Frenzy: A Memory-Aware Serverless LLM Training System for Heterogeneous GPU Clusters

Zihan Chang[1,2,3], Sheng Xiao[1], Shuibing He[1,2,3(✉)], Xuechen Zhang[4], Siling Yang[1], Zhenxin Li[1,2,3], Zhe Pan[1], and Weijian Chen[1]

[1] Zhejiang University, Hangzhou, China
[2] Zhejiang Lab, Hangzhou, China
[3] Zhejiang Key Laboratory of Big Data Intelligent Computing, Hangzhou, China
heshuibing@zju.edu.cn
[4] Kennesaw State University, Kennesaw, USA

Abstract. Training large language models (LLMs) on heterogeneous GPU clusters is increasingly common, but resource provisioning remains a core bottleneck. Existing systems either require users to manually choose GPU types and counts or rely on static memory formulas that miss runtime factors (e.g., allocator behavior and communication buffers), leading to OOM failures or conservative over-provisioning. At the same time, heterogeneous scheduling often depends on expensive optimization procedures that introduce high scheduling latency. To solve this problem, we propose Frenzy, a memory-aware serverless LLM training system for heterogeneous GPU clusters. Frenzy first estimates peak training memory using analytical modeling calibrated by runtime feedback to infer robust GPU type-and-quantity plans. It then applies a pricing-driven greedy scheduler with lightweight local swaps to achieve efficient near-optimal placement with low overhead. We evaluate Frenzy using multi-task LLM training on a physical heterogeneous cluster with 8 nodes and 4 GPU types. Compared with state-of-the-art baselines, Frenzy reduces average job completion time by 12%–50%, lowers scheduling overhead by 10–20×, and achieves over 97% memory prediction accuracy.

Keywords: Serverless · Heterogeneous · Memory-aware

1 Introduction

Training large language models (LLMs) on heterogeneous GPU clusters is increasingly important as production and research environments become more diverse [4–8]. In principle, heterogeneity can improve utilization by matching workloads to devices with different memory and compute profiles. In practice,

Z. Chang and S. Xiao—Equal contribution.

M. Torquati et al. (Eds.): Euro-Par 2026, LNCS 16782, pp. 91–105, 2027.
https://doi.org/10.1007/978-3-032-35251-4_7

however, existing systems typically assume that users already know the right GPU types and counts, and then focus on optimizing scheduling under that fixed decision [8]. This leaves a major gap between model submission and robust execution.

The key difficulty is that resource planning and scheduling are tightly coupled. Provisioning decisions based on static formulas are often inaccurate because they ignore runtime-dependent factors such as allocator behavior, communication buffers, and kernel overheads, which can cause either OOM failures or conservative over-provisioning. Meanwhile, heterogeneous scheduling over a large combinatorial space can incur high overhead when solved with heavy optimization methods such as ILP [1,8]. Together, these limitations make end-to-end heterogeneous LLM training brittle and costly.

This paper introduces **Frenzy**, a memory-calibrated serverless system that explicitly targets this planning–scheduling gap. First, Frenzy determines the required GPU types and quantities through an adaptive estimation of peak memory consumption during LLM training. Upon receiving a training task, Frenzy incorporates the specified parallelism strategy together with model hyperparameters to analytically estimate model-state and activation memory. Instead of treating this analytical result as a final decision, Frenzy refines the estimate via runtime feedback to obtain a calibrated peak memory prediction. Based on this calibrated estimate, Frenzy generates feasible and high-utilization resource planning options across heterogeneous GPU types.

Furthermore, we design a scheduling system for heterogeneous GPU clusters that efficiently allocates GPU resources through a pricing-driven greedy algorithm augmented with lightweight local swaps. This approach achieves near-ILP scheduling optimality with substantially lower computational cost, enabling efficient and low-overhead task-to-cluster scheduling.

Specifically, Frenzy contributes:

- **A serverless abstraction for heterogeneous LLM training** that removes manual GPU-type and GPU-count selection from users and turns model submission into automatic resource provisioning.
- **MARP-F**, a feedback-driven memory-aware resource planner that treats analytical peak-memory estimation as a prior, then calibrates it with runtime observations to produce robust and high-utilization GPU allocation plans.
- **HAS**, a low-overhead heterogeneity-aware scheduler that uses pricing-driven greedy placement with local swap refinement, achieving near-ILP quality at substantially lower scheduling cost.
- Empirical validation on real and simulated heterogeneous clusters, showing 92%–98% memory prediction accuracy, 10× lower scheduling overhead, and 12%–50% lower average JCT.

2 Background and Motivation

Serverless Computing. Serverless platforms abstract GPU provisioning and resource selection so that users submit code and configurations rather than man-

ually managing devices. However, LLM training is stateful and long-running, making timeout and worker volatility important concerns. Frenzy therefore uses serverless mainly as a control-plane abstraction: training state is preserved through checkpoints, while transient workers can be relaunched or rescheduled after timeout, failure, or OOM, and Ray/NCCL communication is rebuilt for the recovered worker group.

