Outline

Yinuo - Introduction
1) what is the problem? - Motivation/background
2) why is it important? - Motivation/background
3) why is it hard? - Motivation/background
Introduced LLM serving……
4) why existing solutions do not work? - Problem

This might be two long of a intro: split to motivation and background? -- Yinuo

Yuanshu - Trade-off analysis
5) what is the core intuition for the solution
Prefilling
Decoding

Ola - Implementation Details? Section 4 from the original paper
5) what is the core intuition for the solution

Zack - Experiments results
6) Does the paper prove its claims?
7) What is the setup of analysis/experiments? is it sufficient?

Ola - Discussion / Conclusion
8) are there any gaps in the logic/proof?
9) describe at least one possible next step

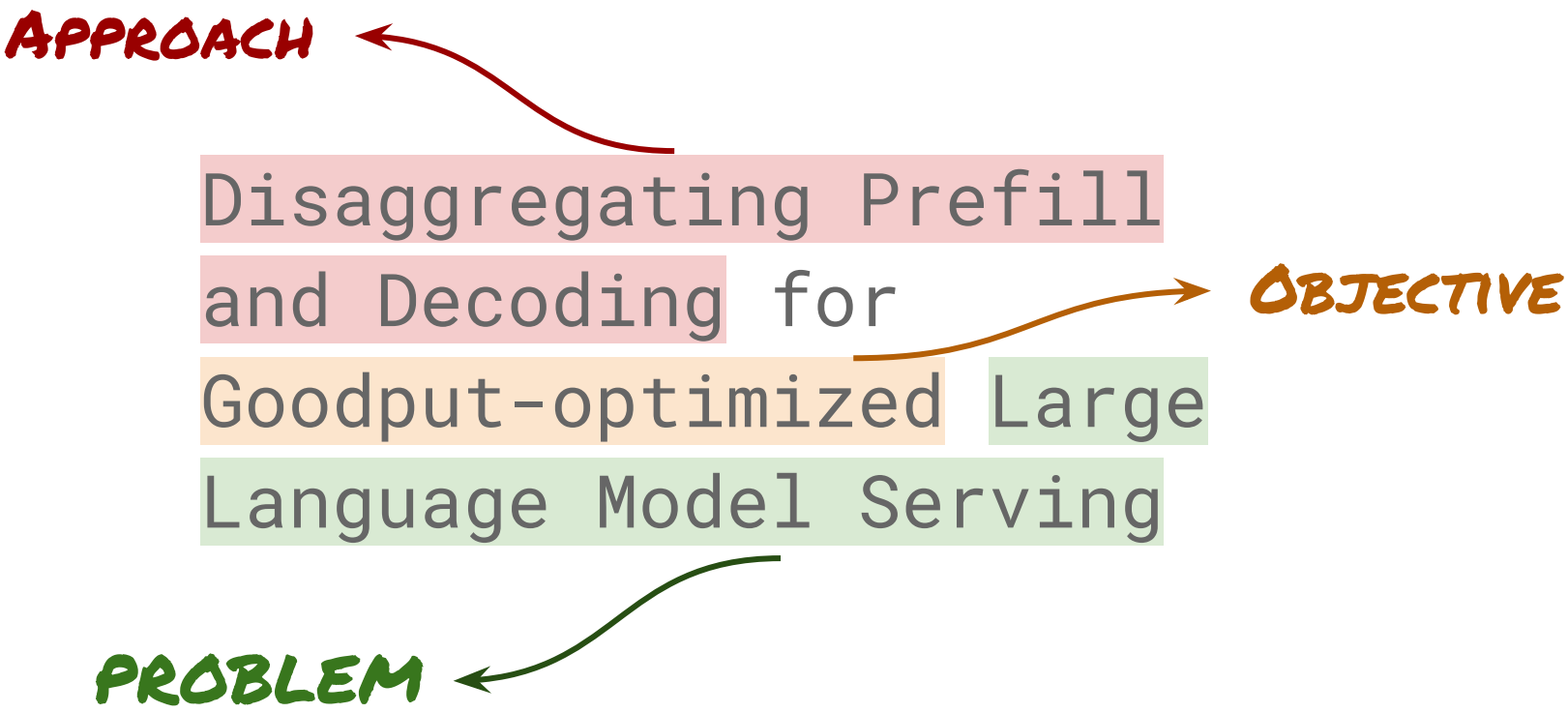
Technical Details
Very technical details, use as back up to answer questions

DistServe

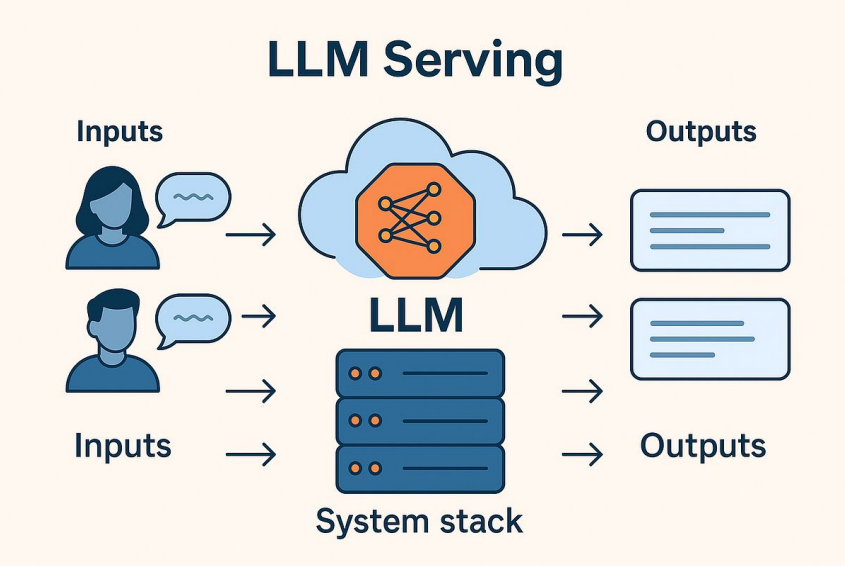
By Zhong et. al, 2024

Disaggregating Prefill and Decoding for Goodput - optimized Large Language Model Serving

Presented by:
Yinuo Zhao, Yuanshu Wang, Zhentian Wu, Ola Kulseng
April 2026, cs265@Harvard University


APPROACH
Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving
OBJECTIVE
PROBLEM

What is large language model serving?

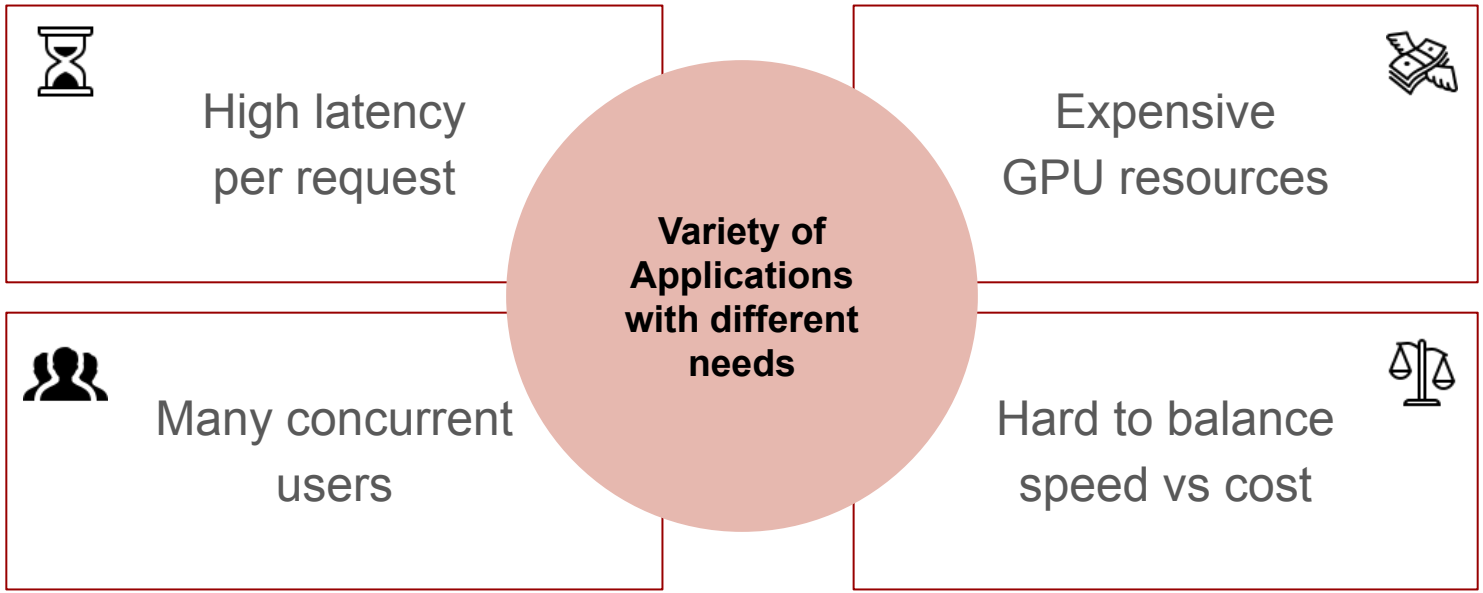


Running a trained model to answer user requests in real time.

Why is it important?

"As of 2025, **201 million**, or **67%** of organizations worldwide have adopted LLMs to support their operations with generative AI." (source)

Why Is LLM Serving So Hard?



There are many approaches for better LLM serving.

