act} to convert to bytes.

2.3.8 Normalization.

$$\text{Param}_{\text{Norm}} = n^{op}_{\text{norm}} \cdot 2h, \qquad \text{Act}_{\text{Norm}} = B^{FP}_{\text{norm}} \cdot n^{op}_{\text{norm}} \cdot s \cdot b \cdot h.$$

Normalization layers (e.g., LayerNorm or RMSNorm) have two parameters per instance: a scale and a bias, each of length h. Thus, if a model has n^{op}_{norm} normalization operations in total (including those in each layer), the total parameters are $2h$ per operation times n^{op}_{norm}. The activations from normalization are of the same shape as the input: for each normalization op, an h-dimensional output is produced for each of the $s \cdot b$ tokens. Therefore ActNorm $= n^{op}_{\text{norm}} \cdot s \cdot b \cdot h$ elements, each stored in B^{FP}_{norm} bytes (which could be FP16 or FP32 depending on implementation). In summary, the normalization contributions scale linearly with the number of normalization layers and have negligible additional structure beyond counting the vectors.

2.3.9 Transient byte formulas (consistent with the comm model). Let $etp \in \{1, t\}$ denote expert-model parallel, and $\text{comm}_d, \text{comm}_t, \text{comm}_c, \text{comm}_e \in \{0, 1, 2, 3\}$ be mode/masks. We use the following byte bounds to instantiate the subterms in (4):

Data Parallel/Optimizer Parallel:

$$\text{Comm}_{\text{DP/OP}} = \begin{cases} (\text{Param}_{\text{Attn}} + \text{Param}_{\text{Norm}})\left(\frac{1}{sp \cdot tp} + \frac{1}{tp}\right) + \text{Param}_{\text{FFn}}\left(\frac{1}{sp \cdot etp \cdot ep} + \frac{1}{\max(ep, etp)}\right), & \text{comm}_d = 2, \\ (\text{Param}_{\text{Attn}} + \text{Param}_{\text{Norm}})\left(\frac{1}{sp \cdot tp} + \frac{2}{tp}\right) + \text{Param}_{\text{FFn}}\left(\frac{2}{sp \cdot etp \cdot ep} + \frac{1}{\max(ep, etp)}\right), & \text{comm}_d = 3, \\ 0, & \text{otherwise.} \end{cases}$$

Tensor Parallel:

$$\text{Comm}_{\text{TP}} = \text{comm}_t \cdot \frac{1}{sp} \times \begin{cases} \frac{1}{4}\, n_{\text{gather}}\, s \cdot b\, (h + h_{ff}) \cdot \text{PPFactor}^{\text{sched}}_i, & n^r_{ex} = 1, \\ \frac{1}{4}\, n_{\text{gather}} \cdot B^{FP}_{act} \cdot h \cdot \left(h_{ff} \cdot n^{\text{FFn}}_{\text{MM}} \cdot \left(\frac{n^r_{ex}}{ep} + n^s_{ex}\right) + h \cdot n^{\text{Attn}}_{\text{MM}}\right), & n^r_{ex} > 1. \end{cases}$$

Expert Parallel:

$$\text{Comm}_{\text{EP}} = \text{comm}_e \cdot \text{PPFactor}^{\text{sched}}_i \cdot \frac{2 \cdot s \cdot b \cdot h}{sp \cdot t}.$$

Sequence Parallel:

$$\text{Comm}_{\text{CP}} = \text{comm}_s \cdot s \cdot b\, \left(3h \cdot n^{\text{Attn}}_{\text{MM}} + 2h_{ff} \cdot n^{\text{FFn}}_{\text{MM}}\right) \cdot \tfrac{1}{t},$$

