Frenzy

Memory-Aware Serverless LLM Training for Heterogeneous GPU Clusters

Zihan Chang
Advisor: Shuibing He
Intelligent Storage and Computing Systems (ISCS) Laboratory
Zhejiang University

August 26，2026

Frenzy

Memory-Aware Serverless LLM Training for Heterogeneous GPU Clusters

Frenzy

Spy Robot

Transmitting Intelligence

Megatron(Transformer)

Outline

Frenzy closes the loop between resource planning and placement in heterogeneous GPU clusters.

Motivation: Heterogeneous Clusters Are Valuable but Hard to Use

Motivation

Heterogeneous GPUs improve utilization but make resource selection for LLM training more error-prone.

Developer Burden

- Manually select GPU types and counts
- Balance memory, compute, topology, and parallelism
- Large-model training often requires trial and error

Runtime Uncertainty

- Static formulas miss allocator behavior
- Communication buffers and kernels add overhead
- Seemingly feasible plans may still OOM

Scheduling Cost

- Resource placement is combinatorial
- ILP solvers are accurate but expensive
- Online systems require fast decisions

Goal: users submit an LLM training job; the system automatically selects a robust GPU plan and placement.

Challenges

Challenge 1

- Robust planning under memory uncertainty

Challenge 2

- Low-overhead scheduling over a large search space

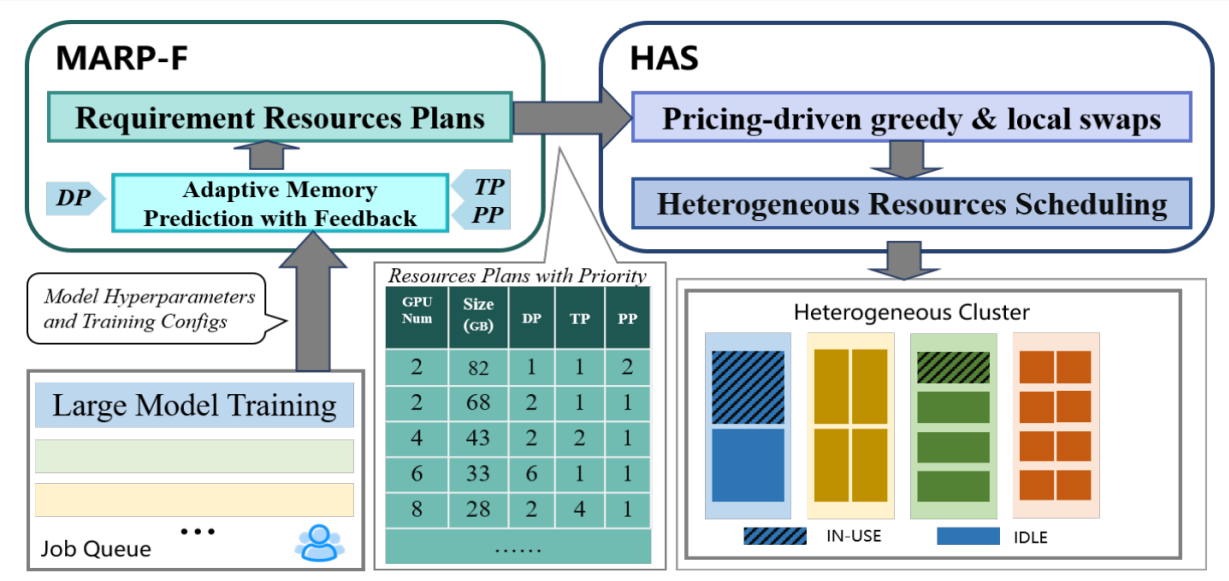
Challenge 3

- End-to-end integration of planning and scheduling

Frenzy Overview: A Closed-Loop System

Design

MARP-F generates robust resource plans; HAS maps the selected plan to specific nodes.



MARP-F (Memory aware resources prediction with feedback)

- Analytical memory prior
- Runtime feedback calibration
- Prioritized resource plans

HAS (Heterogeneous aware scheduling)

- Dynamic-price greedy placement
- Lightweight local swaps
- Topology-aware scheduling

Key idea: a resource plan is not final until runtime feedback confirms its feasibility.

MARP-F Starts from Memory Uncertainty

MARP-F

A one-shot memory formula is useful, but insufficient as the final allocation basis.

