MemX: Memory-Aware Automatic Parallelism for Cross-Vendor Heterogeneous GPU Training

Lin Meng[a,b], Yuzhong Sun[a,*] and Jie Zhu[c]

[a]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China
[b]University of Chinese Academy of Sciences, 101408, China
[c]School of Computer Science, Nanjing University of Posts & Telecommunications, 210023, China

ARTICLE INFO

Keywords:
Automatic parallelism
Heterogeneous GPU training
Memory-aware optimization
Performance modeling
Pipeline parallelism
Activation recomputation

ABSTRACT

Cross-vendor heterogeneous GPU clusters are increasingly used for large-scale model training. However, their GPUs differ substantially in compute power, memory capacity, and interconnect characteristics. Existing automatic parallelism training systems usually minimize iteration time while treating memory as a hard constraint, almost failing in resolving the compute–memory asymmetry of cross-vendor clusters regardless correlation between memory and throughput. In this paper, we present MemX, a new memory-aware automatic parallelism system that jointly optimizes iteration time, peak memory usage, and cross-GPU memory balance. MemX consists of a weighted-product objective function, heterogeneous recomputation strategies, a minimal-set profiler, interpolation-based data augmentation, a dual-output XGBoost predictor, and a hash-signature prediction cache, in order to accelerate search of large-scale auto-parallel best configurations at lower cost. We implement MemX on Megatron-LM and evaluate it against four automatic parallelism baselines on two heterogeneous clusters. MemX achieves the lowest prediction Mean Absolute Percentage Error (MAPE) of 12.17% and is the only method that identifies the real Top-1 configuration under both time-based and memory-balance rankings. In terms of search cost and search space, MemX searches 31,240 configurations in 26.5 seconds, while an end-to-end profiling baseline is extrapolated to require about 17 days at the same scale. The simulation study that extends the measured 20-GPU data to 40-GPU and 60-GPU clusters further illustrates the effectiveness of MemX at larger scales.

arXiv:2610.41525v2 [cs.DC] 13 Nov 2026

1. Introduction

The rapid growth of large language models (LLMs), such as GPT-5 and GLM-5 [31, 45], has created a huge increase in computational demand. Training these models, which comprise hundreds of billions to trillions of parameters, typically requires thousands of high-performance GPUs operating continuously for months [36, 37]. Current approaches generally deploy large-scale homogeneous clusters of NVIDIA GPUs, benefiting from their mature software ecosystem. However, this approach faces several limitations: geopolitical tensions and supply chain shortages have made building large-scale homogeneous clusters increasingly difficult, while the high costs and energy consumption of cutting-edge accelerators compel organizations to seek alternative hardware solutions.

Therefore, the emergence of cross-vendor heterogeneous GPU clusters has become an inevitable trend in large-scale model training. It is essential to distinguish between traditional intra-vendor generational heterogeneity (e.g., mixing NVIDIA V100, A100, and H100) and cross-vendor architectural heterogeneity (e.g., AMD, Huawei Ascend, Biren) [1, 10, 11], where GPUs differ in instruction sets, memory architectures, and interconnect technologies. The latter presents fundamentally distinct system challenges that existing training frameworks fail to handle effectively.

*Corresponding author at: Institute of Computing Technology, Chinese Academy of Sciences, 100190, China.
yuzhongsun@ict.ac.cn (Y. Sun)
ORCID(s): 0000-0002-0362-4420 (Y. Sun)

To bridge this hardware diversity and achieve efficient training, automatic parallelism strategies capable of adapting to heterogeneous GPU capabilities are required. Current automatic parallelism systems for intra-vendor heterogeneous GPUs commonly employ pipeline parallelism (PP) [8, 23] partitioning as the core mechanism for load balancing: allocating layers proportionally to device compute capacity [15, 38, 43, 9], where high-performance GPUs assume more layers and low-performance devices assume fewer. This strategy performs well in intra-vendor generational heterogeneity scenarios because device performance is typically positively correlated with memory capacity. For instance, compared with the NVIDIA V100 (125 TFLOPS/32GB), the NVIDIA A100 (312 TFLOPS/80GB) exhibits exhibits a 2.5× increase in both computing power and memory, with roughly matching ratios.

Table 1
Heterogeneous PP partitioning failure.

Strategy	Layers (X:Y)	TFLOPS	Vendor X Mem%	Vendor Y Mem%
Compute	21:11	*OOM**	>100	—
Memory	8:24	55	88	78
Vendor X only	8:0	70	88	—
Vendor Y only	0:24	86	—	78

* Out-of-Memory.

However, the memory-performance asymmetry in cross-vendor heterogeneous scenarios renders such a partitioning mechanism ineffective. We validate this observation on a

employed cluster with Vendor X (∼300 TFLOPS/32GB) and Vendor Y (∼256 TFLOPS/64GB), using partitioning ratios derived from the effective capabilities measured on real training workloads (i.e., LLaMa 2-7B with 32 layers) rather than the nominal specifications. Although the nominal peak-compute ratio is only 300:256 (≈1.17:1), differences in kernel efficiency and software-stack maturity give X about 1.9× the effective training throughput of Y, i.e., a 21:11 compute ratio; likewise, despite the 32:64 (1:2) nominal memory ratio, the same workload exhibits different memory footprints on the two vendors' software stacks, so the memory-feasible assignment is 8:24 (1:3) rather than 11:21. As shown in Table 1, the 21:11 computing-proportional partition exceeds X's 32GB memory triggering Out-of-Memory, while the 8:24 memory-proportional partition runs but achieves merely 55 TFLOPS, far below the homogeneous baselines of 70 (X-only) and 86 (Y-only) TFLOPS. The culprit is a stage-time mismatch in PP: at these standalone rates, X computes its 8 layers in only (8/70)/(24/86) ≈ 41% of the time which Y takes for its 24 layers, idling for about 59% of each pipeline cycle, while Y becomes the bottleneck despite 78% memory utilization; overall, the pipeline harvests only 35% of the two devices' combined 156 TFLOPS (70+86), wasting about 65% (101 TFLOPS) of the available computing power. As the computing/memory partitioning mechanisms optimize only a single dimension of load balancing, they are inherently incapable of handling the dual constraints of computing power and memory simultaneously [15, 38, 43, 9].

Beyond the memory-performance asymmetry, cross-vendor heterogeneity also introduces severe challenges in performance modeling. Automatic parallelism requires finding the optimal solution within a vast configuration space [13, 22] determined by multiple dimensions, including parallelization strategies (data/tensor/pipeline parallelism [25], etc.), model partitioning strategies, recomputation strategies [4, 14], and other training parameters (such as micro-batch size and number of attention heads), resulting in search state explosion. For example, LLaMA 2/3 [37, 6] layers range from 32 upto 80. For a 32-layer Llama 2 model on only 16 GPUs, with 3 parallelization modes, 3 recomputation strategies, and 5 categories of model parameters (each with about 3 values), the configuration space already reaches the order of 10^8, while the 80 layers of LLaMA 3 has 10^9 configuration space. Exhaustively profiling this space would require decades, rendering it clearly infeasible.

Existing automatic parallelism systems typically employ performance models based on sparse sampling or search pruning to reduce overhead [49, 38, 35, 16]. However, the limited profiling budget conflicts with the discrete-continuous hybrid feature space in cross-vendor scenarios (GPU types, batch sizes, sequence lengths, etc.), where sampling sparsity leads to significant degradation in prediction accuracy. Furthermore, current approaches cannot predict performance for configurations outside the sampled range. When new GPUs are added, the entire profiling process must be repeated, resulting in low deployment efficiency and poor adaptability to expanding heterogeneous clusters.

To address these challenges, we present MemX, a memory-aware automatic parallelism system that jointly optimizes iteration time, peak memory usage, and cross-GPU memory balance for cross-vendor heterogeneous GPU clusters. Beyond avoiding out-of-memory failures, such memory-balanced configurations are more robust to runtime memory fluctuations, easier to schedule in shared clusters, and more portable across heterogeneous GPU types. The core contributions of MemX are threefold.

- **Memory-aware heterogeneous parallelism search.** To tackle the load imbalance caused by memory-performance asymmetry, MemX explicitly incorporates peak memory usage as an optimization objective alongside training throughput. The key innovation is support for heterogeneous recomputation strategies: memory-constrained GPU devices adopt aggressive recomputation to trade time for space, while memory-abundant devices use conservative recomputation to trade space for time, thereby minimizing total training time under memory constraints.
- **Minimal-set profiler with data augmentation.** To address the tension between exploding configuration space and profiling overhead, MemX designs a minimal-set profiler that identifies critical parameter subsets for sparse sampling, reducing profiling overhead by orders of magnitude; augments training data based on linear interpolation assumptions; and employs a machine learning model to achieve millisecond-level performance prediction with a mean absolute percentage error (MAPE) of 12.17%.
- **Real-system validation.** We implement MemX[1] based on Megatron-LM [25, 29] and evaluate it on two real cross-vendor GPU clusters. Experiments on LLaMA2-7B/ 13B/70B and LLaMA3-8B/70B models show that MemX reduces mean MAPE by 18.4% compared with the strongest baseline, reduces profiling measurements by a factor of about 411 in the five-dimensional example, searches 160× as many configurations as AMP, and is over 1,400× faster at the 2,000-configuration scale.

2. Background

This section reviews the background that motivates MemX. Section 2.1 explains why cross-vendor heterogeneous GPU clusters are becoming inevitable and quantifies the compute–memory asymmetry that distinguishes them from homogeneous settings. Section 2.2 surveys automatic parallelism systems designed for heterogeneous clusters and analyzes their limitations under this asymmetry. Section 2.3 reviews performance modeling approaches and discusses the tension between modeling accuracy and profiling cost, which motivates MemX's lightweight modeling design in Section 3.2.

[1] https://github.com/meng980626/MemX.

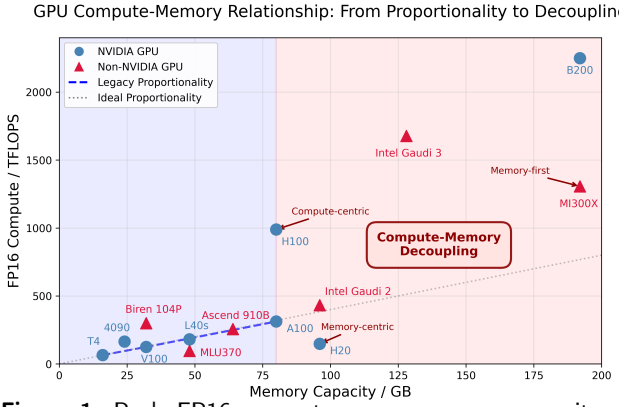


Figure 1: Peak FP16 compute versus memory capacity of mainstream GPUs. Legacy NVIDIA GPUs (T4, V100, A100) follow a stable compute-to-memory proportionality (dashed line), while newer and non-NVIDIA devices deviate in both directions: compute-centric (H100, 12.4:1) or memory-centric (H20, 1.5:1), marking a decoupling of compute and memory in emerging heterogeneous clusters.

2.1. Inevitability of Heterogeneous Trends

The ever-increasing computational demands of large model training, coupled with constrained supply from single-vendor GPUs, have made cross-vendor heterogeneous clusters an inevitable reality. Accelerators such as AMD MI300X [1], Intel Gaudi 3 [11], Huawei Ascend 910B, Biren BR100 series, and ILUVATAR BI-V150 [10] have gradually achieved large-scale deployment capabilities, collectively challenging NVIDIA's dominance. These GPUs vary widely in key metrics including peak compute power, memory capacity, and interconnect bandwidth, compelling training frameworks to adapt to cross-vendor mixed deployments.