All byte expressions above are linear functions of the same parameter and activation bucket counts defined for each layer, and they share the same shard divisors (dp, tp, sp, cp, ep) we have been using. Therefore, any change in a strategy's hyperparameters (s, b, m) or parallelization choices consistently affects both the dynamic memory usage and the communication volumes. For example, if we increase the micro-batch size b, then s, b increases linearly, which in turn linearly scales up both the activation memory in Eq. (3) and the activation-derived communication bytes in formulas like τ^{TP}_{ℓ}, τ^{SP}_{ℓ}, etc. Conversely, if we shard a dimension more (say increase tp), then terms like $\frac{1}{t}$ in the above expressions will reduce the memory per rank for that layer's activations and parameters, while the number of ranks participating in communications increases (reflected in different comm modes and multiplicities, as we'll see), which

might increase total communication time. This coherence between memory and communication models ensures that our strategy search (next section) can trade one for the other without inconsistency. Note also that increasing vpp (virtual pipeline parallelism) does not change the per-micro-batch payloads in any communication term (e.g., the boundary message size in Eq. (33) remains the same); it only changes how frequently such messages occur.

2.4 Communication cost model

From memory buckets to bytes. For a layer ℓ, let the memory buckets from Sec. 2.3 be $\mathrm{Param}(\ell)$, $\mathrm{Grad}(\ell)$, and $\mathrm{Act}_A(\ell)$ with $A \in \{\text{Attn:QKV, Attn:Score, Attn:Proj, FFN, MOE:router, MOE:routed, MOE:shared, MOE:concat, Norm/Util}\}$. Their byte sizes are

$$\mathrm{ParamBytes}(\ell) = \mathrm{Param}(\ell)\, B_p^{FP}, \quad \mathrm{GradBytes}(\ell) = \mathrm{Param}(\ell)\, B_{\mathrm{grad}}^{FP}, \quad \mathrm{ActBytes}_A(\ell) = \mathrm{Act}_A(\ell)\, B_{act}^{FP}.$$

In other words, we multiply each count by the number of bytes per element of that type (parameters, gradients, or activations) to get the total bytes. When a model dimension is sharded, the relationship between a *local shard* and the full *global* tensor is captured by explicit divisors (e.g., each rank might hold $1/(tp \cdot sp)$ of a fully sharded parameter, or $1/sp$ of a sequence-split activation). We introduced layer-specific divisors $\phi^{\mathrm{in/out}}tp(\ell) \in tp, 1$ and $\psi^{\mathrm{in/out}}\mathrm{seq}(\ell) \in sp, sp, 1$ earlier, which mark whether layer ℓ's inputs/outputs are in a sharded or global view for tensor or sequence parallelism. We also have two bucket sets $\mathcal{A}^{\mathrm{in}}\ell$ and $\mathcal{A}^{\mathrm{out}}\ell$ that enumerate which activation buckets each layer consumes as inputs and produces as outputs, respectively. We will use these to sum up activation bytes for communication.

Latency bandwidth model. For any collective pattern $\mathrm{pat} \in \{\text{allreduce, allgather, reducescatter, alltoall, p2p}\}$ over a group of size g and payload b bytes,

$$\tau_{\mathrm{coll}}(b;\, g, \mathrm{pat}) \;=\; \alpha_{\mathrm{pat}}(g) + \frac{b}{\mathcal{B}_{\mathrm{pat}}(g)}, \tag{24}$$

with $\alpha_{\mathrm{pat}}(g)$ (latency) and $\mathcal{B}_{\mathrm{pat}}(g)$ (effective bandwidth) set by topology/algorithm.

Per-micro-batch layer costs $\tau_\ell^{(\cdot)}(b)$. We now derive the per-micro-batch communication cost for each layer (or each parallel dimension) using the above model. All $\tau^{(\cdot)}\ell(b)$ are per-micro-batch costs (for clarity we omit the dependency on i or stage in the notation, as each layer ℓ is associated with a specific stage). Importantly, activation-derived bytes scale with the per-rank micro-batch size b (larger micro-batch means proportionally more activation data to communicate).