Why Do Static Estimates Drift?

- Allocator behavior changes peak reserved memory
- Temporary workspace depends on the kernel
- Communication buffers depend on parallelism
- Fragmentation differs across GPU types

Consequences

- Too optimistic: OOM or probe failure
- Too conservative: more GPUs, lower utilization
- Poor fit: plans are difficult to place

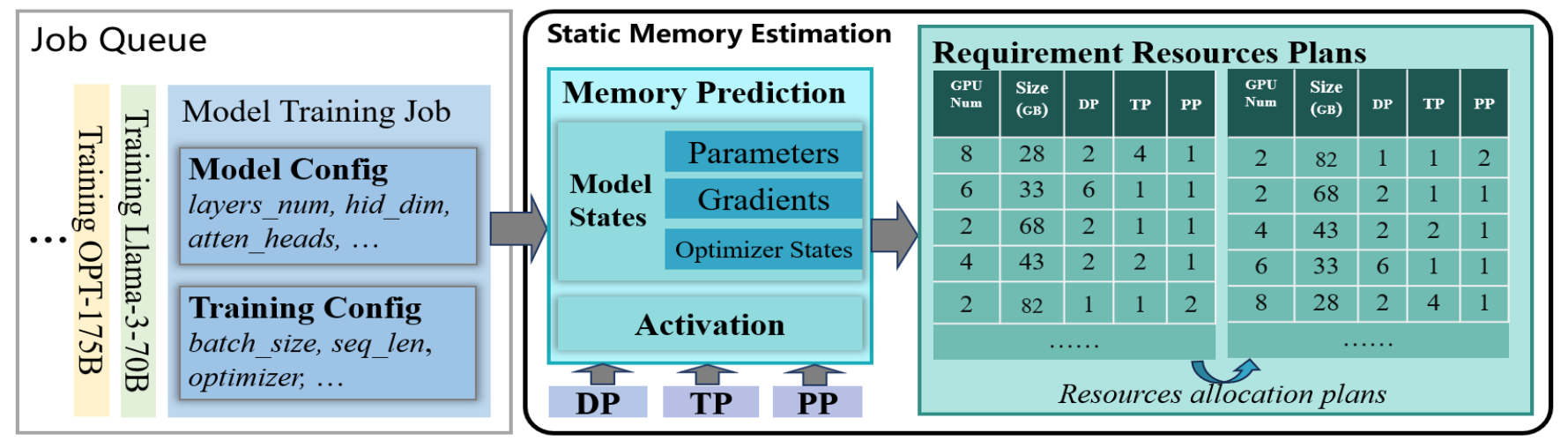
MARP-F Approach

- Use the formula as a prior
- Observe actual memory in a short trial run
- Calibrate and update before final scheduling

MARP-F Memory Prior: Decomposing Training Memory

MARP-F

Static estimation first models the main memory components, then proceeds to feedback calibration.



