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

Yinmin Zhong[1] Shengyu Liu[1] Junda Chen[2] Jianbo Hu[1] Yibo Zhu[3] Xuanzhe Liu[1]
Xin Jin[1] Hao Zhang[2]

[1]*Peking University* [2]*UC San Diego* [3]*StepFun*

Presented by Mingxuan Liu, PhD student at *Northwestern Polytechnical University*

in 2024 Fall Reading Group Meeting at USTC

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

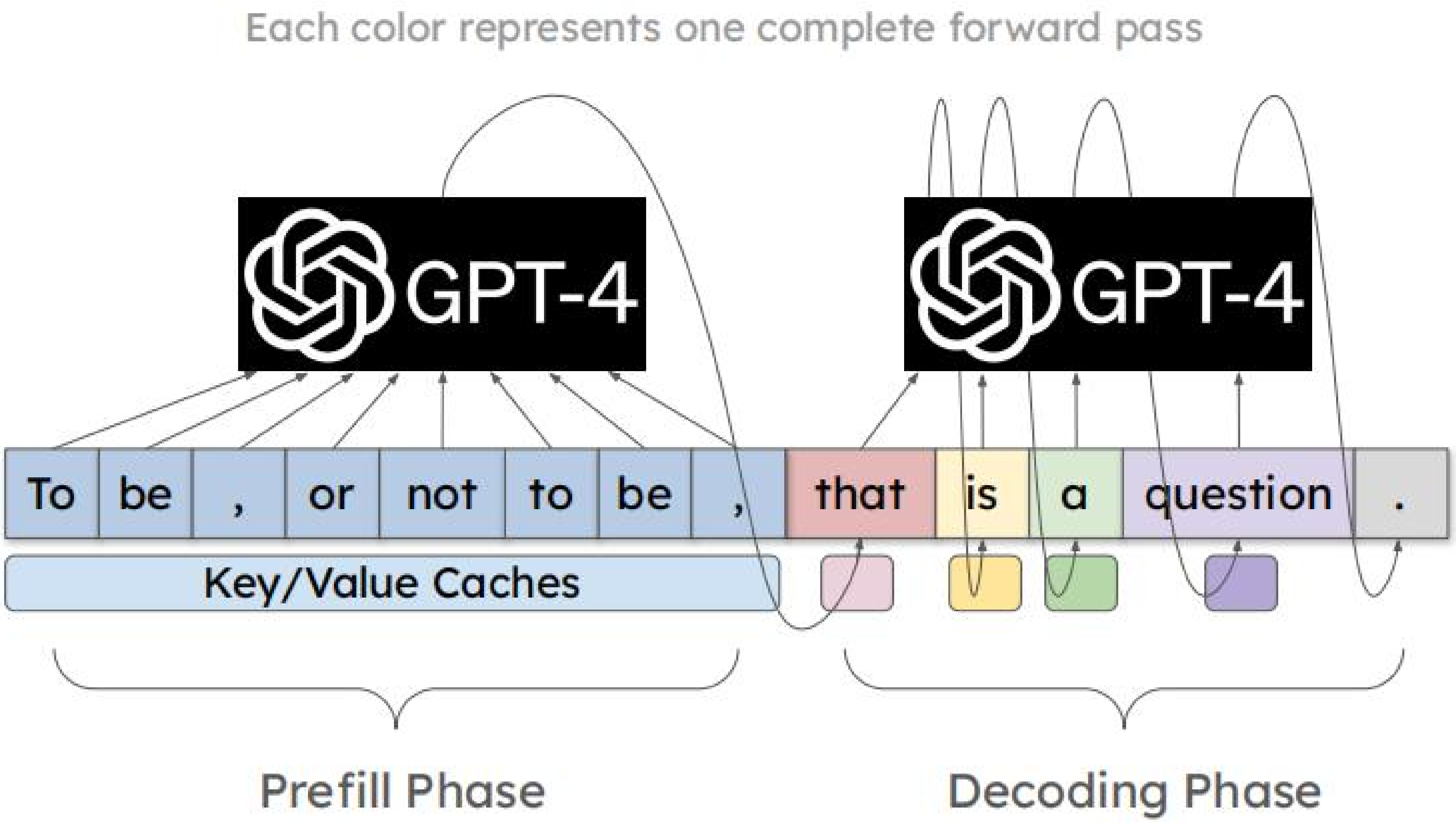
- **Background**
- **Motivations**
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Tradeoff Analysis**
- **Method**
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- **Implementation**
- **Evaluation**
- **Discussion & Summary**

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Background**
- **Motivations**
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Tradeoff Analysis**
- **Method**
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- **Implementation**
- **Evaluation**
- **Discussion & Summary**

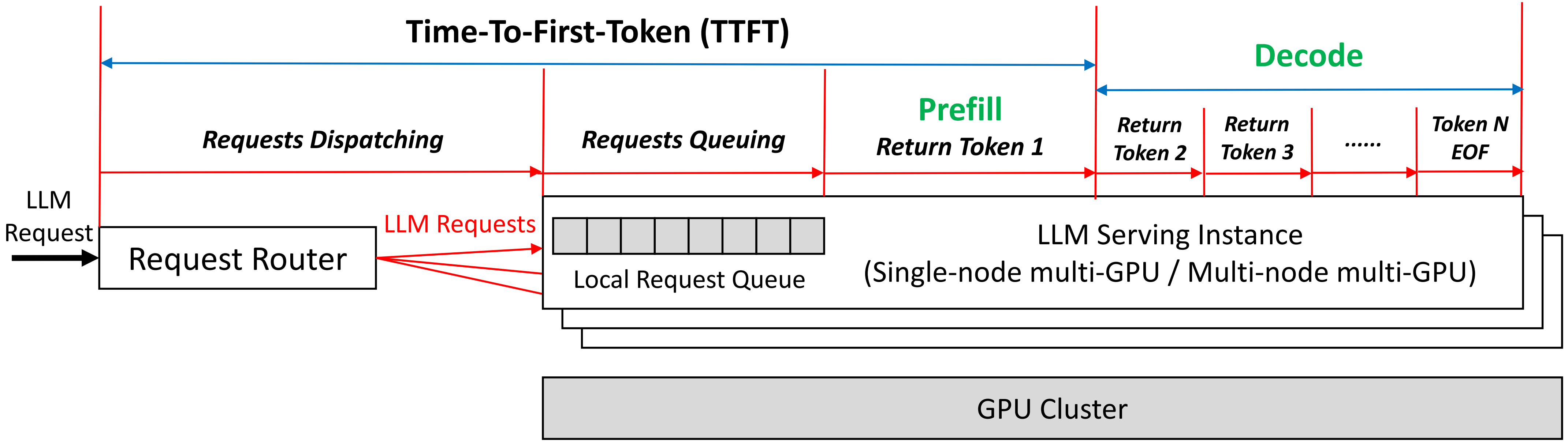
Background: LLM Inference



Background: LLM Inference

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

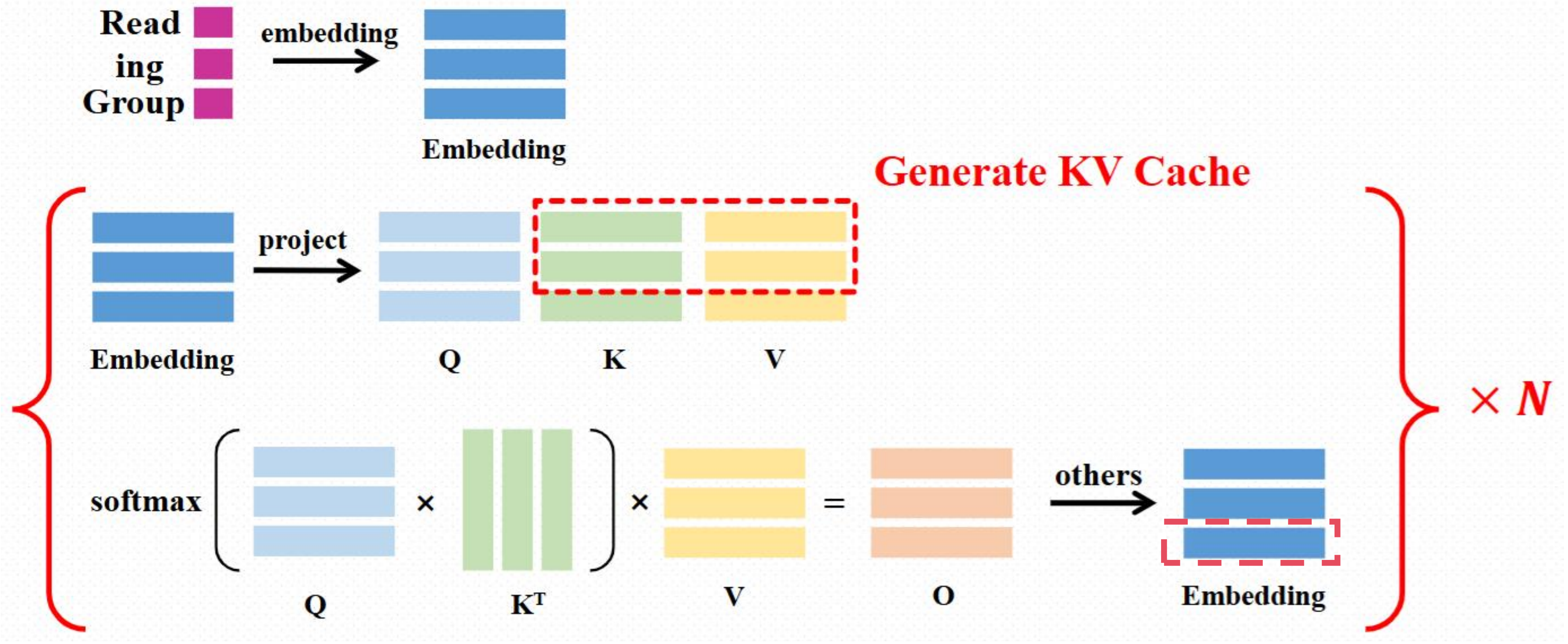
- **LLM Inference: 1 prefill step + N decode step**
- Constraints (***X***, ***Y***, ***M*** are defined according to the scenario):
  - **TTFT (Time to first token)** < ***X*** seconds
  - **TPOT (Time per output token)**: During the decode phase, at least ***M*** tokens must be returned within ***Y*** seconds.