Figure 1 illustrates the distribution of peak computing power and memory capacity across mainstream GPUs. Early-generation NVIDIA GPUs (T4, V100, A100) exhibit a clear positive correlation, with compute-to-memory ratios stabilizing at approximately 4:1. However, newer GPUs deviate significantly from this pattern: the NVIDIA H100 (989 TFLOPS/80GB) achieves substantial compute gains without proportional memory increases, yielding a compute-to-memory ratio of 12.4; consumer-grade GPUs such as the RTX 3090 (142 TFLOPS/24GB) and RTX 4090 (165 TFLOPS/24GB) demonstrate high-compute/low-memory characteristics. Inference-optimized GPUs like the H20 (148 TFLOPS/96GB), with compute reduced to 15% of H100 yet retaining 96GB memory, exhibit an exceptionally low compute-to-memory ratio of merely 1.5. Cross-vendor devices further exacerbate this non-monotonicity: the Biren BR104P (~300 TFLOPS/32GB) delivers A100-comparable compute with only 40% of the memory, while the Huawei Ascend 910B (~256 TFLOPS/64GB) offers lower compute yet double the memory. This decoupling of compute and memory capacities becomes the main difference between emerging cross-vendor heterogeneous clusters and traditional homogeneous deployments. Recent systems have begun to automate distributed training over heterogeneous, geo-distributed, or dynamically changing resources [35, 50, 16, 19, 41], further highlighting the practical importance of heterogeneity-aware parallelization.

2.2. Heterogeneous Automatic Parallelism

Distributed training frameworks combine data, tensor, and pipeline parallelism to scale large models across GPU clusters [29, 8, 23], and automatic parallelism systems further search for model partitioning and device placement strategies instead of relying on manual configuration [48, 22]. For heterogeneous clusters, AMP explores tensor, pipeline, data parallelism, and micro-batch size through heterogeneity-aware strategy search [15]; Metis extends this line with device-group partitioning for heterogeneous GPUs [38]; HexiScale uses graph partitioning and analytical cost modeling [43]; and FaSOP assigns pipeline stages through a Minmax-Stage layer-assignment algorithm [9]. More recent systems extend automatic parallelism to broader heterogeneous and dynamic settings, including geo-distributed or dynamically available resources [35, 50], straggler-aware replanning [16], commodity-server tensor parallelism [18], operator-level asymmetric partitioning [41], and fine-grained heterogeneous cluster planning [19]. HPMD further adapts GPU allocation and hybrid parallelism under changing batch sizes [44].

These systems commonly optimize iteration time, monetary cost, or load balance, and usually handle memory as a feasibility constraint or a component of the cost model. Although several recent systems model memory footprint or incorporate memory-saving techniques [49, 35, 16, 18, 41], they generally do not jointly optimize iteration time, peak memory usage, and cross-GPU memory balance for cross-vendor heterogeneous clusters.

Although those systems employ pipeline parallelism (PP) slicing to achieve load balancing by allocating layers proportionally to device compute power, such strategies fail to generalize across all heterogeneous scenarios. Their efficacy relies on an implicit assumption: computing power ratios align with memory requirement ratios. In cross-vendor heterogeneous scenarios, the decoupling of compute and memory renders single-dimensional slicing strategies incapable of satisfying dual constraints simultaneously. Strictly following compute proportions leads to memory bottlenecks, while accommodating memory capacities results in compute underutilization. This dilemma exposes the fundamental limitation of traditional approaches in handling asymmetric heterogeneity.

Furthermore, existing systems typically adopt global uniform policies for training parameters such as recomputation, including fully enabled, fully disabled, or layer-wise uniform configurations, lacking adaptive capabilities for heterogeneous memory capacities. This rigid design leads to significant performance degradation in cross-vendor scenarios: memory-constrained devices face overflow risks

when forced to adopt conservative policies, while memory-abundant devices waste resources when subjected to aggressive policies. The inability of uniform strategies to accommodate differentiated device requirements becomes another critical bottleneck restricting training efficiency.

Beyond guaranteeing feasibility, jointly optimizing memory balance yields practical benefits that a time-only objective cannot provide. The two memory terms in the objective correspond to two distinct risks. High memory utilization (M) exposes a configuration to out-of-memory failures. Measured memory usage is not a static quantity: allocator fragmentation, variable-length batches, and transient communication buffers all introduce fluctuations over a long training run, and a configuration operating close to device capacity can be aborted by a single transient spike, wasting days of computation. Configurations with moderate utilization are robust to such fluctuations, and they are also friendlier to cluster scheduling: a modest and predictable per-device footprint is easier to place and bin-pack, and can be more readily co-located with other tenants in shared clusters.

High memory variance (V) across devices creates a different risk: the bottleneck GPU device. The feasibility and stability of the entire configuration become hostage to the single most-loaded device, even when the average utilization is low. Configurations with uniform memory headroom avoid this risk in two ways. First, they facilitate subsequent hyper-parameter exploration. Adjustments such as enlarging the micro-batch size or the sequence length introduce additional memory that must be absorbed on every device, which is only possible when the headroom is uniform; researchers can therefore vary hyper-parameters around a known-good configuration without re-running the full configuration search. Second, they improve portability across hardware. A configuration with low and balanced utilization carries a compatibility margin that allows it to fit the different GPU types of a heterogeneous cluster, eliminating the cost of re-searching whenever the deployment target changes. These considerations motivate treating memory balance as a first-class optimization objective rather than a binary constraint.

2.3. Performance Modeling

Performance modeling serves as a core component of automatic parallelism systems, responsible for predicting the execution efficiency of different parallel configurations to provide decision support for search algorithms. Existing modeling approaches are dominated by analytical and measurement-based models, while recent work further introduces simulation-based and learning-based variants.

Some prior works construct analytical cost models by parsing static features of computational graphs, estimating FLOPs, communication volume, and memory footprint to predict execution performance [12, 39, 34, 13, 33, 2, 48, 46, 21]. HGTrainer relies on such analytical computation, communication, and memory models for heterogeneous geo-distributed clusters [50], and Mist further uses symbolic expressions to avoid repeated simulation during configuration search [49]. The advantage of such methods lies in low runtime overhead, enabling rapid evaluation of different configurations without actual execution. However, their limitations are equally significant: they struggle to capture dynamic characteristics during actual execution, including GPU caching behavior, compute utilization fluctuations, kernel launch overhead, and complex hardware interactions such as vendor driver optimization differences. In cross-vendor heterogeneous scenarios, microarchitectural differences and software stack characteristics across vendors further amplify the impact of these unmodeled factors, leading to substantial prediction errors.

Other works have shifted toward execution time sampling, recording durations from short iterations or operators on real hardware and constructing cost models to predict performance of complete configurations [23, 24, 5, 22, 20, 28, 47]. Measurement granularity presents a trade-off. Iteration-level sampling executes full forward and backward propagations. It offers high accuracy but requires longer measurement time. Operator-level sampling targets individual operators, which is faster but requires compositional estimation and may introduce cumulative errors. Both approaches share a common limitation: profiling overhead grows explosively with parameter combinations. Different parallelization strategies and training hyperparameters such as batch sizes, sequence lengths, and recomputation configurations require independent sampling. The configuration space scale directly determines profiling cost. Although sparse sampling can reduce such overhead, balancing modeling accuracy against profiling cost remains a significant challenge.

Another line of work instead measures execution time on real hardware [23, 24, 5, 22, 20, 28, 47]. Short iterations or individual operators are timed, and the measurements are used to fit a cost model that predicts the performance of complete configurations. Measurement granularity presents a trade-off. Iteration-level sampling runs full forward and backward propagation; it is accurate but slow. Operator-level sampling is faster, but it requires compositional estimation and may accumulate errors. Both approaches share a common limitation: profiling overhead grows explosively with the number of parameter combinations. Every combination of parallelization strategy and training hyperparameters must be sampled independently, so the profiling cost scales directly with the size of the configuration space. Sparse sampling reduces this overhead, but balancing modeling accuracy against profiling cost remains a significant challenge.

Taking LLaMA2-7B (N=32 layers) training on 16 GPUs as an example, and constraining pipeline parallelism degree to $PP \leq 4$, the configuration space is determined by four dimensions: 15 base parallel strategies satisfying TP×PP×DP=16 with each dimension as a power of two; $\binom{N+PP-1}{PP-1}$ layer partitioning schemes for PP=4; r^{PP} recomputation combinations from r strategies per stage; and x^y configurations from x model hyperparameters (batch size, sequence length, etc.) each with y possible values. The total configuration space under $PP \leq 4$ constraint reaches

approximately 10^8 (billion-scale). Complete profiling would require decades, while existing sparse sampling methods lack effective prior guidance in cross-vendor scenarios, making the trade-off between modeling accuracy and profiling overhead a fundamental challenge.

Beyond these two dominant lines, simulation-based approaches model runtime and memory behavior through resource-level or graph-based simulators [35, 7, 42]; hybrid systems combine profiling with cost estimation or cached intermediate results [16, 41, 19]; AutoScout reduces profiling cost by prioritizing high-impact configuration features and adaptively combining simulators with different fidelities [30]; and learning-based solvers such as NNPiper and APSS train neural strategy generators offline using cost-model feedback [26, 27]. MemX adopts a hybrid design: a minimal-set profiler measures only high-impact parameter combinations, validated interpolation augments the training set, and a dual-output XGBoost predictor estimates both time and memory, while a hash-signature cache avoids repeated predictor calls when candidate configurations share performance-critical substructures.

2.4. Memory Optimization and Pipeline Scheduling

Activation recomputation reduces memory footprint by discarding intermediate activations and recomputing them during backpropagation [4], while selective recomputation preserves a subset of expensive activations to balance time and space [14]. Recent systems increasingly co-optimize memory-saving techniques with parallelization: Mist jointly tunes activation checkpointing, redundancy elimination, offloading, and parallelism [49]; AutoHAAP combines memory-aware initialization with activation recomputation selection [41]; and Oases models overlapped communication and computation under tensor model parallelism [18]. Pipeline schedules such as 1F1B, interleaved 1F1B, and Chimera reduce pipeline bubbles through different micro-batch execution orders [23, 29, 17]. OctoPipe further co-optimizes partitioning, placement, and scheduling through a graph-based pipeline simulator [7], while HARP derives a heterogeneity-aware 1F1B variant to hide cross-cluster communication latency [19]. MemX differs in assigning recomputation strategies at pipeline-stage granularity according to device memory and compute characteristics, and jointly evaluating scheduling, parallelism, recomputation, and memory balance under a unified objective.

3. Method

Figure 2 gives an overview of MemX, and Algorithm 1 summarizes the end-to-end procedure. MemX operates in two phases: an offline phase that builds lightweight performance models from a minimal set of real measurements (Section 3.2), and an online phase that searches the pruned configuration space under the memory–performance joint objective (Section 3.1). We first describe the search algorithm and its objective, and then the performance modeling pipeline that makes the search affordable.

