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

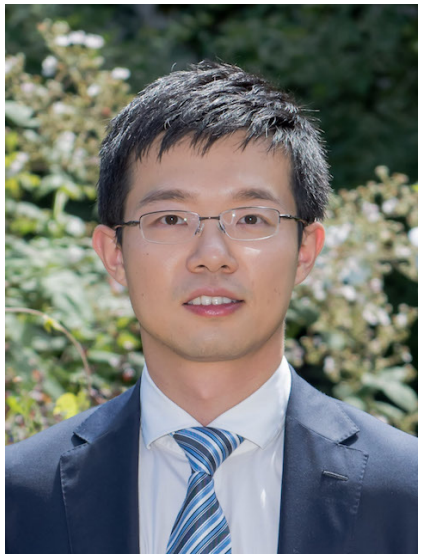
OSDI'24

Authors: Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, Hao Zhang

Presenter: Yitao Wang

2025.3.7

Author / Team Introduce



PKU

https://xinjin.github.io/

Projects:

- Serverless Computing
- Training and Serving Large Language Models
- Network Testing and Verification
- Disaggregated Storage with RDMA and DPUs

Papers:

- [SOSP'24]LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism
- [OSDI'24]dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving
- [SIGCOMM'23]Ditto: Efficient Serverless Analytics with Elastic Parallelism
- [NSDI'23]Transparent GPU Sharing in Container Clouds for Deep Learning Workloads

Author / Team Introduce

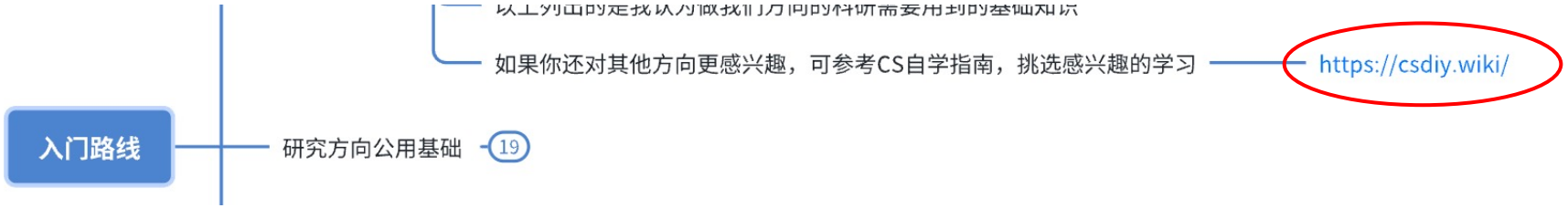
PKU

https://yinminzhong.com/

Papers:

- [NSDI'24]MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs
- [OSDI'24]AlphaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving
- [OSDI'24]DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving
- [arXiv'23]Fast Distributed Inference Serving for Large Language Models

Author / Team Introduce



Yinmin Zhong
PKUFlyingPig

Unfollow

CS Ph.D @ PKU; Machine Learning Systems; Distributed Systems;

8k followers · **33** following

PKUFlyingPig / README.md

Hi there 👋

Yinmin Zhong's GitHub Stats

Total Stars Earned:	69.9k
Total Commits (2025):	56
Total PRs:	67
Total Issues:	44
Contributed to (last year):	3

A–

Pinned

cs-self-learning Public
计算机自学指南
HTML ☆ 60.8k ⑂ 7.1k

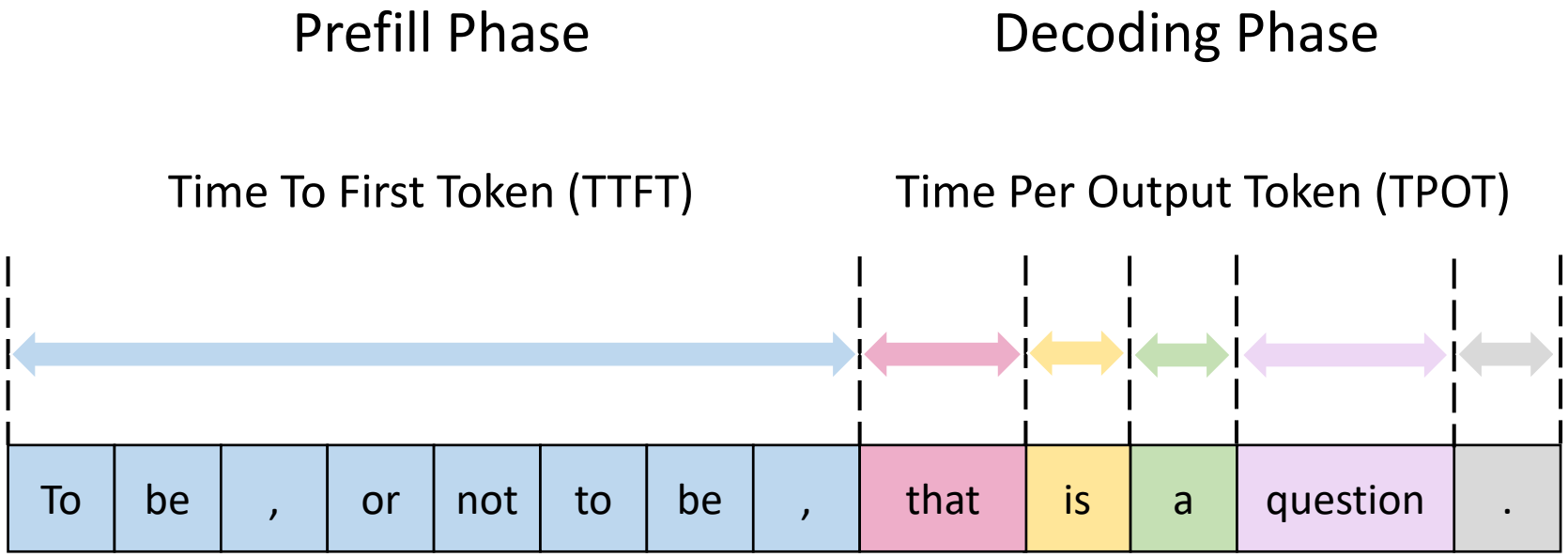
MIT6.S081-2020fall Public

Contents

- **Background**
- Problem
- Opportunity
- Challenge
- Method
- Evaluation
- Thinking