Data parallel (classic) and optimizer sharding (OP/ZeRO-3). Let g_{dp}=dp and $op \le dp$ be the optimizer partition count.

$$\tau_\ell^{\mathrm{DP}} = \tau_{\mathrm{coll}}\Big(b{=}\tfrac{\mathrm{GradBytes}(\ell)}{tp\cdot sp};\quad g \;{=}g_{dp},\quad \mathrm{pat{=}allreduce}\Big), \tag{25}$$

$$\tau_\ell^{\mathrm{OP\,AG}} = \tau_{\mathrm{coll}}\Big(b{=}\tfrac{\mathrm{ParamBytes}(\ell)}{tp\cdot sp};\quad g{=}op,\quad \mathrm{pat} \;{=}\mathrm{allgather}\Big), \tag{26}$$

$$\tau_\ell^{\mathrm{OP\,RS}} = \tau_{\mathrm{coll}}\Big(b{=}\tfrac{\mathrm{GradBytes}(\ell)}{tp\cdot sp};\quad g \;{=}op,\quad \mathrm{pat} \;{=}\mathrm{reducescatter}\Big). \tag{27}$$

Equation (25) is the time to perform an all-reduce on layer ℓ's gradients across the dp data-parallel ranks (classic data-parallel gradient synchronization). The payload b is the number of bytes of gradients each rank has for that layer, which is $\mathrm{GradBytes}(\ell)$ divided by $tp \cdot sp$ (since if tensor or sequence parallelism is applied, each rank only has $1/(tp \cdot sp)$ of the layer's gradients). The collective group size is $g_{dp} = dp$. Equations (26) and (27) are the additional costs introduced by ZeRO-3 style optimizer partitioning. $\tau^{\mathrm{OP;AG}}\ell$ is the time to all-gather the layer's parameters (of size $\mathrm{ParamBytes}(\ell)$ per full model) across op ranks (each rank originally has $1/op$ of the parameters if op groups are used). This occurs at the start of the forward pass for that layer. $\tau^{\mathrm{OP;RS}}\ell$ is the time to reduce-scatter the layer's gradients

across those op ranks after backward (each rank ends up with the $1/op$ portion of gradients to update its partition of the parameters). In many cases $op = dp$ (full partitioning across all data ranks), but we allow $op < dp$ to model partial ZeRO (in which case dp/op ranks form each partition group). These operations use group size $g = op$ and payloads similarly divided by $tp \cdot sp$ if model parallelism or sequence parallelism also split the gradients/parameters.

Tensor/model parallel (TP). With indicators $\chi_\ell^{\text{in-gather}}, \chi_\ell^{\text{out-red}} \in \{0, 1\}$,

$$\tau_{\ell,\text{AG}}^{\text{TP}}(b) = \chi_\ell^{\text{in-gather}}\ \tau_{\text{coll}}\Big(b = \tfrac{1}{\phi_{tp}^{\text{in}}(\ell)} \sum_{A \in \mathcal{A}_\ell^{\text{in}}} \text{ActBytes}_A(\ell);\ g{=}tp,\ \text{pat=allgather}\Big), \tag{28}$$

$$\tau_{\ell,\text{AR}}^{\text{TP}}(b) = \chi_\ell^{\text{out-red}}\ \tau_{\text{coll}}\Big(b = \tfrac{1}{\phi_{tp}^{\text{out}}(\ell)} \sum_{A \in \mathcal{A}_\ell^{\text{out}}} \text{ActBytes}_A(\ell);\ g{=}tp,\ \text{pat=allreduce}\Big), \tag{29}$$

$$\tau_\ell^{\text{TP}}(b) = \tau_{\ell,\text{AG}}^{\text{TP}}(b) + \tau_{\ell,\text{AR}}^{\text{TP}}(b). \tag{30}$$