Heterogeneous Training. Heterogeneous clusters integrate diverse GPUs, improving utilization by repurposing existing hardware instead of relying exclusively on uniform, high-end accelerators. However, these GPUs vary significantly in compute and memory capacity (e.g., 40–80 GB A100s [12] vs. 24 GB RTX 3090 s [13]), which strictly dictates maximum batch sizes and parameter limits. Consequently, efficiently orchestrating tasks across these disparate devices—while exploiting high-bandwidth intra-node interconnects like NVLink—is critical for maximizing training throughput.

2.1 Gap Analysis

Recent systems improve parts of the heterogeneous training pipeline, but none closes the full loop from model submission to robust execution. Serverless frameworks such as ElasticFlow [9] simplify submission and elastic scaling, yet they are primarily designed for homogeneous settings and do not explicitly reason about cross-GPU memory heterogeneity. Heterogeneity-aware schedulers such as Metis [7] and Sia [8] optimize placement once candidate resources are specified, but they still rely on assumptions about feasible GPU types/counts and can incur high optimization overhead.

This creates a practical gap: *resource planning* (which GPUs and how many) and *resource scheduling* (where to place jobs) are handled separately, even though they are strongly coupled in heterogeneous clusters. As a result, users still face trial-and-error behavior, unstable OOM outcomes, and long scheduling latency under dynamic workloads.

2.2 Challenges

Challenge 1: Robust Resource Planning Under Memory Uncertainty. The first challenge is to decide feasible GPU type-and-count combinations before job execution. Static memory formulas are useful priors but are often inaccurate in practice because peak memory is affected by runtime-dependent factors such as allocator behavior, communication buffers, temporary workspaces, and fragmentation. In heterogeneous clusters, this issue is amplified by diverse memory capacities and software/hardware interactions across GPU types. Consequently, plans that appear feasible may still fail with OOM, while conservative plans waste expensive accelerators.

Challenge 2: Low-Overhead Scheduling in a Combinatorial Space. Given feasible resource plans, the scheduler must map jobs onto heterogeneous

nodes while balancing compute mismatch, communication overhead, and post-allocation fragmentation. Exact or near-exact solvers can become expensive as cluster size and workload diversity grow, causing scheduling delay to become a non-negligible component of job completion time. The challenge is to retain high-quality allocation decisions without paying ILP-level optimization cost.

Challenge 3: Coupling Planning and Scheduling End-to-End. The final challenge is to connect planning and scheduling in one closed loop. Planning outputs should be aware of downstream placement difficulty, and scheduling decisions should reflect calibrated memory feasibility rather than static estimates alone. Without this coupling, systems either over-fit to theoretical memory models or over-spend computation on scheduling, failing to provide consistent performance across real heterogeneous clusters.

To address these challenges, Frenzy integrates feedback-calibrated memory planning (MARP-F) with low-overhead heterogeneity-aware scheduling (HAS), enabling robust and efficient end-to-end LLM training.

3 Design

To address the three challenges identified in Sect. 2, Frenzy integrates MARP-F (Memory-Aware Resource Planner with Feedback) and HAS (Heterogeneity-Aware Scheduler). The overall architecture is shown in Fig. 1.

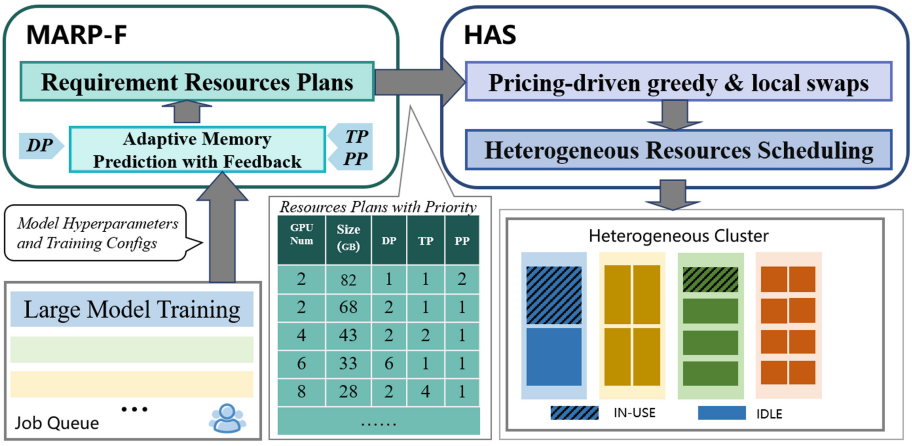

Fig. 1. Frenzy Overview. User submits a LLM training job. **MARP-F** combines analytical memory estimation with runtime feedback to adaptively predict peak training memory under various parallel strategies, and produces robust resource planning options across heterogeneous GPUs. **HAS** takes the selected resource allocation plan and performs heterogeneous GPU scheduling across the cluster using its pricing-driven greedy placement and local swap refinement.