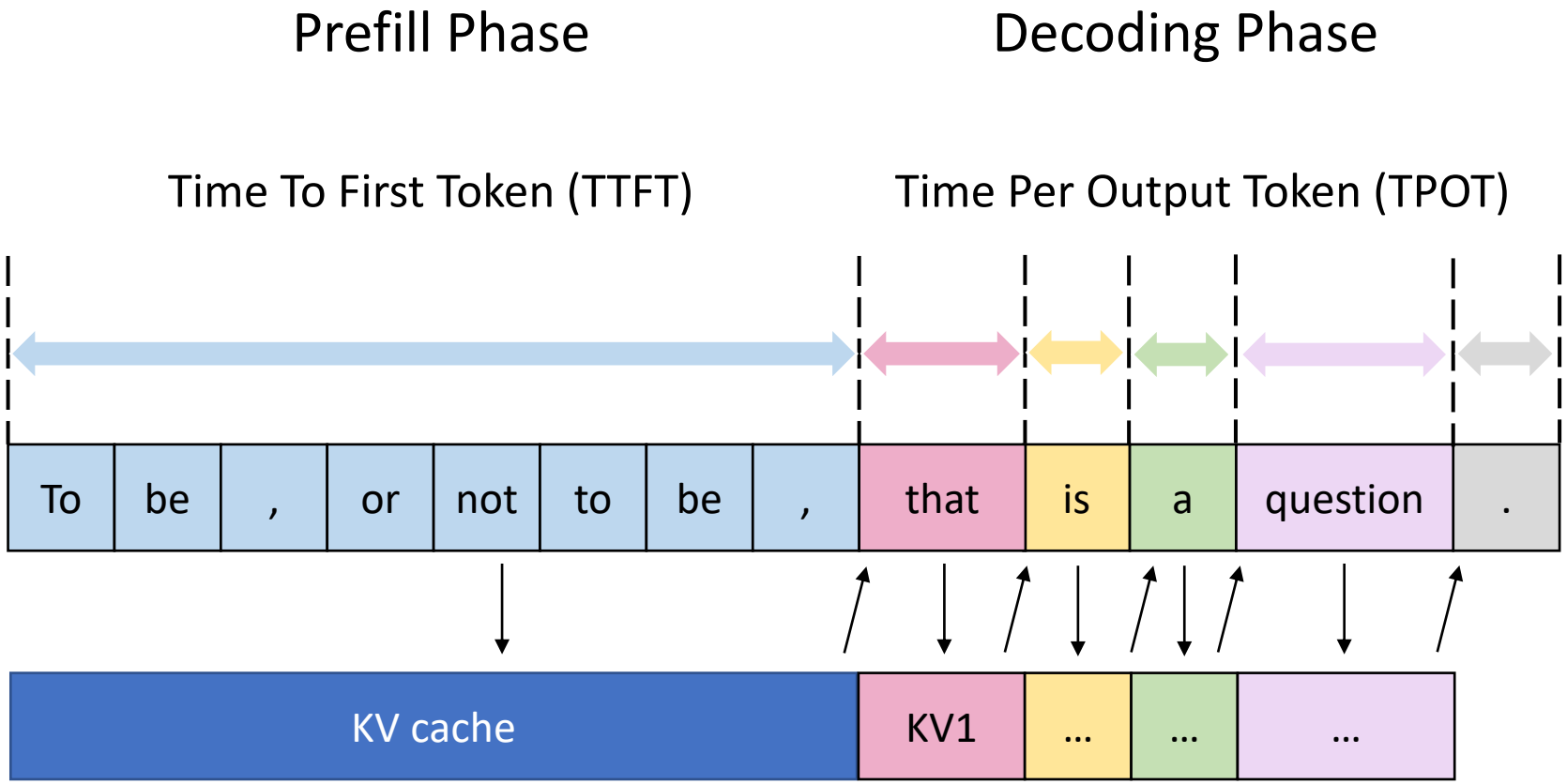
Background

LLM Inference



Background

LLM Inference



Background

LLM Inference

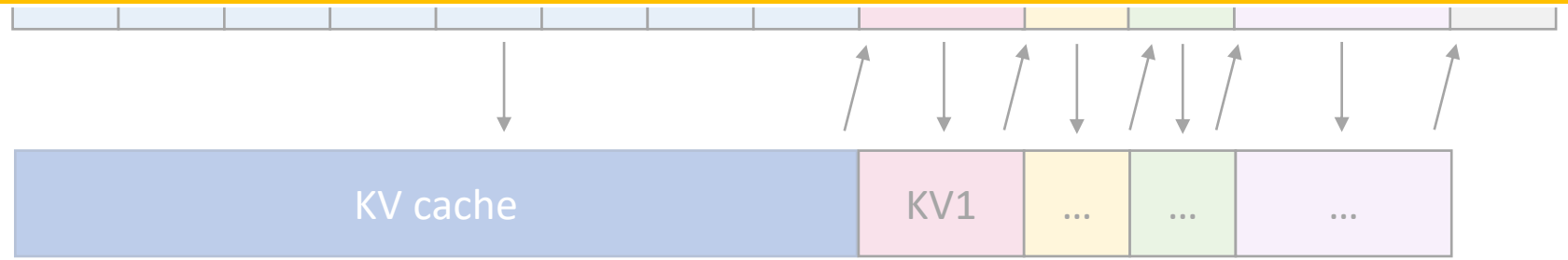
Prefill Phase

Decoding Phase

Time To First Token (TTFT)

Time Per Output Token (TPOT)

Prefill phase is compute bounded;
While Decode phase is memory bounded!



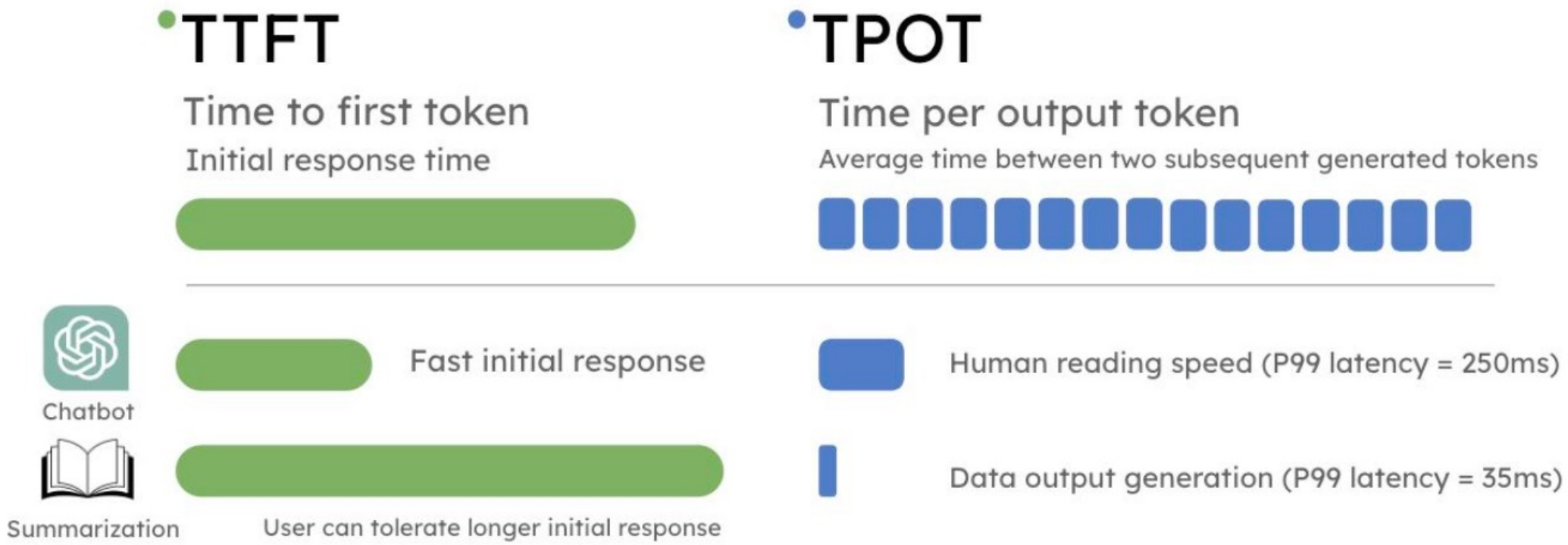
Naïve / Static Batching

Continous Batching

Orca: A Distributed Serving System for Transformer-Based Generative Models 【OSDI'22】

SLO for TTFT & TPOT

Applications have Diverse SLO



SLO for TTFT & TPOT

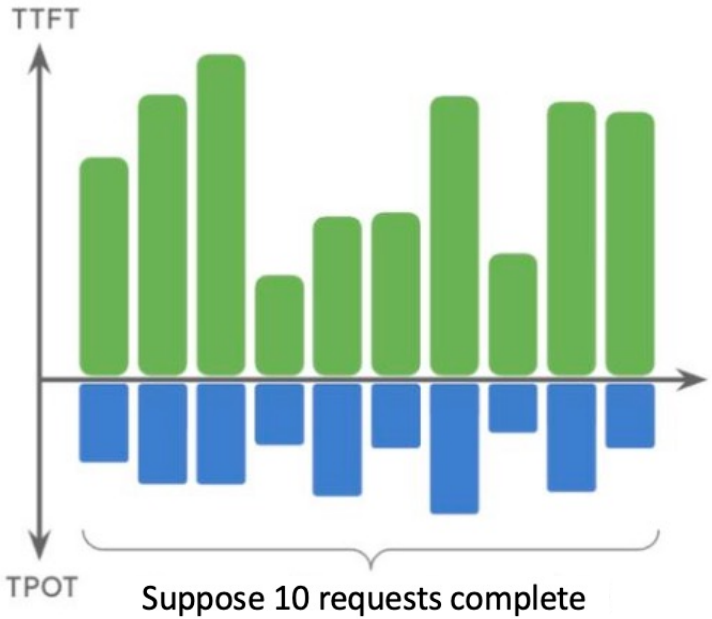
Different apps have various latency requirements

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

Throughput vs. Goodput

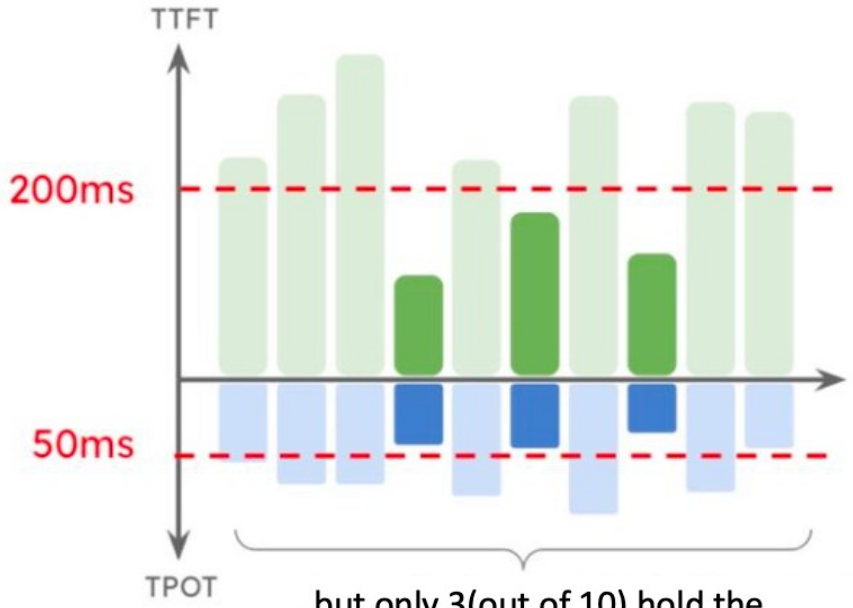
High Throughput ≠ High Goodput

Throughput = completed request / time
= **10** req / s



Suppose 10 requests complete within 1 second ...