If a layer ℓ is tensor-parallel sharded (for example, a linear layer whose weight matrix is split across tp GPUs), then before computing the layer we may need to all-gather the input activations from the tp ranks (so that each rank has the full input). This cost is $\tau^{TP}\ell$, AG. We determine the payload by summing the sizes of all input activation buckets $\mathcal{A}^{in}\ell$ for that layer (e.g., for a linear layer this might just be the output of the previous layer, whereas for attention it might include multiple inputs like Q, K, V), then dividing by $\phi^{in}tp(\ell)$. If ℓ's input was originally split across tp ranks, then $\phi^{in}tp(\ell) = tp$ and $1/\phi^{in}tp(\ell)$ gives the fraction of global input that each rank initially had, thus $\frac{1}{\phi^{in}tp(\ell)} \sum_{A \in \mathcal{A}^{in}\ell} \text{ActBytes}A(\ell)$ is the bytes each rank must send (and receive) in the all-gather. If ℓ's inputs were not split ($\phi^{in}tp(\ell) = 1$), then no all-gather is needed (which is also signaled by $\chi^{in-gather}\ell = 0$ in that case). Similarly, $\tau^{TP}\ell$, AR covers a possible all-reduce after the layer. For instance, if layer ℓ produced outputs that were model-parallel replicated across ranks (like a bias gradient that every rank computed independently and now needs to be summed), we perform an all-reduce over tp ranks on that output. The payload is the sum of the relevant output activation buckets (divided by $\phi^{out}tp(\ell)$ if that output is split among ranks). We gate these costs with $\chi^{\text{out-red}}\ell$, which would be 1 if, say, the layer produced some globally shared value (like a fully-connected layer's bias term's gradient). In practice, many layers have either an input gather or an output reduce or both. We sum $\tau^{TP}AG$ and $\tau^{TP}AR$ to get the total tensor-parallel comm cost $\tau^{TP}\ell$.

Sequence (SP). With indicators $\chi_\ell^{\text{seq-gather}}, \chi_\ell^{\text{seq-red}} \in \{0, 1\}$,

$$\begin{aligned}\tau_\ell^{\text{SP}}(b) = {} & \chi_\ell^{\text{seq-gather}}\ \tau_{\text{coll}}\Big(b = \tfrac{1}{\psi_{\text{seq}}^{\text{in}}(\ell)} \sum_{A \in \mathcal{A}_\ell^{\text{seq}}} \text{ActBytes}_A(\ell);\ g{=}sp,\ \text{pat=allgather}\Big) \\ & + \chi_\ell^{\text{seq-red}}\ \tau_{\text{coll}}\Big(b = \tfrac{1}{\psi_{\text{seq}}^{\text{out}}(\ell)} \sum_{A \in \mathcal{A}_\ell^{\text{seq}}} \text{ActBytes}_A(\ell);\ g{=}sp,\ \text{pat=reducescatter}\Big).\end{aligned} \tag{31}$$

This is analogous to the tensor parallel case but for sequence (or context) parallelism. If layer ℓ has its sequence input split across sp ranks (e.g., each rank processes only a subset of the sequence positions), then before the layer we might need to gather the full sequence on each rank. $\psi^{in}seq(\ell)$ is either sp depending on which parallelism is used (or their product if both apply), or 1 if the inputs are not sequence-sharded. We sum the input activations and divide by $\psi^{in}seq(\ell)$ to get the bytes per rank to all-gather. Similarly, if the outputs were sequence-sharded and need to be combined (e.g., for a subsequent all-reduce), we include the reduce-scatter term. These communications happen over sp ranks.

Expert parallel (EP, MoE). Let $\phi_{\text{route}}(\ell) \in [0,1]$ be the fraction of routed activations that leave the rank (given $(ep, top_k, n^r_{ex}, n^s_{ex}, \texttt{moe\_cap})$).

$$\tau^{\text{EP}}_{\ell}(b) = \tau_{\text{coll}}\Big(b = 2\,\phi_{\text{route}}(\ell)\,\text{ActBytes}_{\text{MOE:routed}}(\ell);\ g{=}ep,\ \text{pat=alltoall}\Big). \tag{32}$$

This means that for MoE layer ℓ, a fraction ϕ_{route} of the token activations (those routed to other ranks' experts) are sent, and an equal fraction are received (hence $2, \phi_{\text{route}}$ portion of the total bytes of routed activations are exchanged). The group size is ep (all expert-parallel ranks participate in the all-to-all).

Pipeline boundary (PP). Let $\mathcal{U}_i$ be the set of point-to-point boundary transfers on stage i. With shard divisor $\varphi_{tp,sp} \in \{tp, sp, tp \cdot sp, 1\}$ on the boundary tensor,

$$\tau^{\text{PP}}_{i}(b) = \sum_{u \in \mathcal{U}_i} \tau_{\text{coll}}\Big(b = \tfrac{\text{BoundaryBytes}(i \to i+1, u)}{\varphi_{tp,sp}};\ g{=}1,\ \text{pat=p2p}\Big). \tag{33}$$

This sums the times for all point-to-point sends from stage i to stage $i+1$ for one micro-batch. Each such boundary message u (e.g., the activations output by a certain layer that gets sent to the next device) has size $\text{BoundaryBytes}(i \to i+1, u)$ bytes if it were not sharded. We divide by $\varphi_{tp,sp}$ to account for any model or context parallel partitioning of that message.