3.1 Memory-Aware Resource Planner with Feedback

MARP-F is a feedback-driven memory planning module that converts user-specified model and training configurations into feasible, high-utilization parallelism plans. As shown in Fig. 2, MARP-F first derives an *initial* plan using *Static Memory Estimation* as a prior. It then runs a lightweight *feedback* probe to measure real peak memory and systematic overheads, calibrates the estimator, and finally outputs an *optimized* plan through a Plan–Probe–Update loop that both avoids OOM and reduces over-provisioning. The key contribution of MARP-F is the closed-loop mechanism that (i) quantifies estimator bias using runtime measurements, (ii) updates memory feasibility with calibrated safety margins, and (iii) feeds calibrated plans to downstream heterogeneous scheduling. This design turns static memory modeling into robust system behavior across diverse GPU types and software environments.

Static Memory Estimation (prior). The static estimator decomposes the per-GPU peak memory into three analytical components: (i) *model-state (static) memory* for parameters/gradients/optimizer states, (ii) *activation (dynamic) memory* mainly determined by micro-batch size and sequence length, and (iii) *logits projection memory* that scales with vocabulary size. Concretely, given model hyperparameters (V, h, l, a), training inputs (s, B), and parallelism degrees (d, t, p), we compute a predicted peak memory

$$M_{\text{pred}} = M_{\text{static}}(t, p) + M_{\text{act}}(d, t; B, s) + M_{\text{logits}}(t; B, s, V)$$

and select a feasible initial plan (d, t, p, b) (with $b = B/d$) by enforcing

$$M_{\text{pred}} < C_{\text{GPU}}$$

where C_{GPU} is the target GPU memory capacity. Among feasible candidates, the initial plan can be chosen to minimize GPU count $N = d \cdot t \cdot p$ or maximize training throughput under user constraints.

Adaptive Memory Estimation with Feedback. The key limitation of static estimation is that M_{pred} may deviate from true peak memory because of environment-dependent effects (e.g., allocator behavior, temporary workspaces, communication buffers, and fragmentation). MARP-F addresses this limitation by treating the analytical estimate as a prior and introducing a feedback loop, as illustrated on the right side of Fig. 2.

(1) Initial Plan. MARP-F starts from the static estimator and produces an initial plan (d, t, p, b) that is predicted to fit in memory. Importantly, this plan is not final; it is a *starting point* that will be refined using runtime observations.

(2) Probe Run. The system then runs a short probe (a few iterations) under the initial plan and monitors runtime telemetry, including peak allocated memory, peak reserved memory, and fragmentation indicators. This probe is designed to expose early unmodeled overheads with low cost, rather than assuming that a single probe fully characterizes the entire training lifecycle. *(3) Calibrate & Adjust.*

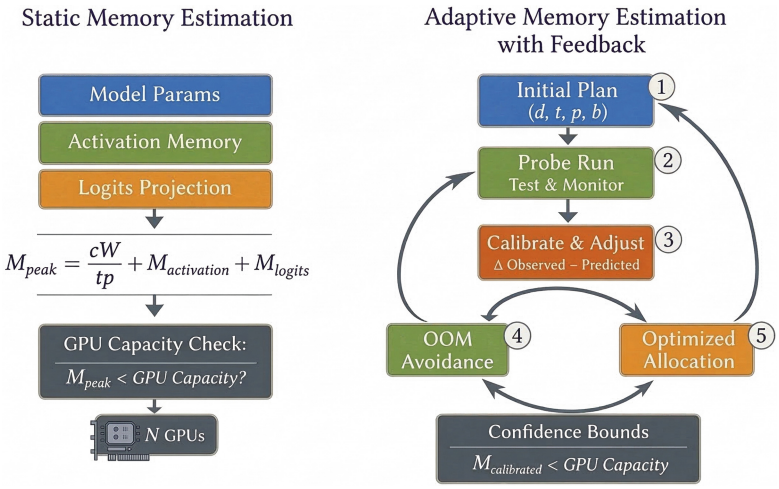

Fig. 2. Left: a one-shot analytical estimator predicts peak memory from model-state and activation terms and checks against GPU capacity. Right: MARP-F treats the estimate as a prior and refines it via a feedback-driven Plan–Probe–Update loop to produce robust, high-utilization GPU allocations.

MARP-F computes the deviation

$$\Delta = M_{\text{obs}} - M_{\text{pred}}$$

and uses it to calibrate an environment-aware correction term, producing a refined estimate M_{cal} (or a safe upper bound). This step absorbs systematic biases introduced by the software/hardware stack and stabilizes prediction across heterogeneous GPUs.