Background: Prefill in LLM Inference

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Prefill: Generate KV cache & first token -> Compute-bound**



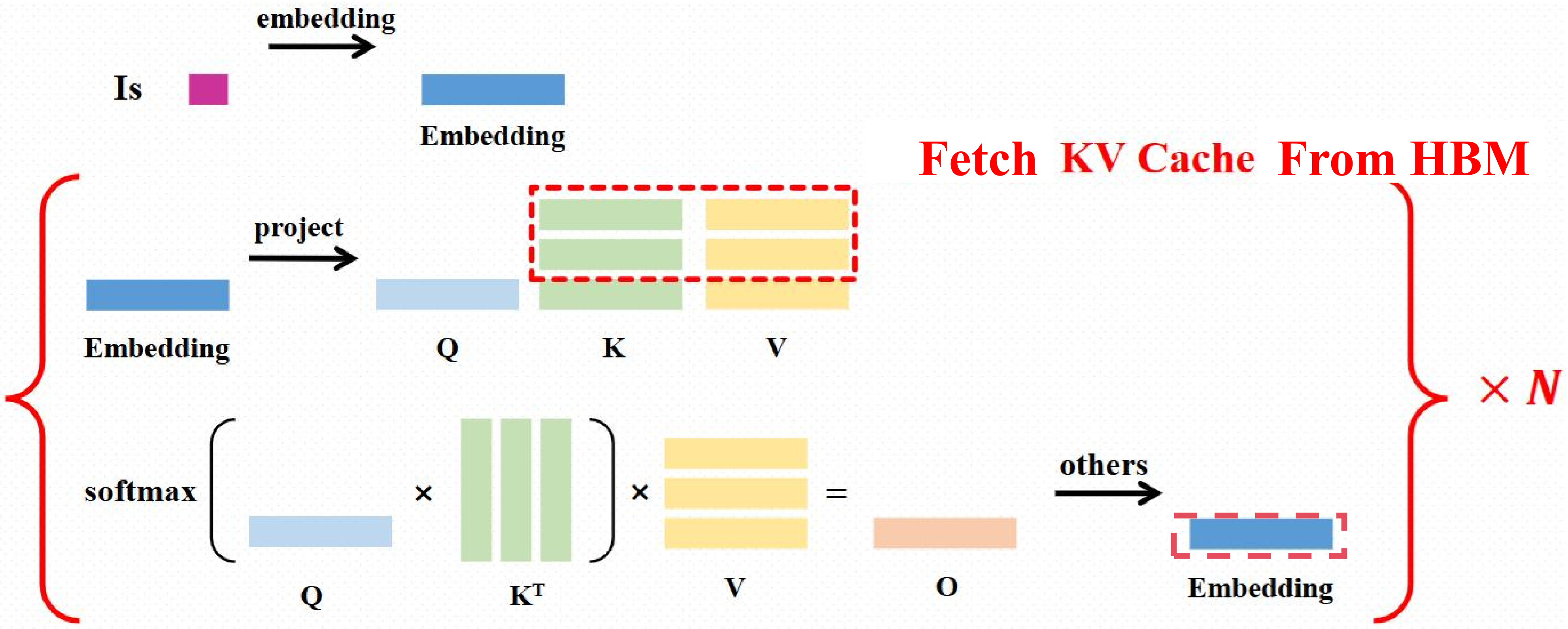
October 15

- 💡 [OSDI'24] InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management
- 👤 Ping Gong, Jiawei Yi, Juncheng Zhang
- slides, Q&A summary, video

Background: Decoding in LLM Inference

NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Decode: Fetch KV cache & generate next token



October 15

- 💡 [OSDI'24] InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management
- 👤 Ping Gong, Jiawei Yi, Juncheng Zhang
- 📕 slides, 📄 Q&A summary, 📺 video

Background: Prefill vs Decode

Prefill: generate KV cache

- Generate KV Cache
- **Compute-bound***

* For a 13B parameter LLM, processing a single prompt of 512 tokens can fully engage an A100 GPU.

Decode: generate next token

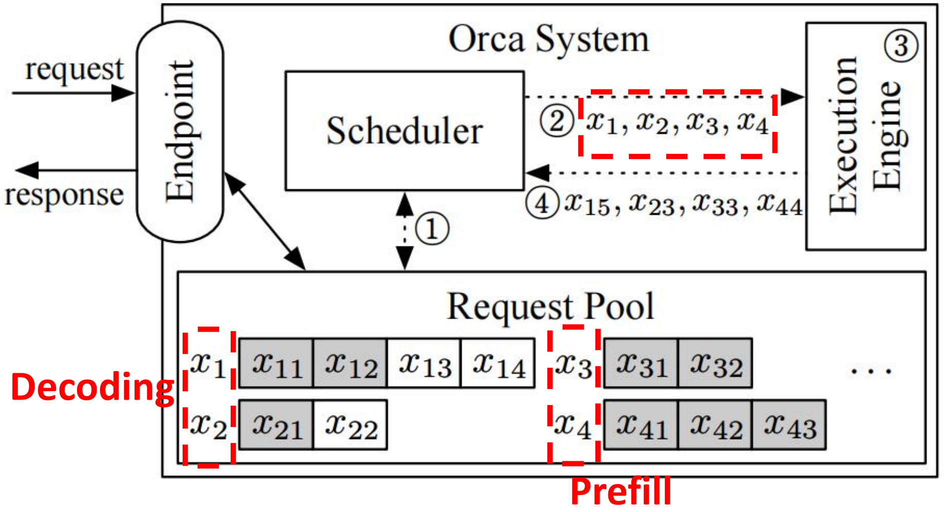
- Fetch KV Cache from HBM
- **Memory-bound**

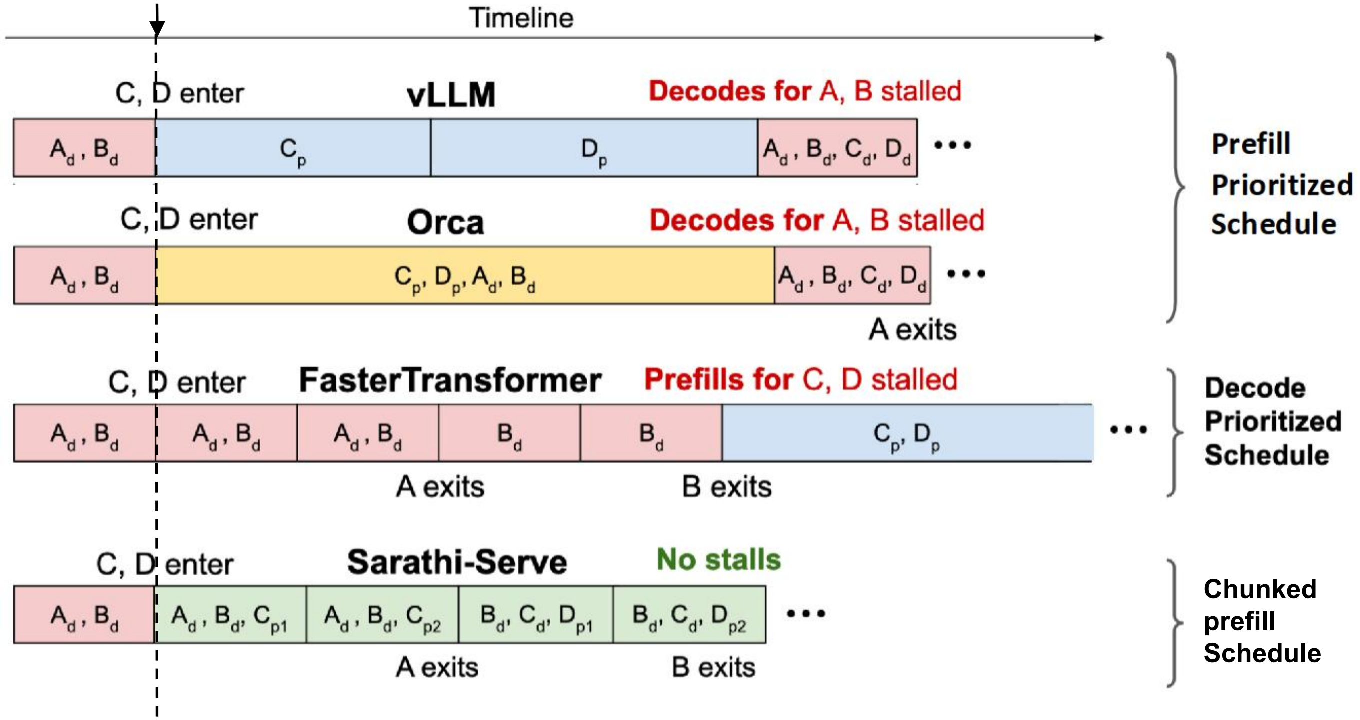
	TTFT	TPOT
Chat	Low (< 1s)	Match read speed (~ 100ms)
Search	Very Low (~ 200ms)	Match read speed (~ 100ms)
Program	Very Low (~ 200ms)	Very Low (~ 50ms)

- **Different apps have various latency requirements***

* Set the SLOs **empirically** based on their service target because there exists no available SLO settings for these applications as far as we know

Background: Batching in LLM Serving

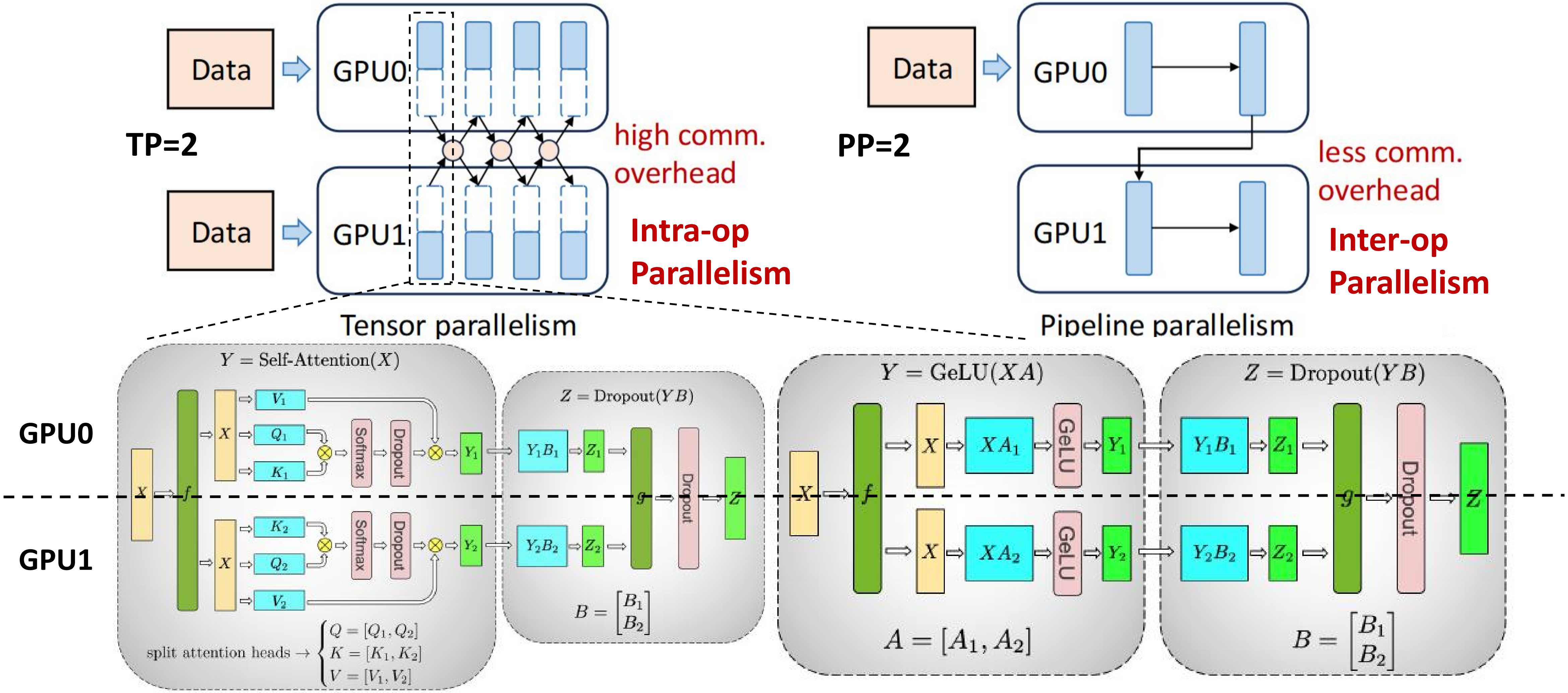




- $x_{i,j}$ is the j-th token of the i-th request
- batch size = 4 in Figure
- Shaded: input tokens received from clients
- Unshaded: generated by Execution Engine