Batching & Scheduling: Continuous batching, dynamic batching, chunked prefill, request prioritization, shortest-job-first scheduling …

Memory Optimizations: PagedAttention, FlashAttention, KV cache compression, KV cache sharing, Activation recomputation …

Parallelism: Tensor, pipeline, data, operator …

System & Architecture: ***Prefill-decode disaggregation***, multi-stage pipelines, asynchronous execution …

Model-Level: Quantization, Grouped Query Attention …

…

Throughput Isn’t Enough -> Goodput

Traditional Metric: **Throughput**

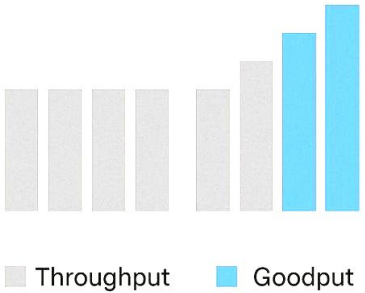
- Tokens per second
- Maximizes Utilization
- But ignores latency constraints

What We Actually Care About: **Goodput**

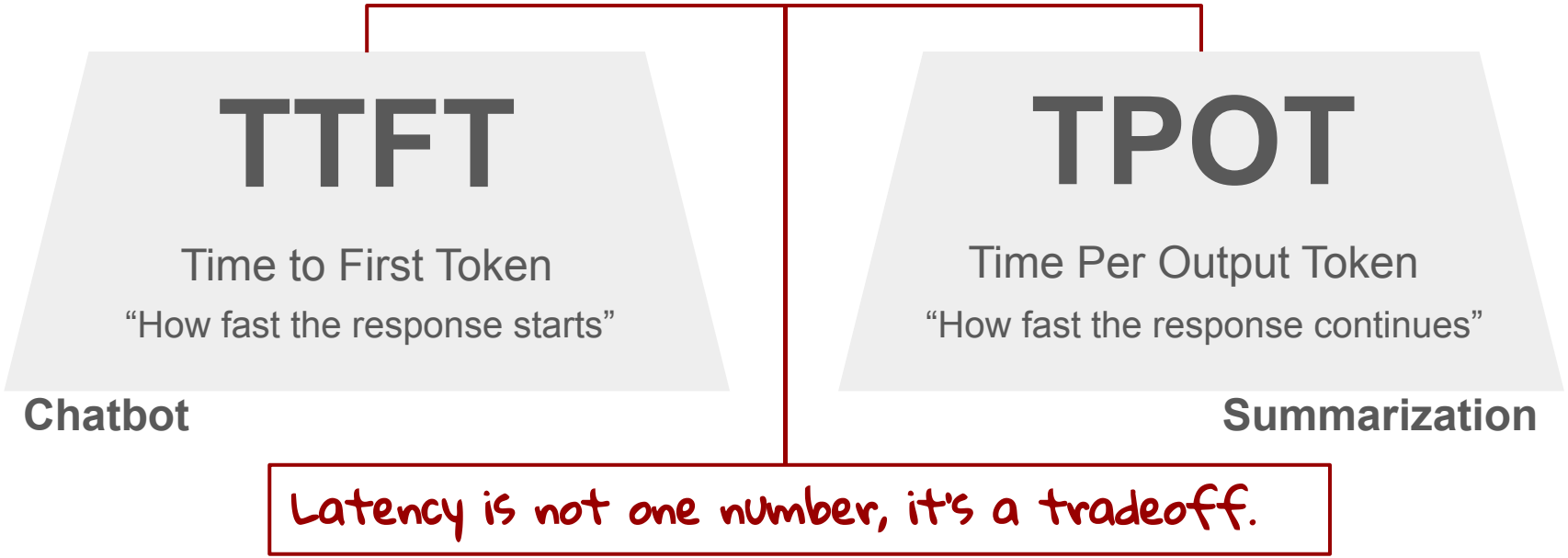
- Requests/sec that meet latency SLOs

Goodput

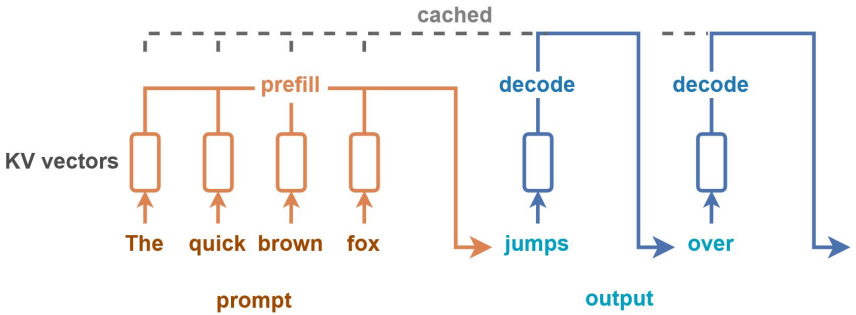
= Throughput under target metric



Latency Matters: TTFT vs TPOT



LLM serving has 2 stages: Prefilling and Decoding



Source: Benjamin Markel, 2025, Prefill and decode for concurrent requests

Prefilling

- Reads every token
- Builds and internal Understanding
- Prepared to respond

Decoding

- Model starts generating the answer
- First word
- Then next word…

Parallel

Determines TTFT

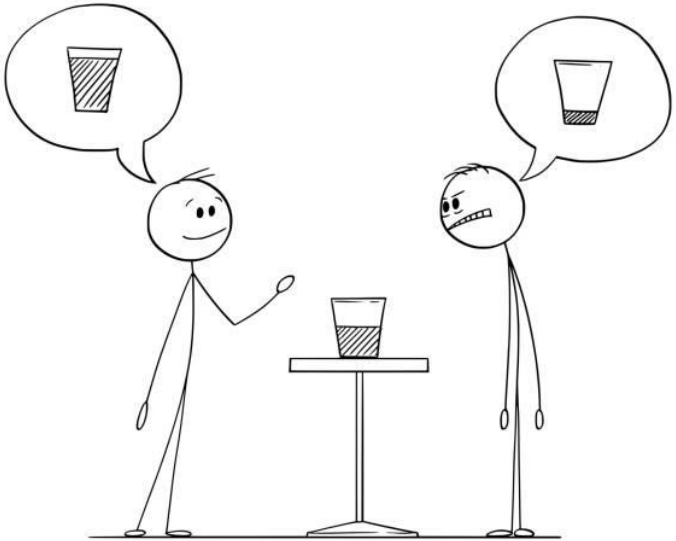
Sequential

Determines TPOT

But One Hardware Plan Cannot Fit Both Phases

Prefill prefers:

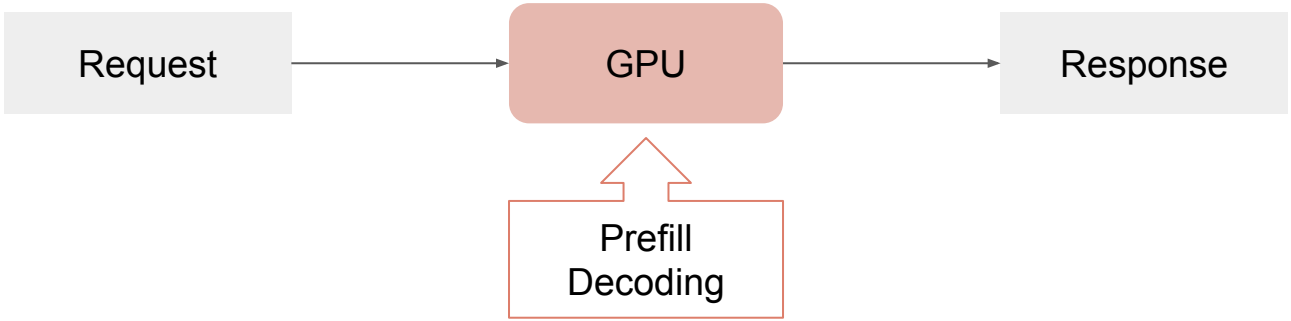
- Low TTFT
- Fast execution of large prompt processing
- Often more aggressive **compute-oriented** parallelism



Decoding prefers:

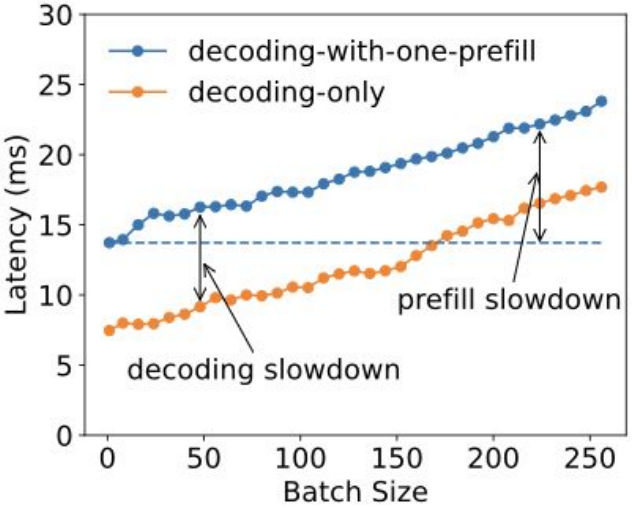
- Low TPOT
- Large and efficient decode batches
- Parallelism choices that depend on batch size and **memory behavior**

Current systems treat LLM inference as a single pipeline

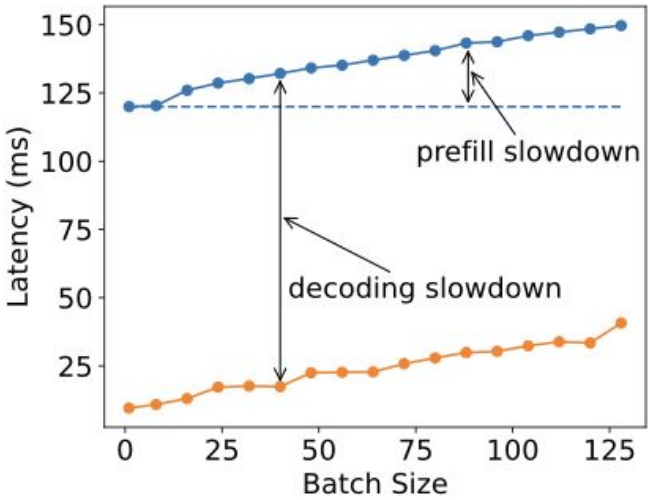


- Batch multiple requests together
- Maximize GPU utilization
- Goal: high throughput

Colocation Leads to Interference



(a) Input length = 128



(b) Input length = 1024

Essential Insights

1. Prefilling is computation bound.
2. Decoding is memory bound.
3. Their inherent differences makes the optimal parallelization techniques for the different stages incompatible.
4. ***Solution: put prefilling and decoding on different GPUs!***

Ready for the technical things?