(4) OOM Avoidance. If the probe indicates insufficient headroom (or triggers OOM), MARP-F applies a minimal-perturbation mitigation strategy, prioritizing adjustments with the smallest performance impact—e.g., reducing micro-batch size, enabling/increasing activation checkpointing, or re-tuning (d, t, p). Failures or memory-drift signals are treated as informative feedback: Frenzy re-enters the Plan–Probe–Update loop and adjusts the micro-batch size, checkpointing level, or parallelism plan instead of terminating the job. *(5) Optimized Allocation.* If the probe reveals excessive safety margin, MARP-F tightens the plan to improve utilization, e.g., increasing micro-batch size or reducing the number of GPUs, while maintaining feasibility. This step reduces conservative over-provisioning that is common in purely analytical planning.

(6) Confidence Bounds and Termination. Finally, MARP-F enforces feasibility using calibrated estimates with explicit safety margins (or statistical confidence bounds), ensuring robustness against runtime variance. The planner iterates the Plan–Probe–Update loop until it converges to a stable allocation that satisfies $M_{\text{cal}} < C_{\text{GPU}}$ and the chosen optimization objective.

3.2 Heterogeneity-Aware Schedule

Given the execution plan from MARP-F, HAS schedules jobs using the currently available resources in the heterogeneous cluster. As illustrated in Fig. 3, HAS first generates a small set of candidate placements for each job, then selects an allocation using a **pricing-driven greedy algorithm**, and finally refines the result through **lightweight local swaps**. This design achieves near-ILP scheduling quality while maintaining substantially lower computational overhead.

Let the job set be J, the GPU type set T, and the node set N. Each node n provides capacity $\mathrm{Cap}_{n,t}$ for GPU type t. For job j and GPU type t, the required GPU count is $g_{j,t}$, and the number of GPUs allocated from node n is denoted by $x_{j,n,t}$. The allocation must satisfy

$$\sum_{n \in N} x_{j,n,t} = g_{j,t}, \quad 0 \le x_{j,n,t} \le \mathrm{Cap}_{n,t}, \quad \sum_{j} x_{j,n,t} \le \mathrm{Cap}_{n,t}$$

To balance performance degradation, communication cost, and resource utilization, HAS evaluates each candidate allocation using a composite score function:

$$\mathrm{Score}(j,t,\mathbf{x}) = \alpha\, r_{j,t} + \beta \left|\{n \mid x_{j,n,t} > 0\}\right| + \gamma\, \mathrm{Topo}(\{n \mid x_{j,n,t} > 0\}) + \delta\, \mathrm{FragAfter}(t,\mathbf{x}) + \sum_{n} p_{n,t}\, x_{j,n,t}$$

Here, $r_{j,t}$ represents the relative slowdown of running job j on GPU type t (e.g., the slowdown of V100 relative to A100 can be approximated by $r_{\mathrm{A100,V100}} = \mathrm{TFLOPS}_{\mathrm{A100}}/\mathrm{TFLOPS}_{\mathrm{V100}}$). The second term encourages *compact placement* by minimizing the number of involved nodes. $\mathrm{Topo}(\cdot)$ captures inter-node communication cost (e.g., 0 for intra-node, 1 for inter-node, and 2 for cross-rack communication). FragAfter penalizes post-placement resource fragmentation, discouraging allocations that leave small unusable GPU fragments across nodes, for example:

$$\sum_{n} \max(0, \tau - \mathrm{rem}_{n,t}^{\mathrm{after}})/\tau$$

Finally, $p_{n,t}$ is a dynamic *price variable* reflecting the scarcity of node–type pairs. The coefficients α, β, γ, and δ balance performance degradation, node compactness, communication overhead, and fragmentation cost.

Pricing-Driven Greedy. HAS adopts a pricing-driven greedy strategy to construct the initial schedule. All price variables are initialized as $p_{n,t} = 0$, and jobs are sorted by scheduling difficulty (e.g., descending $\min_t g_{j,t}$ or ascending feasible node count). For each job–type pair (j,t), the scheduler generates a small number of candidate placements: preferably single-node allocations, and otherwise 1–3 minimal-node covering sets obtained via a Best-Fit-Decreasing heuristic. The candidate with the lowest $\mathrm{Score}(j,t,\mathbf{x})$ is selected.

To incorporate global resource scarcity into local decisions, HAS dynamically updates price variables after each scheduling iteration using a subgradient rule:

$$p_{n,t} \leftarrow \max\big(0,\ p_{n,t} + \eta(\text{used}_{n,t} - \text{Cap}_{n,t})\big), \quad \text{used}_{n,t} = \sum_j x_{j,n,t}$$

Here η is a learning rate that decays across rounds. Intuitively, heavily utilized node–type pairs receive higher prices, which discourages subsequent allocations from selecting the same resources. This pricing mechanism introduces a lightweight global feedback loop (Fig. 3), allowing the greedy procedure to adaptively avoid contention and improve cluster-wide utilization.

The complexity of each pricing round is approximately

$$O(|J| \log |J| + |J||T|Kd)$$

where K is the number of candidate placements per job–type pair and d is the average number of nodes per candidate.

