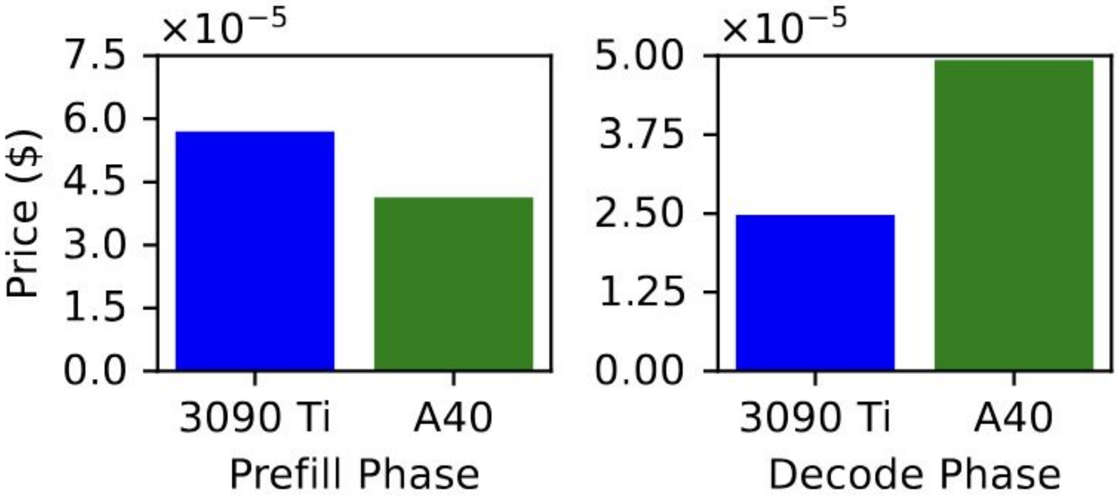
ThunderServe: High-performance and Cost-efficient LLM Serving in Cloud Environments (2025)

Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, Eiko Yoneki

The next 20 minutes

1. Key Takeaway (1 min)
   - If you remember one thing from today…
2. Background (5 min)
   - What is the heterogeneous GPU cloud paradigm?
   - What does LLM serving mean?
3. ThunderServe (10 min)
   - What does the paper propose?
4. Opinion + Discussion (4 min)

Key Takeaway



3090 Ti: Memory access-efficient (comparatively)

A40: Compute-efficient (comparatively)

Figure 1. Prefill and decode prices for a single request with input and output lengths of 512 and 16 on 3090Ti and A40.

https://huggingface.co/blog/tngtech/llm-performance-prefill-decode-concurrent-requests; Figure 2 from Jiang et al., “ThunderServe…”, MLSys 2025

Leveraging heterogeneity

Cloud systems often have heterogeneity in **hardware**: Different GPUs have different memory, bandwidth, compute abilities

And LLM inference incorporates heterogeneity in its two constituent **workloads**: Prefill (compute bound) and Decode (memory bound)

The paper argues: We should frame and solve an optimisation (~search) problem to match hardware to workloads to max performance

Background: Breaking down the paper title

High-performance and Cost-efficient LLM Serving in Cloud Environments

LLM Serving

Use a trained model to generate text. For example:

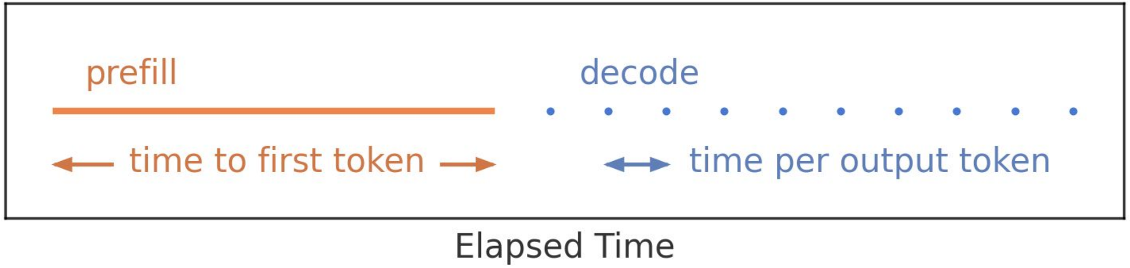
Input prompt: "The capital of France is"

Step 1 (Prefill): Process "The capital of France is" → Generate "Paris"

Step 2 (Decode): Process "The capital of France is Paris" → Generate " ."