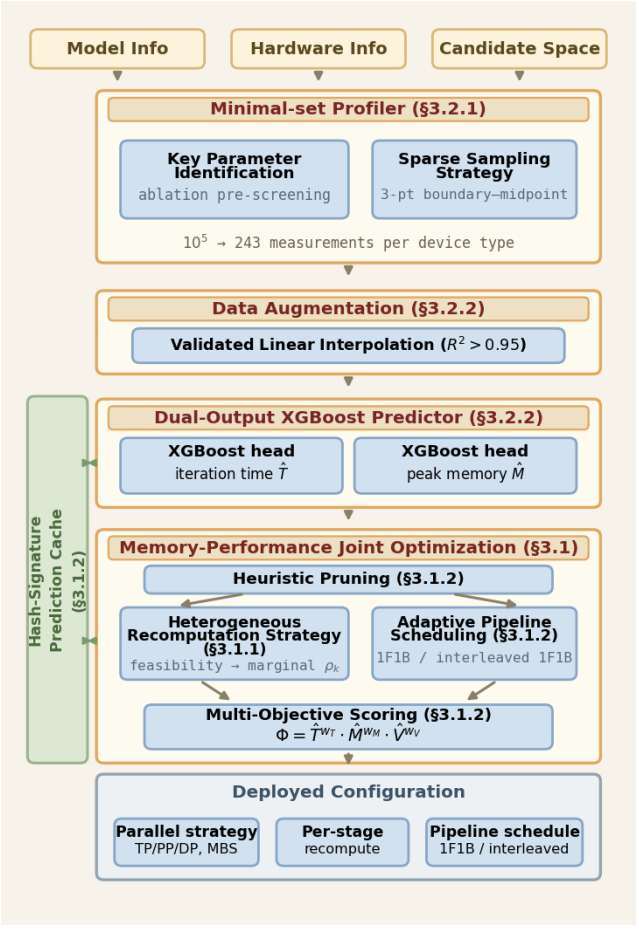


Figure 2: Overview of MemX. In the offline phase (Section 3.2), a minimal-set profiler measures a small set of high-impact parameter combinations on representative devices, validated interpolation augments the measurements, and a dual-output XGBoost predictor is trained to estimate iteration time and peak memory in a single inference call. In the online phase (Section 3.1), MemX enumerates the pruned candidate space, evaluates each configuration under the weighted-product objective Φ with the help of a hash-signature prediction cache, and outputs the deployed configuration together with its per-stage recomputation policy and pipeline schedule.

3.1. Memory-Performance Joint Optimization Search Algorithm

3.1.1. Heterogeneous Recomputation Strategy

Most existing automatic parallelism systems apply globally uniform recomputation policies or treat memory-saving techniques as separate from parallelism search. Recent systems have started to combine parallelism search with activation recomputation or other memory-saving techniques [49, 41, 18]. However, these designs primarily target memory footprint reduction or communication-computation overlap, rather than assigning heterogeneous recomputation policies to explicitly balance memory headroom across devices. Therefore, MemX allows differentiated recomputation strategies across pipeline stages (typically on different GPU types), enabling adaptive time-space tradeoffs.

The recomputation strategy is divided into three levels:

Algorithm 1 MemX configuration search

Require: model $\mathcal{M}$ (N layers, hidden size, heads), cluster $\mathcal{C} = \{(C_k, M_k)\}_{k=1}^{K}$, weight profile (w_T, w_M, w_V)
Ensure: deployed configuration $\mathbf{x}^*$

$\mathcal{S} \leftarrow$ candidate space (TP, PP, DP, MBS, recompute, hyper-parameters) ▷ §3.1.2
$\mathcal{S} \leftarrow \textsc{HeuristicPrune}(\mathcal{S})$ ▷ §3.1.2
$P \leftarrow \textsc{HighImpactParams}(\mathcal{M}, \mathcal{C})$ ▷ §3.2.1
$D \leftarrow \textsc{ThreePointSample}(P)$ ▷ §3.2.1
$D \leftarrow D \cup \textsc{Interpolate}(D)$ ▷ §3.2.2
$f \leftarrow \textsc{TrainDualXGBoost}(D)$ ▷ §3.2.2
cache $\leftarrow \emptyset$; $\Phi^* \leftarrow \infty$; $\mathbf{x}^* \leftarrow \emptyset$
for $\mathbf{x} \in \mathcal{S}$ **do**
  $\sigma \leftarrow \textsc{HashSignature}(\mathbf{x})$ ▷ §3.1.2
  if $\sigma \in$ cache **then**
    $(\hat{T}, \hat{M}) \leftarrow$ cache$[\sigma]$
  else
    $(\hat{T}, \hat{M}) \leftarrow f(\mathbf{x})$; cache$[\sigma] \leftarrow (\hat{T}, \hat{M})$
  end if
  $\mathbf{r} \leftarrow \textsc{RecomputePolicy}(\mathbf{x}, \mathcal{C})$ ▷ §3.1.1
  $s \leftarrow \textsc{SelectSchedule}(\mathbf{x})$ ▷ §3.1.2
  $\hat{V} \leftarrow$ per-device memory variance ▷ §3.1.2
  $\Phi \leftarrow (\hat{T}/T_{\text{ref}})^{w_T} \cdot (\hat{M}/M_{\text{ref}})^{w_M} \cdot ((\hat{V} + \epsilon_V)/(V_{\text{ref}} + \epsilon_V))^{w_V}$ ▷ §3.1.2
  if $\Phi < \Phi^*$ **then**
    $\Phi^* \leftarrow \Phi$; $\mathbf{x}^* \leftarrow (\mathbf{x}, \mathbf{r}, s)$
  end if
end for
return $\mathbf{x}^*$

- **Full Recomputation.** Only the input activations of each layer are retained; all intermediate variables during forward propagation (including attention matrices, Softmax outputs, Dropout masks, etc.) are discarded. During backward propagation, the complete forward computation is re-executed to reconstruct the required gradients [4]. This strategy yields the maximum memory benefit, at the cost of additional forward computation, whose overhead varies across GPU types and kernel implementations. It is suitable for memory-extremely-constrained devices.
- **Selective Recomputation.** On modern GPUs with standard Transformer configurations, this strategy offers limited standalone benefit: its standalone memory benefit can be small, and its time overhead is difficult to distinguish from measurement noise [14]. This is primarily because optimized attention kernels and fused operations already minimize activation retention. Therefore, MemX employs it as a transient state between None and Full in hybrid pipelines.
- **None (No Recomputation).** All intermediate activations from forward propagation are fully preserved and directly reused during backward propagation [8]. Applying this strategy introduces zero computation overhead, while it has maximal memory footprint. It is suitable for memory-abundant high-performance devices, such as Huawei Ascend 910B (64GB), trading space for time to maximize compute throughput.

MemX assigns recomputation strategies at the granularity of pipeline stages. Let the cluster contain K types of heterogeneous devices, where each GPU type k has memory capacity M_k and peak compute power C_k. For an N-layer model with p PP pipeline stages, the decision variable is $r_{k,p} \in \{\text{Full}, \text{Selective}, \text{None}\}$, representing the recomputation strategy for GPU type k at stage p. Given these decision variables, MemX selects the per-stage recomputation strategy in two levels: a hard level that guarantees every stage fits into device memory, and a soft level that picks the most cost-effective strategy among the feasible ones.

First level: hard constraint satisfaction. For device k, the feasibility check compares $\bar{D}_{k,p}^{\text{None}}$ and $D_{k,p}^{\text{Full}}$ against M_k. If $D_{k,p}^{\text{Full}} > M_k$, the configuration is pruned; if $\bar{D}_{k,p}^{\text{None}} \leq M_k$, None is retained; otherwise Full is enforced.

Second level: joint optimization selection. Within the feasible domain, MemX performs strategy refinement to minimize total iteration time. The key observation is that the time-space tradeoff of recomputation exhibits nonlinear characteristics and is jointly affected by device compute power and memory bandwidth. For devices with abundant compute power but limited memory bandwidth, aggressive recomputation may become bottlenecked by memory bandwidth due to frequent recomputation; for devices with scarce compute power but abundant memory bandwidth, the time penalty of aggressive recomputation is relatively smaller.

MemX establishes a marginal time-space exchange rate model for each GPU type:

$$\rho_k^{(s_1 \to s_2)} = \frac{\Delta T_k(s_1 \to s_2)}{\Delta M_k(s_1 \to s_2)} \tag{1}$$

where ΔT denotes the time change caused by strategy switching, and ΔM denotes the memory change. When strategy levels exhibit discontinuous ρ (e.g., Selective yielding near-zero benefit on some devices), MemX collapses the intermediate level and treats the effective choice as binary between None and Full.

3.1.2. Search Algorithm Design

Multi-objective optimization. MemX formulates automatic parallel search as a three-objective optimization problem. The first objective is iteration time, which directly determines training efficiency. The second is peak memory usage, defined as the maximum memory consumption across all devices in the cluster. Unlike existing systems that treat memory merely as a hard constraint, MemX explicitly incorporates it as an optimization objective to improve resource efficiency and prevent scenarios where some devices remain severely underutilized while others operate at their limits. The third is memory utilization variance, which quantifies the deviation of each device's memory usage from the cluster mean. Excessive variance indicates localized bottlenecks: memory-constrained devices are forced to adopt aggressive

recomputation, significantly increasing per-layer execution time and dragging down overall pipeline throughput. These three objectives are inherently conflicting: reducing memory footprint requires aggressive recomputation at the cost of increased iteration time, while pursuing balanced utilization may sacrifice the efficiency of optimal devices. MemX aggregates the three objectives into a single scalar objective through weighted product, which automatically penalizes short boards:

MemX aggregates the three objectives into a single scalar objective through a weighted product:

$$\min_{\mathbf{x}} \quad \Phi(\mathbf{x}) = \widehat{T}(\mathbf{x})^{w_T} \cdot \widehat{M}(\mathbf{x})^{w_M} \cdot \widehat{V}(\mathbf{x})^{w_V} \tag{2}$$

where

$$\widehat{T}(\mathbf{x}) = \frac{T(\mathbf{x})}{T_{\text{ref}}}, \quad \widehat{M}(\mathbf{x}) = \frac{M(\mathbf{x})}{M_{\text{ref}}}, \quad \widehat{V}(\mathbf{x}) = \frac{V(\mathbf{x}) + \epsilon_V}{V_{\text{ref}} + \epsilon_V}.$$

Here, $T(\mathbf{x})$ denotes iteration time, $M(\mathbf{x})$ denotes peak memory usage, and $V(\mathbf{x})$ denotes the variance of memory usage across devices. The reference values T_{ref}, M_{ref}, and V_{ref} are fixed for a given candidate set, for example by using the median of all feasible candidates. This rescaling makes the three terms dimensionless without changing their ordering, because the reference denominators contribute only a constant multiplier to Φ. We set ϵ_V to a small fraction of V_{ref} to prevent a zero-variance configuration from dominating the product solely because $V(\mathbf{x}) = 0$. With $w_T, w_M, w_V \geq 0$ and $w_T + w_M + w_V = 1$, the weights can be interpreted as elasticities: a one-percent increase in objective dimension i raises Φ by approximately w_i percent.

We use three fixed weight profiles rather than tuning the weights on test data. The default heterogeneous profile $(w_T, w_M, w_V) = (0.4, 0.2, 0.4)$ is used for clusters with substantial device diversity, where identifying balanced configurations is as important as minimizing iteration time. The safety-first profile $(0.3, 0.5, 0.2)$ is used when the minimum predicted memory margin falls below 10%, prioritizing feasibility and OOM avoidance. The throughput-first profile $(0.7, 0.2, 0.1)$ is used for homogeneous or weakly heterogeneous clusters, where memory pressure and imbalance are less pronounced. All profiles are selected before evaluation according to these deployment conditions. Section 4.2.4 uses the default heterogeneous profile, while Section 4.4 uses the safety-first profile because the small V100/T4 cluster has the tightest memory headroom among the evaluated systems.

Heuristic pruning. MemX designs three pruning rules to control the search space. First, cross-device communication restriction: tensor parallelism and data parallelism involve frequent synchronization operations that are latency-sensitive, hence their communication groups are constrained within individual pipeline stages. Second, memory overflow pruning: the memory prediction model rapidly estimates per-device memory consumption under the Full recomputation strategy, which preserves the minimal activations and constitutes the theoretical lower bound of memory usage; if this lower bound still exceeds physical capacity, the configuration is infeasible regardless of recomputation strategy and is immediately pruned, avoiding invocations of the performance prediction model on other recomputation configurations. Third, iteration time lower-bound pruning: the minimum achievable iteration time for a partial configuration is computed from device peak compute and theoretical communication volume; if this lower bound exceeds the current best solution, the branch cannot yield a superior configuration and is pruned.