- **However, batching the two phases make them *share the same batching strategy***
- **Sharing GPUs cause competition between prefill and decoding, which may hurt both TTFT and TPOT**

Background: Model Parallelism



- **However, batching the two phases make them *share the same parallel strategy***

Outline

- Background
- **Motivations**
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Tradeoff Analysis
- Method
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- Implementation
- Evaluation
- Discussion & Summary

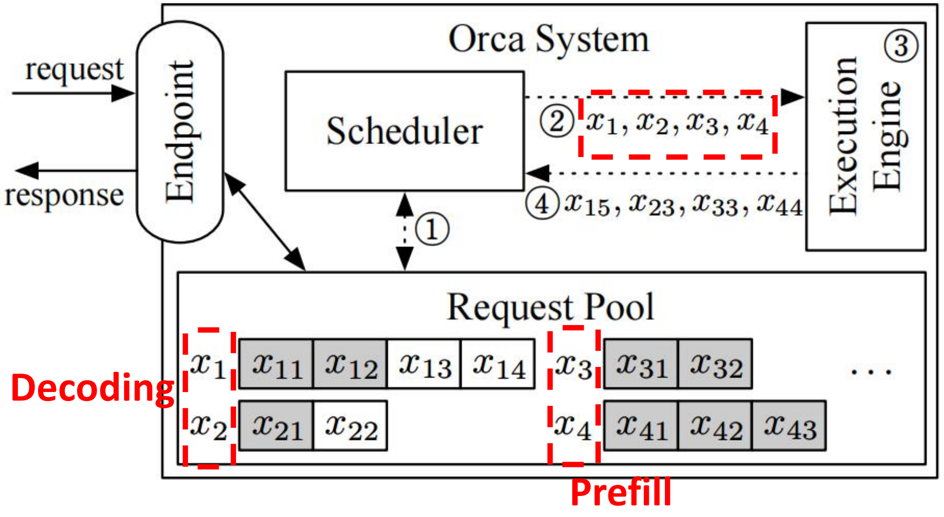
Common Challenges

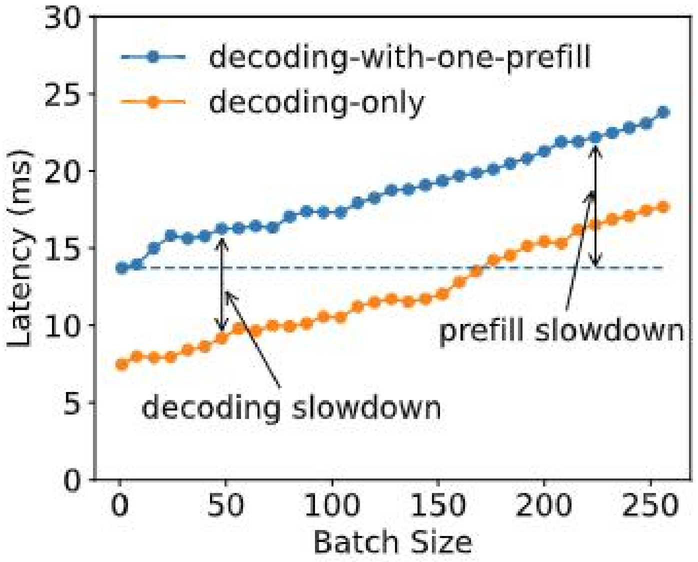
- **Different apps have various latency requirements**
- **Title: DistServe: Disaggregating Prefill and Decoding for *Goodput-optimized* Large Language Model Serving**
- **"*Goodput-optimized*" in Title: To be precise, *Per-GPU goodput*, defined as *the maximum request rate (RPS)* that can be served adhering to the SLO attainment goal (say, 90%) for *each GPU*.**
- **How to do?**

Problem 1: Prefill-Decoding Interference

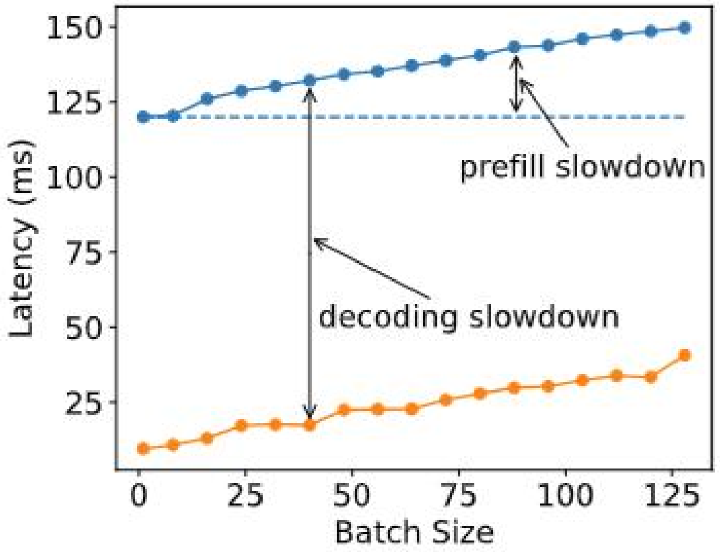
西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Batch execution time when serving a 13B LLM as batch size increases.**
- **Batching prefill and decoding phase together hurt both TTFT and TPOT.**





(a) Input length = 128

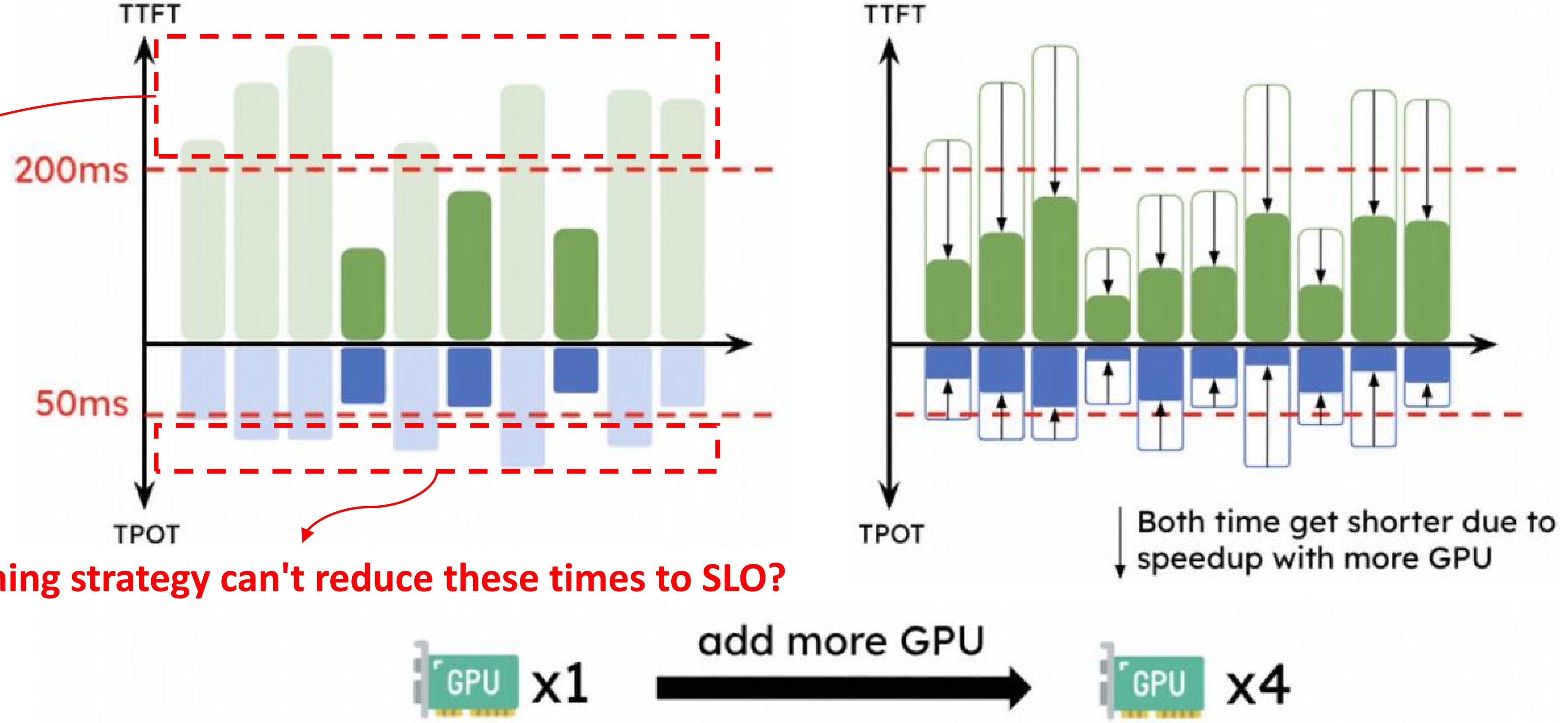


(b) Input length = 1024

Common Challenges | **Existing Solutions** | Design Intuitions | Special Challenges

Problem 2: Resource & Parallelism Coupling

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY



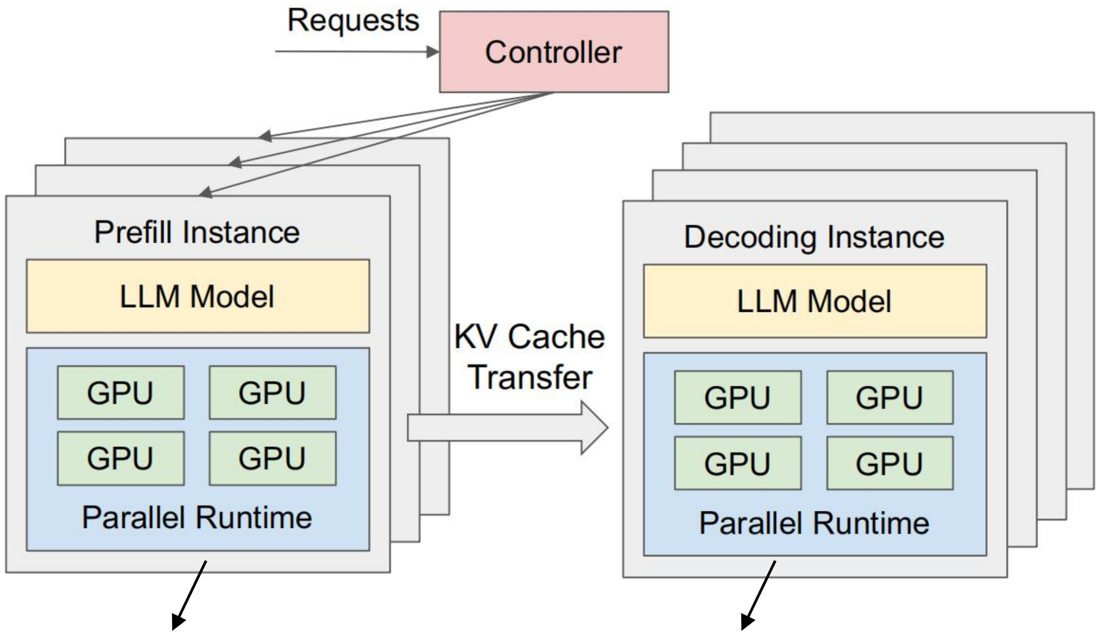
What if Batching strategy can't reduce these times to SLO?