Prefill and Decode on separate GPUs

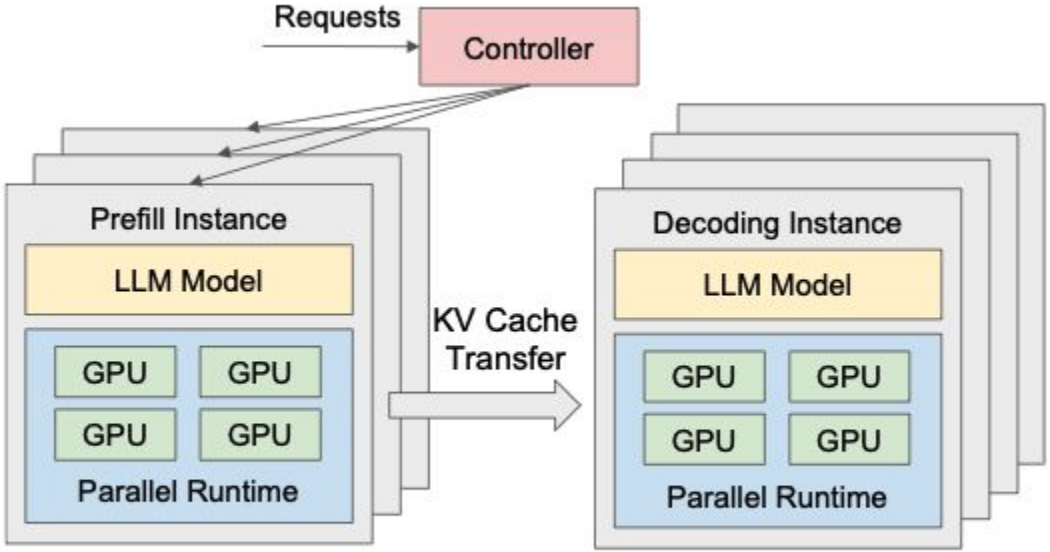
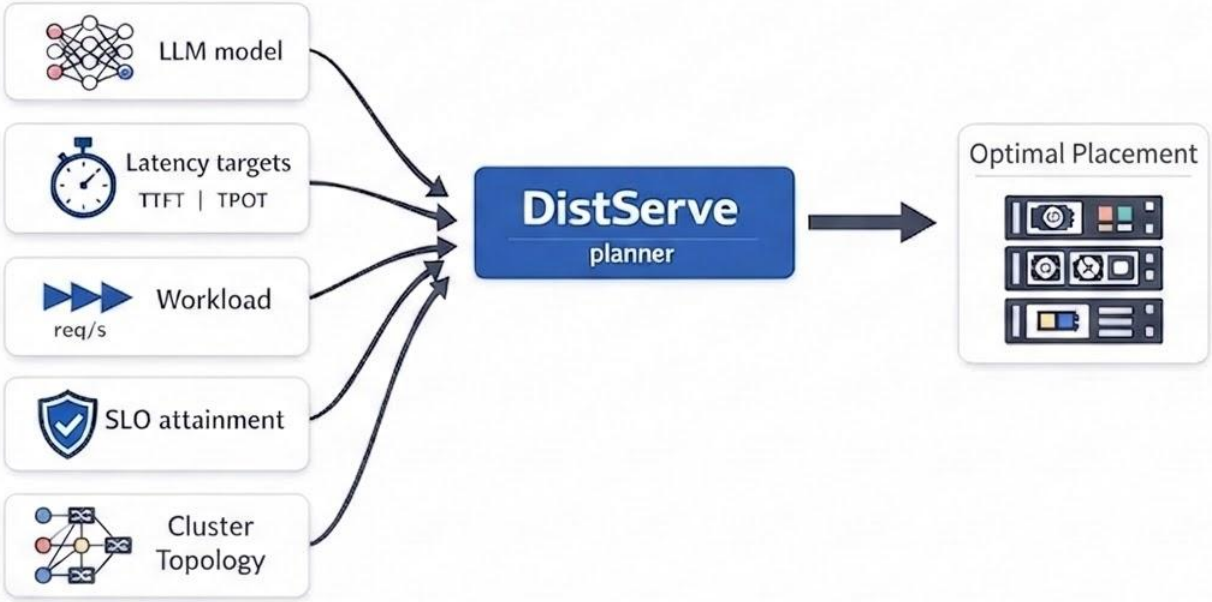
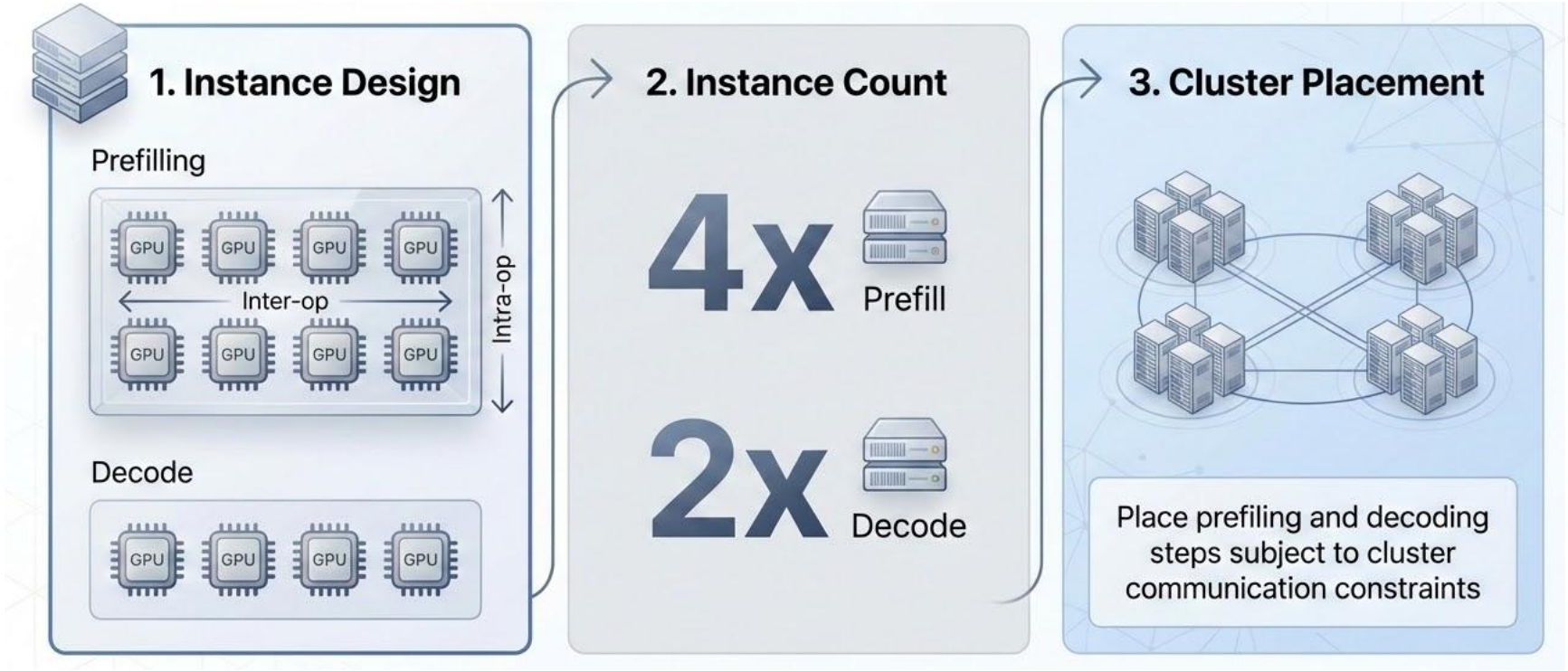