Step 3 (Decode): Process "The capital of France is Paris." → Generate <END>

LLM Serving



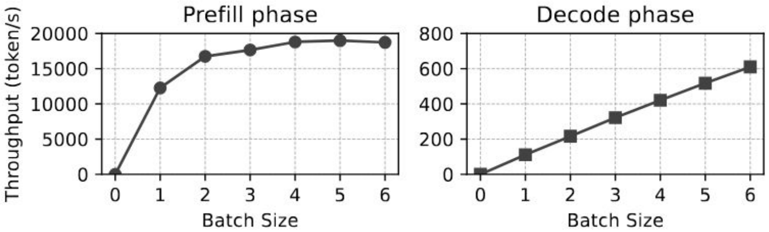
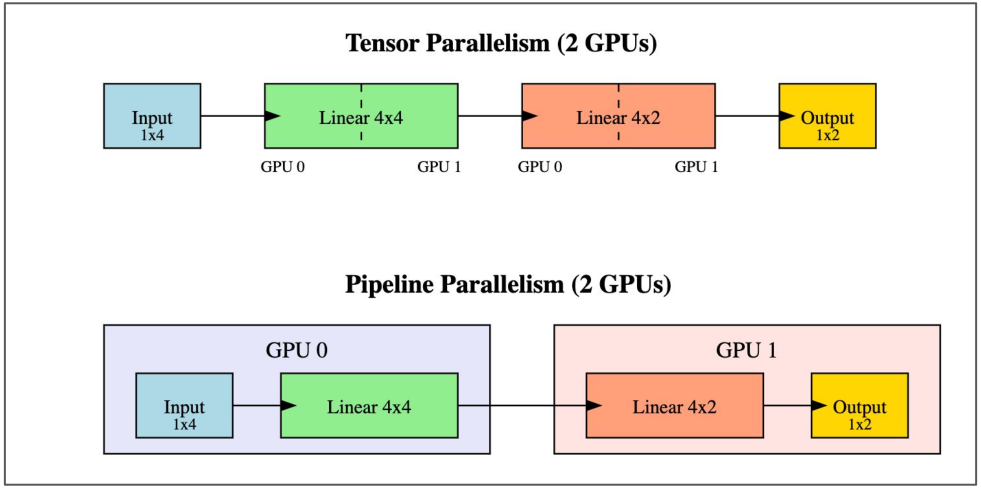


Figure 2. Effects of batching on different phases (LLaMA-7B with each input having a sequence length of 1024).

https://huggingface.co/blog/tngtech/llm-performance-prefill-decode-concurrent-requests; Figure 2 from Jiang et al., "ThunderServe…", MLSys 2025

High-performance

Generate and stream text as quickly as possible.



https://huggingface.co/blog/huseinzol05/tensor-parallelism

Cost-efficient, Cloud Environments

Cloud environments pose unique challenges:

1. **Hardware heterogeneity,** with different types and different numbers of GPUs
2. **Low network bandwidth**, limiting data transfer speed
3. **Workload variance**, with request patterns changing over time

Cost-efficient, Cloud Environments

Table 1. GPU specifications and pricing

GPU Type	Memory Access Bandwidth	Peak FP16 FLOPS	Memory Limite	Price (per GPU)
A100	2 TB/s	312 TFLOPS	80 GB	$1.753/hr
A6000	768 GB/s	38.7 TFLOPS	48 GB	$0.483/hr
A5000	626.8 GB/s	27.8 TFLOPS	24 GB	$0.223/hr
A40	696 GB/s	149.7 TFLOPS	48 GB	$0.403/hr
3090Ti	1008 GB/s	40 TFLOPS	24 GB	$0.307/hr

Table 1 from Jiang et al., “ThunderServe…”, MLSys 2025

Putting it all together

Can we find a good match between different GPUs and different phases of the LLM inference workload, accounting for the constraints of the cloud?