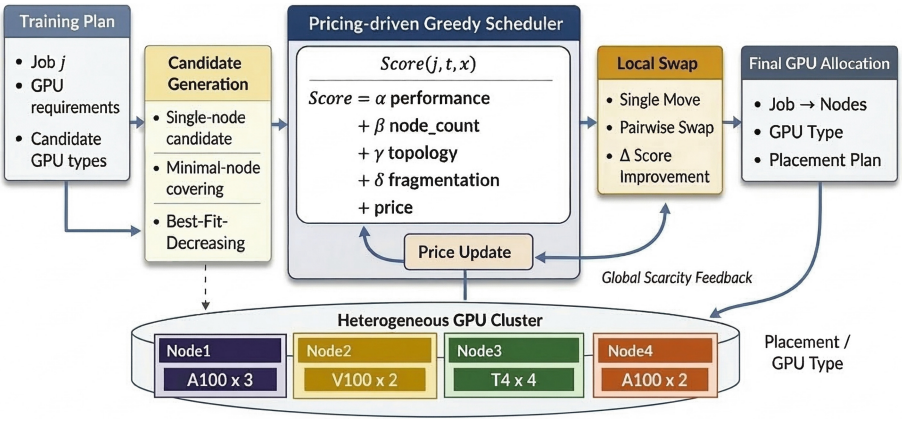

Fig. 3. Overview of HAS, which performs heterogeneous GPU scheduling using a pricing-driven greedy algorithm with dynamic scarcity feedback, followed by lightweight local swaps to refine the final allocation.

Local Swap. To further improve the greedy solution, HAS performs a lightweight local search stage. This stage explores two types of refinement operations: (i) *single-move*, which relocates $x_{j,n,t}$ to another node n', and (ii) *pairwise-swap*, which exchanges allocations between two jobs j_1 and j_2. A candidate move is accepted if it reduces the objective, i.e., $\Delta\text{Score} < -\varepsilon$, where ε is a small acceptance threshold.

The search is bounded by a small number of iterations L and candidate swaps q per iteration, resulting in additional complexity $O(Lq)$ (e.g., $L = 10$ and $q = 3$), which is negligible compared to the greedy scheduling stage.

Approximation Guarantee and System-Level Complexity. Although HAS uses a greedy construction, its composite score exhibits a near-submodular structure, yielding a constant-factor approximation within the candidate space. Its core novelty lies in its system-level application: dynamic pricing acts as dual-like feedback, approximating a Lagrangian relaxation of capacity constraints to globally balance placements without the exponential overhead of global ILP solvers. By bounding the greedy phase to $O(|J|\log|J| + |J||T|Kd)$ and local search to $O(Lq)$, HAS guarantees near-linear scheduling latency. These heuristics empirically perform within a few percent of ILP-optimal solutions at a negligible computational cost.

TP Placement Guarantee. To prevent severe communication degradation across heterogeneous links, HAS enforces a strict topology boundary during candidate generation. It guarantees that all Tensor Parallelism (TP) groups are placed entirely within a single physical node utilizing high-bandwidth interconnects (e.g., NVLink). By ensuring the TP degree t never exceeds a single node's capacity ($t \leq Cap_{n,t}$), HAS effectively shields the training process from the prohibitive synchronization overheads of cross-node tensor parallelism.

4 Evaluation

4.1 Setup

Cluster Configuration and Simulator

(1) We conducted LLM training scheduling experiments with Ray on a physical heterogeneous cluster. The cluster includes 1 head node with 4 × A800 80G GPUs (NVLink), 2 nodes with 2 × A100 40G GPUs (PCIe), 2 nodes with 1 × A100 40G GPU, 1 node with 2 × H800 80G GPUs (NVLink), and 2 nodes with 2 × A100 80G GPUs (PCIe), for a total of 8 nodes and 4 GPU types.
(2) **Simulator.** We extend the Alibaba Cloud PAI simulator [25] to support heterogeneous-cluster scheduling. The simulator emulates GPUs with different memory capacities (e.g., A100 40G and T4 16G) and reproduces job-level events including submission, arrival, execution, and completion. We calibrate simulator inputs using real-cluster measurements of throughput, scheduling overhead, training time, and GPU utilization. The resulting simulator achieves high fidelity, with less than 5% error relative to real-cluster execution.

Workload and Traces. *Philly* [5] is a dataset derived from over 100,000 jobs executed over a two-month period in a multi-tenant cluster with multiple GPU types at Microsoft. This dataset provides valuable insights into the real-world usage patterns and challenges of managing large-scale, heterogeneous GPU clusters. *Helios* [20] is a real-world workload dataset from SenseTime, consisting of four clusters and over 3.3M tasks. Compared to *Philly*, *Helios* requires more GPUs and has longer runtimes. We also used *NewWorkload*, which includes

GPT-2 models at 350M, 1.3B, and 4.2B [21], OPT models at 6.7B and 13B, and BERT models at 110M and 340M [22], each with varying batch sizes, to construct 1,000 LLM task queues.