Figure 6: DistServe Runtime System Architecture

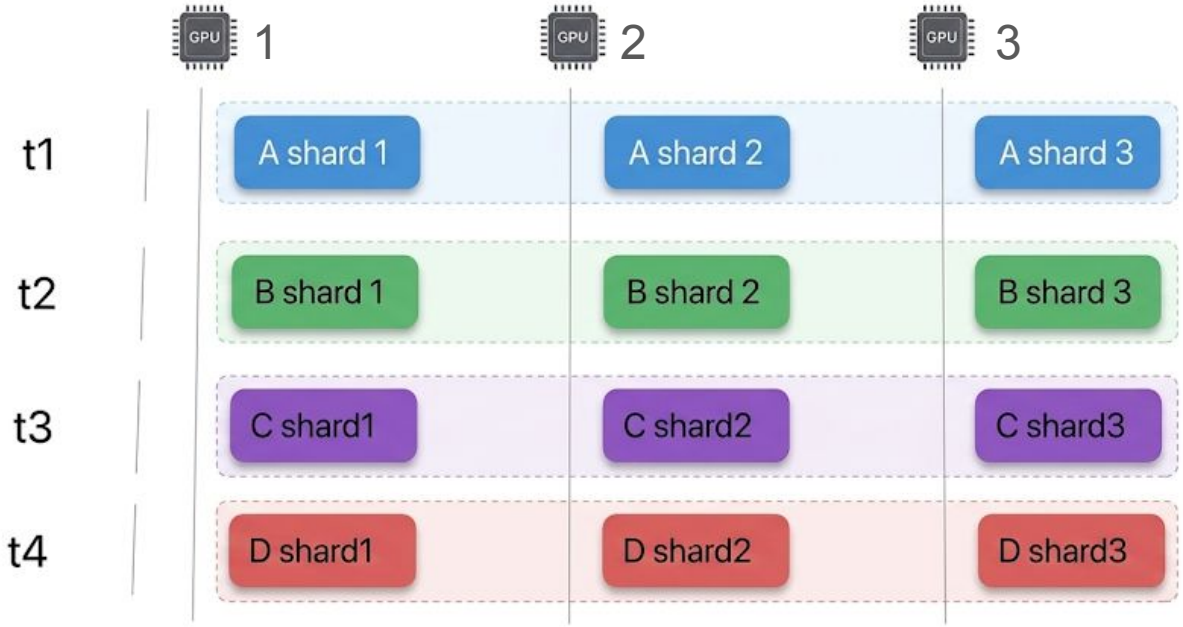
DistServe → Placement



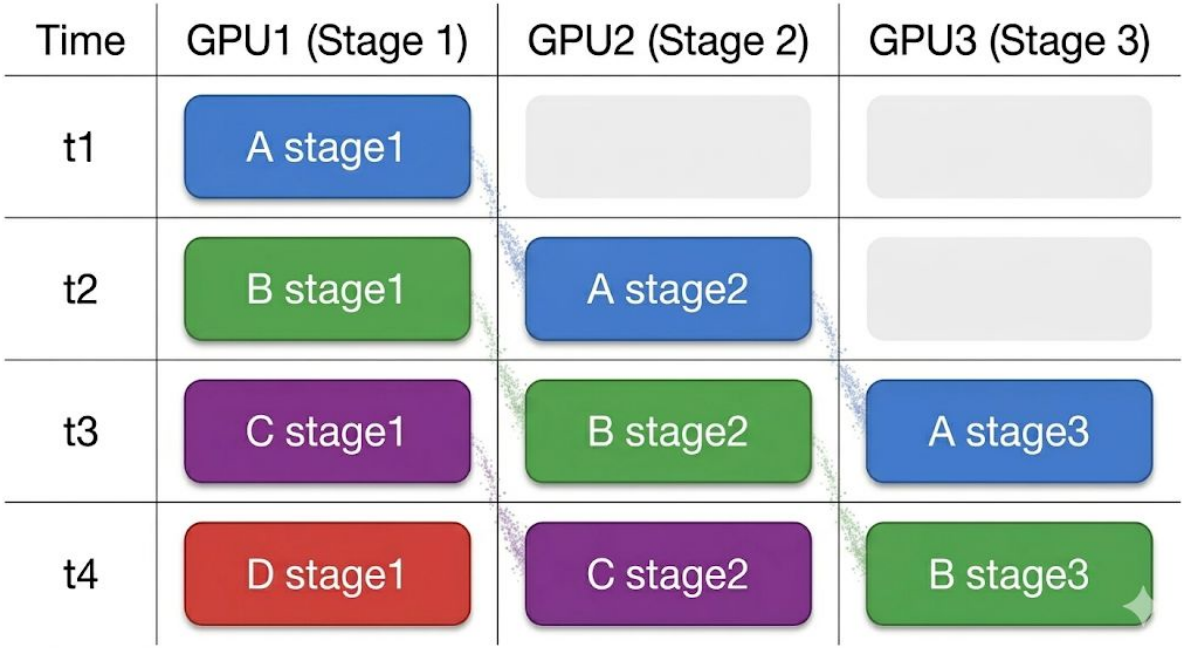
A placement includes three parts



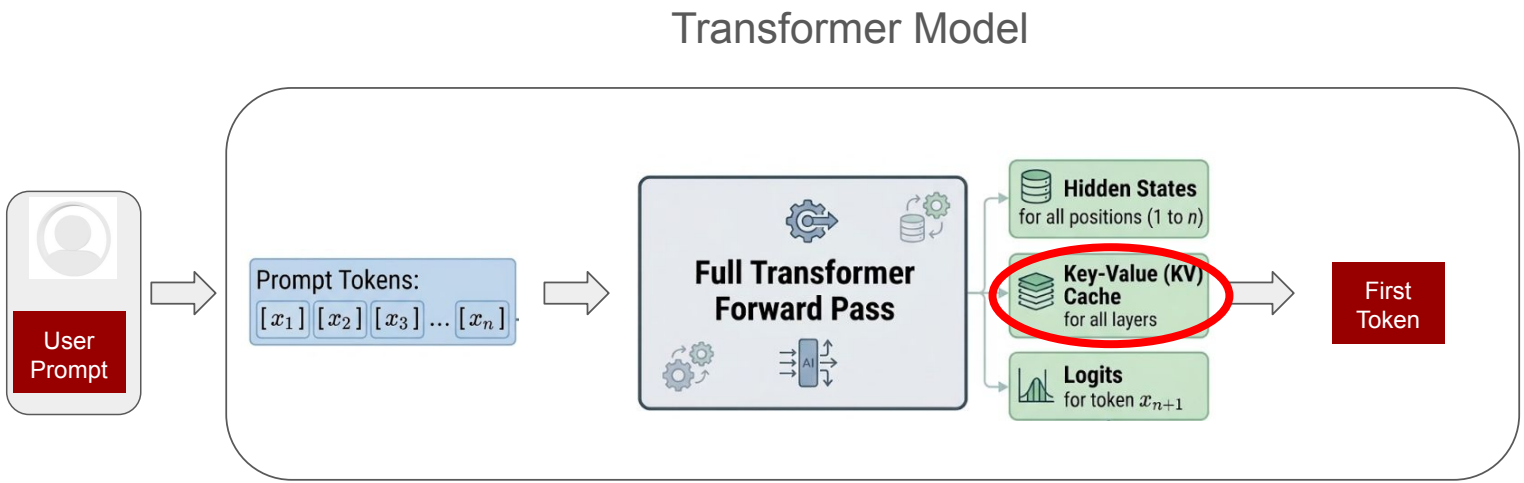
Intra-op


GPU 1
GPU 2
GPU 3
t1
t2
t3
t4
A shard 1
A shard 2
A shard 3
B shard 1
B shard 2
B shard 3
C shard1
C shard2
C shard3
D shard1
D shard2
D shard3

Inter-op

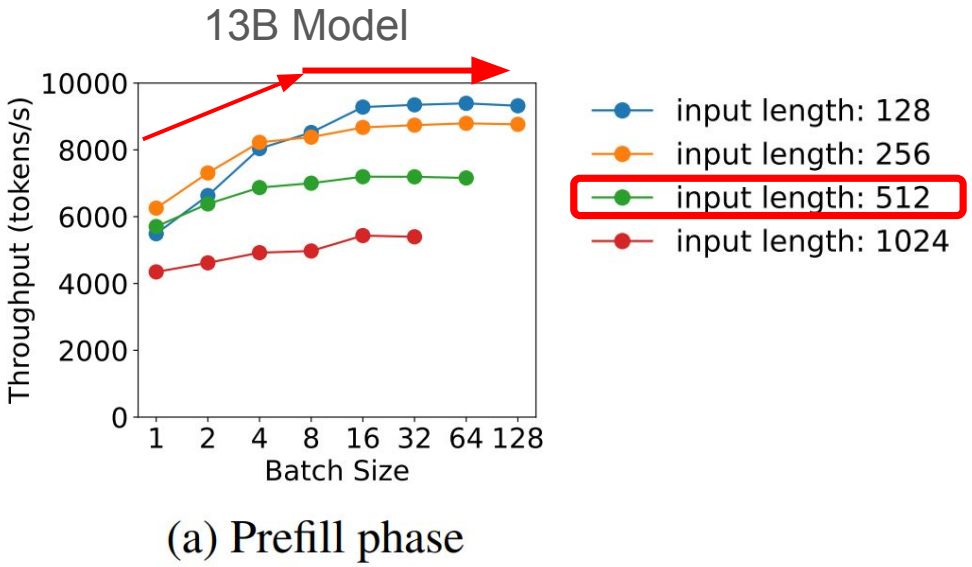

Time
GPU1 (Stage 1)
GPU2 (Stage 2)
GPU3 (Stage 3)
t1
A stage1
t2
B stage1
A stage2
t3
C stage1
B stage2
A stage3
t4
D stage1
C stage2
B stage3

Prefill-only Resource Provisioning



Optimizing methods → Batch & Parallelism

Prefill is not a “the bigger batch, the better” workload



(a) Prefill phase

A single A100 GPU is already saturated for one sequence!

Consequence of a larger batch ?

Threshold for maximum input length

Analysis setup

Model: 66B

GPU: 2x A100

Input size: 512

Arrival process: Poisson arrival

Compare

Inter-op | Intra-op

M/D/1 queue

Request Poisson arrival rate

execution time for each request is fixed at D

RD < 1: meaning the system must not be overloaded

$$Avg\_TTFT = D + \frac{RD^2}{2(1-RD)}$$

execution time

queuing delay

When R is small, execution time dominates TTFT

When RD closes to 1, queuing dominates TTFT

2-way Inter-op

Due to negligible interlayer activation communication

$$D \approx D_s \approx 2D_m$$

Total Request-level latency

Dm: slowest stage

$$Avg\_TTFT^{inter} = D_s + \frac{RD_m^2}{2(1 - RD_m)} = D + \frac{RD^2}{4(2 - RD)}$$

execution time

Queuing delay

$$D_s \approx D$$

$$D_m \approx D/2$$

2-way Intra-op

$$Avg\_TTFT^{intra} = \frac{D}{K} + \frac{RD^2}{2K(K - RD)}$$

Queuing delay

Speedup Factor
1 < K < 2

execution time

The execution time significantly reduced!

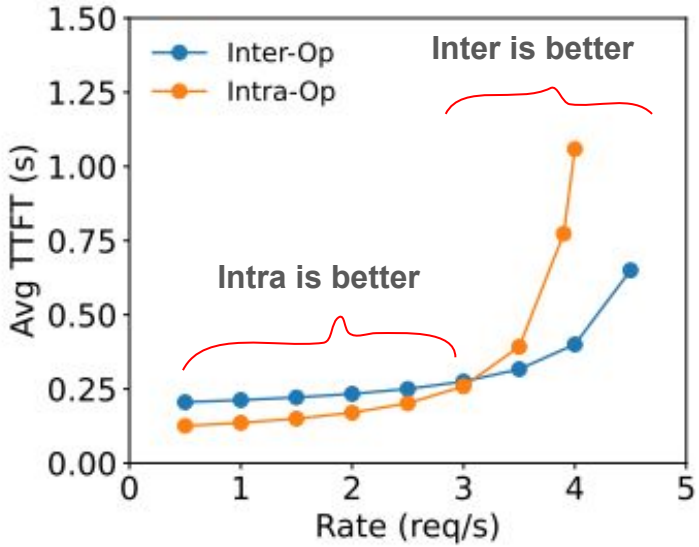
K is large

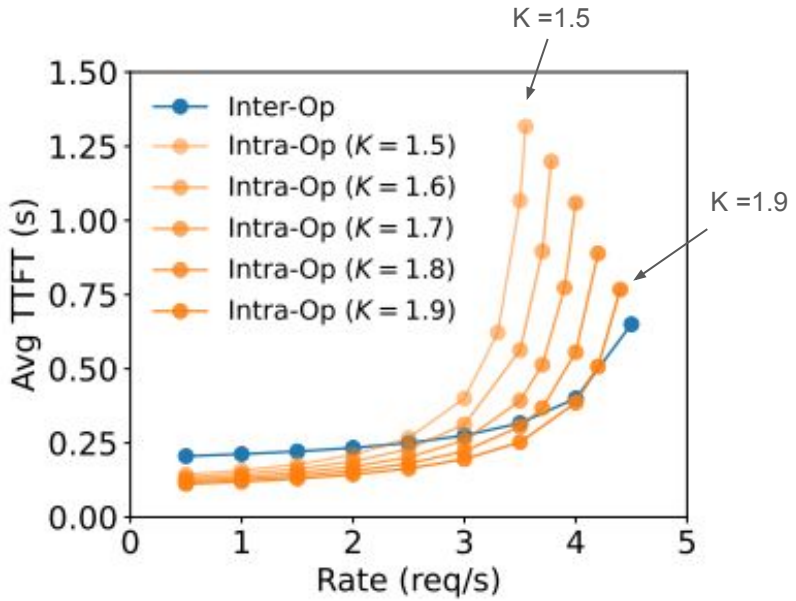
Good GPU collaboration → faster request processing → lighter queue.