2.5 Strategy solving

Decision variables and constraint. A strategy is $s = (dp, tp, pp, vpp, sp, ep, op, \mathcal{R}, b, \texttt{sched})$ with $\texttt{sched} \in \{\text{1F1B}, \text{SeqPipe}, \text{DualPipeV}\}$, recomputation policy $\mathcal{R}$, schedule Sched and optimizer partition $op \le dp$. We jointly choose per-rank micro-batch size $b \in \mathbb{N}_+$ and accumulation steps $m \in \mathbb{N}_+$ under the global-batch constraint

$$G = dp \cdot b \cdot m. \tag{34}$$

Communication multiplicities. Let $\rho_{\cdot}(m)$ be how many times a collective appears in one training step:

$$\rho_{\text{TP}} = \rho_{\text{SP}} = \rho_{\text{EP}} = \rho_{\text{PP}} = \rho_{\text{OP AG}} = m, \qquad \rho_{\text{DP AR}} = \rho_{\text{OP RS}} = 1. \tag{35}$$

These match the semantics: TP/SP/SP/EP collectives and PP P2P are *per micro-batch*; classic DP all-reduce and OP reduce-scatter are *once per step*

Per-step communication time. Using the per-micro-batch layer costs from Sec. 2.4,

$$T^{\text{comm}}_{\text{step}}(s,b,m) = m \sum_i \tau^{\text{PP}}_i(b) + \sum_{\ell \in \mathcal{L}} \big[m\,\tau^{\text{TP}}_{\ell}(b) + m\,\tau^{\text{SP}}_{\ell}(b) + m\,\tau^{\text{EP}}_{\ell}(b) + m\,\tau^{\text{OP AG}}_{\ell} + \tau^{\text{OP RS}}_{\ell} + \tau^{\text{DP}}_{\ell}\big], \tag{36}$$

where $\mathcal{L}$ is the layer set. Equation (36) is a no-overlap bound; a schedule-aware overlap map can refine it without changing bytes or multiplicities.

Bubble term. Let $\Theta_{\text{mb}}(b)$ be the *per-micro-batch compute time* (approximately linear in b from operator enumerations). Let $g_{\texttt{sched}}(pp, vpp, m, \Delta_{\text{imb}}(s))$ be a dimensionless schedule factor (decreasing in m) and $\Delta_{\text{imb}}(s)$ a normalized stage-imbalance measure. With $c_{\text{recomp}}(\mathcal{R})$ the recomputation overhead in micro-batch-equivalents,

$$\mathcal{B}(s,b,m) = \Theta_{\text{mb}}(b)\,\Big[g_{\texttt{sched}}(pp, vpp, m, \Delta_{\text{imb}}(s)) + c_{\text{recomp}}(\mathcal{R})\Big]. \tag{37}$$

Feasibility and objective. We require steady-state schedulability and per-accelerator memory feasibility:

$$\textsc{ScheduleValidWith}(pp, vpp, \mathsf{sched}, m), \qquad \max_i \ \mathrm{Peak}_{\mathrm{mem},i}(s;\, b, m) \ \le \ M_{\mathrm{budget}}, \tag{38}$$

where $\mathrm{Peak}_{\mathrm{mem},i}$ is given by Eq. (2). Since useful compute per step is invariant, we minimize

$$J(s, b, m) = \mathcal{B}(s, b, m) + T^{\mathrm{comm}}_{\mathrm{step}}(s, b, m) \tag{39}$$

subject to (34) and (38).