Per-GPU Peak Memory Prior

$$M_{pred} = M_{static} + M_{act} + M_{logits}$$

Static Terms

- Model parameters
- Gradients
- Optimizer states

Dynamic Terms

- Activations
- Micro-batch
- Sequence length

Logits Term

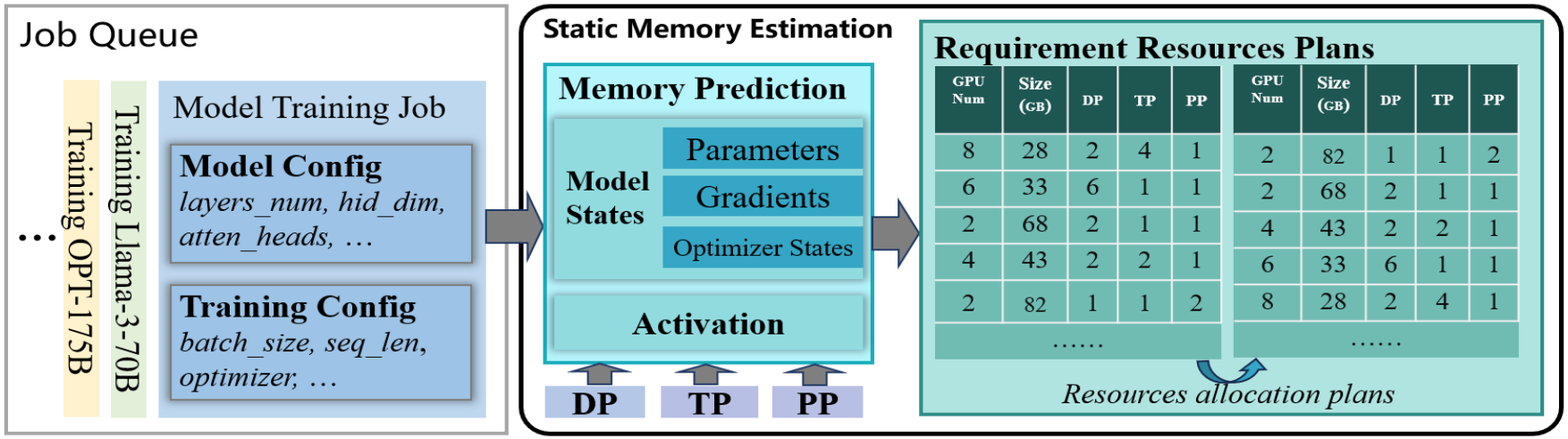
- Vocabulary size
- Output projection memory

Initial feasibility check: Mpred < GPU memory

MARP-F Memory Prior: Decomposing Training Memory

MARP-F

Static estimation first models the main memory components, then proceeds to feedback calibration.



$$W = Vh + l(12h^2 + 13h)$$

$$activations = sbhl(10 + \frac{24}{t} + \frac{5as}{ht})$$

$$\frac{16W}{tp} + sBhl(\frac{10}{d} + \frac{24}{dt} + \frac{5as}{dht}) + \frac{2bsV}{t} < GPUcapacity$$

Module	Component	Shape	Weight	Bias	Paras
Embedding	embedding	$[V, h]$	Vh	–	Vh
Attention	self_atte (W_Q)	$[h, h]$	h^2	h	h^2+h
	self_atte (W_K)	$[h, h]$	h^2	h	h^2+h
	self_atte (W_V)	$[h, h]$	h^2	h	h^2+h
	self_atte (W_O)	$[h, h]$	h^2	h	h^2+h
	mlp (up)	$[h, 4h]$	$4h^2$	$4h$	$4h^2+4h$
	mlp ($down$)	$[4h, h]$	$4h^2$	h	$4h^2+h$
	layer_norm 1	$[h]+[h]$	$2h$	–	$2h$
	layer_norm 2	$[h]+[h]$	$2h$	–	$2h$

Static Estimation: From Configuration to Initial Plan

MARP-F

Before runtime feedback, a fast analytical prior enumerates feasible GPU-type and count configurations.

1. Input
- Model: V, h, l, a
- Training: B, s
- Parallelism: d, t, p

2. Memory Prior
- Model states
- Activation memory
- Logits projection

3. Feasibility
- Check Mpred < CGPU
- Filter OOM-risk plans
- Retain candidates

4. Initial Plan
- GPU type + count
- DP / TP / PP / batch
- Rank by cost/throughput

What the Formula Captures

Mpred = Mstatic + Mact + Mlogits

Each term depends on GPU capacity, parallel strategy, and workload shape

Mstatic: parameters, gradients, and optimizer states; partitioned by TP/PP
Mact: micro-batch × sequence length × layer activations
Mlogits: output projection memory, affected by vocabulary size

Selecting a Static Plan

Enumerate candidate (d, t, p, b) configurations for each GPU type.
Discard candidates whose predicted memory exceeds capacity.
Select plans using fewer GPUs or offering higher expected throughput.

Limitations of Static Estimation

It cannot fully capture allocator behavior, NCCL buffers, temporary workspace, or fragmentation.

Optimistic plans may OOM; conservative plans waste GPUs.