K is small

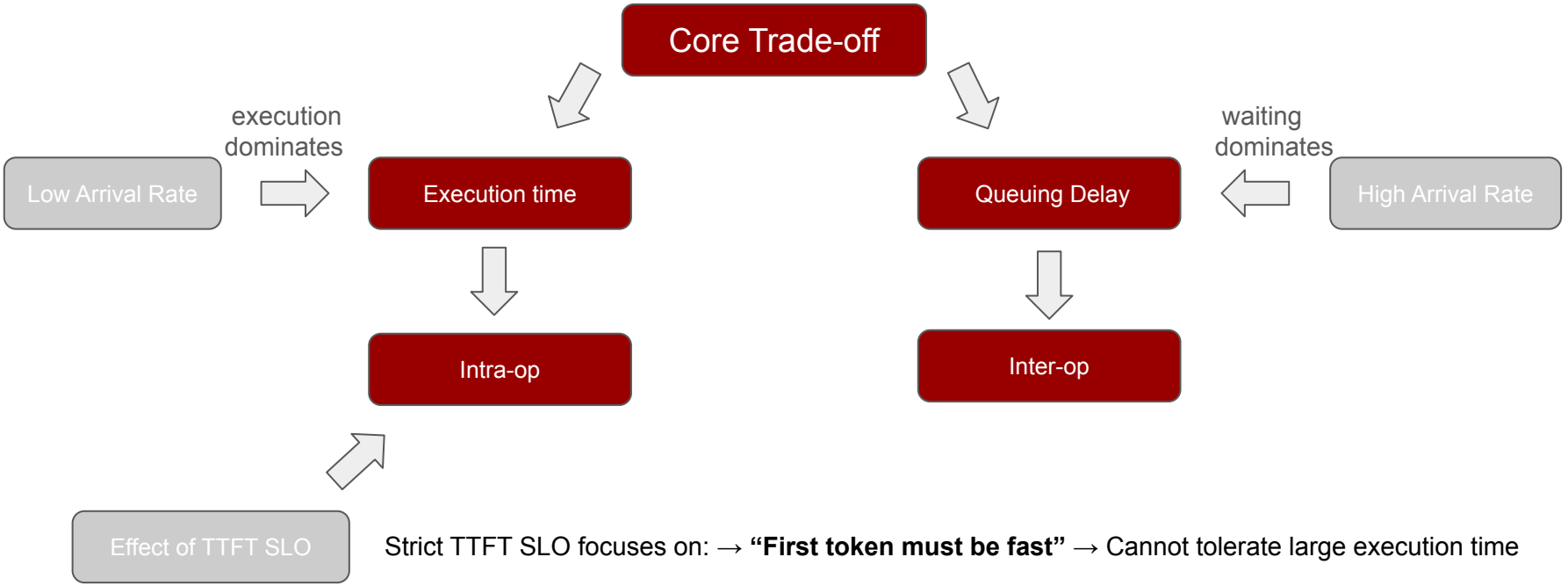
Communication overhead offsets gains → little speedup → limited queue improvement.