End-to-end performance. MemX adopts the micro-batch as the minimal modeling unit, striking a balance between iteration-level accuracy and operator-level efficiency. A micro-batch encompasses complete layer computations, capturing dynamic characteristics such as operator fusion and kernel launch overheads, while maintaining controllable per-prediction overhead and aligning with the fundamental scheduling unit of pipeline parallelism. MemX supports two pipeline scheduling schemes.

1F1B Scheduling [23]. Let the global batch size be B, data parallelism degree D_P, and micro-batch size b_m. The number of micro-batches (gradient accumulation steps) is $G = B/(D_P \cdot b_m)$. Let t_i^{mb} denote the single micro-batch execution time at PP stage i (including forward and backward of all layers in that stage), t_i^{pp} the inter-stage PP communication time, and t^{fix} the fixed overhead (e.g., optimizer step, data loading). The iteration time comprises three components: pipeline fill and drain, steady-state execution, and communication overhead. Specifically, the first micro-batch incurs fill delay through all P stages, the last micro-batch incurs drain delay for backward propagation, intermediate G-1 micro-batches progress at the bottleneck stage period in steady state, plus activation passing PP communication and DP communication.

$$T_{\text{1F1B}} = (G-1) \cdot \max_{i \in [P]} t_i^{\text{mb}} + \sum_{i=0}^{P-1} t_i^{\text{mb}} + 2 \sum_{i=0}^{P-1} t_i^{\text{pp}} + t^{\text{fix}} + t^{\text{dp}} \tag{3}$$

where $t^{dp} = 2(D_P - 1)M/(D_P \cdot B_{dp})$ is the data parallelism all-reduce communication time, M the model parameter memory footprint, and B_{dp} the DP communication bandwidth. The first term $(G-1) \cdot \max t_i^{mb}$ is the steady-state progression time, the second term $\sum t_i^{mb}$ the base computation for a single micro-batch traversing all stages, and the coefficient 2 in PP communication accounts for one forward activation pass and one backward gradient pass each.

Interleaved 1F1B Scheduling [25]. Introduce the virtual pipeline number N_v (vp_size), where each physical stage is divided into N_v virtual stages with micro-batches interleaved. A single physical stage is shared by N_v virtual stages, and its single-micro-batch execution time scales ideally by $1/N_v$. Fill and drain overheads are amortized due to increased virtual stages, while micro-batches progress in N_v-way interleaving in the steady-state term.

$$T_{\text{Interleaved}} = \left(G - 1 + \frac{1}{N_v}\right) \cdot \max_{i \in [P]} \frac{t_i^{\text{mb}}}{N_v} + \frac{2}{N_v} \sum_{i=0}^{P-1} t_i^{\text{mb}} + \sum_{i=0}^{P-1} t_i^{\text{pp}} + t^{\text{dp}}. \tag{4}$$

Compared to 1F1B, the steady-state coefficient changes from $(G-1)$ to $(G-1+1/N_v)$, with the additional $1/N_v$ term reflecting extra fill from virtual stages; the base computation term is divided by N_v; and the bottleneck time takes $\max_i(t_i^{\text{mb}}/N_v)$. The virtual stage number N_v is constrained by memory: $N_v \cdot A_{\text{stage}} \leq M_{\text{min}}$, where A_{stage} is the single-stage activation peak.

MemX supports adaptive selection among two pipeline scheduling schemes. The baseline 1F1B alternates forward and backward execution, offering simplicity and stable memory footprint at the cost of substantial pipeline bubbles. Interleaved 1F1B extends this by partitioning each physical stage into virtual stages that execute in an interleaved manner, effectively compressing bubbles while increasing activation memory overhead; this scheme is suitable for memory-abundant scenarios. The adaptive selection mechanism relies on two indices: the memory pressure index, defined as the ratio of minimum available memory to maximum stage memory demand, and the GPU heterogeneity index, quantified as the coefficient of variation of device compute power (standard deviation over mean). When memory is constrained or heterogeneity is pronounced, 1F1B is enforced to guarantee feasibility; when memory is abundant and devices are sufficiently homogeneous, Interleaved 1F1B is activated to pursue higher throughput.

Reducing Prediction Overhead in Search. The excessive search space leads to frequent invocations of the performance prediction model, causing search overhead to grow linearly with the number of candidate configurations. Reducing redundant evaluation has also been recognized by recent systems. AutoHAAP caches operator states to eliminate repeated latency calculations [41], HARP clusters repeated modules to reduce profiling overhead [19], AutoScout prioritizes high-impact configuration features and adaptively combines simulators with different fidelities [30], and Mist reuses symbolic expressions after a single simulation pass [49]. In comparison, MemX reduces invocation overhead from two complementary perspectives: model architecture design and prediction result reuse.

At the model architecture level, MemX employs a dual-output XGBoost predictor that produces iteration time and memory footprint in one inference call, avoiding two independent model invocations for each candidate configuration.

At the result reuse level, MemX establishes a hash-signature-based prediction cache mechanism. The signature is generated from the combined features of tensor parallel strategy, recomputation strategy and other hyperparameter for each micro-batch. New queries are first subjected to signature matching, with cache hits directly returning previously stored results. Cache invalidation conditions are strictly confined to GPU type changes (execution characteristics are non-transferable across GPUs from different vendors), model structure changes (number of layers, hidden dimensions, etc.), and batch size changes (affecting operator scheduling and memory allocation patterns). In typical search processes, many candidate configurations share identical substructures, allowing repeated predictor calls to be avoided. Section 4.3 shows that this reuse leads to near-constant search overhead when cluster-scale extensions leave the key performance parameters unchanged.

3.2. Lightweight Performance Modeling

This design is complementary to recent approaches that reduce modeling cost through symbolic expressions [49], analytical hardware models [50], profiled cost coefficients [16, 18, 40], graph-based or resource-level simulation [35, 7], or adaptive simulator ensembles [30]. MemX targets a different operating point: it performs sparse real measurements only for high-impact parameter combinations, augments them through validated interpolation, and trains a lightweight dual-output predictor for both time and memory.

3.2.1. Minimal-set Profiler

Key Parameter Identification. The configuration space for automatic parallel search is determined by multiple dimensions, encompassing parallel strategies, model partitioning schemes, recomputation strategies, and model hyperparameters. Exhaustively traversing all parameter combinations incurs sampling overhead that grows exponentially with dimensionality, rendering it particularly infeasible in cross-vendor heterogeneous scenarios. MemX identifies high-impact parameter subsets through ablation experiments: fixing other parameters at typical default values while varying a single parameter and observing the fluctuation amplitude of iteration time and memory footprint. Parameters with large fluctuation amplitudes are classified as high-impact and incorporated into subsequent sampling; those with small fluctuation amplitudes are classified as low-impact and only fixed typical values are selected.

The ablation experiments rely on small-scale pre-sampling: for each type of heterogeneous device, a representative server hosting that GPU type is selected to execute short-iteration measurements covering extreme and central values of each parameter. Taking LLaMA2-7B training as an example, typical high-impact parameters include TP degree (determining tensor sharding granularity and all-reduce communication volume), recomputation strategy (significantly altering the memory-time tradeoff curve), and micro-batch size (linearly scaling activation memory and affecting operator utilization); typical low-impact parameters include maximum position embedding length (only determining the upper bound of pre-allocated positional encoding tables, with actual sequence lengths during training constrained by other parameters). Through this pre-screening, MemX compresses the parameter dimensions requiring dense sampling from over ten to a core subset of 3-5 dimensions, laying the foundation for subsequent sparse sampling.

Scope of parallel dimensions: Throughout this paper, the parallelization strategies in the candidate space refer to tensor, pipeline, and data parallelism (TP/PP/DP); three other dimensions are excluded. First, sequence parallelism (SP) in Megatron-LM is not an independent device-placement dimension but a memory-saving extension of TP that shards the LayerNorm and Dropout activations along the sequence dimension[29]; it introduces no new communication groups or device mappings, and its memory effect is already absorbed by our profiled peak-memory measurements, since all profiling runs execute with SP enabled. Second, context parallelism (CP) targets extremely long sequences (tens of thousands to millions of tokens) by sharding the sequence dimension across devices, whereas our evaluated workloads use sequences of 1K–4K tokens whose activations fit in single-device memory, so CP's benefit regime does not arise. Moreover, CP is orthogonal to cross-vendor heterogeneity: it partitions within a layer and thus interacts with neither the cross-device layer placement decided by PP nor the intra-layer tensor sharding decided by TP. Third, expert parallelism (EP) applies only to Mixture-of-Experts models, while our evaluation targets dense LLMs. We note that CP and EP can be incorporated as additional device-mesh dimensions in the candidate space without structural changes to MemX's profile–predict–search pipeline, which we leave to future work.

Sparse Sampling Strategy. For identified high-impact parameters, MemX adopts a three-point boundary-midpoint sparse sampling strategy in lieu of exhaustive traversal. Specifically, for continuous or discrete parameters with value range $[x_{min}, x_{max}]$, the minimum x_{min}, maximum x_{max}, and midpoint $(x_{min} + x_{max})/2$ or default value are sampled. The theoretical basis of this strategy lies in that most performance metrics (iteration time, memory footprint) exhibit monotonic or approximately monotonic relationships with respect to parameter variation: boundary points capture extreme behavioral characteristics, while the midpoint represents typical operating conditions, with three points sufficing to fit linear or piecewise-linear trends. For parameters with known optimal intervals (e.g., micro-batch sizes typically being powers of two), sampling points are adjusted to boundary subsets of typical values within that interval (e.g., minimum, maximum, and central values from 1, 2, 4, 8, 16, 32, 64).

Complexity reduction of the sparse sampling strategy in Algorithm 1: the reduction is substantial. For a high-impact parameter with N possible values, exhaustive sampling requires $O(N^d)$ measurements, whereas sparse sampling fixes at 3 points, reducing complexity to $O(3^d)$. Taking a 5-dimensional high-impact parameter subset as an example, if each dimension averages 10 values, exhaustive sampling requires $10^5 = 100000$ measurements, while sparse sampling requires merely $3^5 = 243$, reducing by approximately two orders of magnitude. Combined with fixed typical values for low-impact parameters, the total sampling points are compressed from a billion-scale configuration space to hundreds of key measurements, achieving order-of-magnitude reduction in profiling overhead.

3.2.2. Data Augmentation & Training

Linear Interpolation Hypothesis. Sparse sampling substantially reduces measurement overhead, yet the resulting sample size is insufficient to support high-precision prediction models. MemX generates augmented data based on the linear interpolation hypothesis: for certain continuous parameters (e.g., batch size, sequence length), performance metrics are assumed to vary approximately linearly between sparse sampling points, with intermediate configurations synthesized through interpolation for training.

The validity of this hypothesis is verified through small-scale experiments: on representative devices, several parameter intervals are densely sampled with 5-7 points to measure actual performance, linear models are fitted, and the coefficient of determination R^2 is computed. If $R^2 > 0.95$, the interval is deemed to satisfy the local linearity assumption and safe for interpolation; otherwise the interval is shrunk or piecewise-linear fitting is adopted. Experimental results in Table 2 reveal significant variation across parameter-metric associations. "num_layers to memory" and "micro_batch_size to TFLOP/s/GPU" satisfy strong linearity ($R^2 > 0.98$), enabling direct linear interpolation; "seq_length to Iteration time" exhibits superlinear growth ($R^2 = 0.78$) due to quadratic attention complexity, requiring nonlinear extrapolation; "num_query_groups to all three metrics" show negligible variation ($R^2 < 0.05$), thus directly fixing default values.

