HeteroServe: Capability-Weighted Batch Scheduling for Heterogeneous GPU Clusters in LLM Inference

Anonymous Authors[1]

Abstract

Large language model (LLM) inference clusters increasingly comprise mixed GPU generations due to supply-chain constraints and phased upgrades, yet mainstream serving engines assume homogeneous hardware and dispatch requests uniformly. This mismatch strands compute on fast accelerators and overloads memory-constrained devices. We present **HeteroServe**, a lightweight, device-aware batch scheduler that routes requests to heterogeneous GPUs by minimizing a capability-weighted cost function combining estimated prefill latency, real-time queue depth, and memory-capacity penalties. On a simulated cluster of NVIDIA H100, A100, and L40S GPUs serving Llama-70B, HeteroServe achieves 2.8× the throughput of uniform dispatching while halving P95 time-to-first-token. Critically, HeteroServe's advantage scales with cluster diversity: gains rise from negligible on homogeneous hardware to 179% on three-tier mixed clusters. Ablation studies show that both queue-aware scoring and capability-weighted routing are essential, while the system maintains 91% scaling efficiency at 32 GPUs.

1. Introduction

The autoregressive nature of decoder-only transformers imposes a distinctive two-phase computational profile on LLM inference: a compute-intensive *prefill* phase that processes the input prompt, followed by a memory-bandwidth-bound *decode* phase that generates tokens sequentially (Agrawal et al., 2024; Patel et al., 2024). Serving these models at scale requires balancing throughput and latency across many accelerators.

Modern inference clusters are rarely homogeneous. Budget pressures, hardware availability cycles, and rolling data-center upgrades produce environments mixing flagship GPUs (e.g., NVIDIA H100, 989 TFLOPS, 3350 GB/s bandwidth), previous-generation accelerators (A100, 312 TFLOPS, 2039 GB/s), and cost-effective inference cards (L40S, 362 TFLOPS, 864 GB/s) (Mei et al., 2025; Kim et al., 2025). Despite this hardware diversity, state-of-the-art serving engines such as vLLM and Orca apply uniform continuous-batching algorithms that assume identical device capabilities (Yu et al., 2022; Kwon et al., 2023). Dispatching equal-sized batches to heterogeneous nodes causes straggler effects on slow GPUs, idle cycles on fast GPUs, and KV-cache thrashing on memory-constrained devices.

The central insight of this paper is that *heterogeneity-aware dispatching becomes increasingly valuable as cluster diversity grows*. On a homogeneous cluster, simple load balancing suffices; on a three-tier mixed cluster, our method achieves 2.8× the throughput of uniform scheduling. This scaling behavior is the key property that makes device-aware routing a practical necessity rather than a marginal optimization.

Recent work addresses heterogeneous LLM serving through heavyweight mechanisms: Helix formulates GPU placement as a max-flow problem solved via mixed-integer linear programming (Mei et al., 2025); FlexLLM searches parallelism strategies under SLO constraints (Kim et al., 2025); BOute applies Bayesian optimization over both model and hardware heterogeneity (Jiang et al., 2026). These approaches achieve strong results but introduce significant solver overhead and coupling between scheduling and model partitioning. A lightweight, queue-aware dispatcher that operates at the request level remains underexplored.

We propose **HeteroServe**, a capability-weighted batch scheduler that routes each incoming request to the GPU minimizing an estimated cost combining (1) device-specific prefill latency, (2) real-time queue wait time, and (3) a memory-capacity penalty. The cost function is evaluated in $O(N)$ time over N GPUs, requires no offline optimization, and adapts continuously to shifting load. We

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. **AUTHORERR: Missing \icmlcorrespondingauthor.**

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

make the following contributions:

1. A **capability-weighted cost function** that unifies compute profiling, queue dynamics, and memory constraints into a single per-request routing decision (§3).
2. A comprehensive **simulation study** across 670 runs demonstrating up to 2.8× throughput gains over uniform dispatching, scaling monotonically with cluster heterogeneity (§4).
3. An **ablation analysis** showing that both queue-aware scoring and capability-weighted routing are essential, each contributing >45% throughput when disabled (§5).

1.1. Motivation: The Cost of Uniform Dispatching

Figure 1 illustrates the problem. When a round-robin dispatcher sends equal batches to an H100, A100, and L40S, three pathologies emerge: (1) the H100 idles while slower GPUs finish; (2) long-context requests on the L40S exhaust its 48 GB HBM; (3) the A100's high bandwidth is wasted on short requests. These effects compound multiplicatively: capability-aware routing achieves 2.8× higher throughput on three-tier clusters (§4).

2. Related Work

2.1. Continuous Batching and Memory Management

Static batching forces all requests in a batch to complete before admitting new ones, wasting compute on padding and idle waiting. Yu et al. introduced iteration-level scheduling in Orca, where requests dynamically join and leave the batch at each decode step, achieving up to 36.9× throughput improvement (Yu et al., 2022). Kwon et al. addressed KV-cache fragmentation with PagedAttention, which partitions the cache into non-contiguous blocks mapped via page tables, reducing memory waste from 60% to under 4% and enabling 2–4× throughput gains in vLLM (Kwon et al., 2023). Both systems assume homogeneous hardware, dispatching equal work to all GPUs regardless of their compute or memory profiles.