Intra-op vs Inter-op

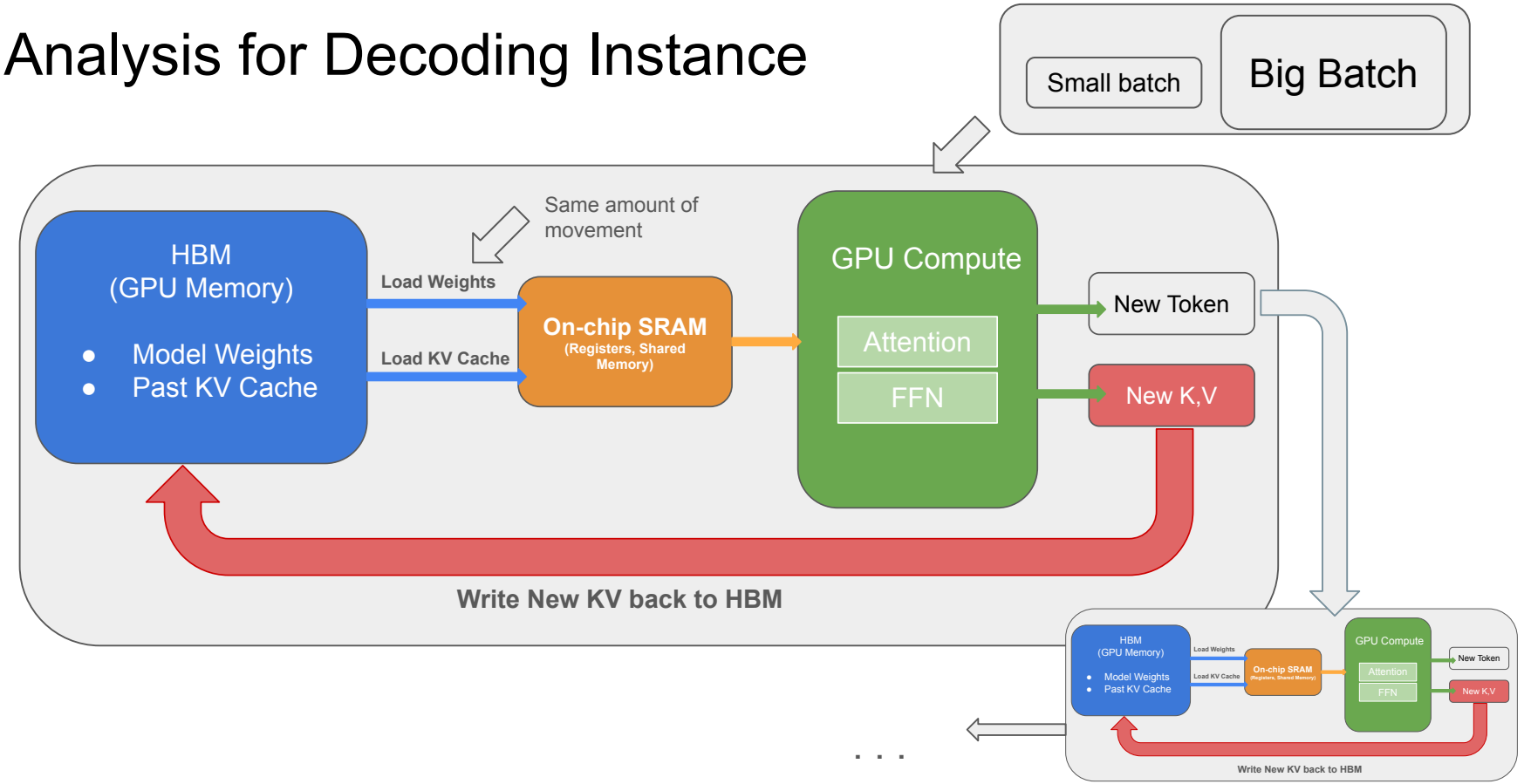




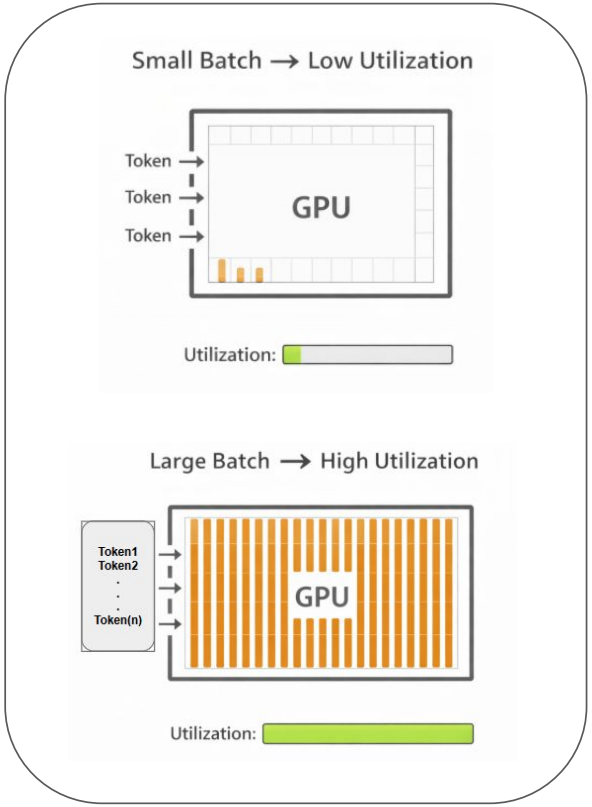
Intra-op vs Inter-op

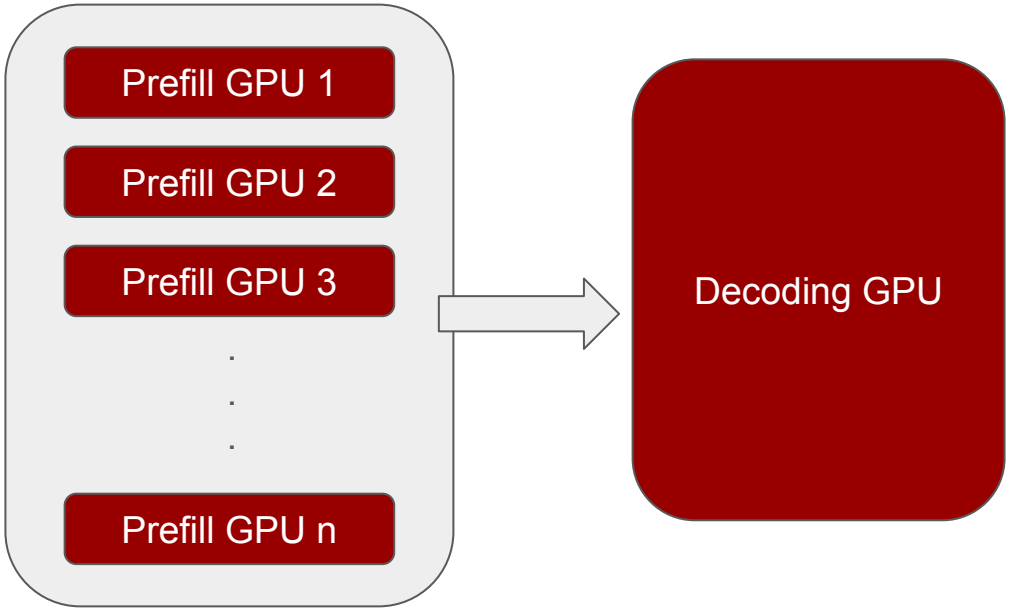


Analysis for Decoding Instance

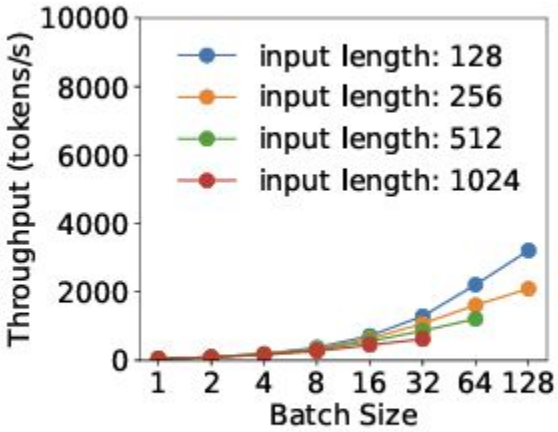


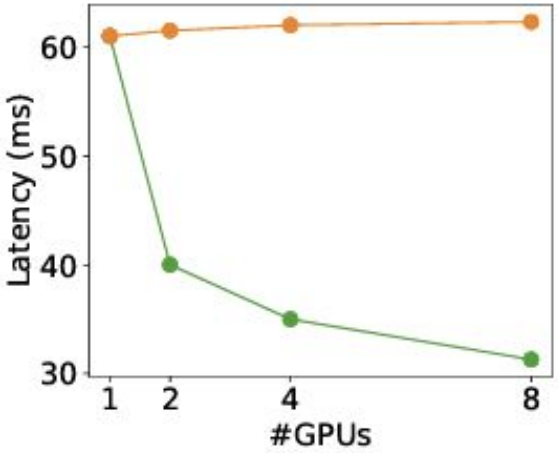
Disaggregation Helps batching

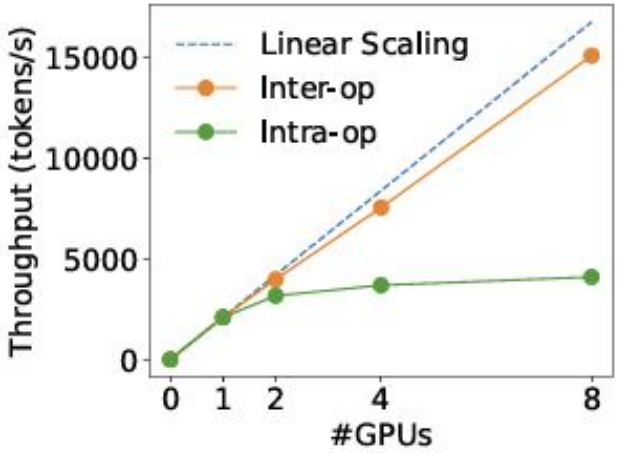




Performance







Batching

Parallelism strategies

Experimental Setup

Low node-affinity placement algorithm
Models: OPT-13B, OPT-66B, OPT-175B
Applications:

- Chatbot (ShareGPT),
- Code completion(HumanEval),
- Summarization(LongBench)

Metric: Goodput - request/sec meeting latency SLO goal (90%)

Baselines:

- vLLM,
- DeepSpeed-MII

Application	Model Size	TTFT	TPOT	Dataset
Chatbot OPT-13B	26GB	0.25s	0.1s	ShareGPT [8]
Chatbot OPT-66B	132GB	2.5s	0.15s	ShareGPT [8]
Chatbot OPT-175B	350GB	4.0s	0.2s	ShareGPT [8]
Code Completion OPT-66B	132GB	0.125s	0.2s	HumanEval [14]
Summarization OPT-66B	132GB	15s	0.15s	LongBench [13]

Main Result: Chatbot Application

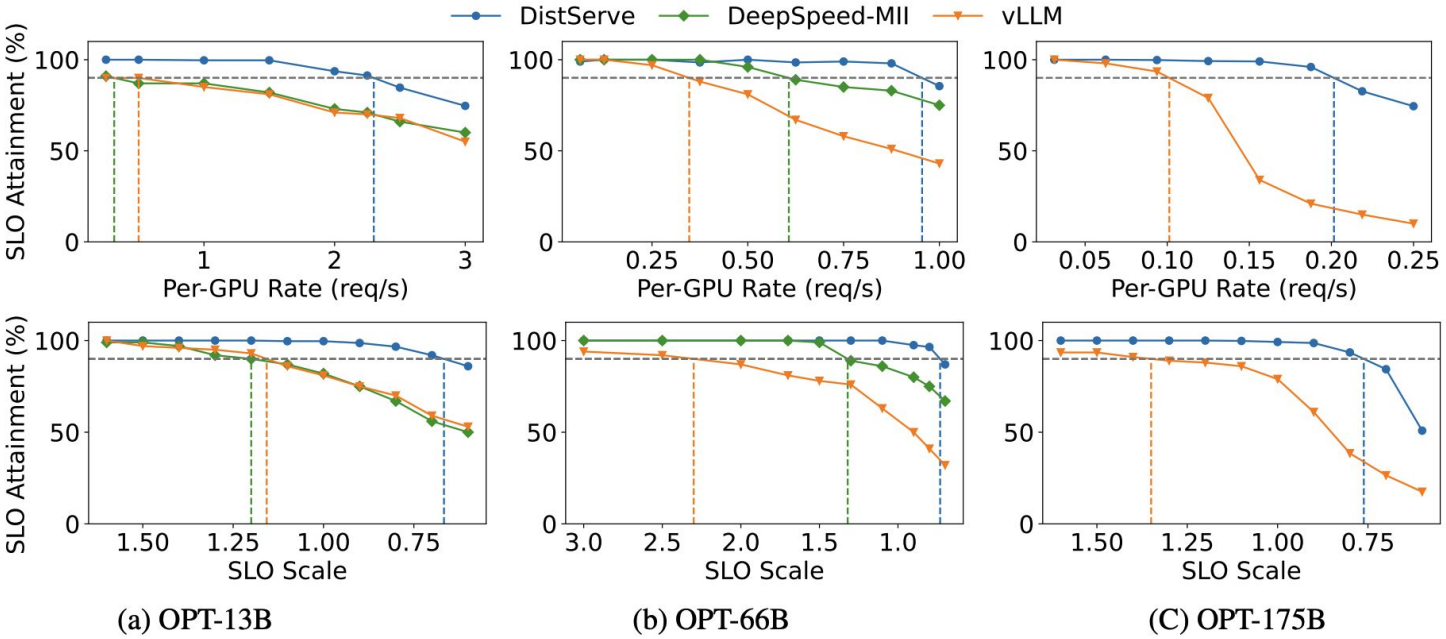


Figure 8: Chatbot application with OPT models on the ShareGPT dataset.

Performance Across Other Applications

Code completion: very strict TTFT
Summarization: looser TTFT, but more TPOT-sensitive and much longer inputs

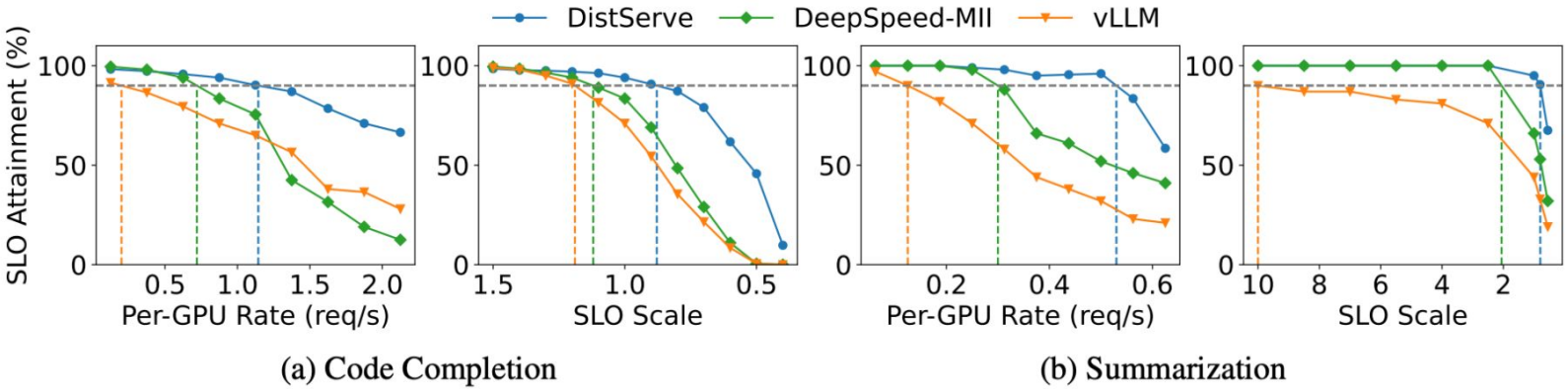


Figure 9: Code completion and summarization tasks with OPT-66B on HumanEval and LongBench datasets, respectively.

Overhead Analysis: Is Communication a Problem?