Goodput = completed requests **within SLO** / time
= **3** req / s

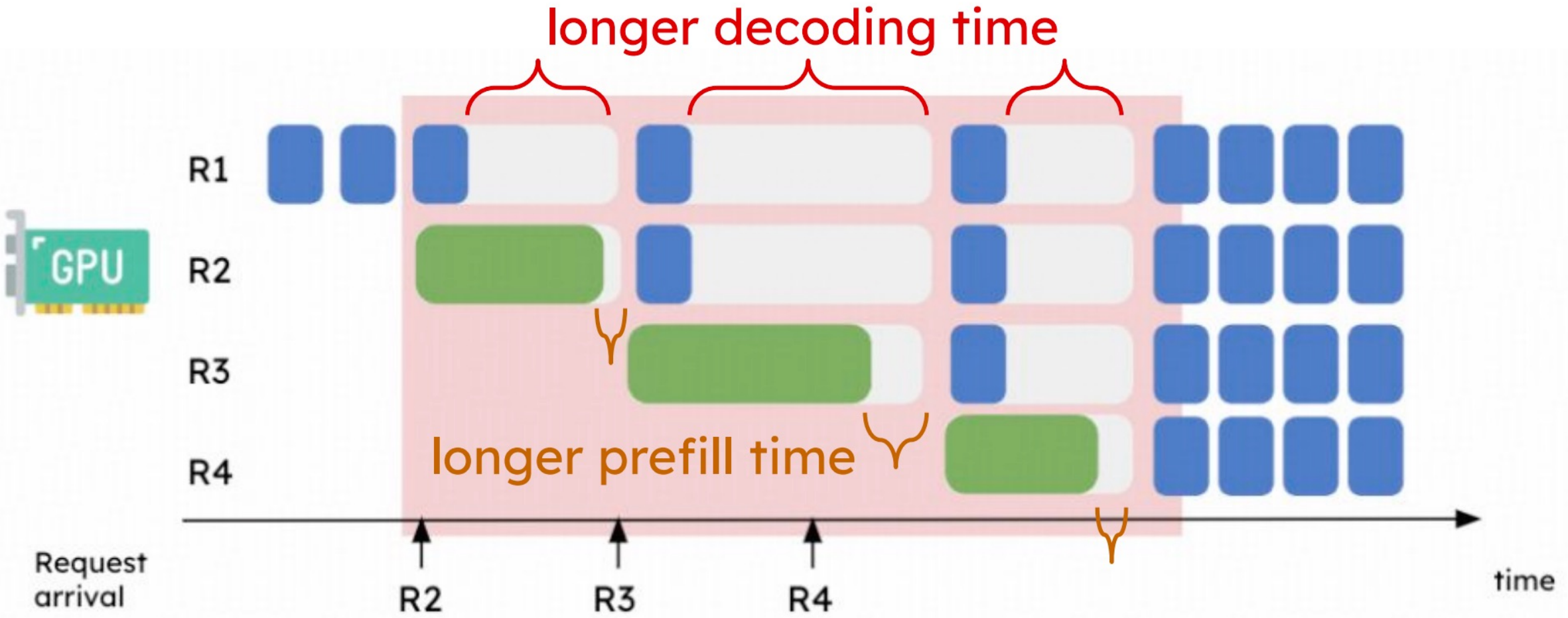


...but only 3(out of 10) hold the latency target

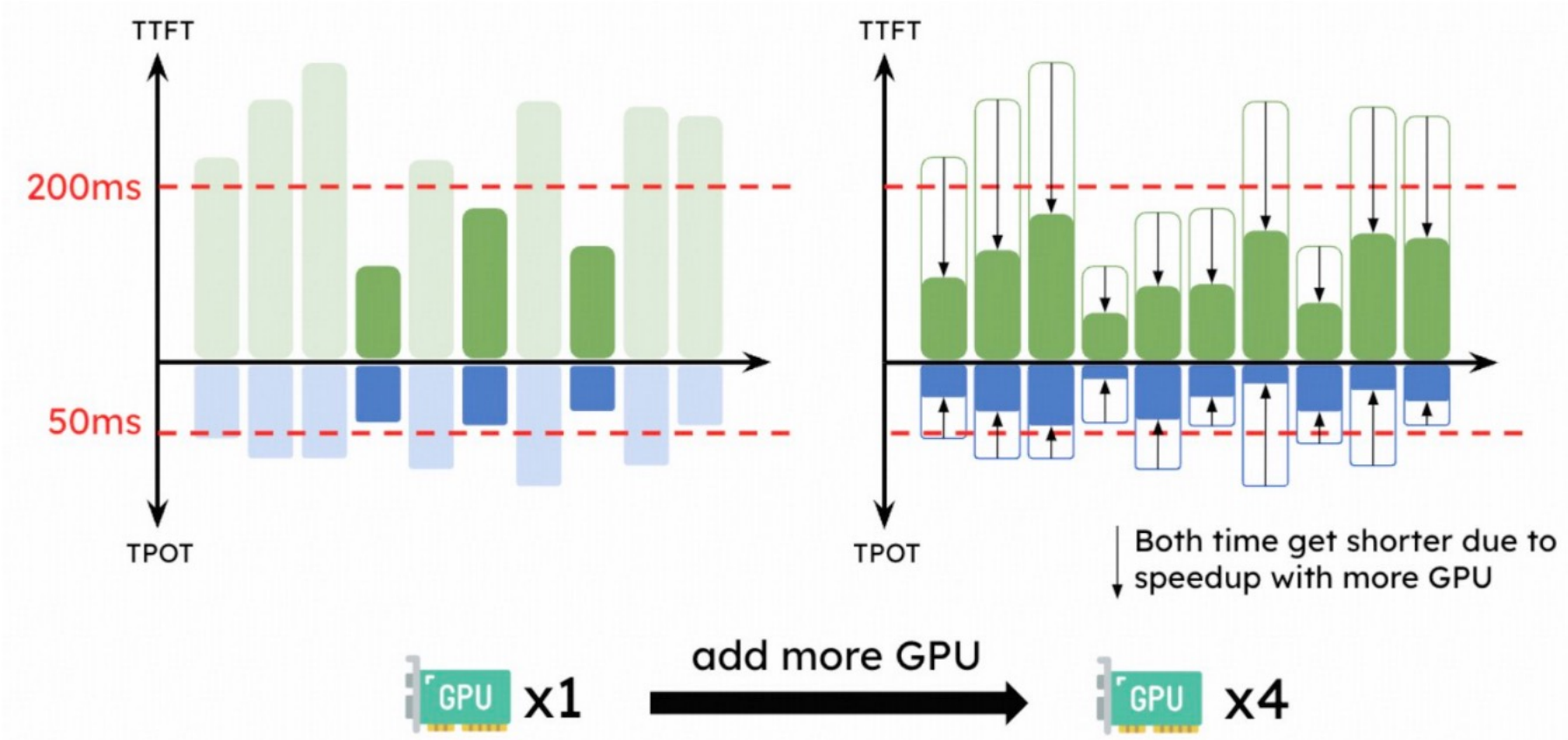
Contents

- Background
- **Problem**
- Opportunity
- Challenge
- Method
- Evaluation
- Thinking

Problem1: Conflict between P and D



Problem2.1: GPU Strategy



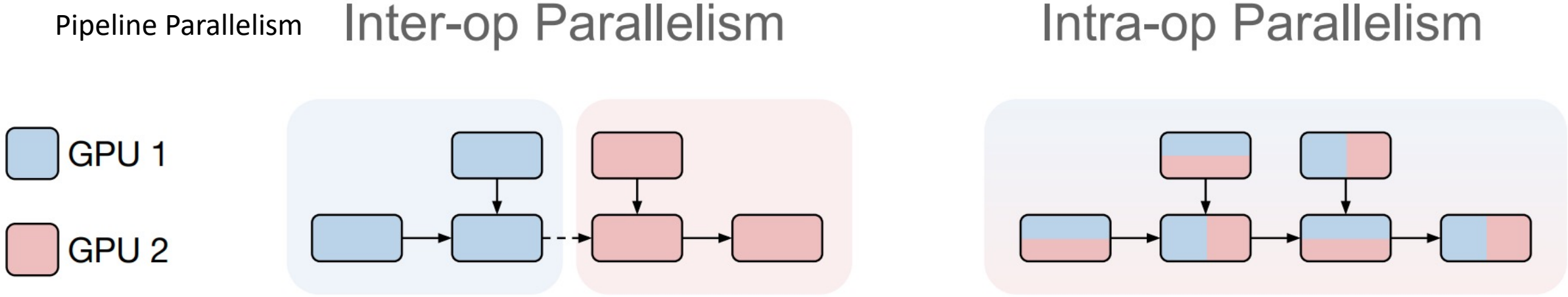
Cost	✅	❌
Goodput	❌	✅

Problem2.2: Parallelism Strategy

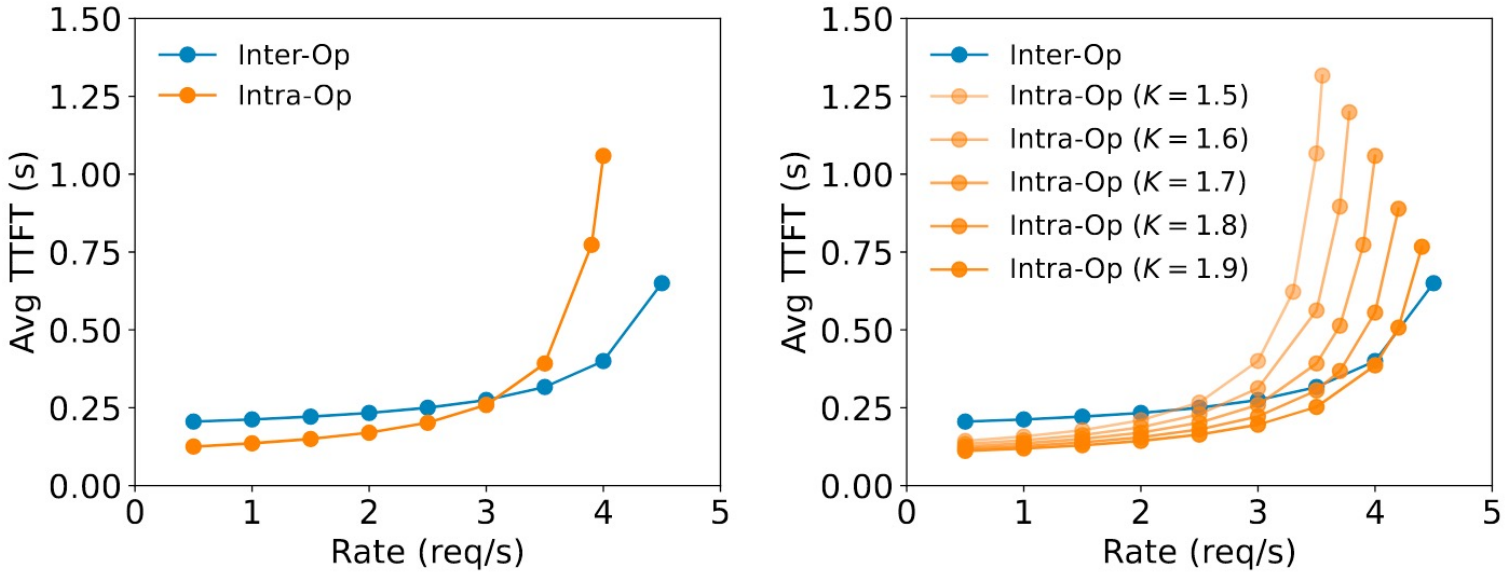
Batching the two phases make them share the same parallel strategy