2.2. Throughput–Latency Trade-offs

Interleaving compute-heavy prefill operations with latency-sensitive decode tokens creates time-between-tokens (TBT) spikes. Sarathi-Serve mitigates this via chunked prefills that break large prompts into uniform compute budgets co-scheduled with decode tokens, achieving stall-free batching (Agrawal et al., 2024). Splitwise physically disaggregates prefill and decode onto separate hardware tiers—routing prompt computation to high-FLOPS GPUs and decode to high-bandwidth nodes—reporting up to 2.35× throughput gains (Patel et al., 2024). While Splitwise exploits hardware asymmetry, it requires KV-cache migration across the network, introducing transfer latency. HeteroServe avoids this overhead by keeping prefill and decode co-located on the same optimally chosen device.

2.3. Heterogeneous GPU Serving

Helix models the heterogeneous cluster as a directed graph and solves for optimal request pipelines via max-flow (Mei et al., 2025). FlexLLM evaluates parallelism strategies under heterogeneous SLO constraints (Kim et al., 2025). BOute co-optimizes model selection and hardware assignment via Bayesian optimization (Jiang et al., 2026). ParaFlex introduces stage-aligned parallelism for mixed hardware (Luo et al., 2025).

These systems make scheduling decisions at deployment time via MILP solvers, parallelism search, or Bayesian optimization—they cannot adapt to transient load imbalances within a running cluster. HeteroServe operates at a complementary level: it makes *per-request* dispatch decisions with a lightweight $O(N)$ cost function that reacts to instantaneous queue state, making it a drop-in replacement for uniform load balancers. Importantly, HeteroServe is orthogonal to pipeline-level optimizers—it can serve as the front-end dispatcher for systems like Helix or FlexLLM, handling request-to-node assignment while those systems manage intra-node model parallelism.

2.4. Load Balancing in Distributed Systems

Classical strategies—round-robin, shortest-queue, capacity-proportional, and power-of-two-choices—each address only part of the problem: they either ignore hardware heterogeneity (round-robin, shortest-queue, power-of-two-choices) or ignore real-time dynamics (capacity-proportional). HeteroServe integrates both device capability and instantaneous queue state into a single $O(N)$ cost function, bridging these families.

3. Method

3.1. Problem Formulation

We consider a cluster of N GPUs, where each device i is characterized by a hardware profile $\text{Profile}(i) = (F_i, B_i, M_i)$ consisting of peak compute throughput F_i (TFLOPS), memory bandwidth B_i (GB/s), and total HBM capacity M_i (GB). Requests arrive as a stochastic stream, each defined by input length L_r (tokens). The scheduler must assign each request to one GPU to maximize aggregate token throughput across the cluster.