- Batching the two phases makes them share the same parallel strategy (TP=xx, PP=xx...)
- Coupling leads to overprovision resources to meet the more demanding SLO

Opportunity: Disaggregting Prefill and Decoding

NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Prefill-Decoding interference is eliminated**
- **The term *instance*:**
  - **a unit of resources that manages exactly *one complete copy of model weights***
  - **One *instance* can correspond to many GPUs when model parallelism (TP or PP) is applied.**
  - **Repliaction: When disaggregate Prefill/Decoding phase to different GPUs, each instance manages its copy of the model weights, resulting in *prefill instances* and *decoding instances*.**
  - **M Prefill instances : N Decode instances (M >= N)**
- **Naturally divide the SLO satisfaction problem into two optimizations:**
  - **Prefill instance optimizes for TTFT.**
  - **Decoding instance optimizes for TPOT.**
  - **Choose the most suitable parallelism and resource allocation for Prefill/Decoding phase.**

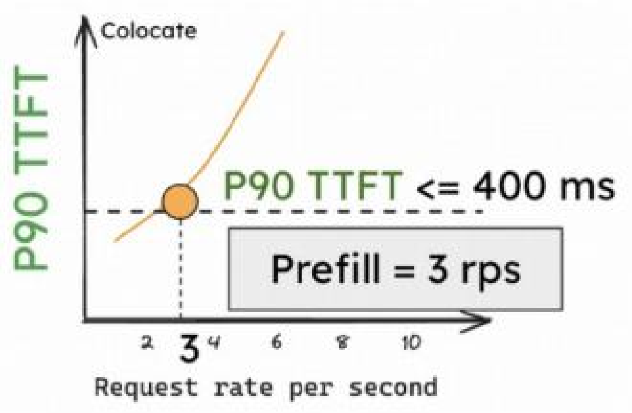


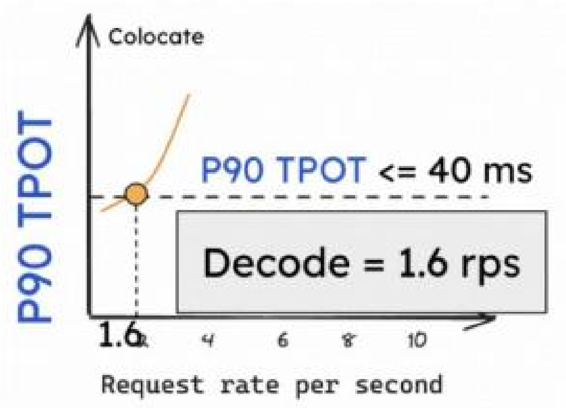
One instance with *4 GPUS* => (TP, PP) = (4, 1) or (1, 4) or (2, 2)

***M Prefill* : *N Decode* instances (batching more decodeing jobs)**

Common Challenges | Existing Solutions | **Design Intuitions** | Special Challenges

Opportunity: Disaggregting Prefill and Decoding





Colocation

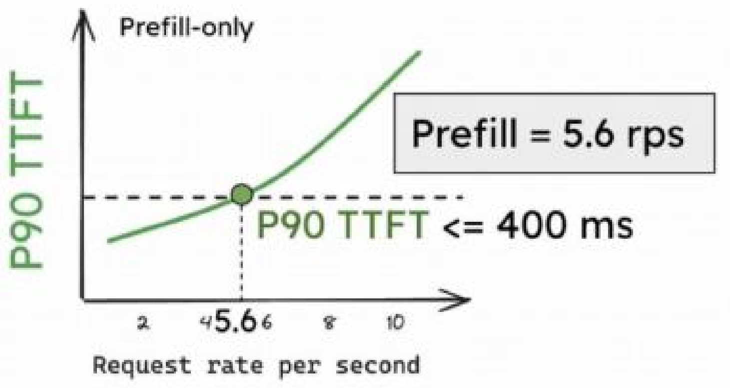
Max System rps

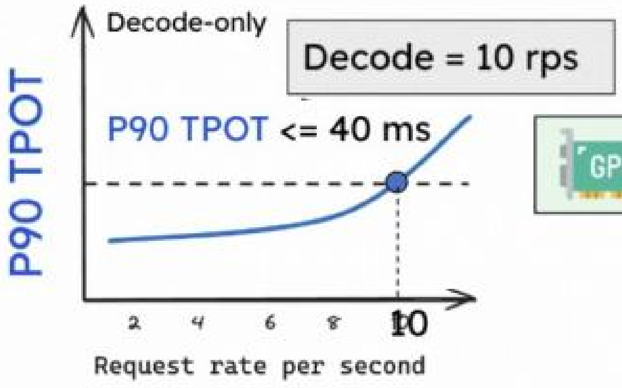
= Min(Prefill, Decode)

= 1.6 rps / GPU × 3 GPU = 5.6 rps

Compare the maximum per-GPU goodput

Disaggregting Prefill and Decoding ***Adding 2 GPUs***





Disaggregation (2P1D)

Max System rps

= Min (5.6 x **2**, 10) rps / 3 GPU

= 3.3 rps / GPU

Common Challenges > Existing Solutions > **Design Intuitions** > Special Challenges

Challenges of Disaggregation

- C1: Communication overhead for KV-Cache transmission
- C2: The optimization target, per-GPU goodput, is difficult to optimize:
  - the workload pattern
  - SLO requirements
  - parallelism strategies
  - resource allocation
  - network bandwidth
  - ...

The author calls this challenge the *Placement* problem

Outline

- Background
- Motivations
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Tradeoff Analysis**
- Method
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- Implementation
- Evaluation
- Discussion & Summary

Tradeoff Analysis: Setup

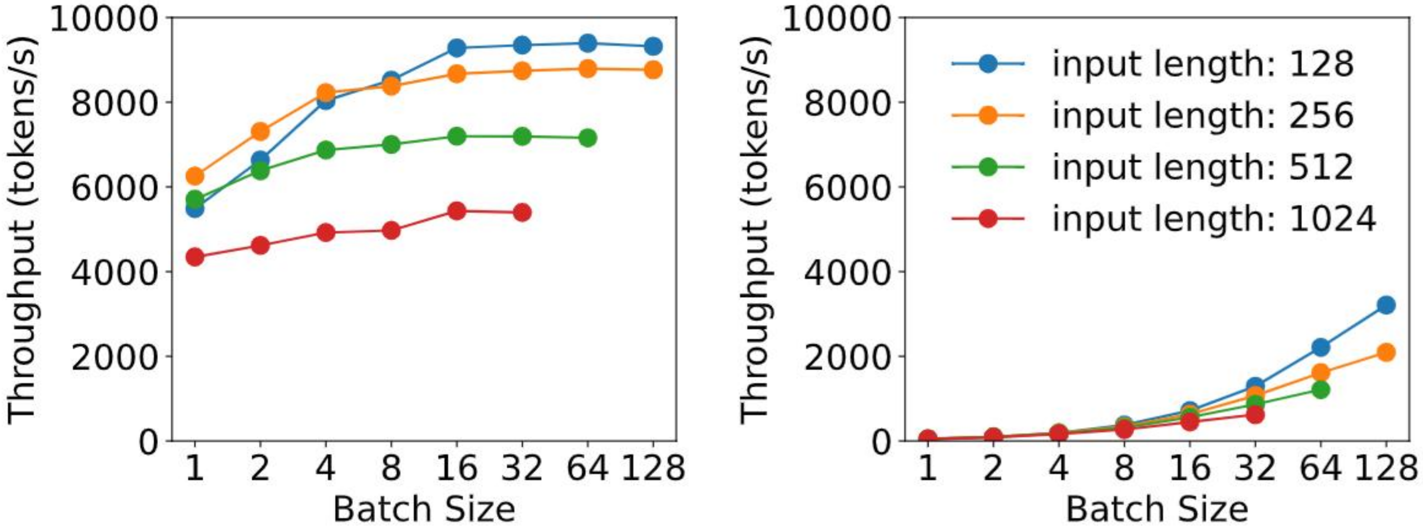
西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Analysis for Prefill Instance (Prefill-only)**
  - **1) Batching strategy: 13B Model + 1 A100-80G**
  - **2) Parallelism plan (TP/PP): 66B Model + 2 A100-80G (Why select this setting?)**
- **Analysis for Decoding Instance (Decoding-only)**
  - **1) Batching strategy: 13B Model + 1 A100-80G (Same as the Prefill)**
  - **2) Parallelism plan (TP/PP): 13B Model + 1/2/4/8 A100-80G (Counter-intuitive, because if the model can be placed on a single GPU, it is *usually* not considered to use multiple GPUs in parallel.)**
- **Some assumptions:**
  - **All prompts are of equal length**
  - **All GPUs on one machine**
  - **Which LLM Engine to test? Not vLLM**

Prefill/Decoding Instance: Batching strategy

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Profile Throughput for Prefill/Decoding phases with different *batch sizes* and *input lengths*
- Serving an LLM with 13B parameters on 1 A100-80G GPU.



(a) Prefill phase (b) Decoding phase

- The optimal batch size expected by prefill and decoding is different:
  - Prefill: Throughput growth plateaus with larger batch sizes due to *compute-bound limitations*. It is necessary to *profile* the specific LLM and GPUs to identify a critical input length threshold *L_m*.
  - Decode: Throughput increases significantly with larger batch sizes due to *memory-bound limitations*. Disaggregation enables *multiple prefill instances* to a *single decoding instance*, allows for accumulating a larger batch size on dedicated GPUs.