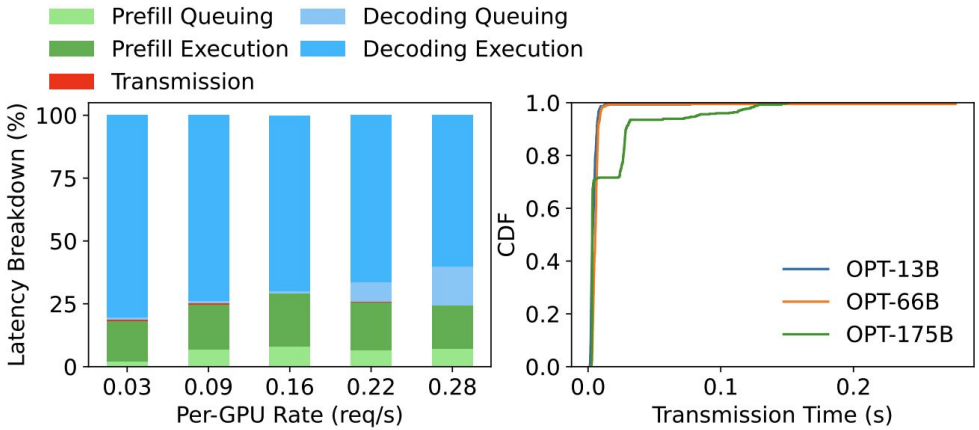
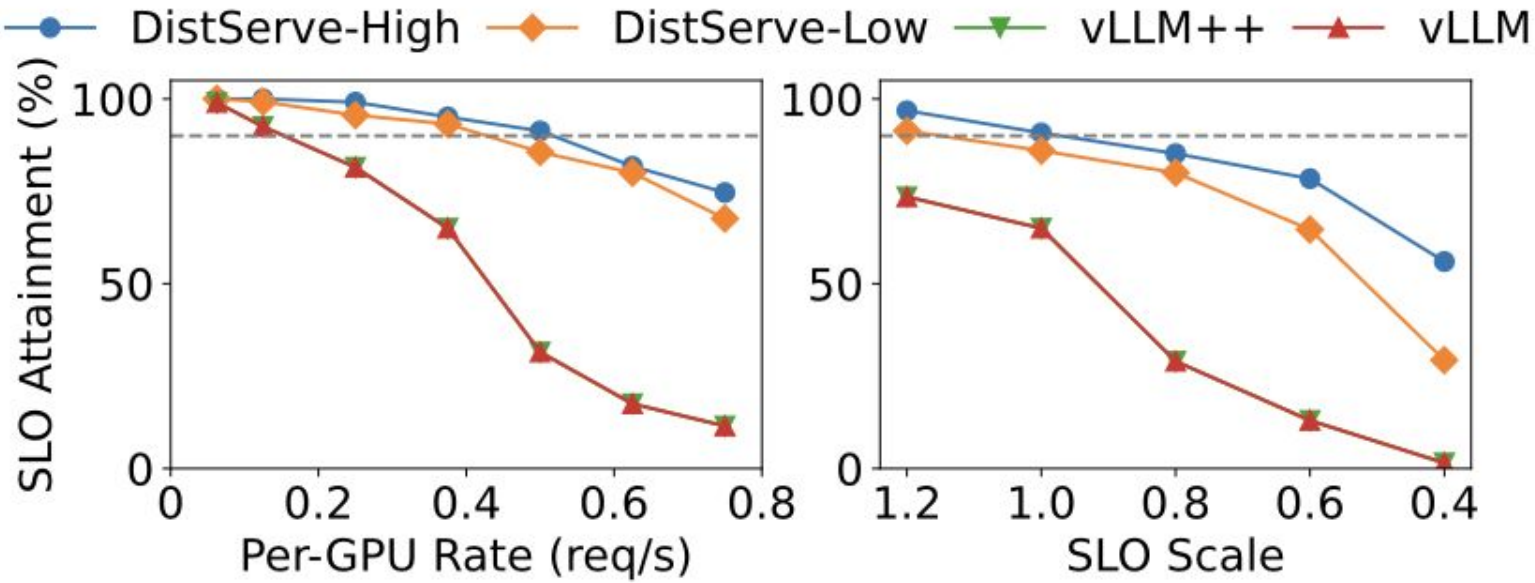


Figure 10: *Left:* Latency breakdown when serving OPT-175B on ShareGPT dataset with DistServe. *Right:* The CDF function of KV Cache transmission time for three OPT models.

Ablation Study

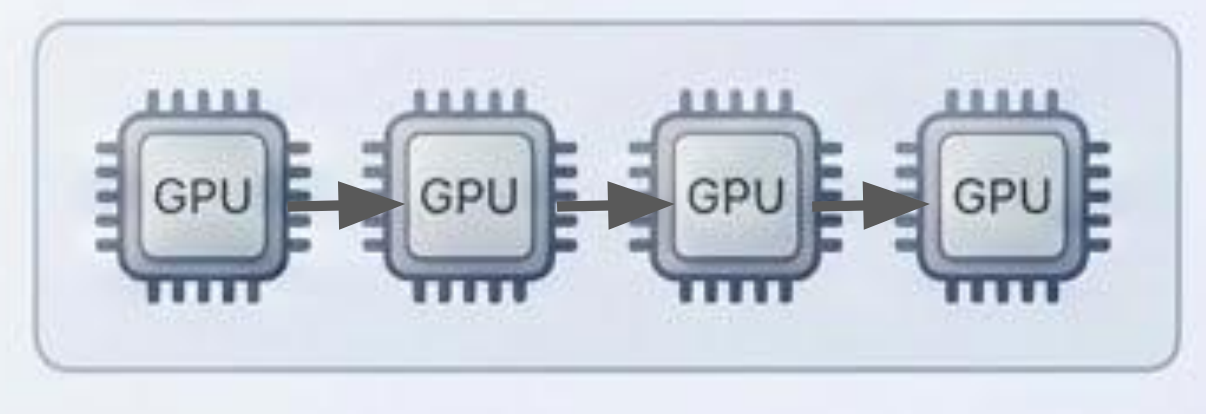


Practical Problem

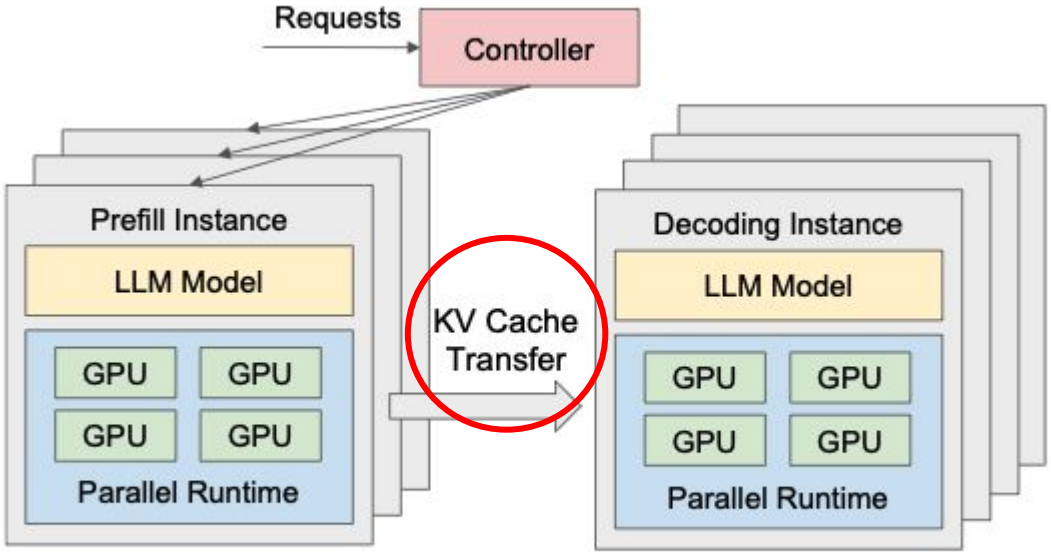
Variable prefill length. Earlier analysis assumed all prompts have the same length, but in reality request lengths vary. That causes **pipeline bubbles**, especially when using inter-op parallelism for prefill, because different requests take different amounts of time. DistServe addresses this later with workload-aware search and better scheduling.

Communication overhead. Disaggregation means the **KV cache must be transferred** from prefill GPUs to decoding GPUs. For large models and long prompts, this can be a lot of data, so enough bandwidth is needed. The paper argues this is manageable on modern clusters, especially with **InfiniBand** or **intra-node NVLINK**, but it adds placement constraints.

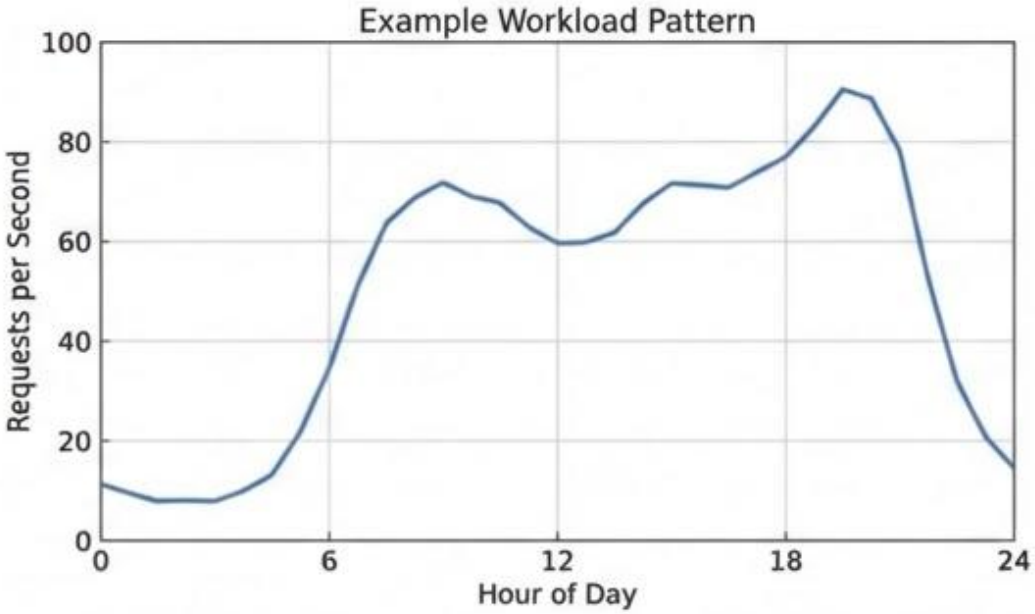
Problem 1: Pipeline bubbles



Problem 2: KV Cache OOM



Problem 3: Workload drift



Thank you!

Yinuo Zhao

T.H Chan School of Public Health
MS in Health Data Science

Yuanshu Wang

Zhentian Wu

Ola Tangen Kulseng

Estimation of SLOs

Here are symbols related to the architecture of the model:

- h: hidden size
- n: number of heads
- s: head size ($h = n \cdot s$)
- m: FFN intermediate size

Note: If tensor parallelism is used, h, n, and m should be divided by the tensor parallelism size.