Baseline. We compare against five baselines: RAR, Sia, *Opportunistic Scheduling*, ElasticFlow, and MLaaS.

RAR [19] performs parameter analysis for LLMs in mixed-precision training, enabling GPU memory usage prediction.

Sia [8] optimizes deep learning (DL) scheduling in heterogeneous clusters, enabling adaptive scheduling for tasks with user-specified GPU counts.

Opportunistic Scheduling [23] denotes the version with opportunistic scheduling strategy, which always prioritizes nodes with higher computational power in heterogeneous cluster scheduling. It follows a first-come, first-served (FCFS) policy, greedily allocating idle resources to newly submitted tasks.

ElasticFlow [9] enables serverless training in homogeneous clusters by using a minimum-satisfaction control mechanism to dynamically estimate GPU counts.

MLaaS [4] performs task scheduling on large-scale clusters based on the Alibaba PAI platform and supports heterogeneous GPU scheduling.

4.2 General Result

We evaluate the throughput and average JCT on the top 300 NewWorkload tasks using the 8-node, 16-GPU heterogeneous cluster. As shown in Fig. 4, Frenzy achieves the highest throughput and saturates later than Sia, ElasticFlow-BE, and Opportunistic Scheduling, indicating better resource utilization under contention. Frenzy also reduces QT and JCT by 15–25% over Sia, around 30% over ElasticFlow-BE, and up to 40% over Opportunistic Scheduling. These gains come from calibrated memory planning in MARP-F and low-overhead heterogeneous placement in HAS.

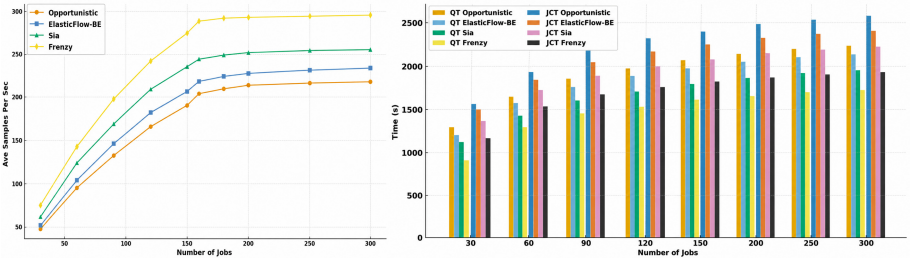

Fig. 4. Comparison of throughput scalability (left) and latency performance (right) under increasing LLM workloads.

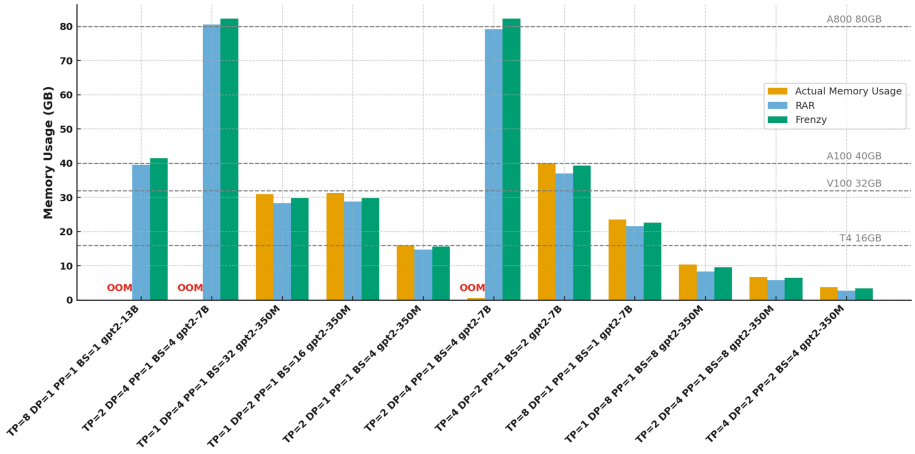

Fig. 5. Comparison between Frenzy and RAR memory predictions and the actual GPU memory usage.

4.3 Impact of Individual Techniques

MARP. Figure 5 compares MARP-F with RAR across GPT-2 models under varying model sizes, parallelism strategies, and batch sizes. MARP-F achieves 98% memory prediction accuracy and reliably detects OOM cases in large-scale settings (e.g., gpt2-7B and gpt2-13B with high TP/DP), where RAR often overestimates feasibility. Validation across common GPU capacities (16–80 GB) further confirms MARP's robustness in modeling realistic memory footprints. Its superior accuracy stems from fine-grained, LLM-specific memory analysis.