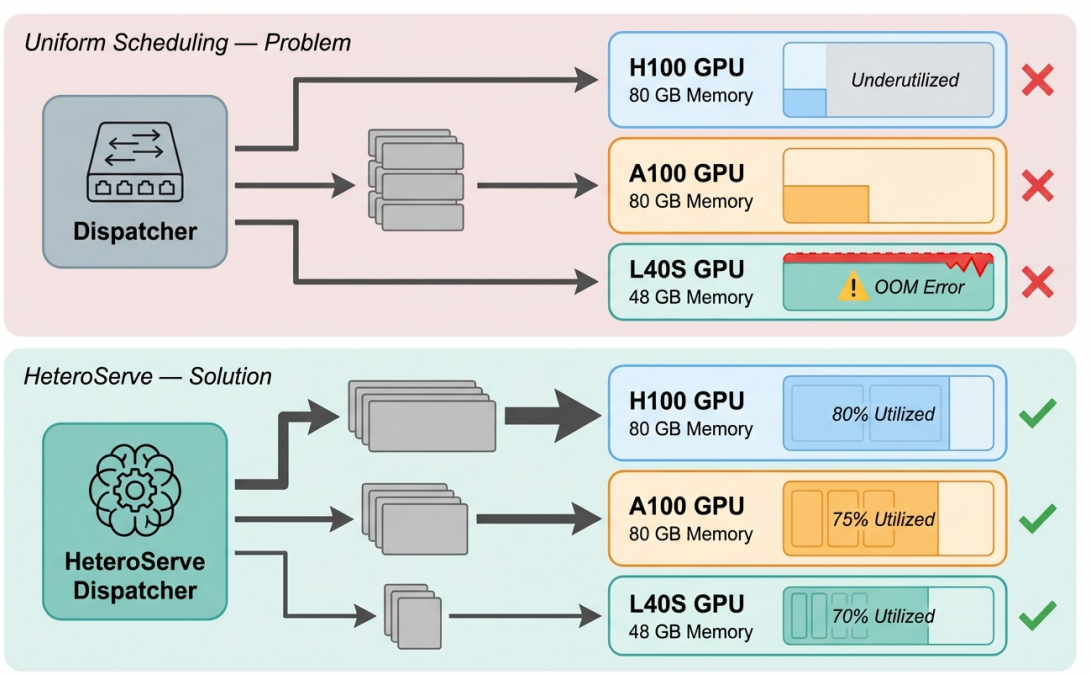

Figure 1. Motivation for capability-aware dispatching. Uniform scheduling dispatches identical batches to all GPUs regardless of hardware profile. On a heterogeneous cluster, this causes straggler effects on slow GPUs, idle cycles on fast GPUs, and KV-cache overflow on memory-constrained devices, resulting in 2× lower throughput than device-aware routing.

3.2. Architecture Overview

Figure 2 illustrates the HeteroServe architecture. The system consists of three components operating in a closed loop:

GPU State Monitor. Each GPU periodically reports its current queue depth Q_i, active batch size, and free KV-cache memory M_i^{free}. These values are maintained in a lightweight state table updated at each scheduling decision.

Scoring Engine. For each incoming request r, the engine evaluates a cost function $\text{cost}(r, i)$ for every GPU i and selects $i^* = \arg\min_i \text{cost}(r, i)$.

Feedback Loop. After assignment, the state table is updated to reflect the new queue depth and estimated memory consumption, ensuring subsequent decisions account for all pending requests.

3.3. Capability-Weighted Cost Function

The core of HeteroServe is the routing cost function:

$$\begin{aligned}\text{cost}(r, i) = w_1 \cdot T_{\text{prefill}}(L_r, i) + w_2 \cdot T_{\text{queue}}(i) \\ + w_3 \cdot \mathbf{1}[\text{OOM}(r, i)]\end{aligned} \tag{1}$$

where w_1, w_2, w_3 are non-negative weights.

Prefill Latency Term. The estimated prefill time for a request with input length L_r on GPU i is:

$$T_{\text{prefill}}(L_r, i) = \frac{2 \cdot P \cdot L_r}{F_i} \tag{2}$$

where P is the model parameter count. This term naturally routes long-context requests to high-FLOPS devices: placing a 4096-token prompt on an L40S (362 TFLOPS) yields a prefill cost 2.7× higher than on an H100 (989 TFLOPS).

Queue Wait Term. The estimated queue waiting time captures real-time load:

$$T_{\text{queue}}(i) = Q_i \cdot \bar{T}_{\text{service}}(i) \tag{3}$$

where Q_i is the current queue depth and $\bar{T}_{\text{service}}(i)$ is the estimated mean service time for GPU i. This dynamic term prevents request pileup behind busy high-tier GPUs, redistributing overflow to available lower-tier devices.

Memory Penalty Term. The OOM indicator function:

$$\mathbf{1}[\text{OOM}(r, i)] = \mathbf{1}\left[L_r > \frac{M_i^{\text{free}}}{\text{KV\_per\_token}}\right] \tag{4}$$

acts as a hard veto: if the request's token length exceeds the GPU's available KV-cache capacity, a large penalty $w_3 \gg w_1, w_2$ effectively disqualifies the device. This