Tensor Parallelism

Pipeline Parallelism



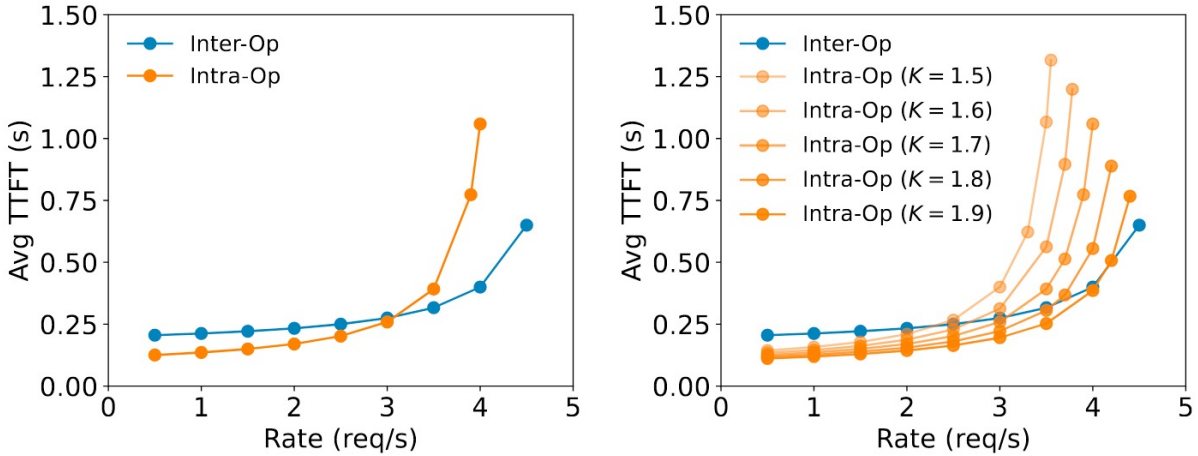
Problem2.2: Parallelism Strategy



(a) Real experiment results (b) Changing intra-op speedup

Figure 4: Average TTFT when serving an LLM with 66B parameters using different parallelism on two A100 GPUs.

Problem2.2: Parallelism Strategy



(a) Real experiment results (b) Changing intra-op speedup

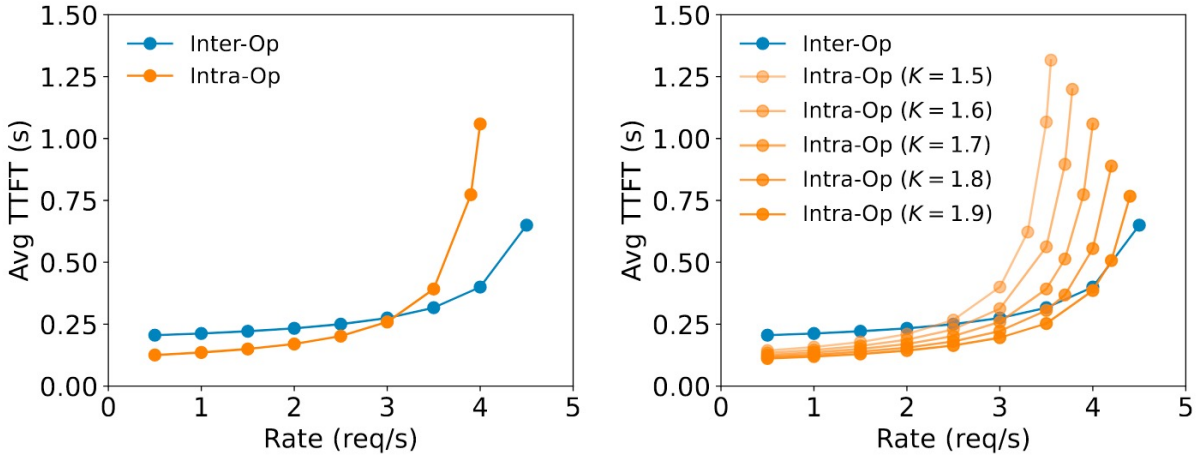
Figure 4: Average TTFT when serving an LLM with 66B parameters using different parallelism on two A100 GPUs.

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

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

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

Problem2.2: Parallelism Strategy



(a) Real experiment results (b) Changing intra-op speedup

Figure 4: Average TTFT when serving an LLM with 66B parameters using different parallelism on two A100 GPUs.

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

Queueing Theory

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

Problem2.2: Parallelism Strategy

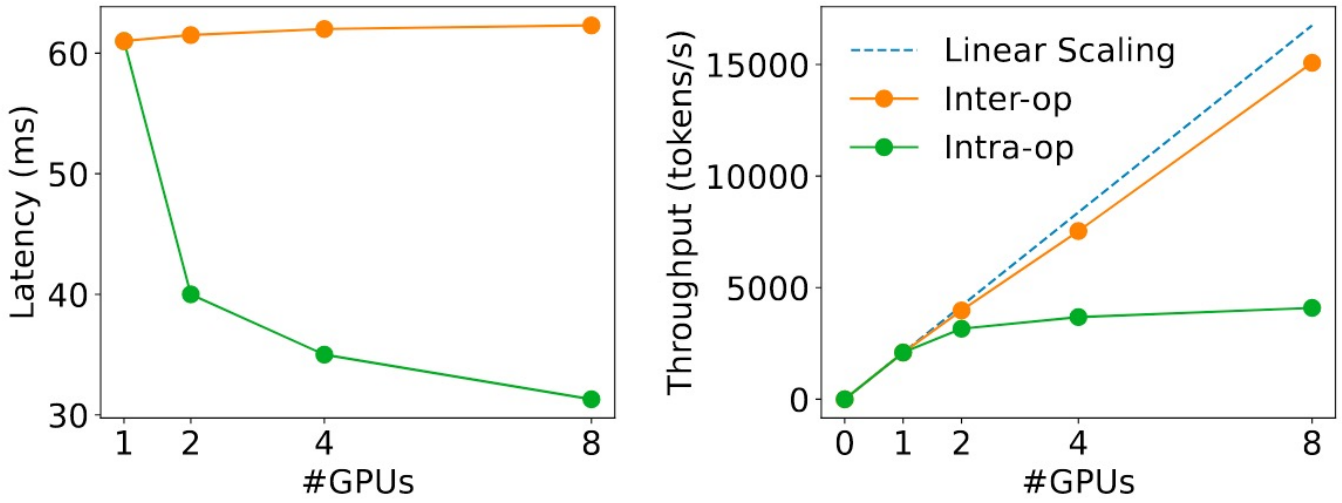
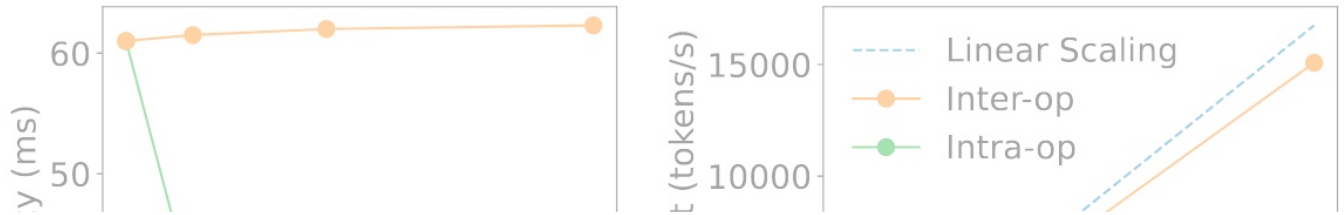


Figure 5: Decoding phase latency and throughput when serving a 13B LLM with batch size = 128 and input length = 256 under different parallel degrees.

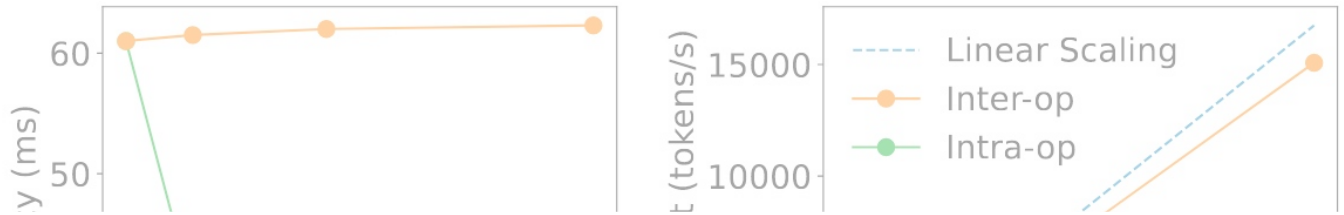
Problem2.2: Parallelism Strategy



Different phase need to meet the target SLO,
so their parallelism strategies are also different.

Figure 5: Decoding phase latency and throughput when serving a 13B LLM with batch size = 128 and input length = 256 under different parallel degrees.