Primarily care to optimise Service Level Objective (SLO) attainment = number of inbound requests that can be served within the time window specified by an SLO

In my opinion, ‘cost-effective’ in the paper title is a bit of a misnomer: the paper does not optimise for cost; rather ‘find a way to make best use of what you have’

ThunderServe

Key idea 1: Phase-splitting with heterogeneous hardware

Use high FLOPS GPUs for prefill, high bandwidth GPUs for decode

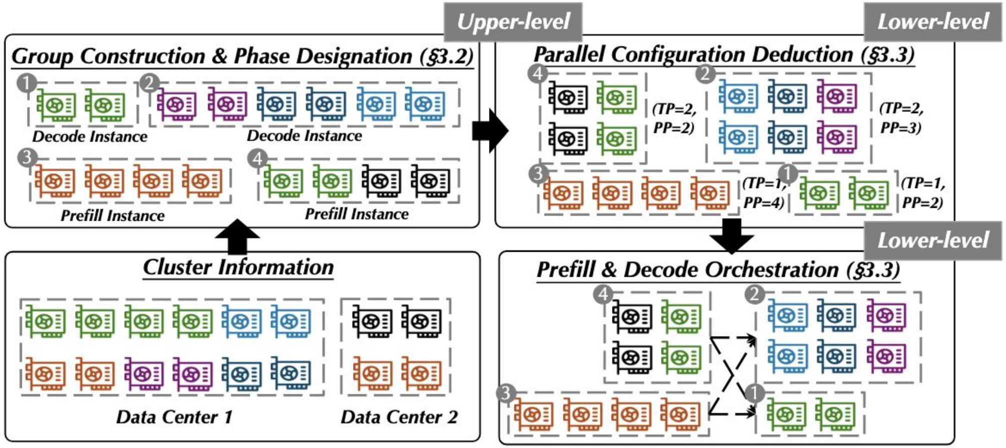


Figure 3. Workflow of our scheduling algorithm.

https://huggingface.co/blog/huseinzol05/tensor-parallelism

How? GPU <> task mapping as two-level optimisation

Yet another ‘optimisation as search’ problem…

How? GPU <> task mapping as two-level optimisation

Upper level: Which GPUs work together and what role they play

1. Partition GPUs into groups, where each group hosts one LLM replica
2. Then, designate each group as either ‘prefill’ or ‘decode’

How? GPU <> task mapping as two-level optimisation

Upper level: Which GPUs work together and what role they play

1. Partition GPUs into groups, where each group hosts one LLM replica
2. Then, designate each group as either ‘prefill’ or ‘decode’

Lower level: How requests should flow groups

1. For each group, determine the optimal parallelism (TP x PP)
2. Then, find optimal flow of requests between P-D groups
3. Estimate SLO attainment using formula

How? GPU <> task mapping as two-level optimisation

Upper level: Which GPUs work together and what role they play => tabu search

1. Partition GPUs into groups, where each group hosts one LLM replica
2. Then, designate each group as either ‘prefill’ or ‘decode’

Lower level: How requests should flow groups => surrogate function for SLO

1. For each group, determine the optimal parallelism (TP x PP) => enumerate
2. Then, find optimal flow of requests between P-D groups => linear program
3. Estimate SLO attainment using formula

Key idea 2: KV cache quantisation to reduce network load

Between the prefill and decode clusters, we need to transfer a 'KV cache' = a large matrix, which can be time-consuming on commodity cloud networks

Scenario	Bandwidth	Relative Speed
NVLink 4 (data center intra-node)	900 GB/s	1× (baseline)
InfiniBand NDR (data center inter-node)	50 GB/s	18× slower
PCIe 5.0 (cloud intra-node, best case)	64 GB/s	14× slower
100 GbE (cloud inter-node, good)	12.5 GB/s	72× slower
25 GbE (cloud inter-node, typical)	3.1 GB/s	290× slower
10 GbE (cloud inter-node, cheap)	1.25 GB/s	720× slower

Claude AI generated

Key idea 2: KV cache quantisation to reduce network load

Use 16 bit -> 4 bit quantisation *only* to compress data over the network; decode replica decompresses the data for computation

Key idea 3: Lightweight rescheduling to adjust workloads

In the face of new cloud workloads, we need to quickly re-adjust hardware <> task mapping, but existing schedulers can cause minutes of downtime

For example: coding heavy workflows are different to conversation heavy ones

The paper proposes solving a relaxed version of the optimisation problem posed earlier to re-assign workloads: only change phase designation and orchestration

Evaluation: Does this work?

Evaluation setup

Two settings at the same price budget: USD 14/hour

Cloud setting: 32 heterogeneous GPUs from Vast.ai: A6000s, A5000s, A40s, and 3090Tis; Baseline = HexGen

In-house setting: 8 A100-80GB GPUs; Baseline = vLLM and DistServe

Workload: LLaMA-30B on coding and conversation workloads from the Azure dataset, measuring SLO attainment and throughput

The assignment algorithm seems to work

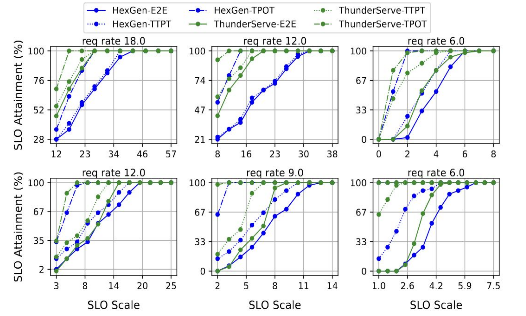


Figure 7. SLO attainment results on coding (top row) and conversation (bottom row) workloads.