Solution sketch. Let η be any residual tiling factor (often η=1). Enumerate structural tuples (pp, dp, tp, ep) such that $pp \cdot dp \cdot tp \cdot ep \cdot \eta = N$. For each, try legal vpp, $op \mid dp$, sched, and $\mathcal{R}$. Set $Q = G/dp$ and jointly scan integer pairs (b, m) with $m \in \mathrm{Divisors}(Q)$ and $b = Q/m$. Keep only candidates that satisfy (38); score them by (39) and return the minimizer $(s^\star, b^\star, m^\star)$. This joint solve captures the couplings: increasing b raises per-call payloads but reduces m (fewer per-micro-batch collectives and smaller bubble), while vpp reduces g_{sched} but raises activation liveness already reflected in $\mathrm{Peak}_{\mathrm{mem},i}$ [14].

Consistency check. By construction, all payloads in (36) are linear forms of the same buckets appearing in Sec. 2.3, with identical shard divisors (dp, tp, sp, ep). Thus any change in (s, b, m) affects memory and communication coherently: if a modification reduces dynamic memory by sharding a dimension, the corresponding collectives in (36) increase (more traffic or more calls), and vice versa.

Algorithm: joint memory-feasible, communication-aware search. We turn the formulation in Eqs. (34)–(39) into a concrete search that (1) enumerates structural degrees (dp, tp, pp, ep) and legal vpp, $op \,|\, dp$, schedules and recomputation policies; (2) jointly scans integer pairs (b, m) that satisfy the global-batch constraint $G = dp \cdot b \cdot m$; (3) enforces memory feasibility using the peak estimator in Eq. (2); and (4) scores feasible candidates with the objective in Eq. (39), where the communication time $T^{\mathrm{comm}}_{\mathrm{step}}$ is given by Eq. (36). All byte payloads are linear in the same buckets as the memory model, and multiplicities follow Eq. (35), ensuring consistency between memory and communication.

3 Experiments

We evaluate PRISM across four widely used dense model families—DeepSeek3, Llama2/3, Mistral, and Multi-modal model–TextHawk and one sparse architecture, Llama-MoE. This suite allows us to assess PRISM's scalability and generality under heterogeneous model architectures.

Hardware setup. Experiments are conducted on a Huawei Ascend-910 (A910) AI cluster, scaling up to a maximum of 1,024 NPUs. Each compute node contains 8× A910–64GB NPUs. Intra-node communication adopts a mesh topology via HCCL [1], delivering up to 392 GB/s aggregate bandwidth per NPU. For inter-node communication, multiple A910 nodes are connected in a RoCE-based ring, with each interface providing 25 GB/s of unidirectional link bandwidth.

Software setup. We integrate our PRISM into MindSpore [2] and execute deep learning models within this framework. Using the framework's profiling utilities and device counters, we report per-run *step time* (ms/step) and *peak HBM memory usage* (GB). Unless otherwise specified, measurements exclude warm-up and are averaged over steady-state steps.

Algorithm 1 PRISM-Search

Require: $\mathcal{L}$, M_{budget}, G, candidate sets for $(s = (dp, tp, pp, vpp, sp, ep, op, \mathcal{R}, b, \text{sched})$
Ensure: $(s^\star, b^\star, m^\star)$ minimizing J

$J^\star \leftarrow +\infty$, $(s^\star, b^\star, m^\star) \leftarrow \varnothing$
for all (dp, tp, pp, ep) **do**
  for all vpp **do**
    for all $op \mid dp$ **do**
      for all sched **do**
        for all $\mathcal{R}$ **do**
          $Q \leftarrow G/dp$
          for all $m \in \text{Divisors}(Q)$ **do**
            $b \leftarrow Q/m$
            if not ScheduleValidWith$(pp, vpp, \text{sched}, m)$ **then**
              continue
            end if
            $s \leftarrow (dp, tp, pp, vpp, sp, ep, op, \text{sched}, \mathcal{R})$
            $\text{PeakMem}_i \leftarrow$ PeakMemByStage$(\mathcal{L}, s, b, m)$
            if $\max_i \text{PeakMem}_i > M_{\text{budget}}$ **then**
              continue
            end if
            $T^{\text{comm}}_{\text{step}} \leftarrow$ CommStep$(\mathcal{L}, s, b, m)$
            $\mathcal{B} \leftarrow$ Bubble(s, b, m)
            $J \leftarrow \mathcal{B} + T^{\text{comm}}_{\text{step}}$
            if $J < J^\star$ **or** $(J = J^\star$ **and** $\max_i \text{PeakMem}_i$ smaller**)** **then**
              $(s^\star, b^\star, m^\star) \leftarrow (s, b, m)$; $J^\star \leftarrow J$
            end if
          end for
        end for
      end for
    end for
  end for
end for
return $(s^\star, b^\star, m^\star)$