Problem2.2: Parallelism Strategy



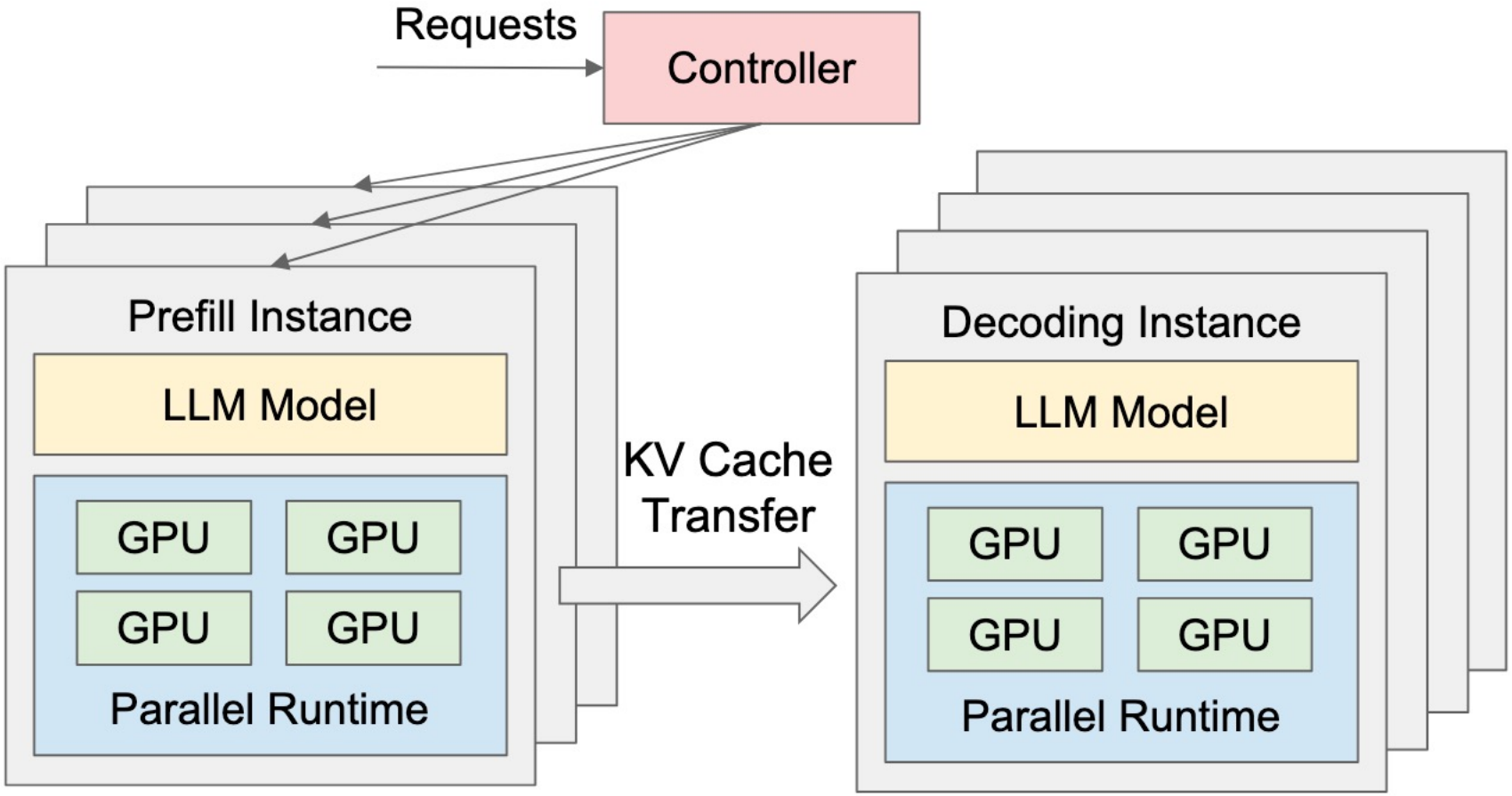
Our goal: Meet both SLOs while minimizing the cost of GPUs.
In other word, *maximizing per-GPU goodput*.

Figure 5: Decoding phase latency and throughput when serving a 13B LLM with batch size = 128 and input length = 256 under different parallel degrees.

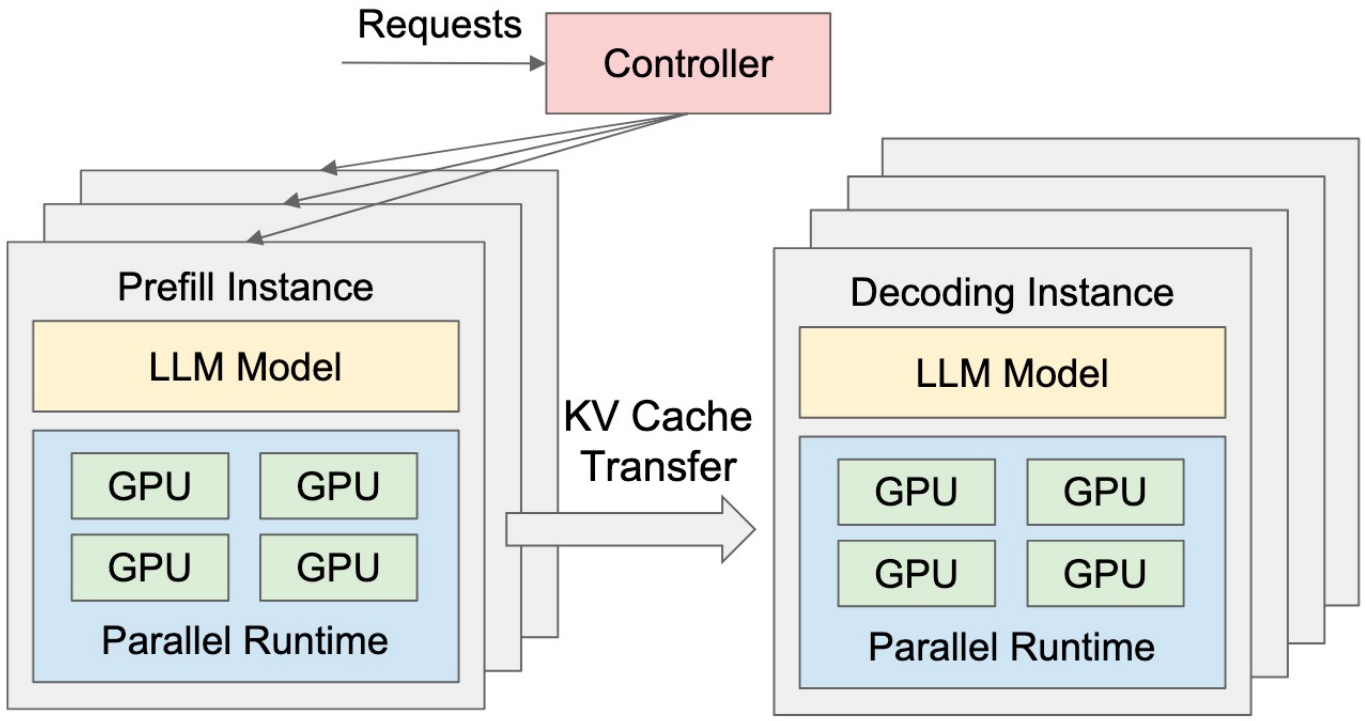
Contents

- Background
- Problem
- **Opportunity**
- Challenge
- Method
- Evaluation
- Thinking

Opportunity



Opportunity



1. TTFT and TPOT are independent.
2. Decoupling GPU / Parallelism Strategy.

Contents

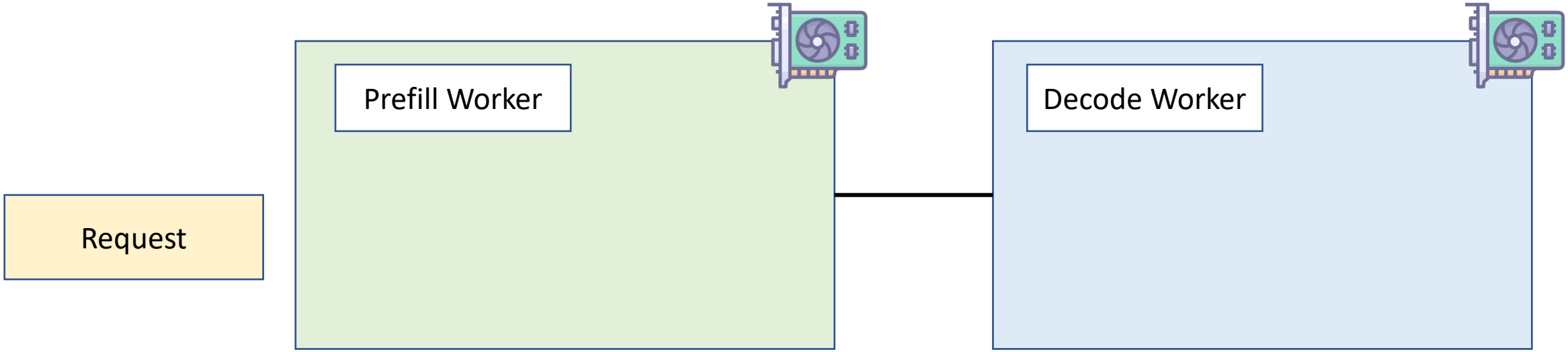
- Background
- Problem
- Opportunity
- **Challenge**
- Method
- Evaluation
- Thinking

Challenge

- **Communication overhead for KV cache migration**

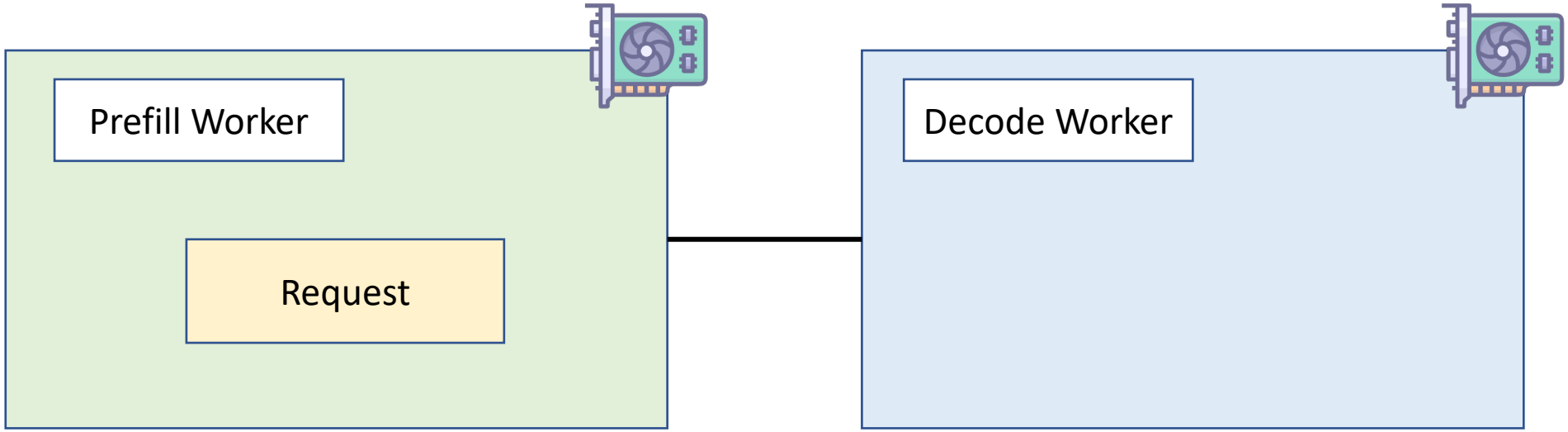
- **The optimization target – *per-GPU goodput*, is difficult to optimize:**
  - The workload pattern
  - SLO requirements
  - Parallelism strategies
  - Resource allocation
  - Network bandwith