Role in Frenzy

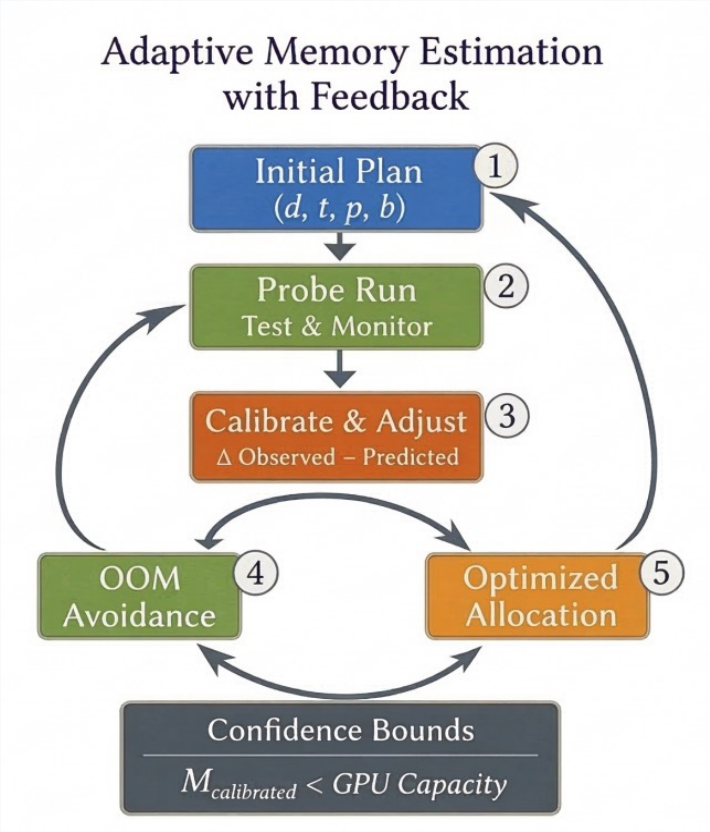
Static estimation is a fast prior, not the final decision.

MARP-F calibrates Mobs − Mpred and refines the plan through Plan–Probe–Update.

MARP-F Feedback Loop: Plan → Probe → Update

MARP-F

Runtime feedback turns analytical planning into robust system behavior.



1. Initial Plan

- Use the static estimator to select (d, t, p, b)

2. Short Trial Run

- Run a few iterations; monitor peak allocated/reserved memory

3. Calibration

- Compute Δ = Mobs − Mpred to obtain Mcal

4. Stop When Stable

- Accept a plan with Mcal < CGPU and a safety margin

Dynamic Adjustment: Turn Memory Drift into Feedback

MARP-F

Frenzy does not treat OOM as a terminal failure, nor does it reserve excessive memory headroom.