Prefill Instance: Parallelism Plan (1)

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **To simplify, assume uniform input length = 512 and a Poisson arrival process.**
- **Disaggregation enables the prefill phase to function analogously to an M/D/1 queue***
  - **M: Requests follow a Poisson distribution, meaning arrivals are independent and equally likely within a time unit.**
  - **D: All requests have the same prefill processing time.**
  - **1: Assume only one GPU is available.**
  - **R: the Poisson arrival rate**
  - **Avg_TTFT = the time a single request is processed + the time the request *waits in the queue***
    = the time a single request is processed + (the number of requests *before this request* * the time a single request is processed)

*Use *queuing theory* to verify the *observation* (next slide). Since one request saturates the GPU, schedule requests via FCFS *without batching*

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

the time a single request is processed

$$\frac{RD}{2(1-RD)} \cdot D$$

the number of requests *before this request*

PP = 2

$$Avg\_TTFT_{inter} = D_s + \frac{RD_m^2}{2(1-RD_m)} = D + \frac{RD^2}{4(2-RD)}. \quad D \approx D_s \approx 2 \times D_m$$

the request-level latency

the time the slowest stage takes

TP = 2

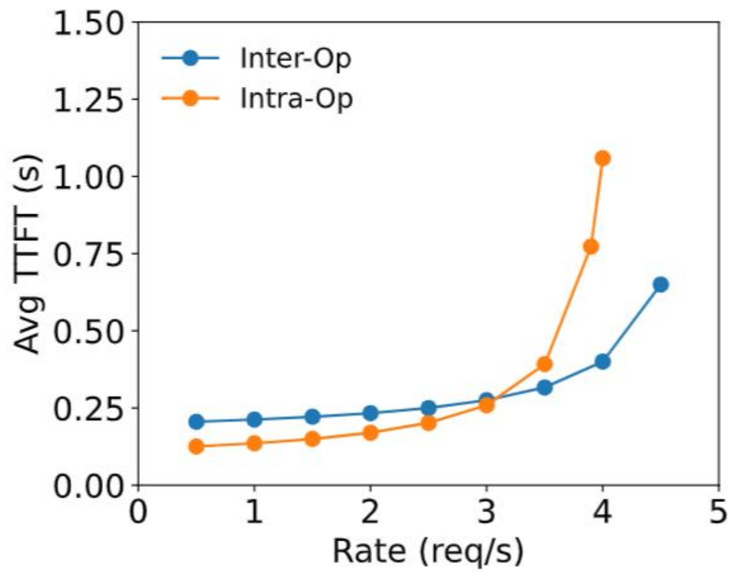
$$Avg\_TTFT_{intra} = \frac{D}{K} + \frac{RD^2}{2K(K-RD)}. \quad 1 < K < 2$$

- **K: depends on the input length, model architecture, communication bandwidth, and *placement*...**

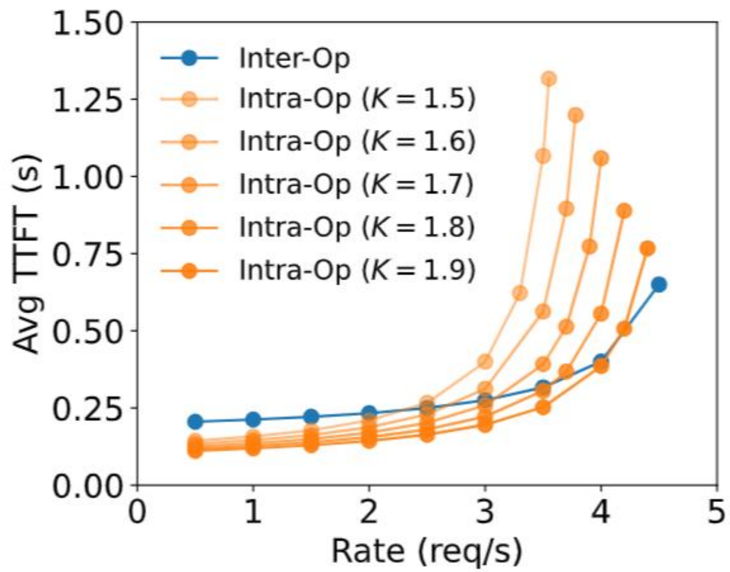
Prefill Instance: Parallelism Plan (2)

NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Profile *Average TTFT* when serving a 66B LLM (input length = 512, *without batching*) using different parallelism on two A100 GPUs (TP=2 vs PP=2)**
- **Observation (use *queuing theory* to verify):**
  - **When RPS is small, TP is more suitable. Since each request's execution time (first term) is dominated.**
  - **When RPS is large, PP is more suitable. Since the queue delay (second term) is dominated.**
  - **TTFT is also influenced by the speedup coefficient *K* (1 < *K* < TP=xx).**



(a) Real experiment results



(b) Changing intra-op speedup

the queue delay

PP = 2

$$Avg\_TTFT_{inter} = D + \frac{RD^2}{4(2-RD)}.$$

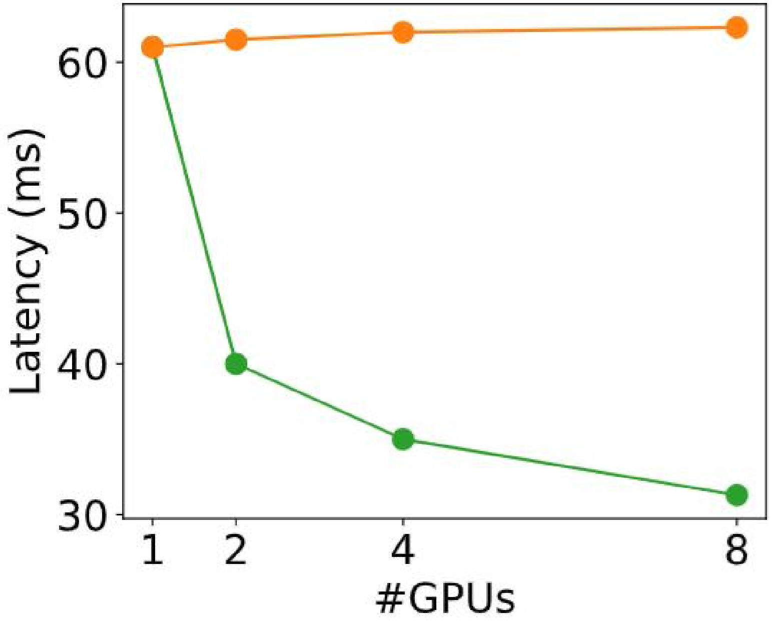
TP = 2

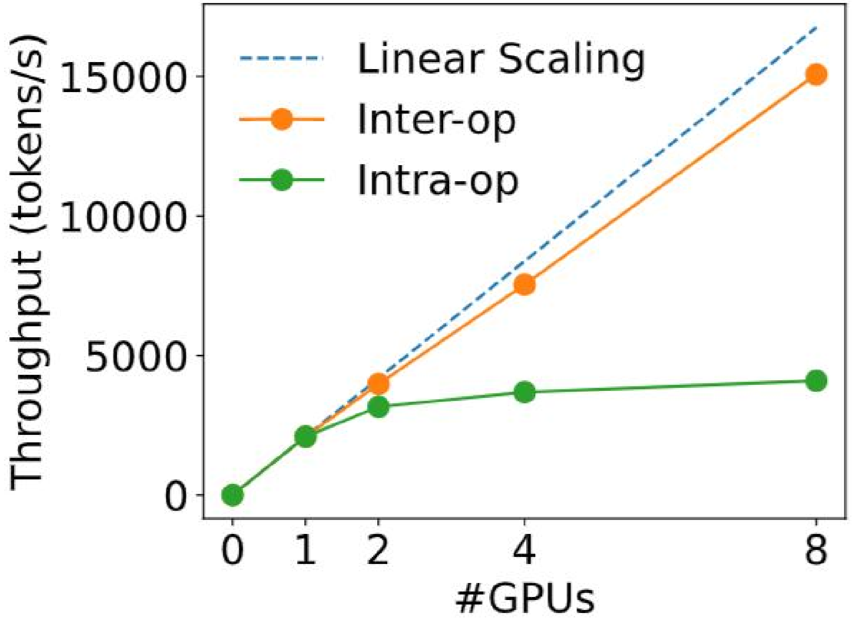
$$Avg\_TTFT_{intra} = \frac{D}{K} + \frac{RD^2}{2K(K-RD)}, \quad 1 < K < 2$$

each request's execution time

Decoding Instance: Parallelism Plan

- As the decoding *batch size continue to increase* to *approach the compute-bound*, the decoding computation begins to resemble the prefill phase.
- Profile Decoding phase *latency* and *throughput* when serving a 13B LLM with *batch size = 128* and *input length = 256* under *different parallel degrees* (TP=xx vs PP=xx).
- Observation:
  - We hope to see that increasing the number of GPUs can bring *linear improvements*. However, TP *cannot* bring linearity to Lantecy or Thpt.
  - Despite this, when the TPOT SLO is stringent, TP is essential to reduce TPOT to meet.
  - PP can bring linearity to Thpt. This is of great value for optimizing Decoding.





Practical Problems

- **Variable prefill length.**
  - **In real deployments, the lengths of requests are non-uniform. This can cause pipeline bubbles for prefill instances applying PP.**
  - **Develop a simple scheduling to reduce pipeline bubbles.**
- **Communication overhead.**
  - **The KV cache size of *a single 512-token request* on OPT-66B is approximately 1.13GB. Assuming an average arrival rate of 10 RPS, it needs to transfer 1.13GB×10=11.3GB data per second—or equivalently *90Gbps bandwidth* to render the overhead invisible.**
  - **Many modern GPU clusters for LLMs, equipped with *cross-node InfiniBand* (e.g., 800 Gbps), can effectively *hide* these communication overheads.**
  - **If cross-node bandwidth is limited, DistServe relies on the commonly available *intra-node NVLINK*, where the peak bandwidth between A100 GPUs is 600 GB/s, again rendering the transmission overhead *negligible*.**
  - **Solving the *placement* problem can reduce communication overhead.**

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Background
- Motivations
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Tradeoff Analysis
- **Method**
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- Implementation
- Evaluation
- Discussion & Summary

DistServe Overview

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Definition of *Placement*:**
  - **1) parallelism strategy for prefill/decoding instance**
  - **2) the number of each instance to deploy (repliactions)**
  - **3) how to place them onto the physical cluster**
  - **Goal: find a *placement* that maximizes the per-gpu goodput**
- **Algorithm Sketch:**
  - **Step 1: Use simulation to measure the goodput for all parallelism config.**
  - **Step 2: Obtain the optimal parallelism config for Prefill/Decoding phase.**
  - **Step 3: Use replication to match the overall traffic.**
- **Alg. 1: Placement for *High* Node-Affinity Cluster**
  - **Assume nodes are connected with *high bandwidth network*, e.g., InfiniBand.**
  - **The communication overhead between nodes is negligible. (We can deploy prefill and decoding instances *across any two nodes* without constraints)**
- **Alg. 2: Placement for *Low* Node-Affinity Cluster**
  - **Assume GPUs inside one node are connected with *NVLINK*.**
  - **The communication overhead within the node is negligible. (Require the same stage of prefill/decoding instances to be on the same node)**

Alg. 1 Placement for *High* Node-Affinity Cluster

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

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*.

①
$config_p, config_d \leftarrow \emptyset, \emptyset$
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**
      $config \leftarrow (inter\_op, intra\_op)$
      $\hat{G} \leftarrow \text{parallel}(G, config)$

②
      $config.goodput \leftarrow \text{simu\_prefill}(\hat{G}, W)$
      if $\frac{config_p.goodput}{config_p.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_p \leftarrow config$ TP_Prefill=xx, PP_Prefill=xx
      $config.goodput \leftarrow \text{simu\_decode}(\hat{G}, W)$
      if $\frac{config_d.goodput}{config_d.num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
        $config_d \leftarrow config$ TP_Decode=xx, PP_Decode=xx

③
$n, m \leftarrow \lceil \frac{R}{config_p.goodput} \rceil, \lceil \frac{R}{config_d.goodput} \rceil$
$best\_plm \leftarrow (n, config_p, m, config_d)$
return *best_plm*

- **Algorithm Sketch:**
  - ① **Enumerating the search space for the best_plm**
  - ② **Use simulation and profiling to obtain the optimal parallelism config**
  - ③ **Use replication to match the overall traffic**
- **Simulator building*:**
  - **Define Goodput Range:** Start with a range of possible goodput values (e.g., goodput = 5 means RPS is between 0 and 5).
  - **Simulate Load:** Send ***simulated requests*** at different goodput (RPS) values to the prefill/decode instance, using the current parallel strategy (e.g., TP=xx, PP=xx), and measure P90 TTFT&TPOT.
  - **Compare with SLO:** Compare the measured P90 TTFT&TPOT with the SLO. If TTFT&TPOT < SLO, increase RPS; otherwise, decrease it.
  - **Binary Search for Optimal Goodput:** Use binary search to adjust the RPS bounds based on the comparison, iteratively finding the final goodput.