Based on this classification, MemX dynamically adjusts how much data is generated through interpolation: strong linear cases use large steps to generate many points, near-linear cases use smaller steps or split into segments, nonlinear cases reduce the interpolation range or skip it entirely, and constant cases simply use fixed values. Figure 3 illustrates the three trend types on measured single-parameter sweeps: near-linear parameters are interpolated with large steps, the superlinear case triggers interval shrinking, and invariant parameters are fixed without any further measurement. The augmented dataset typically grows 5-10× in size, effectively reducing overfitting risks when training samples are limited.

XGBoost Prediction Model. MemX employs gradient boosted trees (XGBoost) as the performance predictor, balancing prediction accuracy, inference speed, and interpretability. XGBoost iteratively adds prediction trees where each tree corrects the errors of its predecessor, and adopts second-order Taylor expansion in loss function optimization for more precise handling of non-linear problems; regularization terms are incorporated to control model complexity and prevent overfitting [3]. Its gradient boosting mechanism is more sensitive to feature importance, quantifying contribution through the gain of features used as split points, facilitating identification of key configuration parameters. Input features encode complete configuration information:

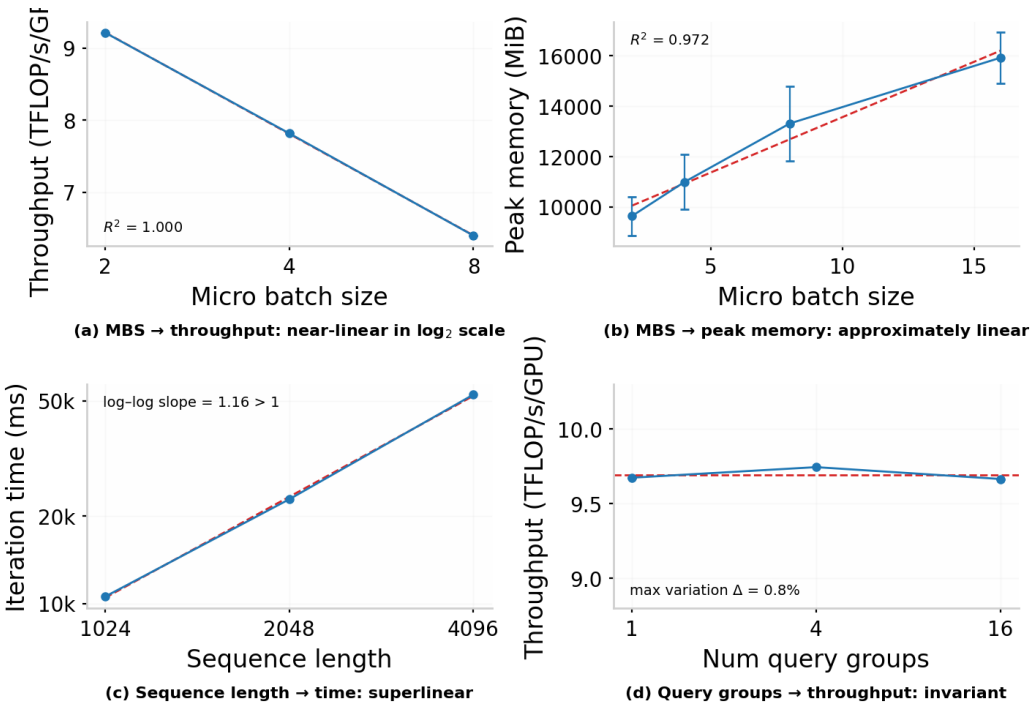


Figure 3: Measured single-parameter sweeps illustrating the trend classification used by the interpolation validator (Section 3.2.2), collected on the small two-node cluster; error bars show the standard deviation across ranks and repetitions, and the dashed line is a linear fit. (a) Micro batch size → throughput: near-linear in $\log_2$ scale ($R^2 = 1.000$); (b) micro batch size → peak memory: approximately linear ($R^2 = 0.972$); (c) sequence length → iteration time: superlinear (log–log slope $1.16 > 1$), requiring nonlinear extrapolation; (d) query groups → throughput: invariant ($\Delta = 0.8\%$), fixed at the default value.

Table 2
Examples of Linear Interpolation Validation Results

Parameter	Metric	Trend	R^2	Interpolation Strategy
num_layers	Peak GPU memory	Strong linear	0.998	Large-step linear
micro-batch size	Peak GPU memory	Approx. linear	0.971	Large-step linear
micro-batch size	TFLOP/s/GPU	Strong linear	1.000	Linear
seq_length	Iteration time	Superlinear	0.780	Nonlinear extrapolation
num_query_groups	All three metrics	Constant	<0.05	Fixed default

GPU type, parallel strategy configuration, recomputation strategy, and model hyperparameters (batch size, sequence length, hidden dimension, layer count, etc.). Dual outputs are produced through a multi-task XGBoost variant: shared tree structures learn feature representations, with separate time prediction heads and memory prediction heads at leaf nodes.

Model training uses the augmented dataset, with both iteration time and memory footprint evaluated by MAPE. Hyperparameter tuning employs Bayesian optimization, efficiently exploring the space of tree depth, learning rate, and subsample ratio based on Gaussian processes [32]. XGBoost feature importance outputs guide the search strategy: high-importance features are varied with priority, while low-importance features are fixed at defaults. The design target is to minimize MAPE with millisecond-level single prediction, satisfying the real-time demands of tens to hundreds of thousands of evaluations during search. In the evaluation of Section 4.2, MemX achieves a mean MAPE of 12.17%.

4. Evaluation

4.1. Experimental Setup

Hardware Environment. We evaluate MemX on a cross-vendor heterogeneous cluster comprising 20 GPUs across three physical nodes. To study the impact of memory–performance asymmetry, the cluster is intentionally configured with GPUs from three different hardware vendors. Table 3 summarizes the hardware specifications.

Table 3
Hardware configuration of the evaluation cluster.

GPU Type	Quantity	HBM Capacity	Normalized Computing Power
NVIDIA A100	4	80 GB	128
Vendor-B GPU	8	32 GB	99
Vendor-C GPU	8	32 GB	46

Communication Bandwidth. The inter-node bandwidth across all three nodes is 28 GB/s. The A100 intra-node

Table 4
Model configurations used in evaluation.

Model	Layers	Hidden Dim	Heads	Head Dim	Vocab Size	Seq Len
LLaMA2-7B	32	4096	32	128	32000	4096
LLaMA2-13B	40	5120	40	128	32000	4096
LLaMA2-70B	80	8192	64	128	32000	4096
LLaMA3-8B	32	4096	32	128	128256	8192
LLaMA3-70B	80	8192	64	128	128256	8192

bandwidth with NVLink is 240 GB/s. The Vendor-B and Vendor-C GPUs exhibit a notable bandwidth asymmetry: the cross-NUMA link (12 GB/s) is slower than the 4-GPU interconnect (17 GB/s), and both are substantially slower than the A100 NVLink domain. This heterogeneity in both compute and communication characteristics poses distinct challenges for load balancing and performance modeling.

Software Environment. We implement MemX on top of Megatron-LM 0.8 [25]. All baseline methods are adapted to the same software stack to ensure a fair comparison.

Models. We conduct experiments on five representative LLM configurations spanning two model families and multiple scales [36, 37, 6]. Table 4 lists their architectural hyper-parameters.

Baseline Methods. We compare MemX against four state-of-the-art automatic parallelism systems:

- **AMP:** Searches over TP, PP, DP, and micro-batch size [15].
- **Metis:** Extends AMP with device-group partitioning and supports heterogeneous parallelism [38].
- **HexiScale:** Employs graph partitioning and analytical cost modeling [43].
- **FaSOP:** Uses a Minmax-Stage layer assignment algorithm [9].

We focus the empirical comparison on four representative systems with directly comparable search objectives and deployment assumptions, and discuss more recent systems in Section 2. To examine behavior at larger scales, Section 4.5 further evaluates MemX on simulated 40-GPU and 60-GPU clusters derived from the measured data.

4.2. End-to-End Comparison

We evaluate MemX against the four baseline methods on the 20-GPU cluster described in Table 3. The evaluation is grounded in a large real-measurement dataset collected on this cluster: 36,740 training records covering 1,628 unique configurations, spanning TP, PP, DP, micro-batch sizes, sequence lengths, and multiple recomputation and attention-grouping settings, across all three GPU types. The workload includes different models in Table 4.

For each method, we report (i) the size of the explored configuration space and the wall-clock search overhead, and (ii) the quality of its predictions and recommendations against real hardware measurements. Since FaSOP is not open-source, we re-implemented it on top of AMP's codebase with the communication-latency modeling revised according to its paper [9]; its search overhead is essentially identical to AMP's, and the AMP rows below therefore represent both methods.

4.2.1. Search Space and Efficiency

Table 5 summarizes the search scope and efficiency of each method. AMP, Metis, and HexiScale all restrict their search to a small subset of the configuration space to keep profiling overhead manageable. AMP explores only 14 configurations spanning TP, PP, DP, and micro-batch size (MBS). Metis expands the space to 123 configurations by incorporating device-group partitioning. HexiScale evaluates 24 configurations using its analytical cost model without profiling. In contrast, MemX searches over 2,240 configurations (160× the space of AMP) by including additional hyper-parameters such as sequence length, recomputation strategy, and attention-head configuration. After filtering out configurations predicted to exceed memory capacity (approximately 30%), MemX retains 1628 valid configurations, still 125× the effective space of AMP.

Despite covering two orders of magnitude more configurations, MemX completes the search in 24.87 seconds, which is 62% of the time taken by AMP. This counter-intuitive result stems from MemX's minimal-set profiler and XGBoost-based predictor, which replace expensive per-configuration profiling with millisecond-level inference. FaSOP does not perform an explicit search in the same sense; instead, it relies on an end-to-end cost model with greedy layer assignment, so we use the same scheme as AMP for FaSOP and omit its search time in this comparison.

4.2.2. Prediction Accuracy

Table 6 reports the prediction accuracy of each method against real iteration times. We measure the Mean Absolute Percentage Error (MAPE) and the proportion of predictions within acceptable error bounds (Good: <20%; Severe: >50%).

MemX achieves the best prediction accuracy across the board. Its mean MAPE of 12.17% improves upon the next-best method Metis by 18.4% in relative terms, i.e., $(14.91 - 12.17)/14.91$, which is the improvement claimed in our contributions; its maximum error is only 29.77%. MemX is also one of the methods with no configuration exceeding 50% error. In contrast, half of HexiScale's configurations suffer errors above 50%, and 61.5% of FaSOP's configurations fall within the >50% error band; on its worst configuration (TP4/PP5/DP1, MBS=4), FaSOP predicts an iteration time more than twice the real measurement (114.6% error).

Table 5
Search space and efficiency comparison (LLaMA2/3, 20 GPUs).

Method	Configs Explored	Valid Configs	Search Time (s)	Avg. Time per Config (ms)
AMP (FaSOP)	14	13	40.07	3,082
Metis	123	8	2.43	304
HexiScale	24	18	0.253	14
MemX	2,240	1,628	24.87	15.3

Table 6
Prediction accuracy comparison against real measurements.

Method	MAPE	Good (<20%)	Severe (>50%)
AMP	66.44%	23.1%	61.5%
Metis	14.91%	62.5%	0%
HexiScale	43.52%	29.2%	50%
FaSOP	54.40%	30.8%	61.5%
MemX	12.17%	69.1%	0%

In addition, MemX's accuracy advantage is structural rather than incidental. The baselines' errors degrade markedly along configuration dimensions such as micro-batch size, pipeline depth, and sequence length—for example, FaSOP's MAPE rises to 76.47% on the MBS=2 subset and 62.07% on the seq_length=4096 subset. MemX's error distribution, by contrast, remains the flattest across all configuration subspaces, with its per-dimension mean MAPE never exceeding 18%, indicating that its performance model is robust to variations in parallel structure and workload.