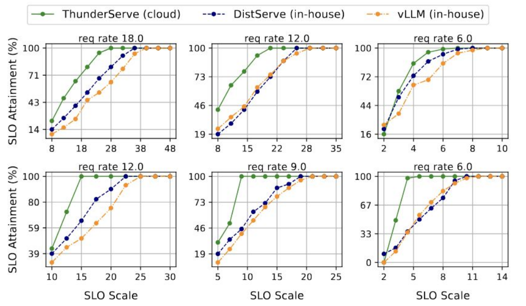


Figure 8. SLO attainment results on coding (top row) and conversation (bottom row) workloads.

Table 3. Model deployment discovered by ThunderServe.

Workload	GPU Configuration	Strategy	Type of Replicas
Coding	8×A40	TP=2, PP=1	4 Prefill Replicas
	4×A5000	TP=4, PP=1	1 Prefill Replica
	4×A6000	TP=2, PP=1	2 Prefill Replicas
	2×A5000+2×3090Ti	TP=2, PP=2	1 Prefill Replica
	4×3090Ti	TP=2, PP=2	1 Decode Replica
	4×A6000	TP=1, PP=2	2 Decode Replicas
	2×A5000+2×3090Ti	TP=2, PP=2	1 Decode Replica
Conversation	6×A40	TP=2, PP=1	3 Prefill Replicas
	2×A5000+2×3090Ti	TP=2, PP=2	1 Prefill Replica
	4×3090Ti	TP=2, PP=2	1 Decode Replica
	2×A40	TP=1, PP=2	1 Decode Replica
	4×A5000	TP=2, PP=2	1 Decode Replica
	8×A6000	TP=1, PP=2	4 Decode Replicas
	2×A5000+2×3090Ti	TP=2, PP=2	1 Decode Replica

ThunderServe performs 1.3x better than other LLM servers on a heterogeneous cloud cluster

Helix, Splitwise are not tested here

ThunderServe performs better than other LLM servers on a homogeneous cloud cluster

Paper attributes this to ThunderServe's ability to spin up 3x more replicas – hence more parallelism

Intuitively assigns the right GPU <> phase. For coding workloads, you would expect more prefill replicas where possible because larger context; vice-versa for conversation

Figure 7; Figure 8; Table 3 from Jiang et al., "ThunderServe…", MLSys 2025

Other results

- KV cache compression helps
- Lightweight scheduling is quicker and does ok

Opinion + Discussion

Phase-splitting + heterogeneous GPU is a compelling idea

Surprising in its simplicity and reduction to known operations research problems; but it yields good results and is a unique contribution

Presumably industry practitioners do versions of this behind closed doors

Phase-splitting + heterogeneous GPU is a compelling idea

Surprising in its simplicity – reduction to known operations research problems; but it yields good results and is a unique contribution

Presumably industry practitioners do versions of this behind closed doors

Can we strengthen the argument?

1. TS testing is arguably limited: small scale (32 GPUs) and only one model (LLaMa), but this is understandable
2. How does TS compare with other recent heterogeneous serving systems? Helix, Splitwise may be good to include as baseline

Open questions

1. How close to ‘optimal’ does the tabu search heuristic get?
    a. How much better or worse is this with more or less GPUs?
2. How much of the performance bottleneck is network?
    a. If latency keeps changing due to network congestion, is the lightweight scheduler sufficient, since the optimal parallelisation might change?
    b. Thinking back to the Perplexity talk
3. Is there a way to more explicitly model and optimise for cost?
    a. Can you frame the LP as ‘performance constrained by budget’?
4. (nit) The inner loop linear program in the paper does not appear to be constrained, which presumably means the solution will always converge on one flow?

References

Y. Jiang, F. Fu, X. Yao, T. Wang, B. Cui, A. Klimovic, and E. Yoneki, "ThunderServe: High-Performance and Cost-Efficient LLM Serving in Cloud Environments," *arXiv preprint arXiv:2502.09334*, 2025.