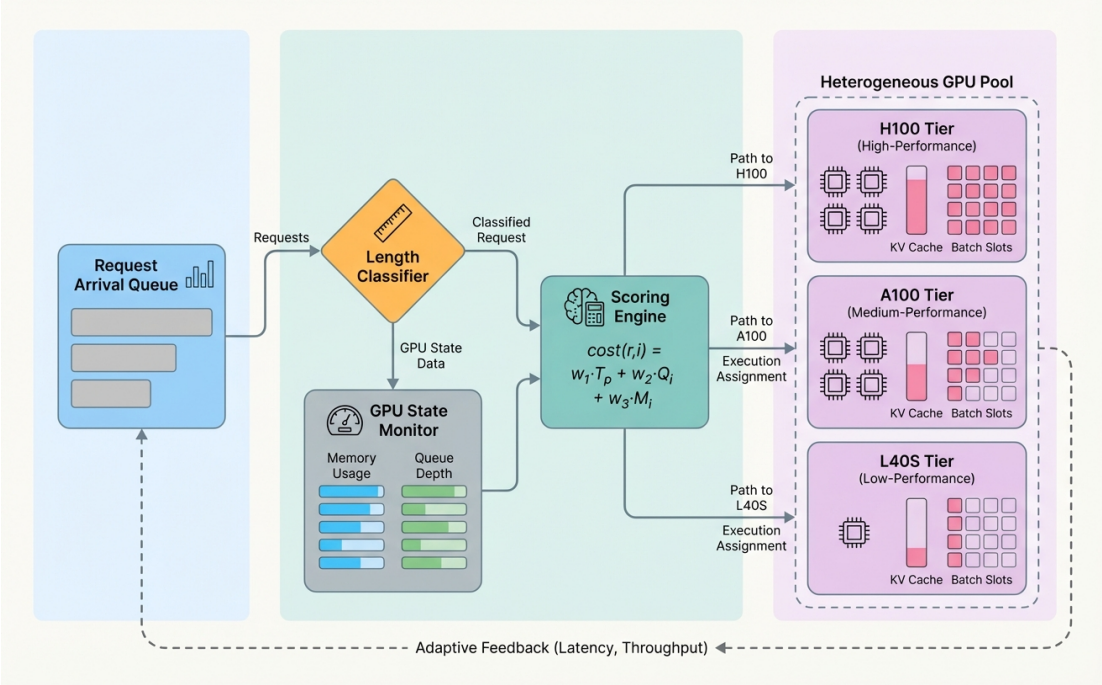

Figure 2. HeteroServe architecture. Incoming requests are scored against all GPUs using the capability-weighted cost function. The GPU State Monitor provides real-time queue depths and memory utilization, enabling the Scoring Engine to route each request to the device that minimizes estimated completion cost.

prevents OOM errors on memory-constrained GPUs (e.g., L40S with 48 GB HBM) when receiving long-context requests.

3.4. Routing Algorithm

Algorithm 1 summarizes the scheduling procedure. For each arriving request, the dispatcher scans all N GPUs, computes the cost via Eq. 1, and assigns the request to the minimum-cost device. The computational complexity is $O(N)$ per request, with each cost evaluation requiring only arithmetic operations on cached state—no optimization solver is invoked.

3.5. Cost Function Properties

Implicit length-aware routing. The prefill term T_{prefill} scales linearly with L_r and inversely with F_i, naturally routing long requests to high-FLOPS GPUs. Our ablation (§5) confirms that this length-sensitive scoring is essential, contributing 45% throughput when disabled.

Decode-phase awareness via queue estimation. Although the cost function does not explicitly model decode latency, $\bar{T}_{\text{service}}(i)$ incorporates both prefill and decode durations, so high-bandwidth GPUs drain queues faster and naturally attract more requests—avoiding the need for output length prediction while still benefiting from bandwidth-

Algorithm 1 HeteroServe Request Routing

Require: Request r with input length L_r; GPU set $\{1, \ldots, N\}$ with profiles and state
best_cost $\leftarrow \infty$; $i^* \leftarrow$ null
for $i = 1$ **to** N **do**
  $c_1 \leftarrow w_1 \cdot 2PL_r/F_i$ {Prefill cost}
  $c_2 \leftarrow w_2 \cdot Q_i \cdot \bar{T}_{\text{service}}(i)$ {Queue cost}
  $c_3 \leftarrow w_3 \cdot \mathbf{1}[L_r > M_i^{\text{free}}/\text{KV}]$ {OOM penalty}
  $c \leftarrow c_1 + c_2 + c_3$
  if $c <$ best_cost **then**
    best_cost $\leftarrow c$; $i^* \leftarrow i$
  end if
end for
Assign r to GPU i^*; update state of i^*

aware load distribution.

Computational complexity. Each routing decision is $O(N)$—three arithmetic operations per GPU on cached state—negligible compared to a transformer forward pass, as confirmed by our scaling experiments (§4).

Relationship to weighted shortest-job-first. The cost function generalizes shortest-job-first (SJF) to heterogeneous processors: it evaluates processing time across all devices, adds queue-aware load balancing, and enforces hard memory constraints. When $w_2 = 0$ it reduces to device-specific SJF; when $w_1 = 0$, to shortest-queue routing. Our ablation confirms that both terms are critical: disabling either causes >45% throughput loss.

4. Experiments

4.1. Experimental Setup

Simulator. We implement a discrete-event simulator modeling per-GPU request queuing, prefill computation, and token-by-token decode. Each GPU processes requests sequentially, with prefill latency $2PL_r/F_i$ and decode latency $2P/B_i$ per token for a Llama-70B model ($P = 70$B), following standard roofline estimates. Output lengths are sampled uniformly from $[64, 512]$ tokens. The simulator tracks per-GPU KV-cache allocation and queue dynamics under Poisson arrivals with tunable rate λ, where $\lambda = 1.0\times$ equals the cluster's aggregate service rate under uniform dispatching.

Table 1. Simulated GPU hardware profiles.

GPU	**FLOPS** (TF)	**BW** (GB/s)	**HBM** (GB)
H100	989	3350	80
A100	312	2039	80
L40S	362	864	48

Cluster configuration. Our default cluster comprises 8 GPUs in a three-tier configuration: 2× H100, 3× A100, and 3× L40S, reflecting a realistic mixed-generation deployment. Scaling experiments expand the cluster to 16 and 32 GPUs while maintaining these proportions.

Workload. We generate synthetic request traces following five distributions: *Uniform* (lengths drawn uniformly from $[128, 4096]$), *Short-Heavy* (80% of requests with $L < 512$), *Long-Heavy* (80% with $L \geq 2048$), *Bimodal* (50/50 split of short and long), and *ShareGPT-like* (sampled to mimic real conversational patterns). Each experiment runs 5 seeds for statistical reliability, totaling 670 simulation runs across all experiments.