4.2.3. Ranking Consistency and Practical Utility

The end product of configuration search is a ranked recommendation list, so ranking quality is closer to practical utility than absolute accuracy. Table 7 reports the consistency between each method's predicted ranking (by predicted iteration time) and the real ranking (by measured iteration time): Spearman's ρ, Kendall's τ, and Top-k hit rates.

MemX attains the strongest ranking consistency, slightly ahead of Metis and well ahead of HexiScale, FaSOP and AMP. The decisive difference, however, lies in recommendation quality: MemX is the only method whose top recommendation is the real optimum, whereas all four baselines miss it; even Metis, whose overall correlation is strong, does not hit it either. For a configuration search system, a missed Top-1 means the entire search cost yields a suboptimal configuration, so this gap is of direct practical consequence.

4.2.4. Ranking Consistency under Memory-Balance Objective

We next examine whether each method's ranking can identify configurations with balanced memory usage. We re-rank the real configurations by their measured $\Phi(\mathbf{x})$ under the default heterogeneous weight profile $(w_T, w_M, w_V) = (0.4, 0.2, 0.4)$ defined in Section 3.1.2. This profile is selected a priori and is not tuned on the measured rankings. Results are shown in Table 8.

All baselines degenerate to random under the memory-balance objective. Their Spearman correlations fall within [-0.069, 0.126] and are all statistically insignificant. Moreover, this failure stems from a missing objective, not from poor prediction. Comparing Tables 7 and 8, the baselines' time-based ranking ability varies widely from 0.31–0.87, yet their Φ-based ranking ability uniformly vanishes, and the two are completely decoupled. Figure 4 visualizes this decoupling: time-based and Φ-based ranking ability are orthogonal across all evaluated methods, and only MemX occupies the consistent-on-both region. For example, the strongest time ranker, Metis ($\rho = 0.8748$), scores only $\rho = -0.0692$ under Φ. In other words, even a cost model that predicts iteration time well cannot identify memory-balanced configurations.

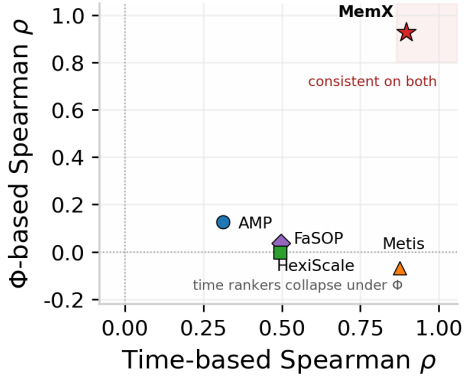


Figure 4: Time-based versus Φ-based ranking consistency (Spearman's ρ, from Tables 7 and 8). The baselines' time-ranking ability varies widely (0.31–0.87), yet all of them collapse to near-zero correlation under the memory-balance objective Φ; MemX is the only method that remains consistent on both axes.

4.3. Scalability of Search Overhead

A practical configuration search method must remain affordable as the search space grows. In this subsection, we extend each method's search space along two axes and measure the resulting search wall-clock time: (i) the hyper-parameter dimension (e.g., enlarging the global/micro-batch size options) and (ii) the cluster scale (increasing the number of nodes, which enlarges the feasible parallel-strategy space, most notably PP). Table 9 summarizes the results and Figure 5 visualizes the scaling trends.

The baselines' search cost grows rapidly, or unpredictably, with the space size. Because AMP (and FaSOP) profiles every configuration end-to-end, its cost grows as a power law along the batch dimension: a power-law fit $t = 1.147n^{1.36}$ ($R^2 = 0.997$) of its measurements extrapolates

Table 7
Ranking consistency against real time-based ranking.

Method	Spearman ρ	Kendall τ	Top-1	Top-3	Top-5	Top-10
AMP	0.3132	0.1795	0%	0%	60%	—
Metis	0.8748	0.6939	0%	66.7%	—	—
HexiScale	0.4967	0.3835	0%	33.3%	20%	40%
FaSOP	0.4960	0.3726	0%	0%	20%	—
MemX	**0.8964**	**0.7335**	**100%**	66.7%	**60%**	**70%**

Table 8
Ranking consistency against real memory-balance (Φ) ranking.

Method	Spearman ρ	Kendall τ	Top-1	Top-3	Top-5	Top-10
AMP	0.1264	0.0256	0%	0%	0%	40%
Metis	-0.0692	-0.0292	0%	0%	0%	—
HexiScale	0.0372	0.0381	0%	0%	0%	20%
FaSOP	-0.0025	0.0008	0%	0%	0%	40%
MemX	**0.9285**	**0.7797**	**100%**	**66.7%**	**80%**	**80%**

Table 9
Search overhead under search-space extension.

Method	Extended setting	Configs	Search time (s)
AMP / FaSOP	base (Table 5)	14	40.07
	−− GBS	8	19.64
	− GBS	11	29.78
	+ GBS	20	68.69
	+ GPUs	47	118.4
	++ GPUs	31	134.7
	+++ GPUs	40	155.6
Metis	base	123	2.43
	− − − TP/MBS options	0*	1.82
	−− TP/MBS options	14	1.88
	− TP/MBS options	64	2.02
	+ PP size	475	19.69
	++ PP size	1,087	96.18
	+++ PP size	1,831	26.76
HexiScale	base	24	0.253
	+ nodes	24	2.03
	++ nodes	8	5.5
MemX	base	2,240	24.87
	+ MBS options	3,360	36.11
	++ MBS options	5,040	55.13
	+ parallel configs	8,040	24.34
	++ parallel configs	31,240	26.5

* This extension yields no valid configuration

to approximately 35,150 s (9.8 hours) for merely 2,000 configurations. Metis stays nearly flat when adding TP/MBS options but degrades unpredictably with PP: extending PP alters its layer-assignment strategy, which can generate large numbers of invalid configurations that must be evaluated and filtered out, spiking the search time to 96.18 s at PP=12. HexiScale's analytical model keeps parameter additions cheap, yet its overhead rises with cluster scale (0.253 s to 5.5 s from 20 to 80 GPUs) while its explored space remains the narrowest (as few as 8 valid configurations).

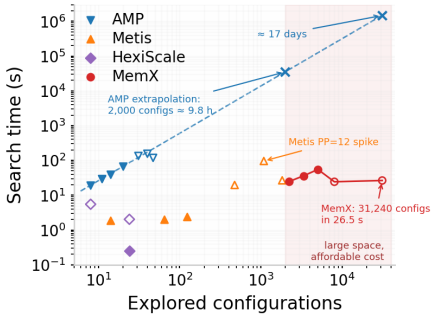


Figure 5: Search time versus search-space size (log–log), combining all extension measurements of Table 9. Filled and open markers denote extensions along the batch dimension and the cluster scale, respectively. The dashed curve is a power-law fit $t = 1.147n^{1.36}$ ($R^2 = 0.997$) of AMP's measurements, extrapolated to 2,000 configurations (~9.8 hours). The shaded region marks large spaces (≥ 2,000 configurations) at affordable cost—only MemX operates there, searching 31,240 configurations in 26.5 s.

In contrast, MemX's search overhead is decoupled from the combinatorial growth of the parallel-strategy space, thanks to the caching mechanism described in Section 3.1.2. When the key performance parameters such as MBS of a configuration are unchanged, MemX reuses the cached prediction instead of re-invoking the XGBoost model. Extending the cluster scale only adds PP/DP combinations while leaving these key parameters intact, so the additional configurations are evaluated almost entirely from cache: 31,240 configurations, about 14× of the base space, are searched in 26.5 s, only 6.5% more time than the base space. Along the batch dimension, however, adding MBS options changes a key performance parameter and invalidates the cache, requiring new predictor calls for the affected configurations; the cost therefore grows linearly. Figure 6 contrasts the two extension axes directly: the same predictor and

Table 10
Deployment validation on the small cluster. Baselines recommend only the parallel dimensions and are deployed with recompute = none; their entries span the measured variants of the remaining dimensions. The best deployed variant of each method is shown in the time columns.

Method	Recommended configuration	Time (ms)	Time rank	Φ rank	Mem. var. (MiB^2)	OOM
AMP	TP2/DP1/PP3, MBS=2	9,211–9,912	1–19	12–26	5.96×10^5–9.34×10^5	No
FaSOP	(same as AMP)	(same as AMP)	(same)	(same)	(same)	No
Metis	TP2/DP1/PP3, MBS=4	10,459–11,685	28–60	67–138	1.37×10^6–2.91×10^6	No
HexiScale	TP1/DP2/PP3, MBS=2	—	—	—	—	Yes
MemX	TP2/DP1/PP3, MBS=2, full recomputation, query groups=16, ...	10,066	26	**1**	4.19×10^5	No

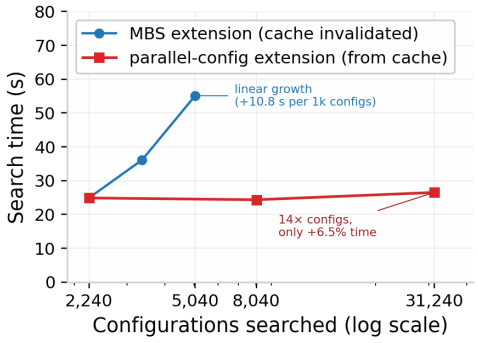


Figure 6: MemX's search time when the search space is extended along two axes (Table 9). Adding MBS options changes a key performance parameter, invalidating the hash-signature cache, so the search time grows linearly with the space size; extending only the parallel-strategy dimensions (TP/PP/DP) leaves the key parameters intact, so the additional configurations are served almost entirely from cached predictions—14× more configurations for only 6.5% more time.

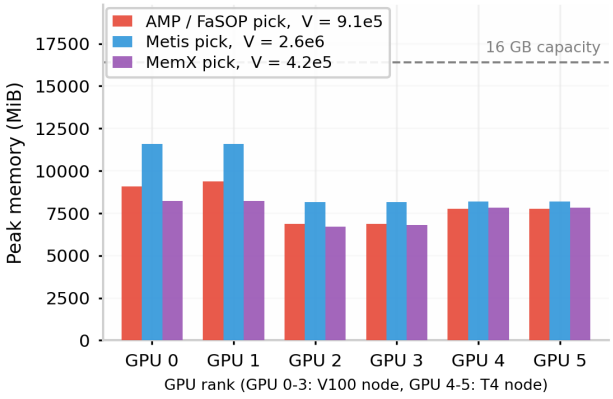


Figure 7: Per-GPU peak memory of the deployed recommendations on the small cluster. MemX's pick (full recomputation) keeps every GPU at a low and even occupancy (variance 4.2×10^5), while the deployed AMP/FaSOP and Metis picks (recompute = none) are both higher and more imbalanced (variance 9.1×10^5 and 2.6×10^6, respectively).

cache make cluster-scale extension nearly free, while batch-dimension extension is the only axis that costs new predictor calls. MemX thus searches spaces orders of magnitude larger at a fraction of the cost: at 2,000 configurations it is over 1,400× faster than AMP/FaSOP's extrapolated time, and at 31,240 configurations the contrast reaches roughly 17 days versus 26.5 s. This affordability is what enables the broad coverage and the accuracy and ranking advantages reported in Section 4.2.

4.4. Deployment Validation on Small Cluster

The end-to-end comparison in Section 4.2 evaluates prediction and ranking quality as proxy metrics. To verify that these metrics translate into real gains, we conduct a deployment validation on a small two-node heterogeneous cluster (one 4×V100 node and one 2×T4 node, 6 GPUs in total), training a 6-layer model based on LLaMA-2. The configuration space covers PP, TP, DP, MBS, sequence lengths, four recomputation strategies (none, selective, full-block, full-uniform), and several attention-head and query-group settings. We collected 4,002 per-rank measurements from 667 runs, covering 226 unique configurations with about 3 repetitions each. Each method recommends its best configuration; since some baselines recommend only the parallel dimensions (TP, PP, DP, MBS), we deploy their recommendations with recompute = none and report the resulting range over the measured variants of the remaining dimensions. The memory-balance objective Φ is computed with the safety-first weight setting ($w^T = 0.3, w^M = 0.5, w^V = 0.2$). The results are summarized in Table 10.