Challenge: Migrate Overhead



Timeline

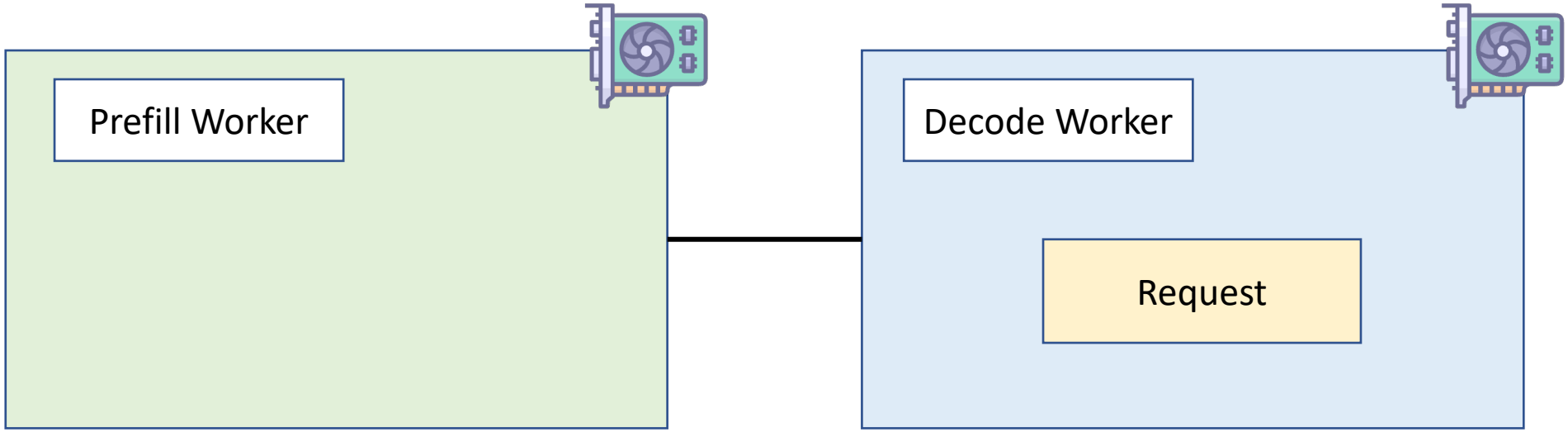
Challenge: Migrate Overhead



Timeline

Prefill

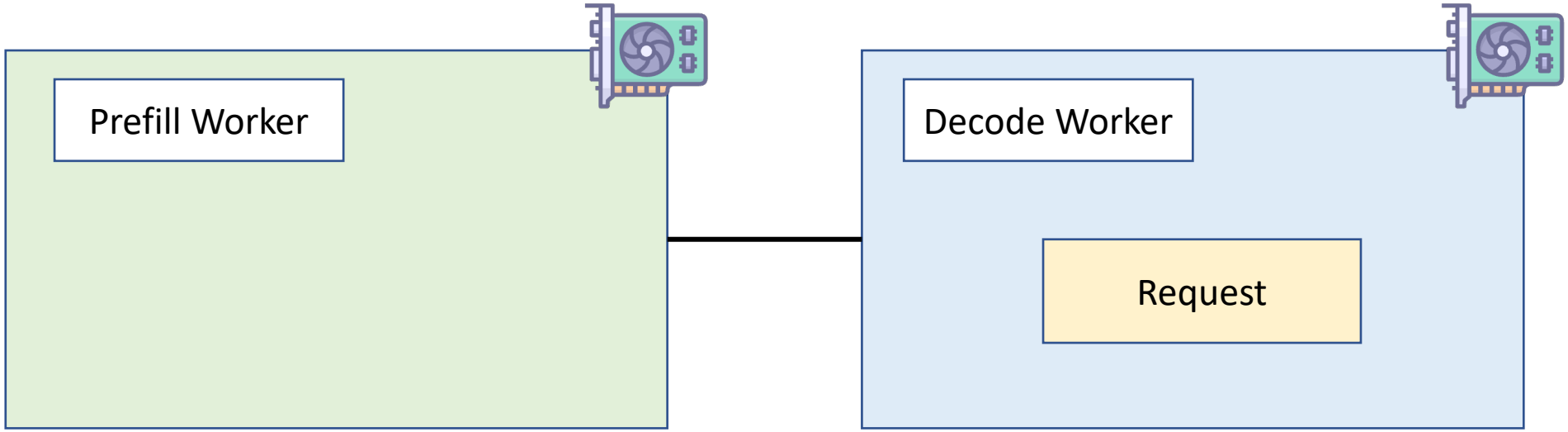
Challenge: Migrate Overhead



Timeline

Migrate

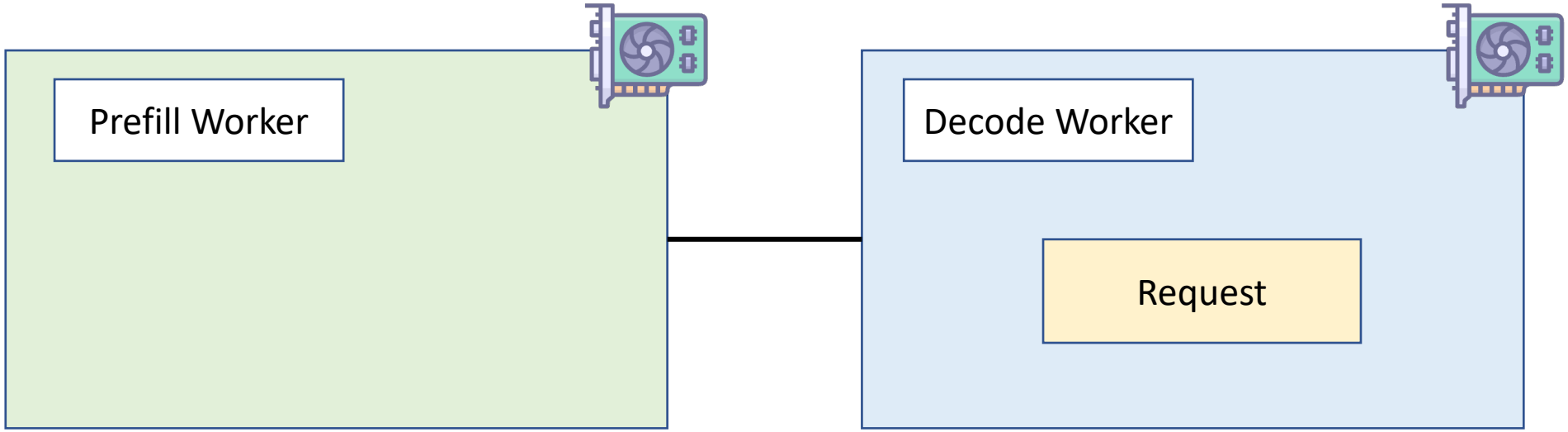
Challenge: Migrate Overhead



Timeline

Decode

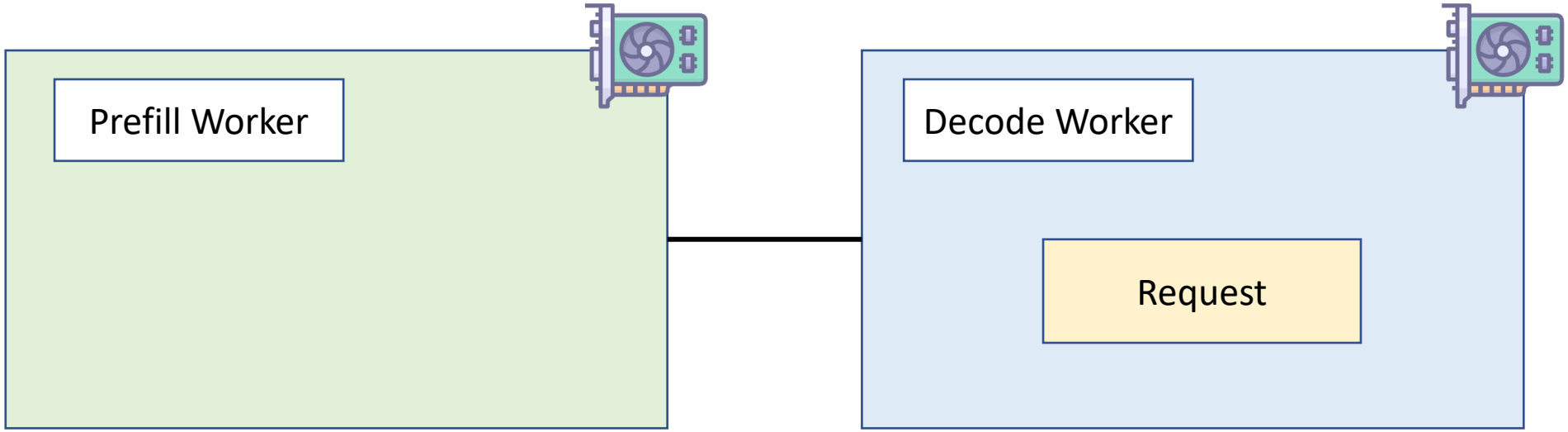
Challenge: Migrate Overhead



Timeline

Decode

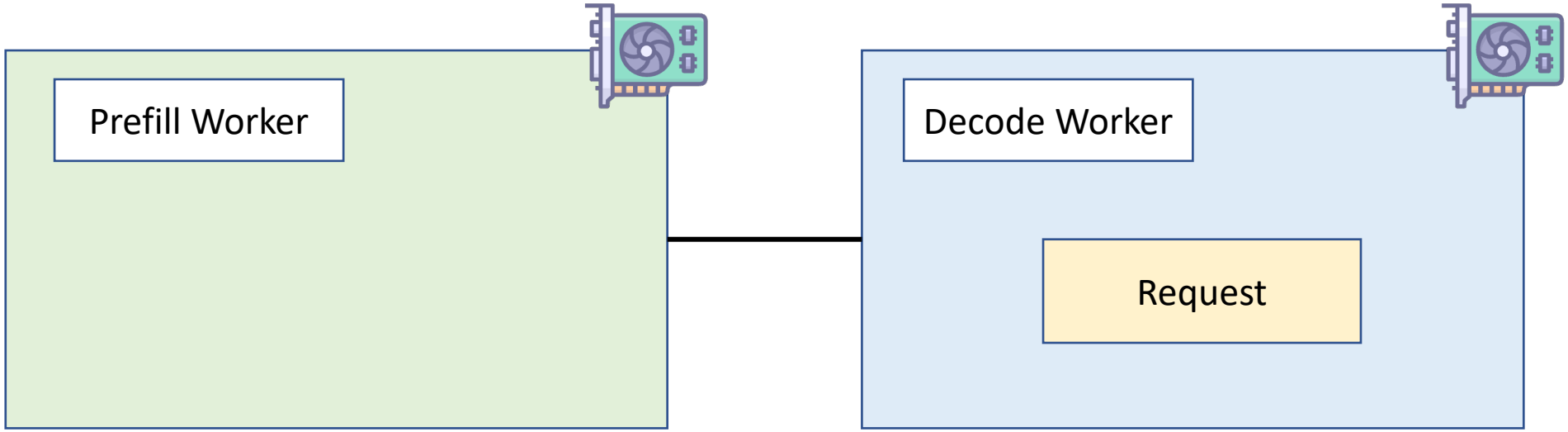
Challenge: Migrate Overhead



Timeline

Decode

Challenge: Migrate Overhead



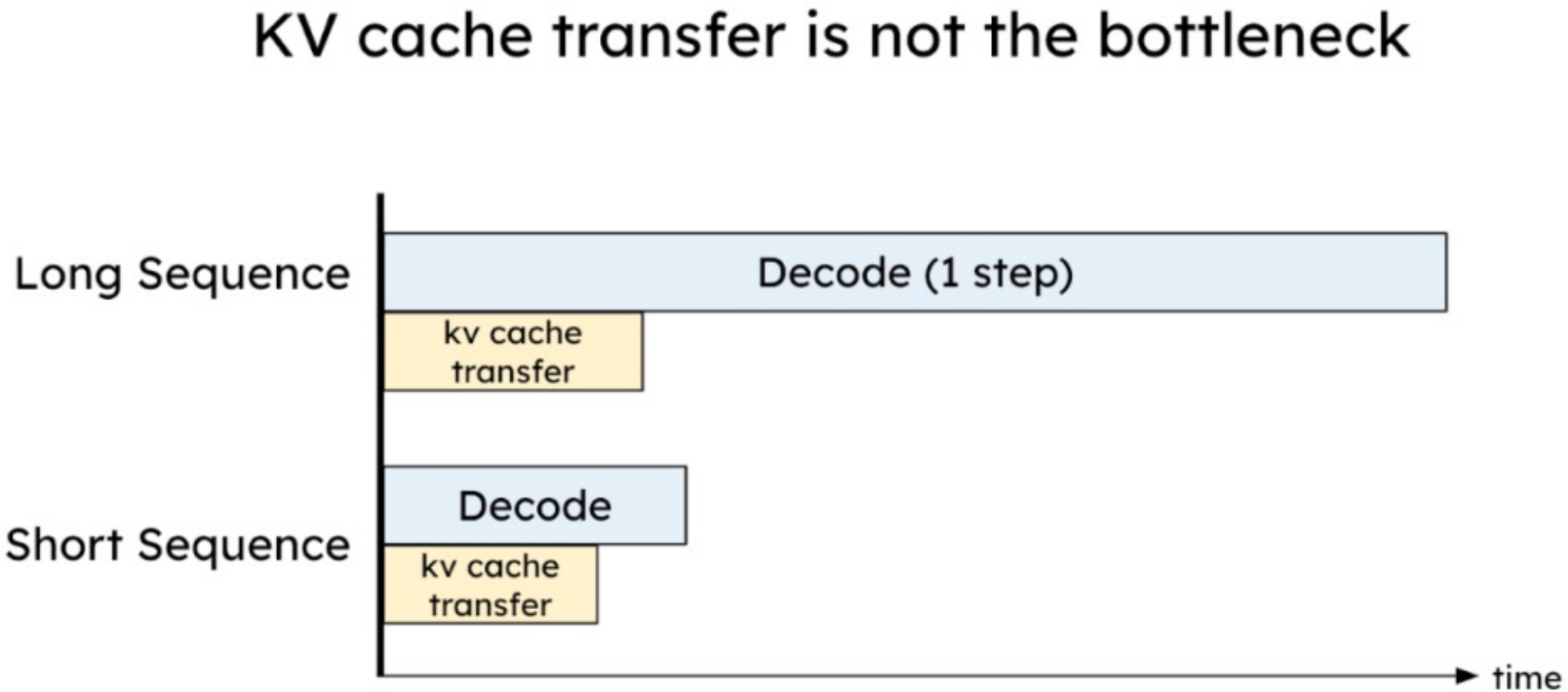
Timeline

Decode

Challenge: Migrate Overhead

- 8-channel Pcle 5.0 * 16 (64GB/s per link)
- Seq Length = 2048
- OPT-175B
- Latency = 2048 * (4.5MB/token) / (64GB/s * 8) = 17.6ms
- Decode latency = 30-50ms

Challenge: Migrate Overhead



Challenge: Goodput-optimization

- Communication overhead for KV cache migration

- **The optimization target – *per-GPU goodput*, is difficult to optimize:**

- The workload pattern
- SLO requirements
- Resource allocation
- Network bandwith

DistServe

- Parallelism strategies for P/D
- The #P,#D to deploy
- How to place them onto cluster

Contents

- Background
- Problem
- Opportunity
- Challenge
- **Method**
- Evaluation
- Thinking

Method

Inter-machine Search

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 $best\_plm$

Method

Inter-machine Search

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 $best\_plm$.

$config_p, config_d \leftarrow \emptyset, \emptyset$

TP ← **for** $intra\_op \in \{1, 2, ..., M\}$ **do**

PP ← **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 $best\_plm$

Method

Inter-machine Search

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 $best\_plm$.

$config_p, config_d \leftarrow \emptyset, \emptyset$

TP ← **for** $intra\_op \in \{1, 2, ..., M\}$ **do**

PP ← **for** $inter\_op \in \{1, 2, ..., \frac{N \times M}{intra\_op}\}$ **do**

if $\frac{G.size}{inter\_op \times intra\_op} < C$ **then**

$config \leftarrow (inter\_op, intra\_op)$

$\hat{G} \leftarrow \text{parallel}(G, config)$

$config.goodput \leftarrow \text{simu\_prefill}(\hat{G}, W)$

Simulate ← **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 $best\_plm$

Method

Inter-machine Search

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 $best\_plm$.