* How to build an accurate simulator, see *Appendix A* of the original article.

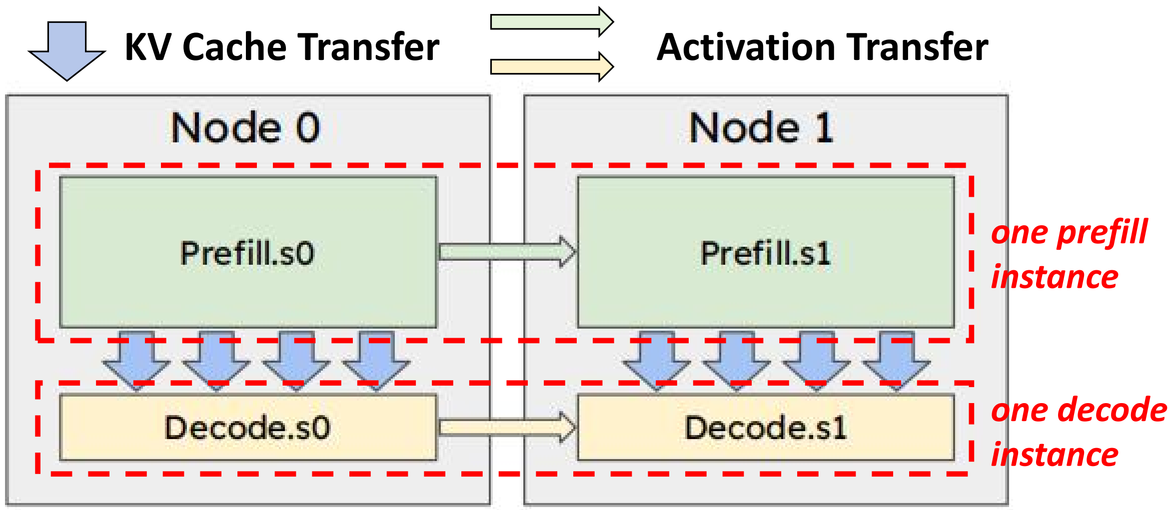
Alg. 2 Placement for *Low* Node-Affinity Cluster

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 $best\_plm$.

①
$config^* \leftarrow \emptyset$
for $inter\_op \in \{1,2,...,N\}$ **do** PP=xx
  $\mathcal{P} \leftarrow$ get_intra_node_configs$(G,M,C,inter_op)$
  for $P_p \in \mathcal{P}$ **do** TP_Prefill=xx
    for $P_d \in \mathcal{P}$ **do** TP_Decode=xx
      if $P_p.num\_gpus + P_d.num\_gpus \leq M$ **then**
        $config \leftarrow (inter\_op, P_p, P_d)$
        $\hat{G}_p, \hat{G}_d \leftarrow$ parallel$(G, config)$
②
        $config.goodput \leftarrow$ simulate$(\hat{G}_p, \hat{G}_d, W)$
        if $\frac{config.^*goodput}{config.^*num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
          $config^* \leftarrow config$
③
$n \leftarrow \lceil \frac{R}{config.^*goodput} \rceil$
$best\_plm \leftarrow (n, config^*)$
return $best\_plm$

- **Difference between Alg. 1:**
  - Add the constraint to require *the same stage* of prefill/decoding instances to be *on the same node* (which can eliminate the communication overhead)
  - PP_Prefill = PP_Decode



KV-Cache Transfer only happens between *the same layer*.

Online Scheduling Optimization

NORTHWESTERN POLYTECHNICAL UNIVERSITY

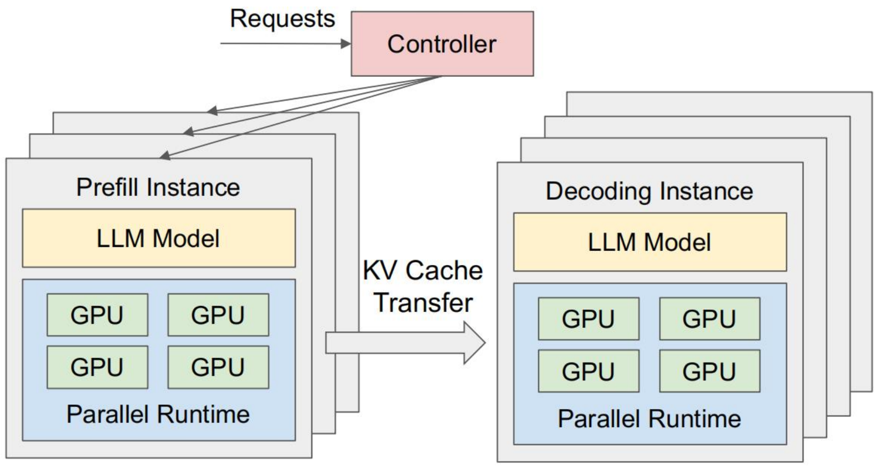
- **Scheduling to reduce pipeline bubbles.**
  - For prefill, *profile* the model and GPU to figure out *the shortest prompt length L_m* needed to *saturate the GPU*. Then schedule batches with *a total sequence length* close to *L_m*.
  - For decoding, set *L_m* as the largest batch size.
- **Combat workload burstiness.**
  - Decoding instances fetch KV cache from prefill instances as needed, using the GPU memory of prefill instances as a queuing buffer.
- **Periodic replaning.**
  - A workload profiler monitors key parameters.
  - If a workload pattern shift is detected, DistServe will trigger a rerun of the placement algorithm based on recent historical data.

Outline

- **Background**
- **Motivations**
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Tradeoff Analysis**
- **Method**
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- **Implementation**
- **Evaluation**
- **Discussion & Summary**

Implementation

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY



- a placement algorithm module (Python)
  - implements the algorithm
  - implements the simulator
  - placement decision for a specific model & cluster
- a RESTful API frontend (Python)
  - an OpenAI API-compatible interface
- an orchestration layer (Python)
  - manages the prefill and decoding instances (parallel execution engine)
  - responsible for request dispatching, KV cache transmission, and results delivery
  - NCCL for cross-node GPU communication
  - asynchronous CudaMemcpy for intra-node communication
- a parallel execution engine: 8.1K lines of C++/CUDA (similar to vLLM Engine)
  - Each instance is powered by a parallel execution engine
  - Ray actor to implement GPU workers that execute the LLM inference and manage the KV Cache
  - Integrates many LLM optimizations: continuous batching, FlashAttention, PagedAttention

Outline

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

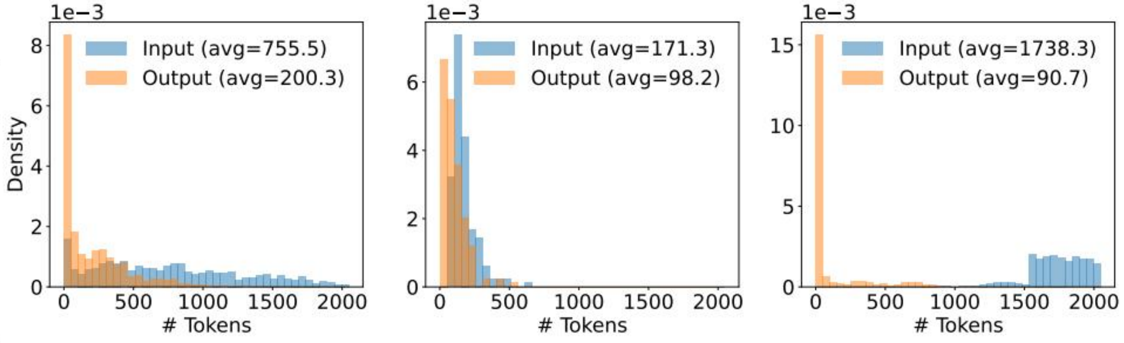
- Background
- Motivations
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Tradeoff Analysis
- Method
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- Implementation
- **Evaluation**
- Discussion & Summary

Evaluation: Setup

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Test bed:** 4 GPU server with {8 A100-80GB GPUs/NVLINK} connected with 25Gbps cross-node network (Most experiments used **one GPU Server** and evaluate **Algorithm 2)**
- **Model: OPT-13B/66B/175B**
- **Workloads:** 3 apps with setting the SLOs ***empirically*** & All the datasets do not include timestamps, generate request arrival times using ***Poisson distribution***.

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]



(a) ShareGPT (b) HumanEval (c) LongBench

- **Metric: SLO Attainment**
- **Baseline:**
  - **vLLM** - supports *continuous batching* and *paged-attention*
  - **DeepSpeed-MII** - supports *chunked-prefill*

Evaluation 1: End-to-end Experiments

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

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]

- **The parallelism strategies chosen by DistServe in the end-to-end experiments.**

Model	Dataset	Prefill		Decoding	
		TP	PP	TP	PP
OPT-13B	ShareGPT	2	1	1	1
OPT-66B	ShareGPT	4	1	2	2
OPT-66B	LongBench	4	1	2	2
OPT-66B	HumanEval	4	1	2	2
OPT-175B	ShareGPT	3	3	4	3