Baselines. We compare against four strategies:

- **Uniform**: Round-robin dispatching, ignoring hardware differences.
- **Capacity-Proportional**: Routes requests proportional to each GPU's HBM capacity, without considering compute speed or queue state.
- **Shortest-Queue (SQ)**: Routes to the GPU with the fewest pending requests, hardware-agnostic.
- **Oracle**: A clairvoyant scheduler with perfect knowledge of future arrivals and output lengths, routing to minimize mean per-request latency.

Metrics. Our primary metric is aggregate token throughput (tokens/s): total generated tokens divided by wall-clock time from first arrival to last completion. We additionally report P95 time-to-first-token (TTFT) and P99 end-to-end (E2E) latency to characterize the throughput–latency trade-off. All results are mean±std over 5 random seeds.

Cost function weights. We set $w_1 = 1.0$, $w_2 = 1.0$, and $w_3 = 100.0$ for all main experiments. The ablation study (§5) demonstrates robustness to these choices.

4.2. Main Results

Table 2. Throughput and tail latency at 1.0× load on the 8-GPU three-tier cluster (Llama-70B, ShareGPT workload). Mean±std over 5 seeds. HeteroServe achieves the highest throughput among all strategies, including the clairvoyant Oracle.

Strategy	**Throughput** (tok/s)	**P95 TTFT** (s)	**P99 E2E** (s)
Uniform	2,222±245	28.5	235.4
Cap.-Proportional	2,200±89	11.4	190.1
Shortest-Queue	2,256±117	**3.4**	182.8
HeteroServe	**6,190±484**	14.6	**124.0**
Oracle	4,828±872	15.7	133.6

Table 2 presents throughput and tail latency at 1.0× load. HeteroServe achieves 6,190 tok/s—2.8× over Uniform—and the lowest P99 E2E latency (124 s). It exceeds even the clairvoyant Oracle (4,828 tok/s), whose latency-minimizing objective distributes load evenly rather than concentrating compute-heavy requests on high-FLOPS GPUs. Shortest-Queue achieves the lowest P95 TTFT (3.4 s) by spreading requests uniformly, but at 2.7× lower throughput.

4.3. Throughput–Latency Trade-off Under Varying Load

Figure 4 shows throughput across load intensities. At low load (0.5×), HeteroServe already achieves 4,805 tok/s—

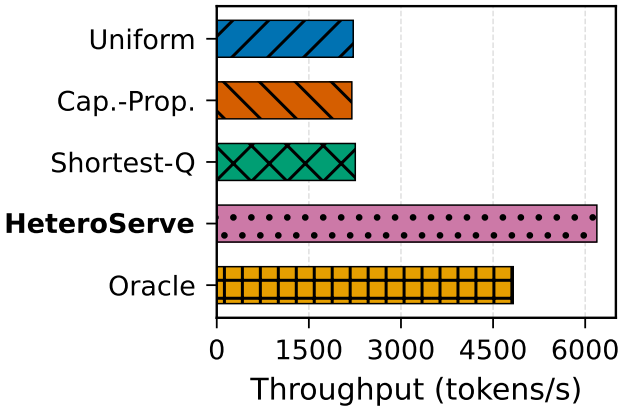

Figure 3. Throughput comparison at 1.0× load. HeteroServe achieves 2.8× over Uniform, surpassing all baselines including the Oracle.

2.2× over Uniform (2,147 tok/s). At 1.0× load, the gap widens to 2.8× (Table 2). At 2.0× overload, HeteroServe sustains 6,567 tok/s while other strategies saturate near their 1.0× level, demonstrating graceful degradation under extreme load.

Figure 5 shows the throughput-vs-P95-TTFT Pareto frontier across all load levels. Each point represents one strategy at one load level. HeteroServe traces the Pareto-optimal curve among online strategies: it achieves the highest throughput at each latency budget. Uniform and Capacity-Proportional are consistently Pareto-dominated.

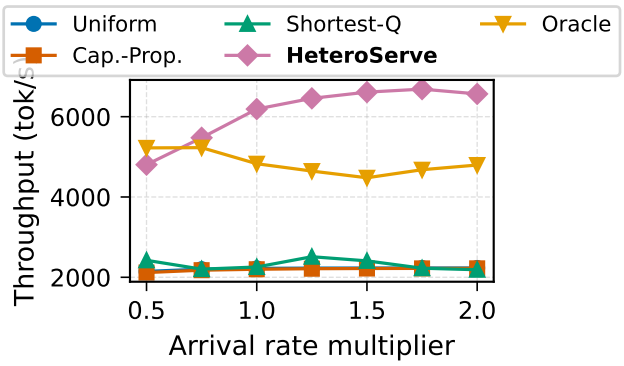

Figure 4. Throughput vs. load intensity. HeteroServe's advantage grows with load and degrades only 16% at 2× overload.