Under the iteration-time objective, the deployment result is a tie: AMP, FaSOP, and MemX all identify the true fastest configuration (9,211 ms), showing that the baselines do not fall behind on a simple objective. Metis recommends a slower MBS=4 variant (only 28th to 60th fastest), and HexiScale's recommendation cannot even be run as specified: without recomputation, a dimension HexiScale does not search, its configuration exhausts memory and OOMs. However, the time-optimal configuration itself ranks only 20th under Φ, with a memory variance of 9.13×10^5, more than twice that of MemX's pick. Figure 8 quantifies this trade on 18 matched configuration groups: recomputation buys memory balance at a bounded, predictable time cost, and full-block recomputation sits on the mild end of that trade, which is exactly the point MemX's Φ objective exploits.

Under the memory-balance objective, only MemX's recommendation matches the real optimum. Figure 7 represents per-GPU peak memory of the deployed recommendations on the small cluster. Its pick shares the exact same parallel skeleton as AMP and FaSOP (TP2/DP1/PP3, MBS=2), but

Table 11
Prediction accuracy and ranking consistency of MemX on simulated larger clusters extended from the measured 20-GPU data.

Cluster	Time MAPE	Memory MAPE	Spearman ρ	Kendall τ
20 GPUs (measured)	12.17%	6.44%	0.8964	0.7335
40 GPUs (simulated)	14.22%	6.65%	0.8876	0.7123
60 GPUs (simulated)	16.19%	6.56%	0.8693	0.6830

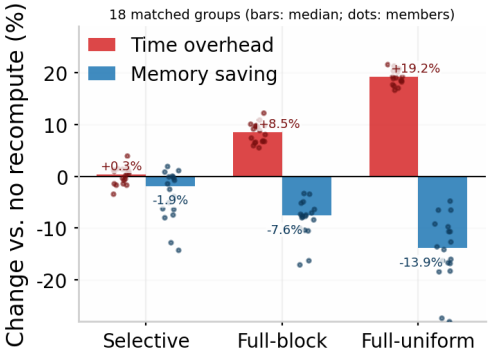


Figure 8: Time overhead versus peak-memory saving of each recomputation strategy relative to recompute = none, over 18 matched configuration groups on the small cluster (same parallel skeleton and hyperparameters; only the recomputation strategy varies). Full-block recomputation (MemX's deployed choice) saves 7.6% peak memory for a median 8.5% time overhead, a milder trade than full-uniform (13.9% saving for 19.2%), while selective recomputation is nearly free but saves little.

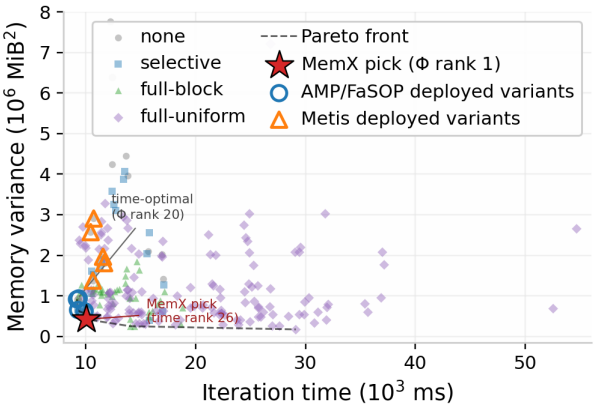


Figure 9: Iteration time versus cross-GPU memory variance for all 226 measured configurations on the small cluster, colored by recomputation strategy; the dashed line is the empirical Pareto front. MemX's deployed pick (star) attains the lowest memory variance (Φ rank 1) at time rank 26, only 9.3% slower than the time-optimal configuration, which itself ranks only 20th under Φ. The deployed AMP/FaSOP and Metis variants (recompute = none) cluster at higher variance.

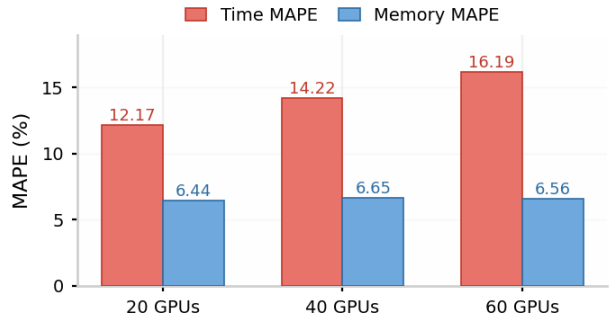
adds full-block recomputation, sequence length 1024, and 64 attention heads with 16 query groups, none of which the baselines can express. At 10,066 ms it is only 9.3% slower than the absolute fastest configuration, yet it attains the lowest and most even memory footprint with mean peak 7,587 MiB and variance 4.19×10^5, ranking 1st in Φ out of all 226 configurations. AMP and FaSOP, blind to these dimensions, land at Φ ranks 12-26 depending on the deployed variant; Metis's pick lands at 67-138 with up to 7x the memory variance of MemX's pick; and HexiScale's pick does not run at all. Across all 226 configurations, the correlation between the time ranking and the Φ ranking is only 0.6568, confirming on this cluster as well that the two objectives cannot proxy each other. Figure 9 maps the full measured space: fast configurations are scattered across a wide range of memory variance, so a time-only search has no signal to land on the balanced region that MemX targets.

In summary, MemX is the only method that wins under both criteria: tied for Top-1 on time, and the sole Top-1 on Φ with the most balanced memory usage.

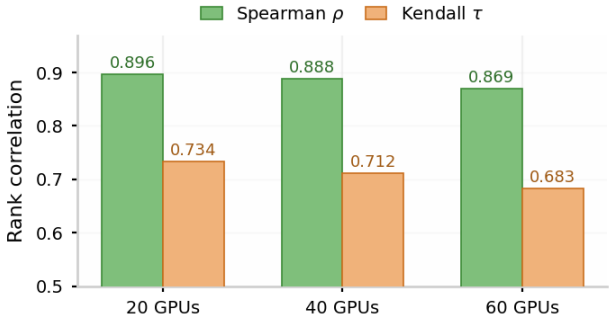
4.5. Simulated Scaling to Larger Clusters

The previous experiments are conducted on a 20-GPU cross-vendor cluster and a 6-GPU small cluster. Since physical cross-vendor resources are limited, we further prove MemX's prediction and ranking quality hold at larger scales of clusters through a simulation experiment by extending the measured 20-GPU cluster data to 40-GPU and 60-GPU clusters.

The extension proceeds in two steps. First, configuration extension: each measured configuration is extended along the pipeline dimension only—a base configuration with parallel strategy (TP, PP, DP) is mapped to extended configurations with PP in 2p, 3p (e.g., PP = 5 extends to 10 or 15 stages, and PP = 10 to 20 or 30 stages, covering 40-rank and 60-rank layouts), while TP, DP, and all other hyperparameters remain unchanged. The LLM model depth grows proportionally with PP, so that the per-stage layer count is preserved while the per-stage execution time and memory footprint of such an extended configuration are inherited from the corresponding measured stages. Second, cluster extension: the added GPUs follow the original vendor mix (approximately 1:2:2 across the three GPU types, i.e., 8/16/16 for 40 GPUs and 12/24/24 for 60 GPUs). Within each extension scheme, the mapping between pipeline stages, GPU types, and ranks is fixed, and tensor-parallel groups are always formed within a single GPU type (no heterogeneous TP). Stage memory usage is taken directly from the measured values, and stage execution time is scaled by the normalized compute-power ratio of the target GPU type (Table 3) relative to the measured one. This procedure adds 10,036 derived configurations to the evaluation set.



(a) Prediction error vs. cluster size



(b) Ranking consistency vs. cluster size

Figure 10: Prediction accuracy and ranking consistency of MemX under simulated scaling from 20 to 60 GPUs (Section 4.5). (a) Mean MAPE grows mildly from 12.17% to 16.19% while the median stays flat; (b) Spearman's $\rho \geq 0.86$ and Kendall's $\tau \geq 0.68$ throughout, indicating that ranking quality degrades gracefully with cluster size.

Table 11 and Figure 10 report the results of Memx's simulation. Prediction accuracy degrades gracefully: time MAPE grows from 12.17% at 20 GPUs to 14.22% at 40 GPUs and 14.91% at 60 GPUs, while memory MAPE remains stable around 6.5%. Ranking quality degrades much more slowly: Spearman's rho stays at or above 0.87 and Kendall's tau at or above 0.69 at all scales. The mild degradation is expected, because PP extension lengthens the pipeline and accumulates per-stage prediction error along the critical path, whereas TP/DP dimensions (which dominate the remaining configuration diversity) are inherited without rescaling. These results indicate that MemX's modeling pipeline remains consistent under PP-dominant scaling, complementing the search-overhead scalability shown in Section 4.3.

Although Time MAPE increases as the simulation scale expands, we argue that the error will not continue to grow linearly to unacceptable levels at extreme scales (e.g., 1000+ GPUs). First, error propagation along the PP critical path is sub-linear: per-stage errors are largely uncorrelated and partially cancel out, so the relative error grows slower than pipeline depth. Second, scaling to 1000 GPUs in practice increases both TP and DP proportionally, which introduces comparatively smaller error than PP; thus, our PP-dominant scaling experiments represent a conservative upper bound. Most importantly, configuration search depends on ranking quality rather than absolute accuracy. Even under the worst-case scenario in our current experiments, Spearman's ρ remains above 0.85, ensuring the utility of search results.

Besides, we note that the extended ground truth is derived from measured stages under compute-ratio scaling. The study therefore validates the internal consistency of MemX's modeling pipeline rather than replacing real large-cluster validation; effects absent from the derivation, such as emerging cross-node communication patterns at 60 GPUs, are not captured.

5. Conclusion and Future Work

We propose MemX, a memory-aware automatic parallelism system for cross-vendor heterogeneous GPU training. MemX formulates configuration search as a joint optimization problem over iteration time, peak memory usage, and cross-device memory variance, and supports heterogeneous recomputation strategies at pipeline-stage granularity. To make the search cost affordable, MemX combines a minimal-set profiler, interpolation-based data augmentation, a dual-output XGBoost predictor, and a hash-signature prediction cache. Experiments against AMP, Metis, HexiScale, and FaSOP show that MemX achieves the best prediction accuracy and ranking consistency, and is the only method that identifies the real Top-1 configuration under both time-based and memory-balance objectives. It also searches substantially larger configuration spaces at much lower cost. Deployment on a small heterogeneous cluster confirms that the predicted memory-balance advantage of MemX finds a runnable, more balanced configuration.

The MemX evaluation in this paper focuses on LLaMA-family models and two moderate-scale clusters, and the objective weights are selected from preset deployment profiles rather than learned online. Future work includes evaluating MemX on larger production clusters with 1000+ GPUs and more model families, studying adaptive weight selection under changing workload and memory conditions, and integrating the memory-aware configurations with cluster-level scheduling and multi-tenant resource management.

Beyond those incremental extensions, a longer-term direction is to automate the entire automatic-parallelism workflow rather than individual decisions within it. A multi-agent scheme could assign profiling, performance modeling, configuration search, and deployment validation to cooperating agents that exchange intermediate results and iteratively refine the configuration search process with little human intervention. Such automation, however, introduces its own systems challenges. Multi-agent orchestration adds non-trivial overhead: inter-agent communication, repeated model invocation, and reasoning latency can easily exceed the wall-clock cost of the deterministic search itself, which in MemX completes within tens of seconds. Reliability is a further concern that LLM-driven agents may produce inconsistent or hard-to-reproduce choices, as an erroneous intermediate decision by a single agent can propagate through

the pipeline. A practical design would therefore keep fast, deterministic components as the execution substrate (e.g., MemX's predictor and prediction cache), and delegate only high-level planning, bottleneck diagnosis, and constraint negotiation to agents, where their flexibility outweighs their cost.