Total 2×1+1×1 = 3 GPUs

Total 4×1+2×2 = 8 GPUs

Total 3×3+4×3 = 21 GPUs

Compare the maximum *per-GPU goodput*

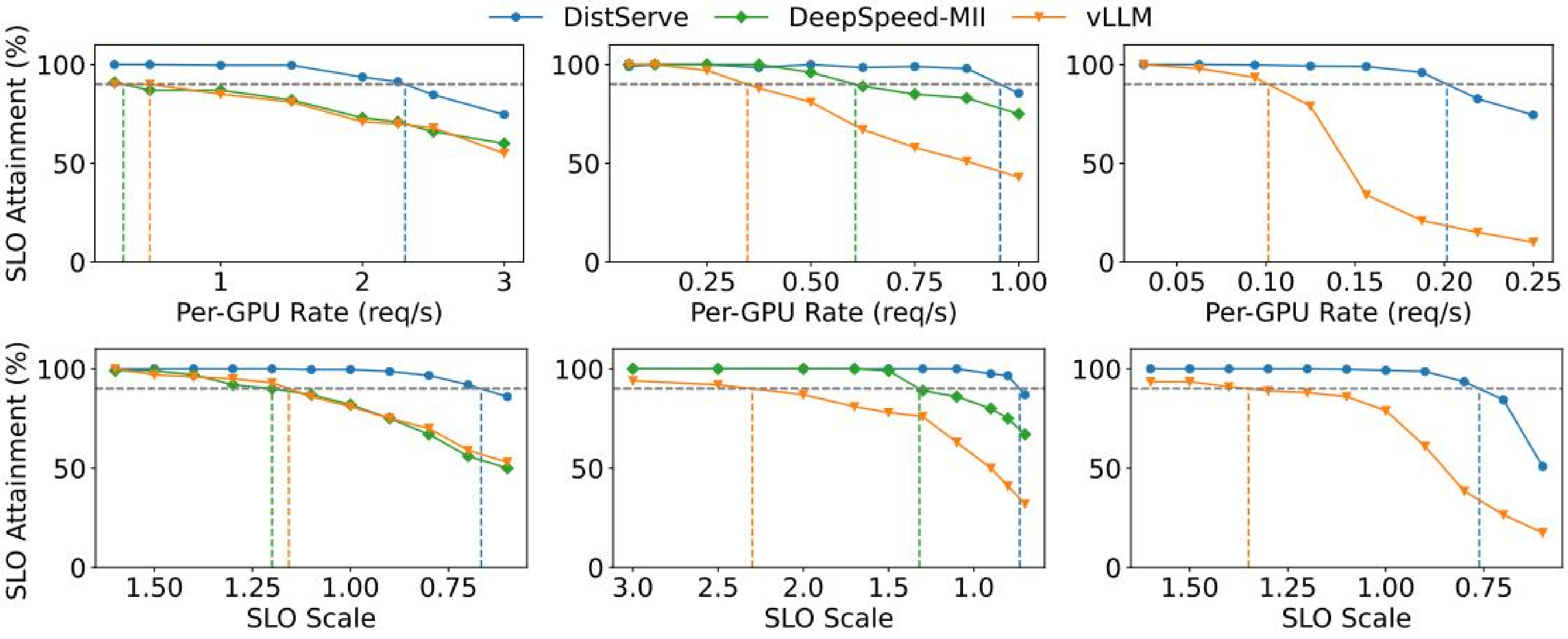
Ignore GPU cost/Model Replication cost?

- **vLLM: Since vLLM only supports TP, we follow previous work to set TP equals 1, 4, 8 for OPT 13B/66B/175B.**
- **DeepSpeed-MII: We set its TP *the same as vLLM* for OPT-13B and OPT-66B for a fair comparison. DeepSpeed-MII does not support 175B.**

Evaluation 1: End-to-end Experiments

NORTHWESTERN POLYTECHNICAL UNIVERSITY

- ***Chatbot application*** **with OPT models on the ShareGPT dataset.**
  - **1st row: SLO attainment of *90%* (the vertical lines) to observe the maximum per-GPU goodput**
  - **2nd row: vary the SLO latency requirements to observe how the SLO attainment changes. ("We fix the rate and then linearly scale the TTFT/TPOT latency requirements", RPS=?)**

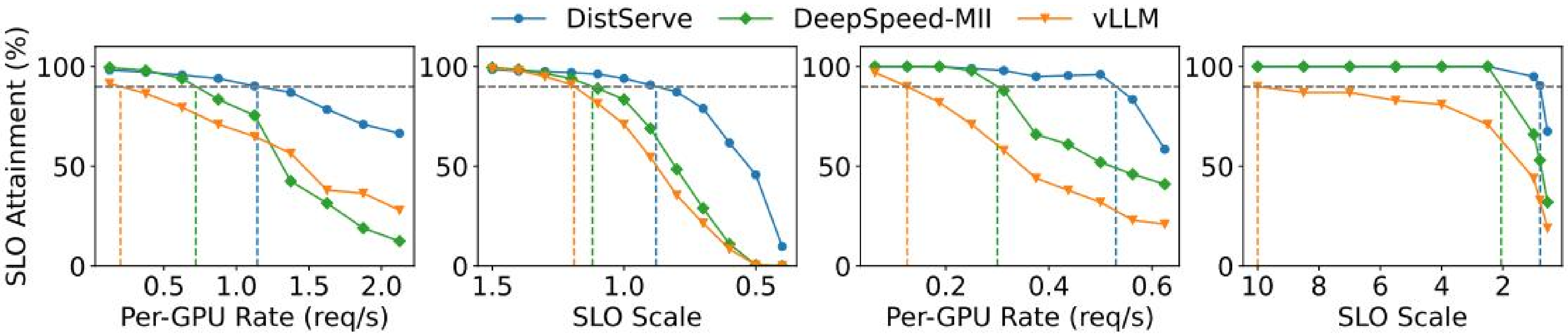


(a) OPT-13B (b) OPT-66B (C) OPT-175B

Evaluation 1: End-to-end Experiments

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

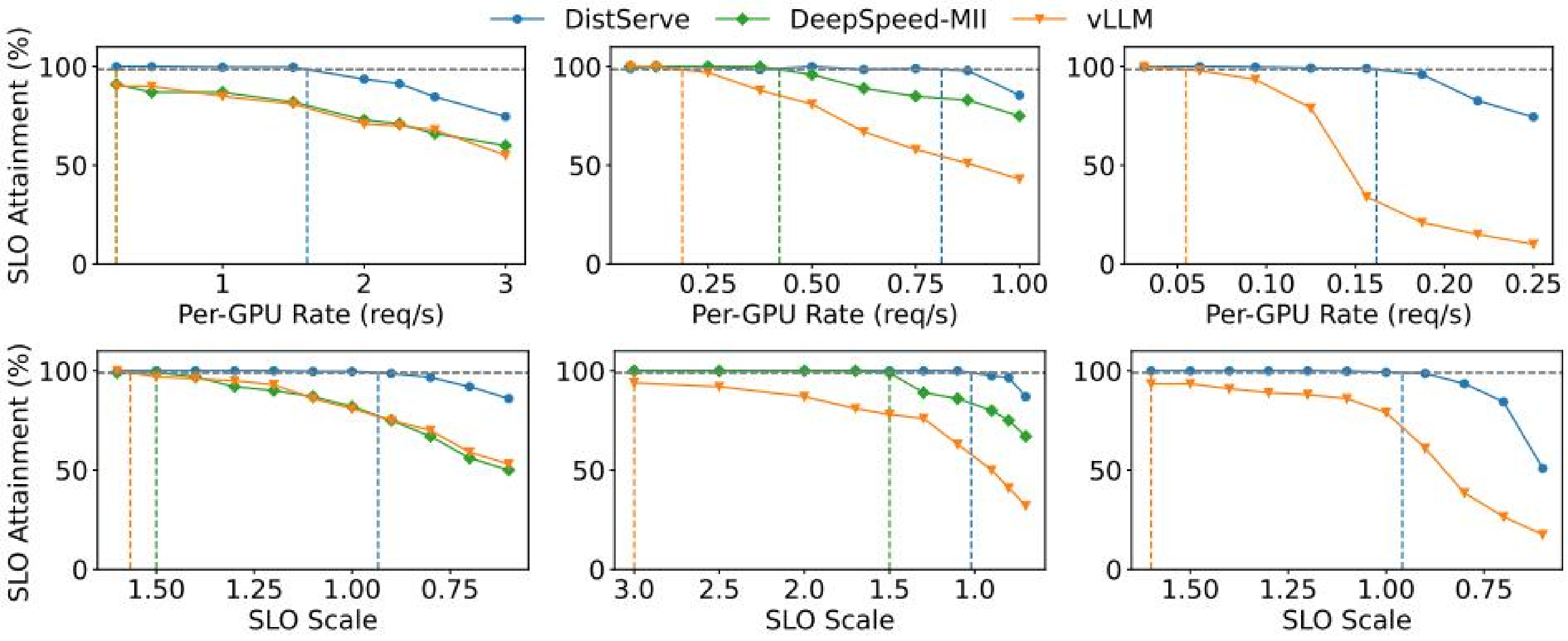
- ***Code completion*** **and** ***summarization tasks*** **with OPT-66B on HumanEval and LongBench datasets, respectively.**
- **The results is similar to Chatbot application.**



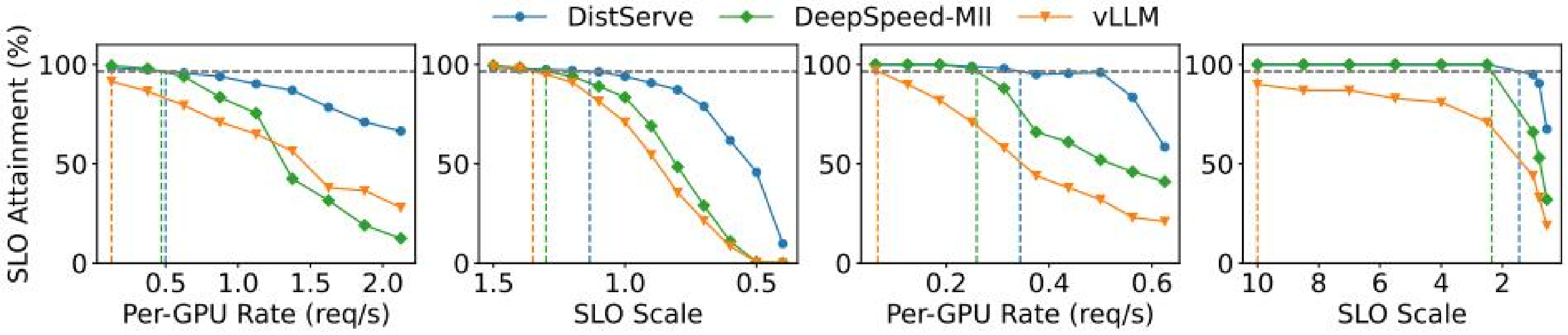
(a) Code Completion (b) Summarization

Evaluation 1: End-to-end (99% SLO)