HAS. We evaluate HAS against Sia and ElasticFlow on 8 heterogeneous nodes. As workload scales from 1K to 10K tasks (Fig. 6, left), Frenzy exhibits the slowest overhead growth, reaching only 2.69 at 10K tasks, compared with 27.45 for Sia and 57.32 for ElasticFlow-BE. Trace-driven evaluation on *Helios* [20] and *Philly* [5] (Fig. 6, right) shows that Frenzy maintains a JCT comparable to Sia while reducing overhead by 10×, and reduces JCT by 28% with 20× lower overhead relative to ElasticFlow-BE. This gain comes from HAS's pricing-driven greedy design with lightweight local swaps, which approximates ILP-level quality at much lower cost.

4.4 Large-Scale Testing

To evaluate scalability, we conducted large-scale simulations using a validated heterogeneous-cluster simulator. The simulated cluster contains three GPU types (T4, V100, and A100), each with 128 nodes of 8 GPUs. We replay 10,000 tasks from the Helios trace on clusters ranging from 1,000 to 9,000 GPUs.

Table 1 shows that Frenzy consistently achieves the lowest average JCT across all cluster sizes, outperforming MLaaS, Sia, and Opportunistic scheduling. On

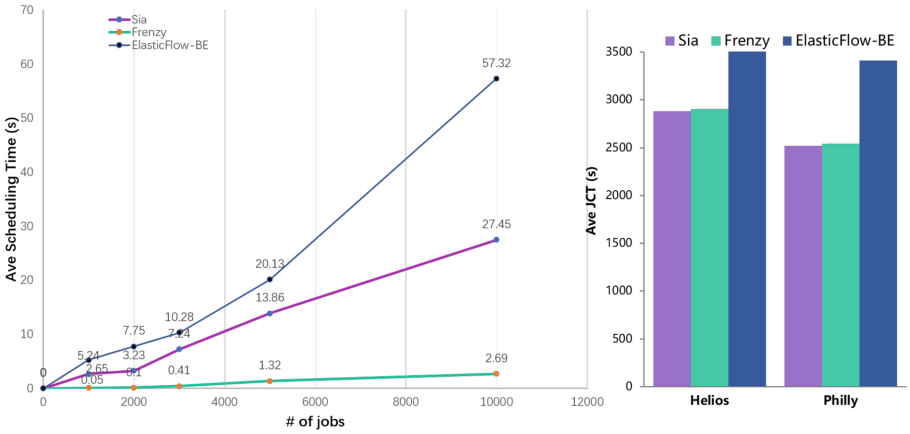

Fig. 6. Frenzy scheduling result compared with Sia and ElasticFlow's best effort (BE) strategy.

Table 1. Average JCT (s) under different GPU cluster sizes in large-scale simulation experiments.

GPU Num	MLaaS	Frenzy	Opportunistic	Sia
1500	35013	24881	52400	34412
2000	25100	18103	46308	23703
3000	18223	10534	29482	18902
4000	13801	7216	21943	12511
5000	11500	5834	17210	11208
6000	10014	5124	15982	9224
7000	8974	4832	15821	8453
8000	8613	4702	14683	8305
9000	8501	4613	14532	8201

average, Frenzy reduces JCT by 40%–50% over MLaaS, 35%–45% over Sia, and over 60% over Opportunistic scheduling, with gains becoming more pronounced at larger scales ($\geq$7000 GPUs).

4.5 Sensitivity Analysis

We first evaluate the sensitivity of HAS to the weight parameters in the scoring function. As shown in Fig. 7(a), we vary each parameter $(\alpha, \beta, \gamma, \delta)$ by scaling it with factors $\{0.5, 1, 2, 4\}$ while keeping the other parameters fixed. The experiment is conducted on a heterogeneous cluster with 8 nodes equipped with four different GPU types, totaling 16 GPUs, using the top 300 tasks from *NewWorkload*. The results show that the average JCT varies by less than 5% across all configurations, indicating that HAS is robust to the choice of weight parameters.

We further evaluate sensitivity to the pricing learning rate η. As shown in Fig. 7(b), we vary η in $\{0.01, 0.05, 0.1, 0.2\}$ under the same setup. Average JCT remains stable across this range, with performance differences within 5%. This indicates that HAS is robust to the exact choice of η.

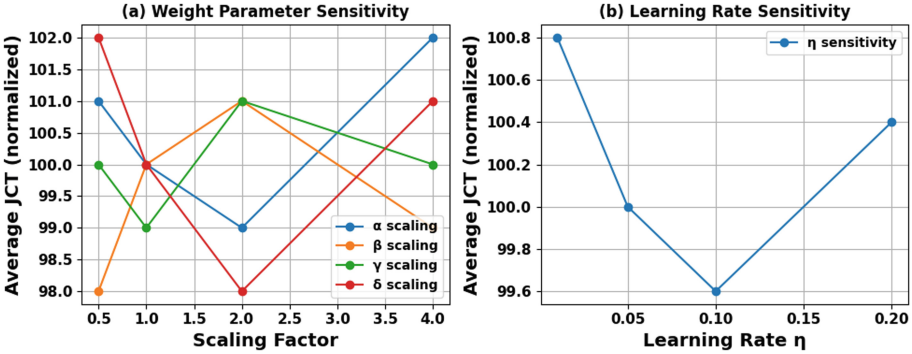

Fig. 7. Sensitivity analysis of HAS parameters. (a) The Sensitivity to weight parameters α, β, γ, and δ. (b) The Sensitivity to pricing learning rate η. The average JCT varies by less than 5%, indicating that HAS is robust to parameter choices and stable across a wide range of learning rates.