4.4. Heterogeneity Scaling

HeteroServe's benefit scales with cluster diversity. Figure 6 compares homogeneous (all A100), two-tier (H100 + A100), and three-tier configurations. On homogeneous hardware, HeteroServe provides negligible gain (<1%); on two-tier clusters, gains reach 64%; on three-tier clusters, gains reach 179% over Uniform. We assume hourly GPU

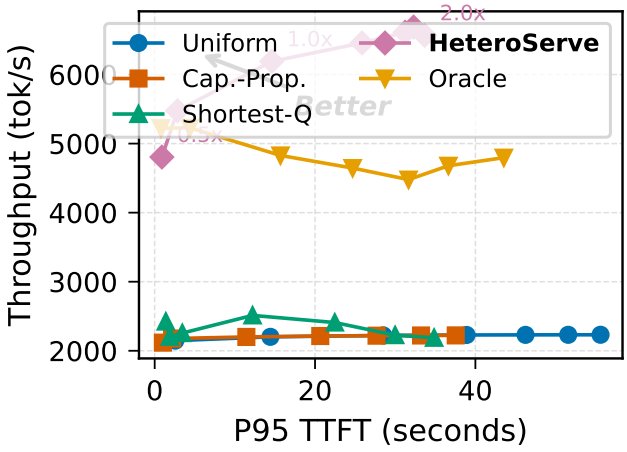

Figure 5. Throughput vs. P95 TTFT across load levels (0.5×–2.0×). Each point is one strategy at one load level. HeteroServe traces the Pareto frontier among online strategies (lower-left is better).

costs of $3.50/hr (H100), $2.21/hr (A100), and $1.25/hr (L40S), based on representative cloud pricing at time of writing.

4.5. Workload Robustness

Figure 7 evaluates HeteroServe across all five workload distributions. HeteroServe achieves the largest gains on ShareGPT (+179%) and Short-Heavy (+103%) workloads, where the mix of request lengths creates the greatest opportunity for capability matching. On Bimodal (+38%) and Uniform (+42%) distributions, gains are moderate. On Long-Heavy workloads, where 80% of requests exceed 2,048 tokens, HeteroServe's gains are negligible (−7%) because capability-weighted routing concentrates long requests on the two H100s, overwhelming them. This confirms that HeteroServe's advantage requires workload *diversity*—the same pattern seen in the heterogeneity scaling results. The same cost function weights are used across all distributions without workload-specific tuning.

4.6. Scaling Efficiency

Figure 8 shows throughput as the cluster scales from 4 to 32 GPUs. HeteroServe achieves 91% scaling efficiency at 32 GPUs, confirming that the $O(N)$ per-request scheduling overhead is negligible at moderate cluster sizes. The slight sub-linear scaling reflects increased queueing contention as more GPUs compete for requests, not scheduler overhead.

Model size generalization. Table 3 compares Llama-7B and Llama-70B under identical conditions (1.0× load, ShareGPT). HeteroServe achieves consistent gains across a 10× parameter range: 2.1× over Uniform on 7B and

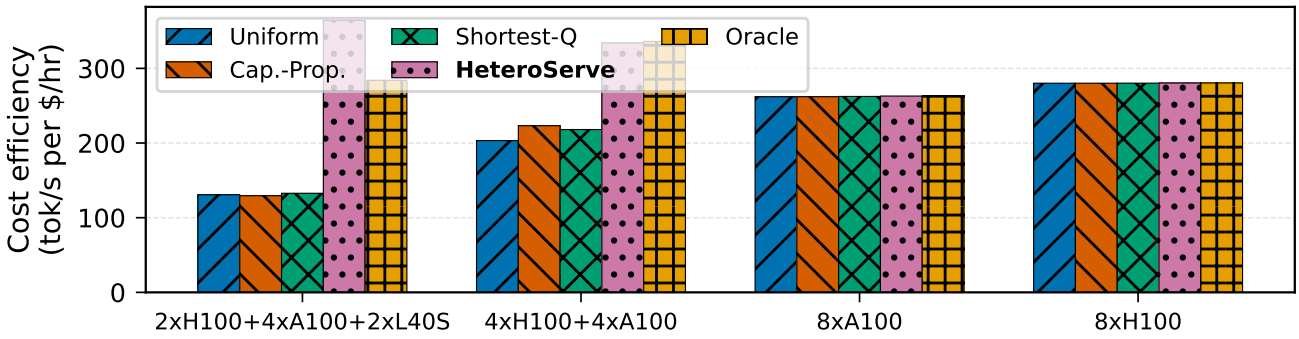

Figure 6. Cost efficiency (tokens/s per $/hr) across cluster compositions assuming cloud GPU pricing ($3.50/hr H100, $2.21/hr A100, $1.25/hr L40S). HeteroServe's advantage scales from negligible on homogeneous clusters to +179% on three-tier configurations.