- **Similar to 90% SLO attainment**



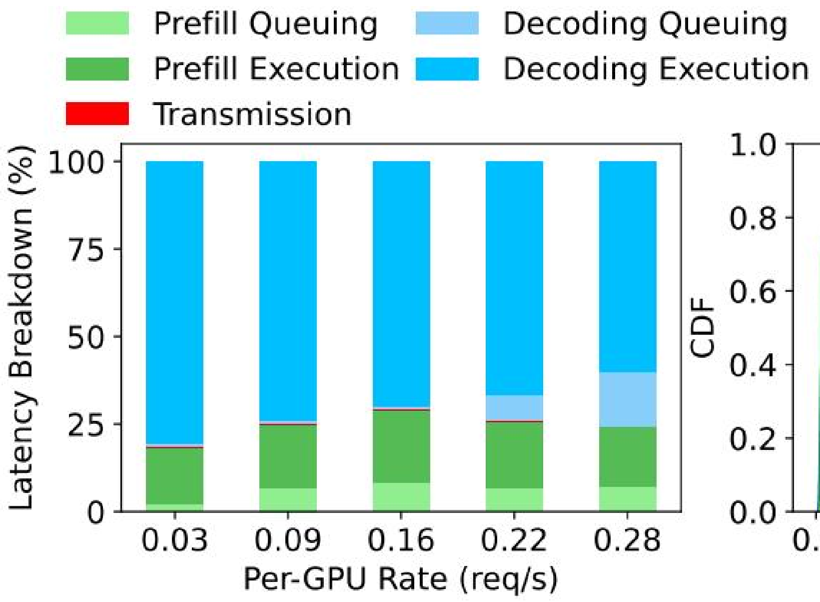
(a) OPT-13B (b) OPT-66B (C) OPT-175B

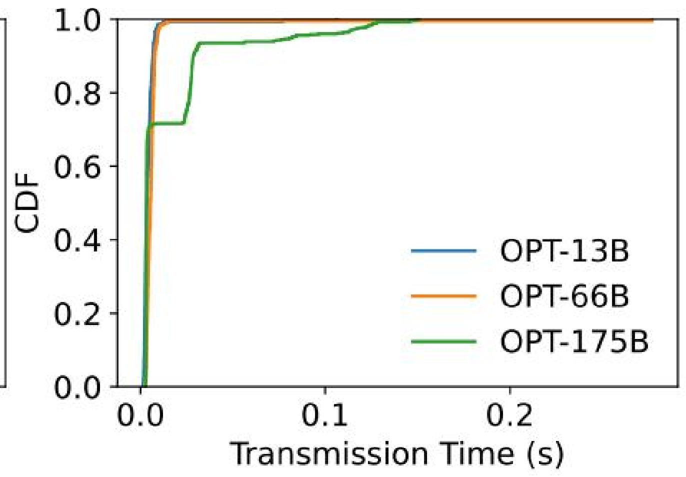


(a) Code Completion (b) Summarization

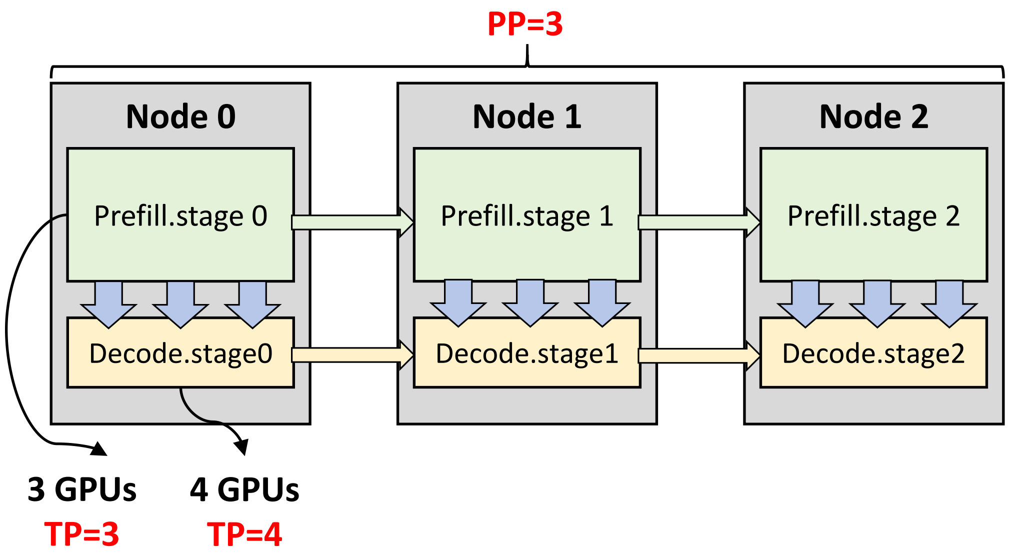
Evaluation 2: Latency Breakdown

- **Divide the processing lifecycle of a request in DistServe into *five* stages: prefill queuing, prefill execution, KV Cache transmission, decoding queuing, and decoding execution.**
- **Left: Latency breakdown with OPT-175B on ShareGPT dataset with DistServe (Alg. 2).**
- **Right: The CDF function of *KV Cache transmission* time for three OPT models (Alg. 2).**





Model	Dataset	Prefill		Decoding	
		TP	PP	TP	PP
OPT-175B	ShareGPT	3	3	4	3

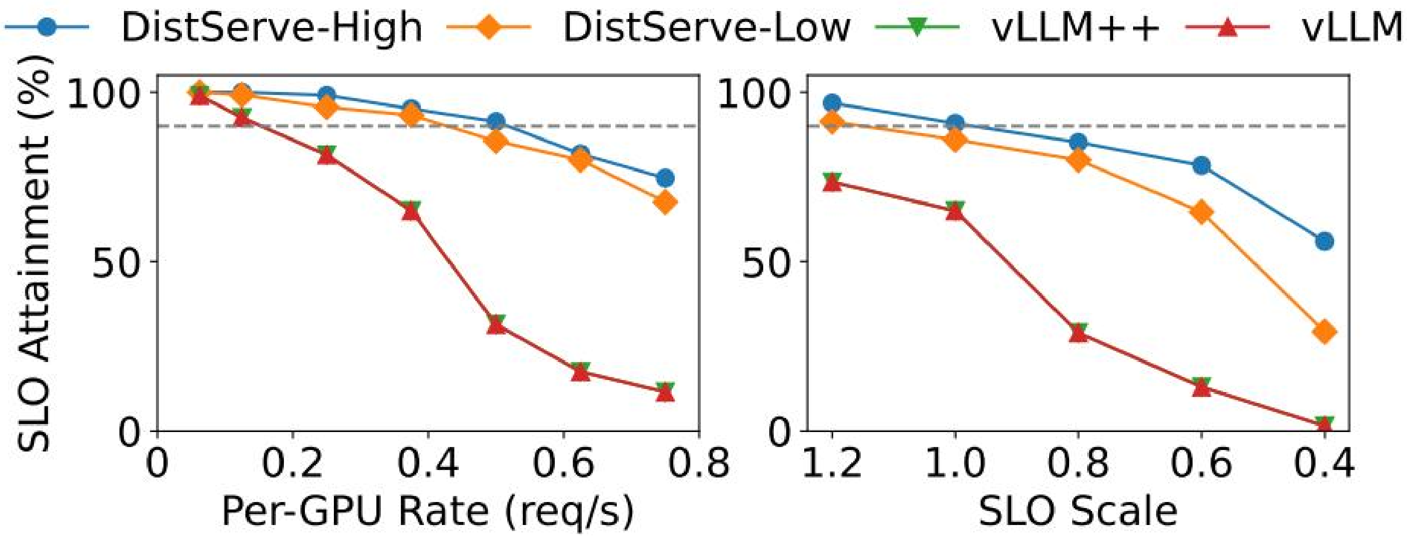


Evaluation 3: Ablation Studies

- **Baseline:**
  - **vLLM: The default parallelism setting**
  - **vLLM++: enumerates different parallelism strategies and chooses the best. (Simulations)**
  - **DistServe-Low: the placement found by Alg. 2**
  - **DistServe-High: the placement found by Alg. 1 which has *fewer searching constraints* and *assumes high cross-node bandwidth*. (Simulations)**
- **OPT-66B on the ShareGPT dataset**

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%

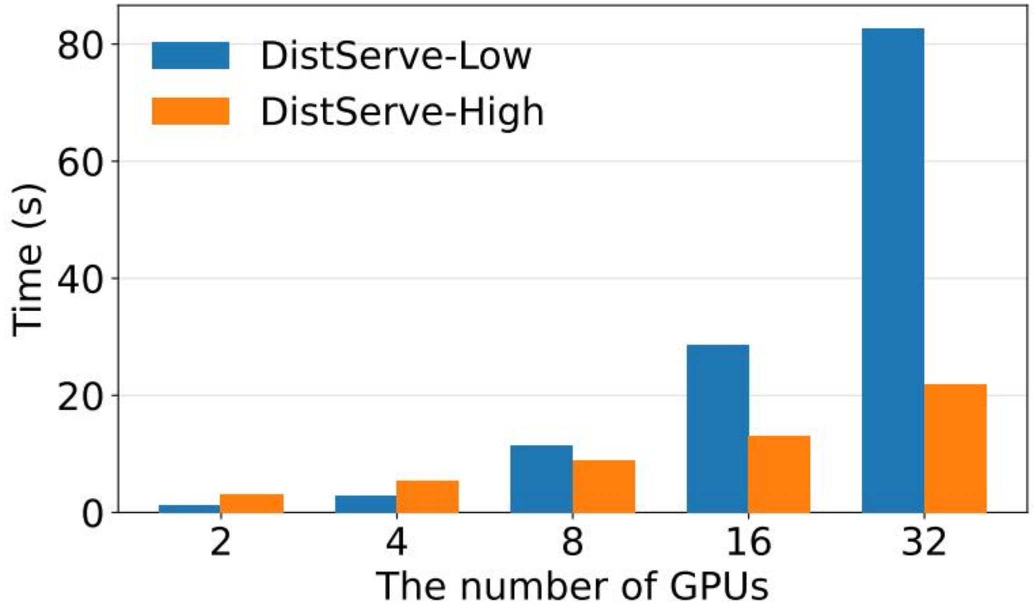
- **Comparison of the *SLO attainment* reported by the simulator and the real system.**



Evaluation 4: Algorithm Running Time

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Alg. 1 (DistServe-Low) and Alg. 2 (DistServe-High) on an AWS m5d.metal instance (VM) as the number of GPUs (N ×M) to a single instance (VM) increases.**
- **The execution time of "Dist-Low" becomes higher than that of "Dist-High":**
  - **the search for parallelism strategies in "Dist-High" is independent and can be parallelized.**
  - **For "Dist-Low", due to additional restrictions on deployment, we need to enumerate all the possible intra-node parallelism combinations for prefill and decoding instances.**



- **Even so, the execution time of the algorithm is *in minutes*, and since it only needs to be executed once before each redeployment, this overhead is acceptable.**

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Background
- Motivations
  - (Common) Challenges
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Tradeoff Analysis
- Method
  - Placement for High Node-Affinity Cluster
  - Placement for Low Node-Affinity Cluster
  - Online scheduling
- Implementation
- Evaluation
- **Discussion & Summary**

Discussion

- **DistServe:**
  - **Throughput-optimized scenarios.**
  - **Resource-constrained scenarios.**
  - **Long-context scenarios -> LoongServe@SOSP24**

December 03

- [SOSP'24] LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism
- Zewen Jin, Hongrui Zhan, Shen Fu
- slides, Q&A summary, video

- **My personal opinion on the advantages of PD disaggregation:**
  - **3-Level disaggregation. DistServe indicates Level 1 and Level 2.**
    - **Level 1: Disaggregate P and D *within one node* (homogeneous devices) and *intra-node interconnect* (e.g., both P and D on the same A800 node) without cluster-level modifications.**
    - **Level 2: Disaggregate P and D *across nodes* (homogeneous devices) and different networks (e.g., P and D on separate A800 nodes) with efficient inter-cluster communication, such as RDMA.**
    - **Level 3: Disaggregate P and D across *heterogeneous devices* and networks (e.g., P on an MI300 cluster, D on an H20 cluster) requiring significant cluster modifications and high-performance inter-node communication.**
  - **Resource asymmetry (Similar to the insights of *Disaggregated Memory?*)**
    - **P clusters and D clusters can use different batching methods and optimal parallel configurations**
    - **How to manage the KV Cache Transfering bewteen P instances and D instances?**
    - **How to manage the workflow between P instances and D instances?**

Summary

NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Pros
  - A good explanation of "Why use a PD disaggregation architecture"
  - Solved the conflict of TTFT and TPOT
- Cons
  - Deals with the sub-problem of the PD disaggregation problem: how to find the optimal parallel configuration for P instances and D instances respectively.
  - Communication Overhead is not adequately solved.
  - Just compare the maximum per-GPU goodput, it seems that ignore GPU# cost/Model Replication cost.

Thank you!