Acknowledgments

This work was supported by 2023YFB4503904 of MOST in China.

References

[1] Advanced Micro Devices, Inc., . AMD Instinct™ MI300X Accelerators. https://www.amd.com/en/products/accelerators/instinct/mi300/mi300x.html.

[2] Cai, Z., Yan, X., Ma, K., Wu, Y., Huang, Y., Cheng, J., Su, T., Yu, F., 2021. Tensoropt: Exploring the tradeoffs in distributed dnn training with auto-parallelism. IEEE Transactions on Parallel and Distributed Systems 33, 1967–1981.

[3] Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794.

[4] Chen, T., Xu, B., Zhang, C., Guestrin, C., 2016. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 .

[5] Fan, S., Rong, Y., Meng, C., Cao, Z., Wang, S., Zheng, Z., Wu, C., Long, G., Yang, J., Xia, L., et al., 2021. Dapple: A pipelined data parallel approach for training large models, in: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 431–445.

[6] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al., 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783 .

[7] Guo, J., Ma, T., Gao, W., Sun, P., Li, J., Chen, X., Jin, Y., Lin, D., 2025. Octopipe: Reducing pipeline bubbles for heterogeneous models via co-optimizing partitioning, placement, and scheduling. arXiv preprint arXiv:2509.23722 .

[8] Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M.X., Chen, D., Lee, H., Ngiam, J., Le, Q.V., Wu, Y., et al., 2018. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1811.06965 .

[9] Hwang, S., Lee, E., Oh, H., Yi, Y., 2024. Fasop: Fast yet accurate automated search for optimal parallelization of transformers on heterogeneous gpu clusters, in: Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing, pp. 253–266.

[10] Iluvatar CoreX, . Iluvatar BI-V150. https://www.iluvatar.com/productDetails?fullCode=cpjs-yj-xlxl-tg150.

[11] Intel Corporation, . Intel Gaudi 3 AI Accelerators. https://www.intel.com/content/www/us/en/products/details/processors/ai-accelerators/gaudi.html.

[12] Jia, Z., Lin, S., Qi, C.R., Aiken, A., 2018. Exploring hidden dimensions in parallelizing convolutional neural networks, in: ICML, pp. 2279–2288.

[13] Jia, Z., Zaharia, M., Aiken, A., 2019. Beyond data and model parallelism for deep neural networks. Proceedings of Machine Learning and Systems 1, 1–13.

[14] Korthikanti, V.A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., Catanzaro, B., 2023. Reducing activation recomputation in large transformer models. Proceedings of Machine Learning and Systems 5, 341–353.

[15] Li, D., Wang, H., Xing, E., Zhang, H., 2022. Amp: Automatically finding model parallel strategies with heterogeneity awareness. Advances in Neural Information Processing Systems 35, 6630–6639.

[16] Li, H., Fu, F., Ge, H., Lin, S., Wang, X., Niu, J., Wang, Y., Zhang, H., Nie, X., Cui, B., 2025a. Malleus: Straggler-resilient hybrid parallel training of large-scale models via malleable data and model parallelization. Proceedings of the ACM on Management of Data 3, 1–28.

[17] Li, S., Hoefler, T., 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–14.

[18] Li, S., Lai, Z., Li, D., Hao, Y., Liu, W., Ge, K., Deng, X., Lu, K., 2025b. Oases: Efficient large-scale model training on commodity servers via overlapped and automated tensor model parallelism. IEEE Transactions on Parallel and Distributed Systems .

[19] Liang, A., Zhao, Z., Zhang, K., Shi, X., Li, C., Wang, C., He, Z., Jing, Y., Wang, X.S., 2026. Harp: Orchestrating automated parallel training on heterogeneous gpu clusters, in: Proceedings of the 21st European Conference on Computer Systems, pp. 1160–1180.

[20] Liu, G., Miao, Y., Lin, Z., Shi, X., Maleki, S., Yang, F., Bao, Y., Wang, S., 2024. Aceso: Efficient parallel dnn training through iterative bottleneck alleviation, in: Proceedings of the Nineteenth European Conference on Computer Systems, pp. 163–181.

[21] Liu, Y., Li, S., Fang, J., Shao, Y., Yao, B., You, Y., 2023. Colossal-auto: Unified automation of parallelization and activation checkpoint for large-scale models. arXiv preprint arXiv:2302.02599 .

[22] Miao, X., Wang, Y., Jiang, Y., Shi, C., Nie, X., Zhang, H., Cui, B., 2022. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. arXiv preprint arXiv:2211.13878 .

[23] Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N.R., Ganger, G.R., Gibbons, P.B., Zaharia, M., 2019. Pipedream: Generalized pipeline parallelism for dnn training, in: Proceedings of the 27th ACM symposium on operating systems principles, pp. 1–15.

[24] Narayanan, D., Phanishayee, A., Shi, K., Chen, X., Zaharia, M., 2021a. Memory-efficient pipeline-parallel dnn training, in: International Conference on Machine Learning, PMLR. pp. 7937–7947.

[25] Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al., 2021b. Efficient large-scale language model training on gpu clusters using megatron-lm, in: Proceedings of the international conference for high performance computing, networking, storage and analysis, pp. 1–15.

[26] Ou, J., Guo, J., Chen, Y., Jiang, S., Xue, R., Tian, W., 2025a. A neural network-based pipeline parallel strategy solver for heterogeneous environments, in: 2025 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–9.

[27] Ou, J., Li, X., Chen, Y., Qian, J., Su, T., Tian, W., 2025b. Efficient and automatic 3d parallelism strategies search via contrastive reinforcement learning pretrained neural networks, in: 2025 IEEE 12th International Conference on Data Science and Advanced Analytics (DSAA), IEEE. pp. 1–10.

[28] Qi, G., Su, J., Yang, L., Li, T., Xie, T., Sun, Y., Xie, Y., Wu, C., 2026. Hetauto: Cross-cluster auto-parallelism for heterogeneous distributed training, in: Proceedings of the 21st European Conference on Computer Systems, pp. 759–779.

[29] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., Catanzaro, B., 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053 .

[30] Shong, J., Ding, Y., Jiang, Y., Jing, L., Chen, H., Zhang, G., Akella, A., Lai, F., 2026. Autoscout: Structured optimization for automating ml system configuration. arXiv preprint arXiv:2603.11603 .

[31] Singh, A., Fry, A., Perelman, A., Tart, A., Ganesh, A., El-Kishky, A., McLaughlin, A., Low, A., Ostrow, A., Ananthram, A., et al., 2025. Openai gpt-5 system card. arXiv preprint arXiv:2601.03267 .

[32] Snoek, J., Larochelle, H., Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems 25.

[33] Song, L., Chen, F., Zhuo, Y., Qian, X., Li, H., Chen, Y., 2020. Accpar: Tensor partitioning for heterogeneous deep learning accelerators, in:

2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE. pp. 342–355.
[34] Song, L., Mao, J., Zhuo, Y., Qian, X., Li, H., Chen, Y., 2019. Hypar: Towards hybrid parallelism for deep learning accelerator array, in: 2019 IEEE international symposium on high performance computer architecture (HPCA), IEEE. pp. 56–68.
[35] Strati, F., Zhang, Z., Manos, G., Périz, I.S., Hu, Q., Chen, T., Buzcu, B., Han, S., Delgado, P., Klimovic, A., 2025. Sailor: Automating distributed training over dynamic, heterogeneous, and geo-distributed clusters, in: Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, pp. 204–220.
[36] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al., 2023a. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 .
[37] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al., 2023b. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 .
[38] Um, T., Oh, B., Kang, M., Lee, W.Y., Kim, G., Kim, D., Kim, Y., Muzzammil, M., Jeon, M., 2024. Metis: Fast automatic distributed training on heterogeneous gpus, in: 2024 USENIX Annual Technical Conference (USENIX ATC 24), pp. 563–578.
[39] Wang, M., Huang, C.c., Li, J., 2019. Supporting very large models using automatic dataflow graph partitioning, in: Proceedings of the Fourteenth EuroSys Conference 2019, pp. 1–17.
[40] Wang, W., Lai, Z., Li, S., Liu, W., Ge, K., Shen, A., Su, H., Li, D., 2024. Pro-prophet: A systematic load balancing method for efficient parallel training of large-scale moe models. arXiv preprint arXiv:2411.10003 .
[41] Wang, Y., Tang, N., Wang, Y., Pan, S., Yu, D., Wang, Z., Sun, M., Fu, K., Wang, F., Chen, Y., et al., 2026. Autohaap: Automated heterogeneity-aware asymmetric partitioning for llm training, in: 2026 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE. pp. 1–17.
[42] Wu, R., Li, X., Wang, Y., Chen, X., Kutscher, D., 2025. Rethinking dynamic networks and heterogeneous computing with automatic parallelization, in: Proceedings of the 9th Asia-Pacific Workshop on Networking, pp. 164–171.
[43] Yan, R., Jiang, Y., Nie, X., Fu, F., Cui, B., Yuan, B., . Hexiscale: Facilitating large language model training over heterogeneous hardware, in: Ninth Conference on Machine Learning and Systems.
[44] Yun, G., Choi, Y.r., 2026. Hpmd: Enabling hybrid parallelism with multi-dimensional adaptive dnn training, in: Proceedings of the 40th ACM International Conference on Supercomputing, pp. 1027–1039.
[45] Zeng, A., Lv, X., Hou, Z., Du, Z., Zheng, Q., Chen, B., Yin, D., Ge, C., Huang, C., Xie, C., et al., 2026. Glm-5: from vibe coding to agentic engineering. arXiv preprint arXiv:2602.15763 .
[46] Zhang, S., Diao, L., Wu, C., Cao, Z., Wang, S., Lin, W., 2024. Hap: Spmd dnn training on heterogeneous gpu clusters with automated program synthesis, in: Proceedings of the Nineteenth European Conference on Computer Systems, pp. 524–541.
[47] Zhang, W., Hu, Y., Shi, J., Bai, X., 2025. Poplar: Efficient scaling of distributed dnn training on heterogeneous gpu clusters, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 22587–22595.
[48] Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E.P., et al., 2022. 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.
[49] Zhu, Z., Giannoula, C., Andoorveedu, M., Su, Q., Mangalam, K., Zheng, B., Pekhimenko, G., 2025. Mist: Efficient distributed training of large language models via memory-parallelism co-optimization, in: Proceedings of the Twentieth European Conference on Computer Systems, pp. 1298–1316.
[50] Zong, Z., Guo, M., Zhai, M., Tang, Y., Li, J., Zhai, J., 2025. Training large models on heterogeneous and geo-distributed resource with constricted networks. Big Data Mining and Analytics 8, 966–980.



Lin Meng received the B.S. degree from Southeast University, Jiangsu, China in 2020. He is pursuing his Ph.D. degree at the Institute of Computing Technology (ICT), Chinese Academy of Sciences. His research interests include distributed machine learning systems and communication optimizations.



Yuzhong Sun received the PhD degree in computer engineering from the Institute of Computing Technology (ICT), Chinese Academy of Sciences. He is a professor at ICT, China. His research interests focus on distributed system software and computing/programming for model training and inference.



Jie Zhu received the Ph.D degree in applied computer science from the School of Computer Science and Engineering, Southeast University, Nanjing, in 2011. She joined Nanjing University of Post & Telecommunication in 2014, where she is currently an Associate Professor with the School of Computer Science. Her research interests include Machine Scheduling, Project Scheduling, Workflow Optimization and Cloud Computing.