Below are symbols that characterize the batch to be executed:

- B: batch size
- $l_0, l_1, \ldots, l_{B-1}$: input length of each request within the batch
- t: number of tokens in the batch, ($t = \sum_{i=0}^{B-1} l_i$)
- t_2: squared sum of the input lengths ($t_2 = \sum_{i=0}^{B-1} l_i^2$)
- b: block size in the attention kernel. This parameter is used in FlashAttention [20], a common kernel optimization technique adopted by current LLM serving systems.

$$T_{Prefill} = C_1 \cdot (4th^2 + 2thm) + C_2 \cdot \frac{3ht_2}{b} + C_3$$

$$T_{Decoding} = C_4 \cdot (4h^2 + 2hm) + C_5 \cdot 3ht$$

Algorithms

Algorithm 1 High Node-Affinity Placement Algorithm

Input: LLM G, #node limit per-instance N, #GPU per-node M, GPU memory capacity C, workload W, traffic rate R.
Output: the placement *best_plm*.

$config_p, config_d \leftarrow \emptyset, \emptyset$
for $intra\_op \in \{1, 2, ..., M\}$ **do**
  for $inter\_op \in \{1, 2, ..., \frac{N \times M}{intra\_op}\}$ **do**
    if $\frac{G.size}{inter\_op \times intra\_op} < C$ **then**
      $config \leftarrow (inter\_op, intra\_op)$
      $\hat{G} \leftarrow \text{parallel}(G, config)$
      $config.goodput \leftarrow \text{simu\_prefill}(\hat{G}, W)$
      if $\frac{config_p.goodput}{config_p.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_p \leftarrow config$
      $config.goodput \leftarrow \text{simu\_decode}(\hat{G}, W)$
      if $\frac{config_d.goodput}{config_d.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_d \leftarrow config$
$n, m \leftarrow \lceil \frac{R}{config_p.goodput} \rceil, \lceil \frac{R}{config_d.goodput} \rceil$
$best\_plm \leftarrow (n, config_p, m, config_d)$
return *best_plm*

Algorithm 2 Low Node-Affinity Placement Algorithm

Input: LLM G, #node limit per-instance N, #GPU per-node M, GPU memory capacity C, workload W, traffic rate R.
Output: the placement *best_plm*.

$config^* \leftarrow \emptyset$
for $inter\_op \in \{1, 2, ..., N\}$ **do**
  $\mathcal{P} \leftarrow \text{get\_intra\_node\_configs}(G, M, C, inter\_op)$
  for $P_p \in \mathcal{P}$ **do**
    for $P_d \in \mathcal{P}$ **do**
      if $P_p.num\_gpus + P_d.num\_gpus \leq M$ **then**
        $config \leftarrow (inter\_op, P_p, P_d)$
        $\hat{G}_p, \hat{G}_d \leftarrow \text{parallel}(G, config)$
        $config.goodput \leftarrow \text{simulate}(\hat{G}_p, \hat{G}_d, W)$
        if $\frac{config.^*goodput}{config.^*num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
          $config^* \leftarrow config$
$n \leftarrow \lceil \frac{R}{config.^*goodput} \rceil$
$best\_plm \leftarrow (n, config^*)$
return *best_plm*

Simulator Accuracy

Rate (req/s)	vLLM		DistServe-Low	
	Real System	Simulator	Real System	Simulator
1.0	97.0%	96.8%	100.0%	100.0%
1.5	65.5%	65.1%	100.0%	100.0%
2.0	52.8%	51.0%	99.3%	99.3%
2.5	44.9%	46.1%	87.3%	88.3%
3.0	36.7%	38.3%	83.0%	84.1%
3.5	27.8%	28.0%	77.3%	77.0%
4.0	23.6%	24.1%	70.0%	68.9%

Table 2: Comparison of the SLO attainment reported by the simulator and the real system under different rates.

Repetition

It is worth noting that when the model can fit into the memory of a single GPU, replication is a competitive option in addition to model parallelism for both prefill and decoding instances, to linearly scale the system's rate capacity. It may also reduce the queuing delay – as indicated by Eq. 1 – by substituting R with R/N assuming requests are equally dispatched to N replicas, at the cost of maintaining additional replicas of the model weights in GPU memory

Why RD<1

In the paper, RD<1is the stability condition for the M/D/1 queue that models the prefill system. Here, R is the request arrival rate and D is the execution time per request, so their product RD represents server utilization. This value must stay below 1 because the system can only remain stable when requests arrive more slowly than they can be completed on average. If RD≥1, then work enters the system at least as fast as it leaves, so the queue keeps growing and the waiting time becomes unbounded. That is why the paper's TTFT formula is only valid under RD<1: as RRD approaches 1, the denominator in the queuing-delay term becomes very small, causing latency to increase sharply.

How to choose?

After disaggregation, DistServe profiles each phase separately. For prefill, it usually uses small batches and chooses intra-op or inter-op depending on whether TTFT is dominated more by execution time or queueing. For decoding, it tries to increase batch size first, then uses intra-op when TPOT is tight and inter-op when it wants more throughput. In the end, it searches for the combination that gives the best goodput under both TTFT and TPOT constraints.”

Why is prefill modeled as an M/D/1 queue?

Because in the paper’s simplified analysis, requests arrive randomly following a Poisson process, and each request has the same service time since the input length is fixed. That matches the M/D/1 setting: random arrivals, deterministic service time, and one server.

Does disaggregation introduce communication overhead?

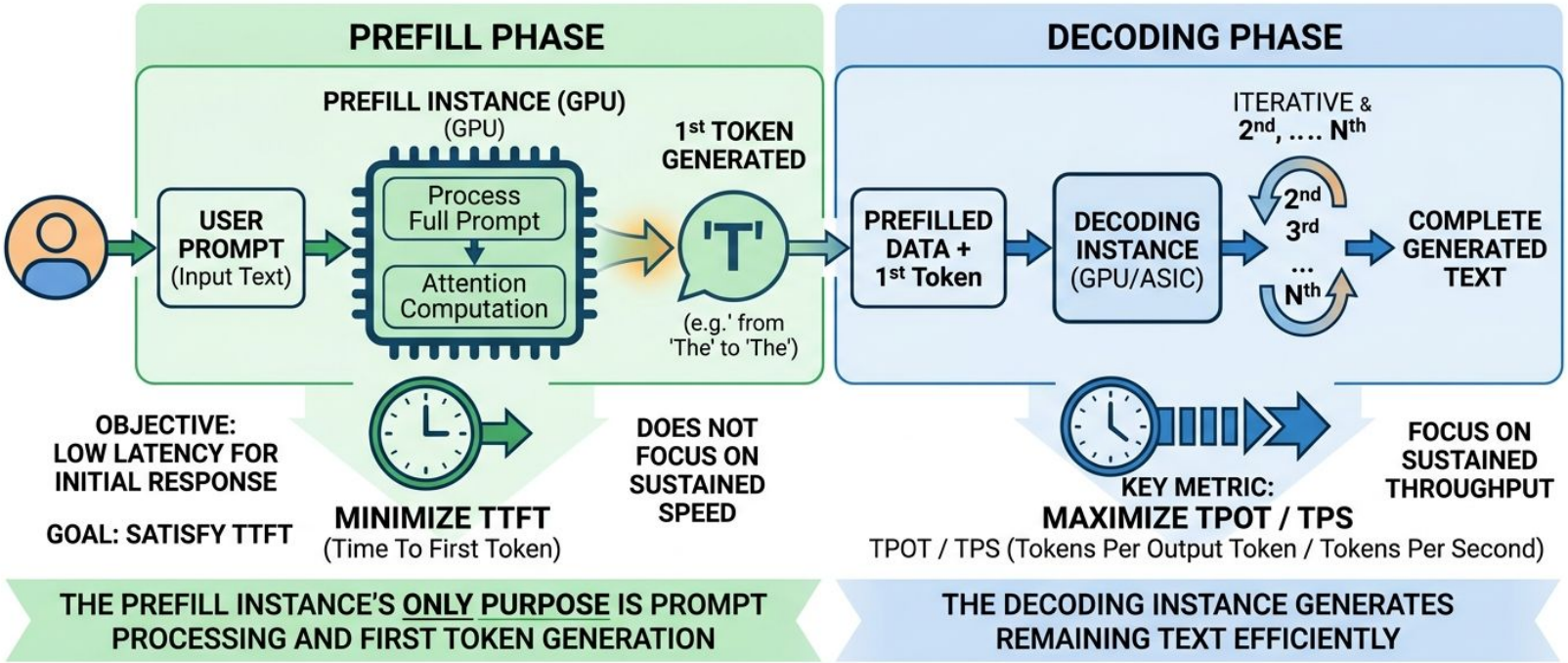
Yes, because the KV cache has to be transferred from prefill GPUs to decoding GPUs. But the paper argues that with good placement and high-bandwidth connections like NVLink or InfiniBand, this overhead is relatively small compared with the performance gain from separating the two phases.

What does “goodput” mean here?

Goodput means the maximum request rate the system can serve while still meeting the latency target for a required percentage of requests, such as 90%. So it is not just raw throughput — it is throughput under SLO constraints.

So how does DistServe actually choose the strategy?

It treats prefill and decoding separately. For prefill, it chooses the batching and parallelism that best satisfy TTFT. For decoding, it chooses the batching and parallelism that best satisfy TPOT. Then it searches over different GPU allocations and parallelism combinations to find the highest goodput while still meeting both SLOs.


PREFILL PHASE
DECODING PHASE
PREFILL INSTANCE (GPU)
(GPU)
USER PROMPT (Input Text)
Process Full Prompt
Attention Computation
1st TOKEN GENERATED
'T'
(e.g.' from 'The' to 'The')
ITERATIVE & 2nd, ... Nth
PREFILLED DATA + 1st Token
DECODING INSTANCE (GPU/ASIC)
2nd
3rd
...
Nth
COMPLETE GENERATED TEXT
OBJECTIVE: LOW LATENCY FOR INITIAL RESPONSE
GOAL: SATISFY TTFT
MINIMIZE TTFT
(Time To First Token)
DOES NOT FOCUS ON SUSTAINED SPEED
KEY METRIC:
MAXIMIZE TPOT / TPS
TPOT / TPS (Tokens Per Output Token / Tokens Per Second)
FOCUS ON SUSTAINED THROUGHPUT
THE PREFILL INSTANCE'S ONLY PURPOSE IS PROMPT PROCESSING AND FIRST TOKEN GENERATION
THE DECODING INSTANCE GENERATES REMAINING TEXT EFFICIENTLY