3.1 Evaluation of memory prediction accuracy

We evaluate PRISM's per-device peak HBM memory prediction against runtime measurements. Accuracy is computed as

$$\text{Acc} = 100 \times \left(1 - \frac{|\text{Pred} - \text{Real}|}{\text{Real}}\right)\%,$$

and we also report mean absolute percentage error (MAPE). Tables 2 and 3 list configuration details and the predicted (Pred) versus measured (Real) peaks in MB; for readability, discussion below converts absolute errors to GB.

DeepSeek3 scaling from 64 to 1,024 devices (Table 2). Across the five configurations, mean accuracy is 92.03% (median 91.52%; range 86.11–98.72%); the corresponding MAPE is 7.97%. The mean (median) absolute error is 4.14 GB (5.05 GB); RMSE is 4.70 GB with a minimum error of 0.46 GB and a maximum of 7.01 GB. The DualPipe case at 128 devices (row 2) shows the largest deviation (86.11% accuracy), consistent with additional warm-up/flush micro-steps not overfit by our profiling-free model. Restricting to 1F1B rows (4/5 cases), mean accuracy improves to 93.51% (MAPE 6.49%). At the largest scale, accuracy remains high: 94.54% at 1,024 devices (row 5).

Cross-model generalization at varied scales (Table 3). Over eight configurations spanning Llama2/3, Mistral, MoE, and Texthawk, mean accuracy is 95.80% (median 96.42%; range 87.72–99.82%) with MAPE 4.20%. The mean (median) absolute error is 1.80 GB (1.42 GB); RMSE is 2.41 GB; the smallest error is 0.08 GB (Llama3, 80 layers, 64 devices), and the largest is 5.03 GB (Llama2, 32 layers, 8 devices). Errors are balanced (4 underestimates vs. 4 overestimates). At extreme scale, Llama-MoE on 1,024 devices retains high accuracy (97.05%).

Discussion. Together, Table 2 (intra-model scaling) and Table 3 (inter-model diversity) show that PRISM achieves robust peak-memory prediction across schedulers (1F1B, DualPipe), parallelism mixes (DP/TP/PP/EP/OP/SP with/without VPP), and cluster sizes up to 1,024 devices, all without profiling. The largest deviations occur in schedule variants with additional warm-up/flush and chunk-boundary effects, whereas mainstream 1F1B settings track closely to measurements. Despite intentionally excluding communication buffers in our memory model, PRISM's predictions are sufficiently accurate to act as a reliable feasibility constraint for strategy planning at scale.

Num layers	Num device	DP	TP	PP	EP	SP	OP	VPP	GBS	Sched	Pred	Real	Accuracy(%)
9	64	8	8	1	64	1	8	1	8	1f1b	36032	36498	98.72
16	128	32	2	2	64	1	32	2	1920	dualpipe	44542	51724	86.11
62	256	256	1	1	256	1	256	1	8192	1f1b	43640	48899	89.24
60	512	4	8	16	8	1	4	1	512	1f1b	55765	60931	91.52
62	1024	128	4	2	256	1	8	1	8192	1f1b	60707	57563	94.97

Table 2. Configurations for DeepSeek3. Reported Pred and Real are per-device peak HBM memory in MB

Model	Num layers	Num device	DP	TP	PP	EP	SP	OP	VPP	GBS	Sched	Pred	Real	Accuracy(%)
Llama2	32	8	1	2	4	1	1	1	1	256	1f1b	47137	41983	87.72
Llama2	80	64	2	4	8	1	1	2	1	256	1f1b	55380	54899	99.12
Llama3	80	64	1	8	8	1	1	1	1	256	1f1b	46320	46236	99.82
Llama3	80	128	1	8	8	1	2	1	1	256	1f1b	43013	46718	92.07
Mistral	32	16	8	1	2	8	1	8	1	128	1f1b	48420	47956	99.03
Llama-MoE	70	1024	32	2	16	8	1	32	1	1536	1f1b	50150	51675	97.05
Texthawk	(6, 1, 4, 4)	8	2	2	2	1	1	2	1	4	1f1b	31728	33120	95.80
Texthawk	(26, 1, 4, 61)	512	8	4	16	4	1	8	1	128	1f1b	44307	46265	95.77

Table 3. Consolidated configurations across model families. Metrics follow Table 2.