4.6 Simulator Validation

We validate the simulator against the physical 8-node, 16-GPU heterogeneous cluster under identical scheduling settings and the same top 300 NewWorkload tasks. As shown in Table 2, the simulator closely matches real-cluster behavior, with 3.4% error in average JCT, 4.3% error in scheduling overhead, 2.2% error in GPU utilization, and 3.3% average error. This confirms that the simulator is sufficiently faithful for large-scale evaluation.

Table 2. Simulator validation compared with the real cluster

Metric	Real Cluster	Simulator	Error
Avg. JCT (s)	1824	1762	3.4%
Scheduling Overhead (s)	2.74	2.62	4.3%
GPU Utilization (%)	81.6	79.8	2.2%
Average Error	–	–	**3.3%**

5 Conclusion

Frenzy integrates serverless computing into heterogeneous GPU clusters to simplify resource management for developers. By predicting peak GPU memory usage during LLM training, Frenzy derives an appropriate resource allocation plan and performs low-overhead scheduling across heterogeneous clusters.

Acknowledgments. This work was supported by the Major Projects of Zhejiang Province (LD24F020012), National Science Foundation of China (U25A20423), Hangzhou Chengxi Sci-Tech Innovation Corridor Special Development Fund (K20241957), Key Research Project of Zhejiang Lab (2025SSYS0005), and the Science and Technology Program of Zhejiang Province (2026SDXT012).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Zheng, L., et al.: Alpa: automating inter- and intra-operator parallelism for distributed deep learning. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022
2. Qiao, A., et al.: Pollux: co-adaptive cluster scheduling for goodput-optimized deep learning. In: 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021
3. Xiao, W., et al.: Gandiva: introspective cluster scheduling for deep learning. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2018
4. Weng, Q., et al.: MLaaS in the wild: workload analysis and scheduling in large-scale heterogeneous GPU clusters. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2022
5. Jeon, M., et al.: Analysis of large-scale multi-tenant GPU Clusters for DNN training workloads. In: USENIX Annual Technical Conference (USENIX ATC), 2019
6. Narayanan, D., et al.: Heterogeneity-aware cluster scheduling policies for deep learning workloads. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020
7. Um, T., et al.: Metis: fast automatic distributed training on heterogeneous GPUs. In: USENIX Annual Technical Conference (USENIX ATC), 2024
8. Jayaram Subramanya, S., et al.: Sia: heterogeneity-aware, goodput-optimized ML-cluster scheduling. In: Proceedings of the 29th Symposium on Operating Systems Principles, 2023
9. Gu, D., et al.: ElasticFlow: an elastic serverless training platform for distributed deep learning. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, vol. 2, 2023
10. Achiam, J., et al.: GPT-4 Technical Report. arXiv preprint arXiv:2303.08774, 2023
11. Barrault, L., et al.: SeamlessM4T: Massively Multilingual and Multimodal Machine Translation. arXiv preprint arXiv:2308.11596, 2023
12. NVIDIA. "NVIDIA A100 Tensor Core GPU." https://www.nvidia.com/en-us/data-center/a100

13. NVIDIA. “NVIDIA GeForce RTX 3090.” https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti
14. Rajbhandari, S., et al.: ZeRO: memory optimizations toward training trillion-parameter models. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020
15. Shoeybi, M., et al.: Megatron-LM: Training Multi-Billion-Parameter Language Models Using Model Parallelism. arXiv preprint arXiv:1909.08053, 2019
16. Micikevicius, P., et al.: Mixed Precision Training. arXiv preprint arXiv:1710.03740, 2017
17. Kingma, D.P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014
18. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT’2010: 19th International Conference on Computational Statistics, Paris, France, August 22–27, 2010: Keynote, Invited and Contributed Papers. Physica-Verlag HD, 2010
19. Korthikanti, V.A., et al.: Reducing activation recomputation in large transformer models. Proc. Mach. Learn. Syst. **5**, 341–353 (2023)
20. Hu, Q., et al.: Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021
21. Radford, A., et al.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
22. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT, vol. 1 (2019)
23. Li, J., Xu, H., Zhu, Y., et al.: Lyra: elastic scheduling for deep learning clusters. In: Proceedings of the Eighteenth European Conference on Computer Systems, pp. 835–850 (2023)
24. Smith, S., Patwary, M., Norick, B., et al.: Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, a Large-Scale Generative Language Model. arXiv preprint arXiv:2201.11990, 2022
25. Alibaba. “Alibaba Cluster Data.” https://github.com/alibaba/clusterdata