If Memory Is Tight

- Reduce micro-batch size
- Increase checkpointing
- Reconfigure parallelism

If the Plan Is Too Conservative

- Increase micro-batch size
- Reduce GPU count
- Improve utilization

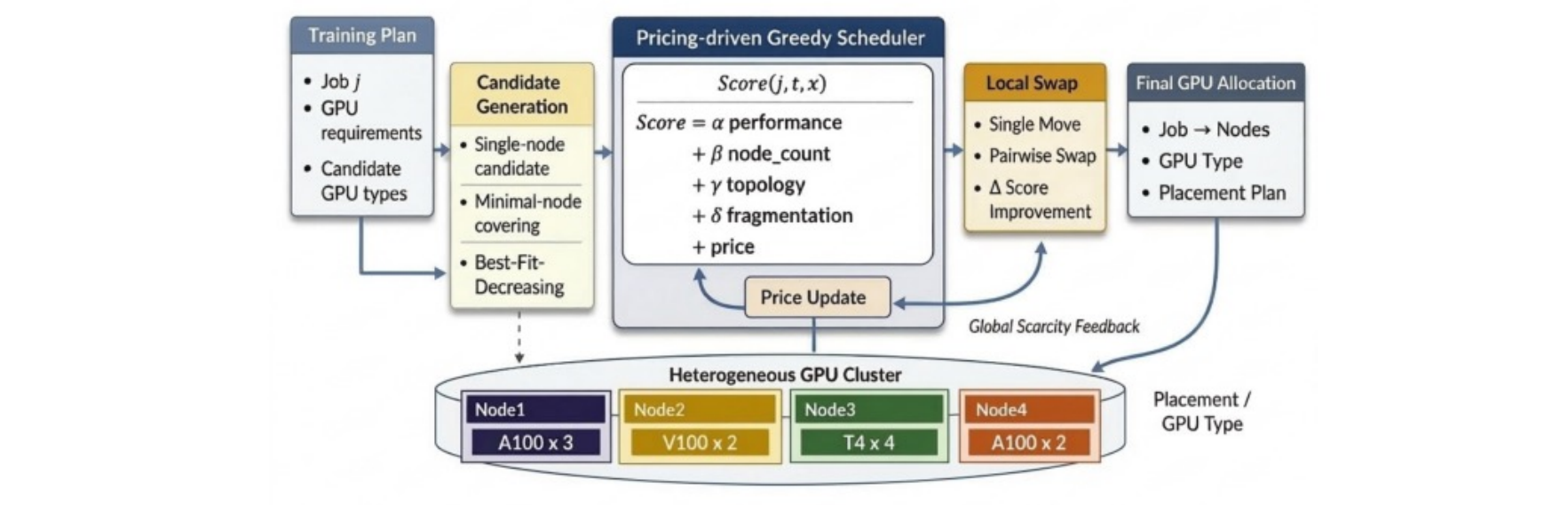
If Memory Continues to Drift

- Re-enter Plan–Probe–Update
- Treat failure as new feedback

HAS: Turning Heterogeneous Scheduling into Fast Placement

HAS

Given a feasible plan, HAS maps jobs to nodes with low online overhead.



Input: training plan (parallelism) + GPU demand + candidate GPU types

Core: dynamic-price-driven greedy placement

Output: job-to-node placement and GPU-type assignment

HAS Scoring: Simple Terms Capture Placement Quality

HAS

The score balances speed, locality, topology, fragmentation, and resource scarcity.

$$\text{Score}(j,t,x) = \text{Performance} + \text{Nodes} + \text{Topology} + \text{Fragmentation} + \text{Price}$$

Performance
Place jobs with larger gains on faster GPUs

Compactness
Use fewer nodes to reduce synchronization

Topology
Penalize cross-node or cross-rack communication

Fragmentation
Avoid leaving unusable GPU fragments

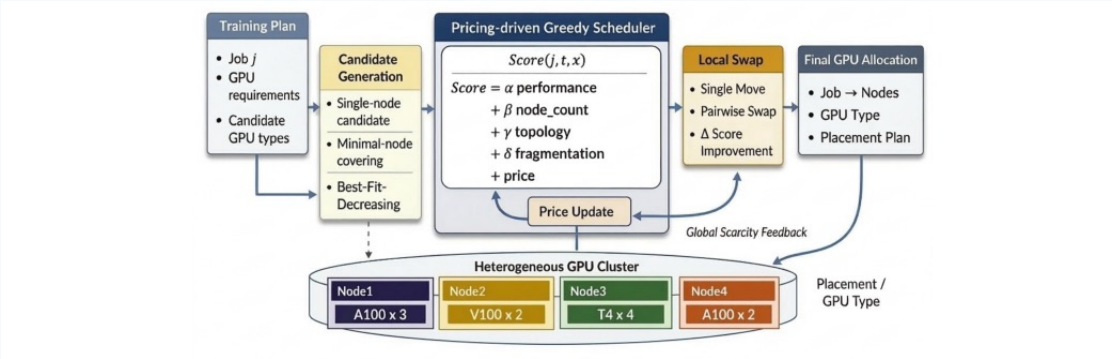
Price
Raise the cost of scarce node–type resources dynamically

Intuition: greedily select the lowest-score candidate while dynamic prices prevent overuse of scarce resources.

HAS Execution: Greedy Placement, Then Local Swaps

HAS

This approach lightly optimizes the final allocation while retaining near-linear scheduling overhead.



Candidate Generation

- Prefer single-node placement
- Otherwise choose the minimum-node cover

Dynamic-Price Greedy

- Schedule difficult jobs first
- Update resource prices after each round

Local Swaps

- Single-job moves and pairwise swaps
- Accept only score-reducing changes

TP placement guarantee: each tensor-parallel group stays within one physical node / high-bandwidth domain.

Evaluation Setup

Evaluation

Frenzy is evaluated on both a real heterogeneous cluster and a calibrated simulator.