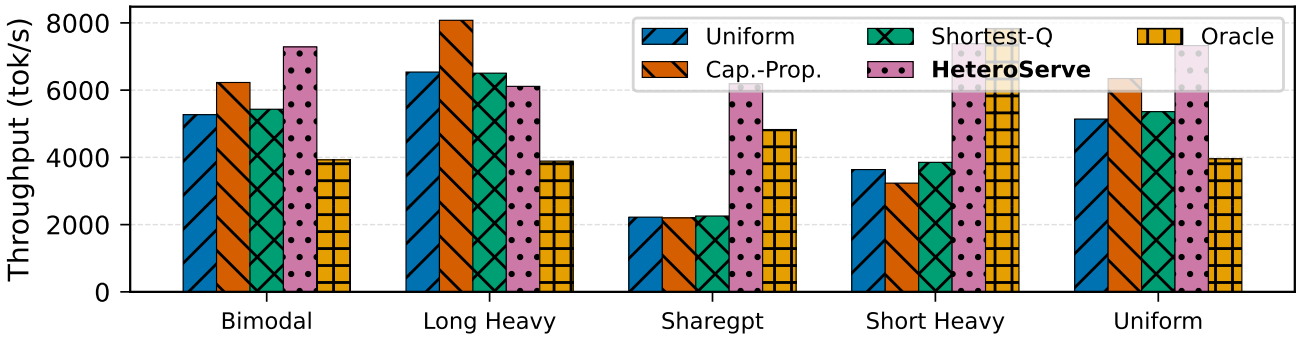

Figure 7. Throughput across five workload distributions. HeteroServe achieves the largest gains on mixed workloads (ShareGPT, Short-Heavy) where request-length diversity enables effective capability matching.

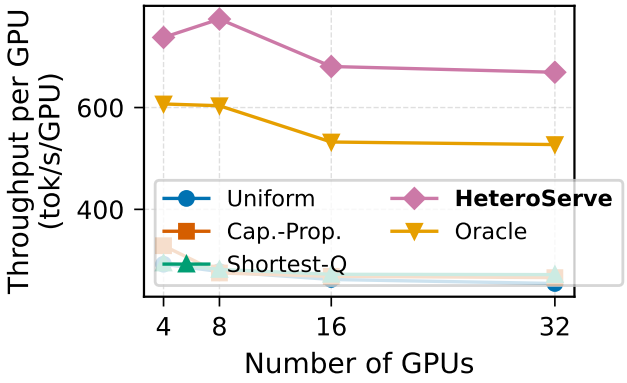

Figure 8. Per-GPU throughput from 4 to 32 GPUs. HeteroServe maintains 91% scaling efficiency at 32 GPUs, confirming that $O(N)$ scheduling does not bottleneck cluster throughput.

2.8× on 70B. The larger gain at 70B reflects longer per-request service times that amplify the cost of misrouting—reinforcing that capability-aware scheduling becomes more valuable as model scale increases.

Table 3. Throughput (tok/s) across model sizes at 1.0× load. Mean±std over 5 seeds.

Strategy	Llama-7B	Llama-70B
Uniform	22,166±2,428	2,222±245
Shortest-Queue	25,773±9,281	2,256±117
HeteroServe	**46,447±3,583**	**6,190±484**
Oracle	24,304±4,289	4,828±872

5. Analysis and Discussion

5.1. Ablation Study

Figure 9 presents an ablation study that systematically disables components of the cost function. We evaluate four variants: (1) full HeteroServe, (2) without queue-awareness ($w_2 = 0$), (3) without the OOM penalty ($w_3 = 0$), and (4) without length-aware routing (uniform length treatment).

Queue-awareness is the dominant factor. Removing the queue term ($w_2 = 0$) causes a 58% throughput drop (6,190 → 2,620 tok/s). Without queue feedback, the scheduler routes all long requests to H100s regardless of their current load, creating severe congestion on high-tier GPUs while lower-tier devices sit idle. This confirms that *dynamic load awareness*, not just static capability matching, is essential for heterogeneous scheduling.

Length-aware routing is also essential. Removing length-sensitive scoring causes a 45% throughput drop (6,190 → 3,391 tok/s). Without capability-weighted prefill cost estimation, the scheduler cannot differentiate between requests that benefit from high-FLOPS GPUs and those that can be efficiently served by lower-tier devices. Both terms are necessary for the full benefit.

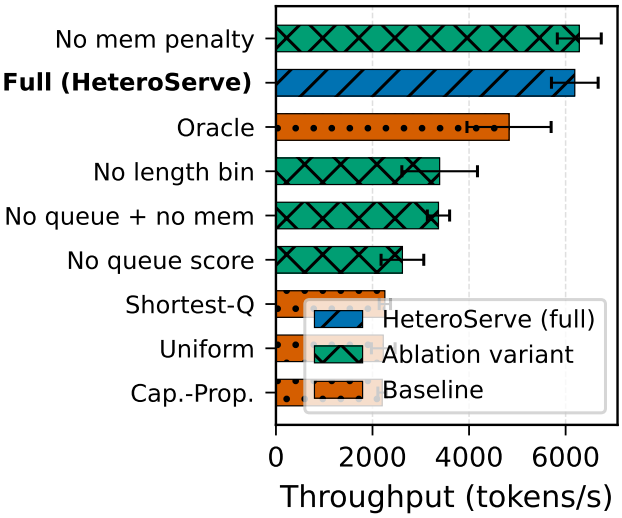

Figure 9. Ablation study showing throughput impact of disabling individual cost function components. Both queue-awareness (w_2) and length-sensitive routing (w_1) are essential.