Model	Devices	Megatron-LM MFU (%)	PRISM MFU (%)	Speedup (×)
DeepSeek3	128	29.40	36.20	1.23×
Llama-MoE	128	13.63	19.50	1.43×
Llama-MoE	256	19.90	25.20	1.27×

Table 4. MFU comparison between the baseline (Megatron-LM) and PRISM across models and scales. Speedup is PRISM / baseline.

3.2 Performance Evaluation

We assess PRISM using Model FLOPs Utilization (MFU, %), a standard proxy for training efficiency. As shown in Table 4, PRISM consistently improves MFU across models and scales.

On DeepSeek3 with 128 devices, MFU rises from 29.40% to 36.20% with 1.23× speedup. For Llama-MoE at 128 devices, it increases from 13.63% to 19.50% with 1.43× speedup. Even at 256 devices, Llama-MoE improves from 19.90% to 25.20% with 1.27× speedup. Overall, these results correspond to 1.23×–1.43× speedups.

4 Conclusion

In this paper, we proposed PRISM, the first profiling-free symbolic planner that unifies memory and communication modeling to automatically select optimal parallelization strategies for large-scale DNN training. PRISM captures the intrinsic memory behavior of layered architectures through a symbolic grammar and extends it to account for diverse parallelism and optimization techniques. This enables accurate peak memory prediction and principled exploration of the trade-off between pipeline bubbles and communication costs. Our results show that PRISM consistently improves utilization and scalability across different models and device scales. Moving forward, we plan to extend PRISM to a broader range of model architectures and heterogeneous training environments, and to integrate interactive visualization to help users better understand how parallel strategies impact memory and performance.

References

[1] 2025. Huawei Collective Communication Library (HCCL). https://gitee.com/ascend/cann-hccl. Accessed: 2025-09-20.

[2] 2025. MindSpore: An Open Source Deep Learning Training/Inference Framework. https://gitee.com/mindspore/mindspore. Accessed: 2025-09-20.

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245 [cs.CL] https://arxiv.org/abs/2305.13245

[4] Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. 2024. vTrain: A Simulation Framework for Evaluating Cost-effective and Compute-optimal Large Language Model Training. arXiv:2312.12391 [cs.LG] https://arxiv.org/abs/2312.12391

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. *CoRR* abs/1604.06174 (2016). arXiv:1604.06174 http://arxiv.org/abs/1604.06174

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/6aca97005c68f1206823815f66102863-Paper.pdf

[7] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.* 23, 1, Article 120 (Jan. 2022), 39 pages.

[8] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism.* Curran Associates Inc., Red Hook, NY, USA.

[9] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Reza Yazdani Aminadabi, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2024. System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing* (Nantes, France) *(PODC '24)*. Association for Computing Machinery, New York, NY, USA, 121–130. doi:10.1145/3662158.3662806

[10] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing Activation Recomputation in Large Transformer Models. In *MLSys*. https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html

[11] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. {GS}hard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *International Conference on Learning Representations*. https://openreview.net/forum?id=qrwe7XHTmYb

[12] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism. *Proc. VLDB Endow.* 16, 3 (Nov. 2022), 470–479. doi:10.14778/3570690.3570697

[13] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3341301.3359646

[14] 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. 2021. 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* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. doi:10.1145/3458817.3476209

[15] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 18332–18346. https://proceedings.mlr.press/v162/rajbhandari22a.html

[16] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 20, 16 pages.

[17] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need. arXiv:1911.02150 [cs.NE] https://arxiv.org/abs/1911.02150

[18] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019). arXiv:1909.08053 http://arxiv.org/abs/1909.08053