Real Cluster

- 8 nodes, 16 GPUs, 4 GPU types
- A800 80G, H800 80G, A100 40G, A100 80G
- Ray-based LLM training scheduler

Workloads & Baselines

- NewWorkload: GPT-2, OPT, and BERT models
- Trace-driven: Helios and Philly
- Baselines: RAR, Sia, Opportunistic, ElasticFlow, MLaaS

Large-Scale Simulator

- Three GPU types: T4, V100, and A100
- 128 nodes × 8 GPUs per type
- 10,000 jobs and 1,000–9,000 GPUs
- Calibrated against real-cluster results

Main Results: Lower JCT, Better Memory Accuracy, Lower Overhead

Results

The gains come from feedback-calibrated planning and lightweight heterogeneous placement.

12–50%
Lower Average JCT
vs. SOTA baselines

92–98%
Memory Prediction Accuracy
MARP-F better models the actual footprint

10–20×
Lower Scheduling Overhead
HAS avoids ILP-level scheduling cost

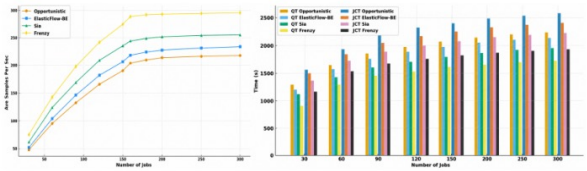


Fig. 4: Comparison of throughput scalability (left) and latency performance (right) under increasing LLM workloads.

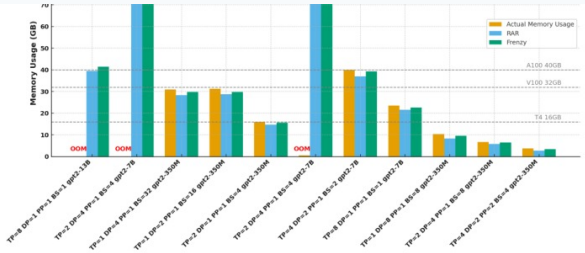


Fig. 5: Comparison between Frenzy and RAR memory predictions and the actual GPU memory usage.

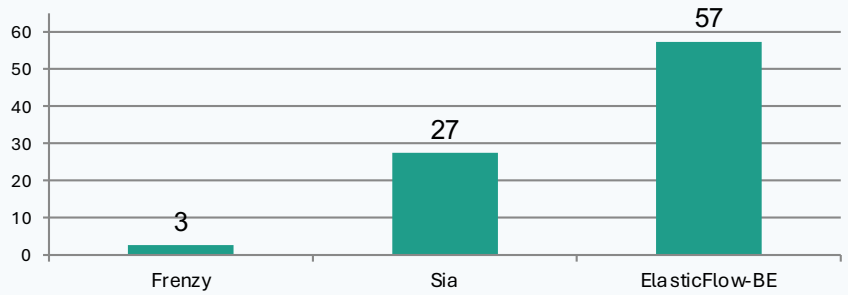
Fig. 4 / Fig. 5: Frenzy saturates later as load grows; MARP-F tracks actual memory more closely than RAR.

Scalability and Robustness

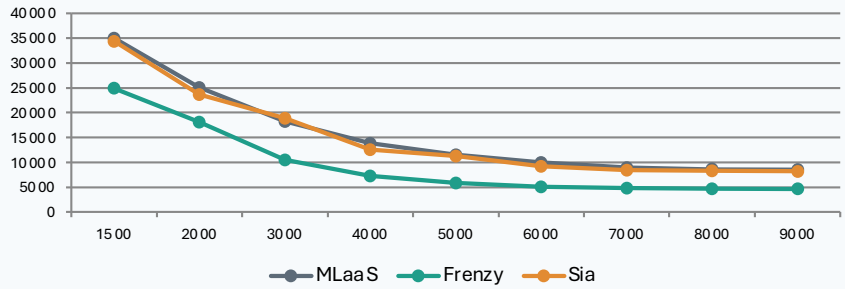
Results

Frenzy maintains low overhead at scale, while the simulator effectively supports large-cluster evaluation.

HAS overhead stays small



Large-scale JCT improves



Scheduler Scalability

- At 10K jobs: 2.69s vs. 27.45s / 57.32s

Parameter Robustness

- Average JCT varies by <5% across weights and learning rates

Simulator Validation

- 3.3% average error vs. the real cluster

Summary: Frenzy connects memory-aware planning with low-overhead heterogeneous scheduling for robust LLM training.

Contact Me

Thank You!

Email：changzihan@zju.edu.cn

Code is available at: https://github.com/ISCS-ZJU/Frenzy