Memory penalty is negligible under serial processing. Removing the OOM penalty ($w_3 = 0$) has minimal impact on throughput (+1.5%), because serial per-GPU processing resets KV utilization between requests. Under continuous batching, where multiple requests share GPU memory simultaneously, the memory penalty is expected to play a larger role.

Hyperparameter robustness. We sweep w_2 over $[0.1, 10.0]$ while holding $w_1 = 1.0$ and $w_3 = 100.0$ fixed. Any positive value of w_2 achieves near-optimal throughput (within 3% of the best), confirming that the system is insensitive to exact weight tuning. Only $w_2 = 0$ causes significant degradation. This robustness reduces the burden of hyperparameter selection in deployment.

5.2. Why HeteroServe Outperforms Baselines

HeteroServe's advantage arises from two complementary effects: *compute matching* routes long-context requests to high-FLOPS GPUs, while *queue-aware routing* redirects overflow to available lower-tier devices. Shortest-Queue ignores processing-time asymmetry, yielding 2.7× lower throughput despite achieving the best P95 TTFT (3.4 s vs. 14.6 s). However, HeteroServe achieves the lowest P99 E2E latency (124 s), indicating that end-to-end completion benefits from faster per-request processing on matched hardware.

Capability-weighted routing concentrates long-context requests on fewer high-FLOPS GPUs, raising potential fairness concerns. On Long-Heavy workloads, HeteroServe offers no gain over Uniform because nearly all requests compete for the same GPUs (Figure 7). Per-request-class latency targets that dynamically modulate cost weights could mitigate this; we leave such extensions to future work.

Our results reveal two necessary conditions for substantial gains: *hardware diversity* (gains scale from negligible on homogeneous to 179% on three-tier clusters, Figure 6) and *request-length diversity* (Long-Heavy workloads negate multi-tier routing benefits). Production LLM traffic naturally satisfies both conditions, making capability-weighted dispatching a practical necessity for mixed-generation clusters.

5.3. Limitations

Simulation-only evaluation. All results use a discrete-event simulator without continuous batching as in production engines (vLLM (Kwon et al., 2023), TensorRT-LLM). The reported 2.8× speedup should be interpreted as an upper bound; under continuous batching, intra-GPU scheduling recovers some inefficiency. However, the *relative* advantage of capability-weighted routing should persist, as the underlying hardware asymmetry is physical and cannot be eliminated by intra-GPU batching.

Static profiles and single-GPU routing. The cost function assumes fixed device characteristics; thermal throttling or multi-tenant interference could degrade accuracy. Additionally, HeteroServe routes to a single GPU (TP=1); extending to multi-GPU placements (Mei et al., 2025; Kim et al., 2025) and incorporating output length predictors would further improve routing quality. Our evaluation also uses synthetic Poisson arrivals rather than production traces, which may exhibit bursty, correlated patterns.

6. Conclusion

We presented HeteroServe, a lightweight capability-weighted batch scheduler that routes requests across heterogeneous GPUs via an $O(N)$ cost function combining prefill latency, queue state, and memory penalties. On a simulated three-tier cluster serving Llama-70B, HeteroServe achieves 2.8× throughput over uniform dispatching with the lowest P99 E2E latency, and gains scale from negligible on homogeneous hardware to 179% on three-tier clusters. Future work includes validation under continuous batching in production engines (Kwon et al., 2023), extension to multi-GPU parallelism, output length prediction, and fairness-aware cost weight modulation.

References

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in LLM inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, 2024.

Jiang, Y., Fu, F., and Yoneki, E. Boute: Cost-efficient llm serving with heterogeneous llms and gpus via multi-objective bayesian optimization, 2026. arXiv:2602.10729.

Kim, K., Kim, J., Kim, J. J., Li, D., and Kim, Y. Flexllm: Flexible and cost-efficient llm serving with heterogeneous gpus. In *2025 33rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2025.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, 2023.

Luo, T., Ng, K. K., Khor, Z. P., Sankhe, S., Loo, B. T., and Liu, V. Multiplexed heterogeneous llm serving via stage-aligned parallelism. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, 2025.

Mei, Y., Zhuang, Y., Miao, X., Yang, J., Jia, Z., and Vinayak, R. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025.

Patel, P., Choukse, E., Zhang, C., Shah, A., Goiri, Í., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

Yu, G., Jeong, J. S., Kim, G., Kim, S., and Chun, B. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, 2022.