$config_p, config_d \leftarrow \emptyset, \emptyset$

TP ← **for** $intra\_op \in \{1, 2, ..., M\}$ **do**

PP ← **for** $inter\_op \in \{1, 2, ..., \frac{N \times M}{intra\_op}\}$ **do**

if $\frac{G.size}{inter\_op \times intra\_op} < C$ **then**

$config \leftarrow (inter\_op, intra\_op)$

$\hat{G} \leftarrow \text{parallel}(G, config)$

$config.goodput \leftarrow \text{simu\_prefill}(\hat{G}, W)$

Simulate ← **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**

$config_d \leftarrow config$

Solution ←

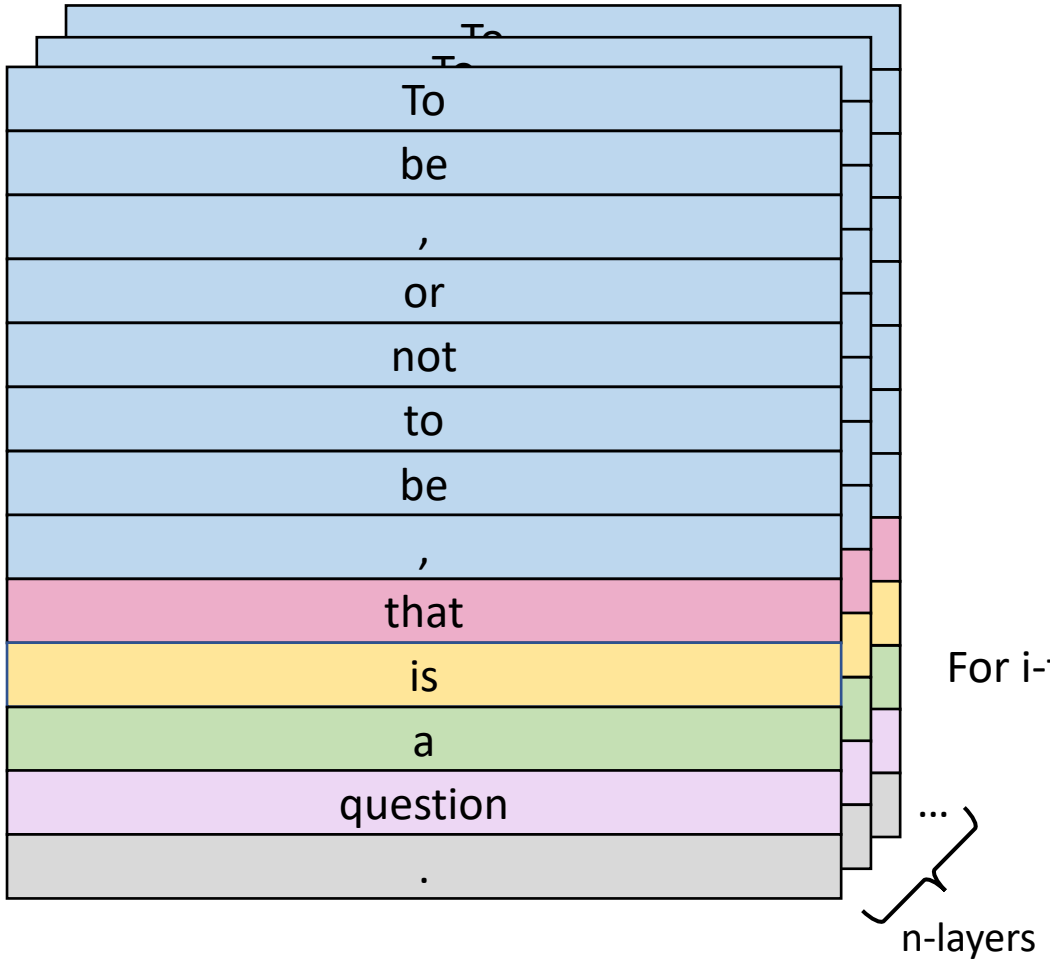
$n, m \leftarrow \lceil \frac{R}{config_p.goodput} \rceil, \lceil \frac{R}{config_d.goodput} \rceil$

$best\_plm \leftarrow (n, config_p, m, config_d)$

return $best\_plm$

Method

Intra-machine Search



If LLM *G* (such as OPT-175B) is too large, single machine cannot afford.

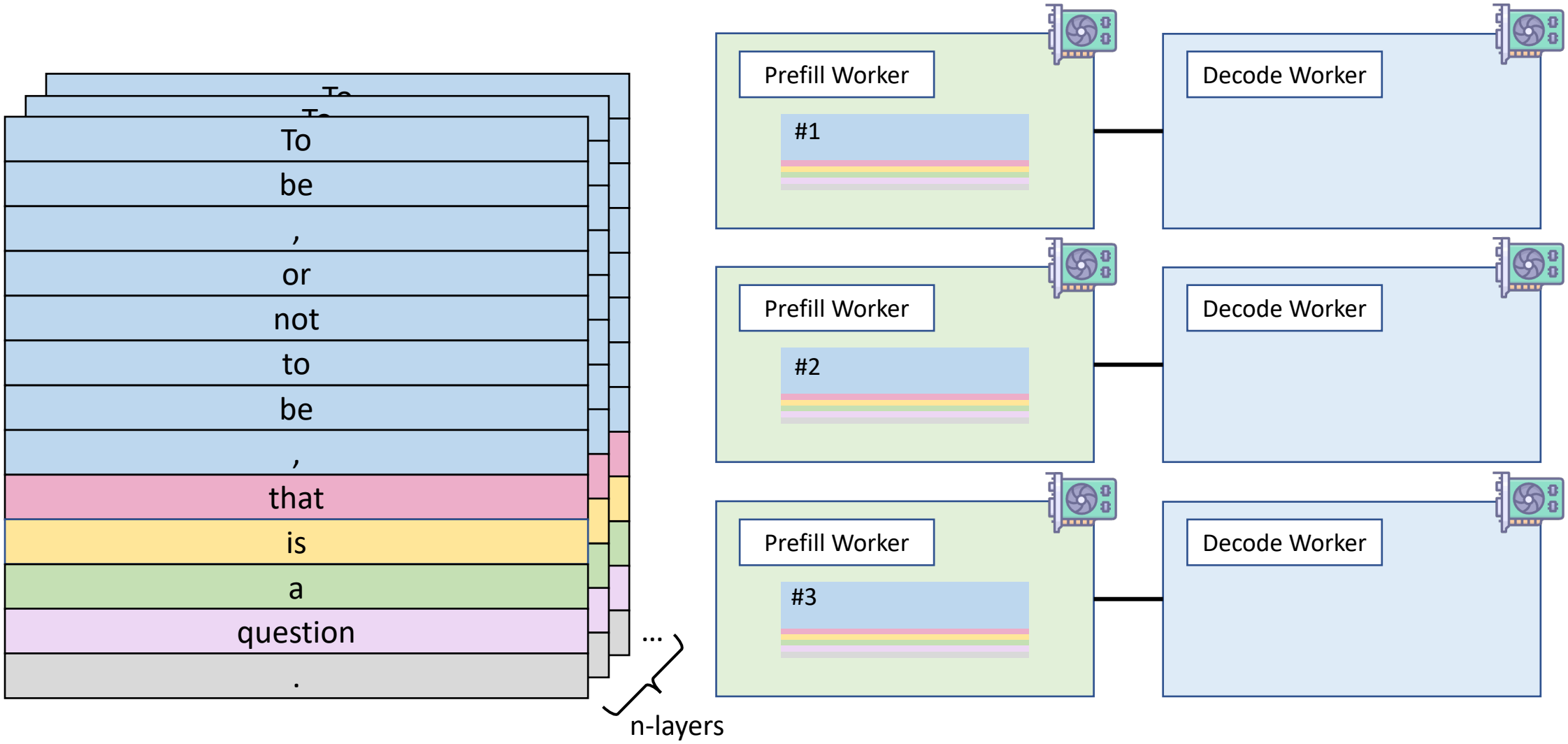
Cross-machine transimission is required, increases the migration time 😭

Need to place workers reasonably:

For i-th layer, make the Prefill worker and Decode worker with in the same machine.

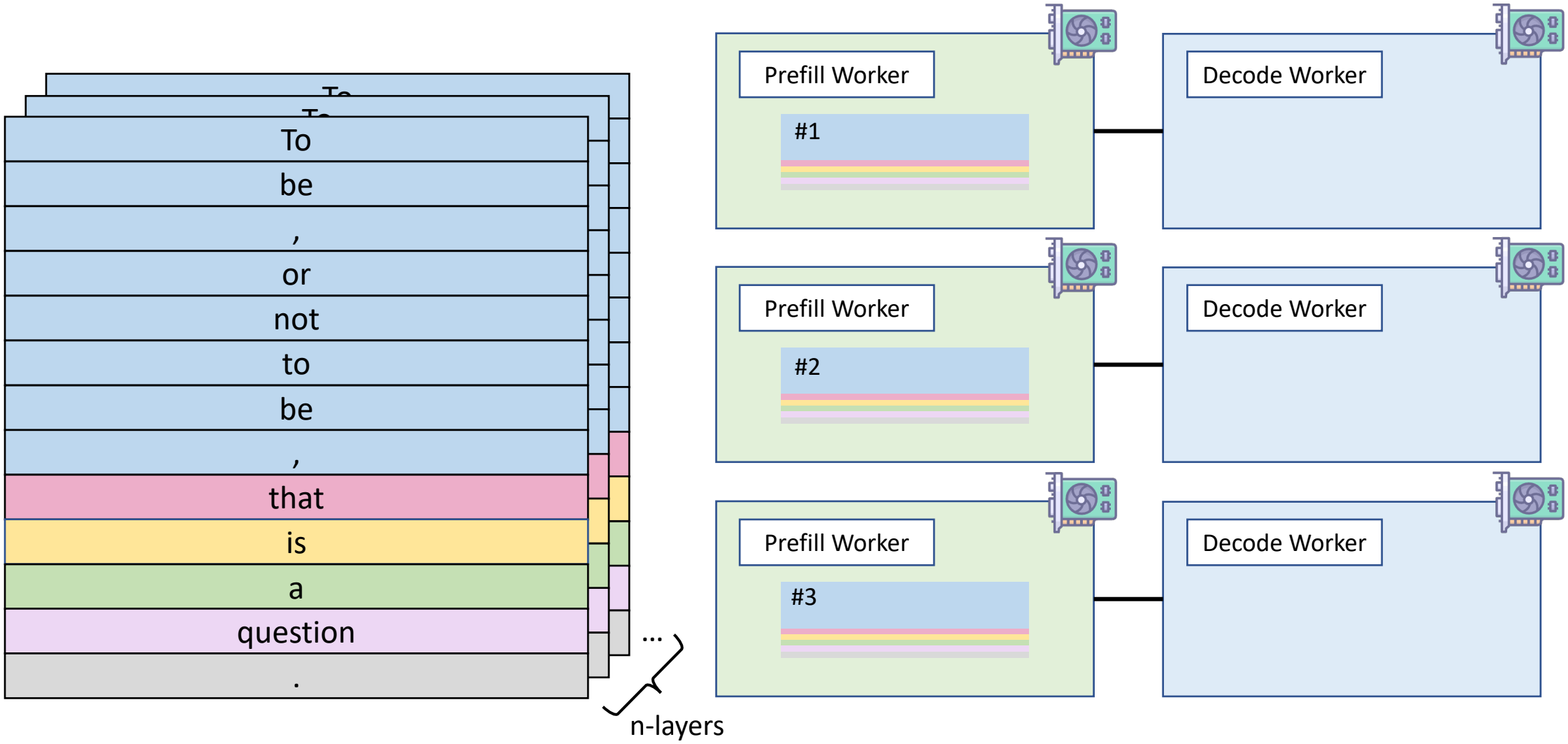
Method

Intra-machine Search



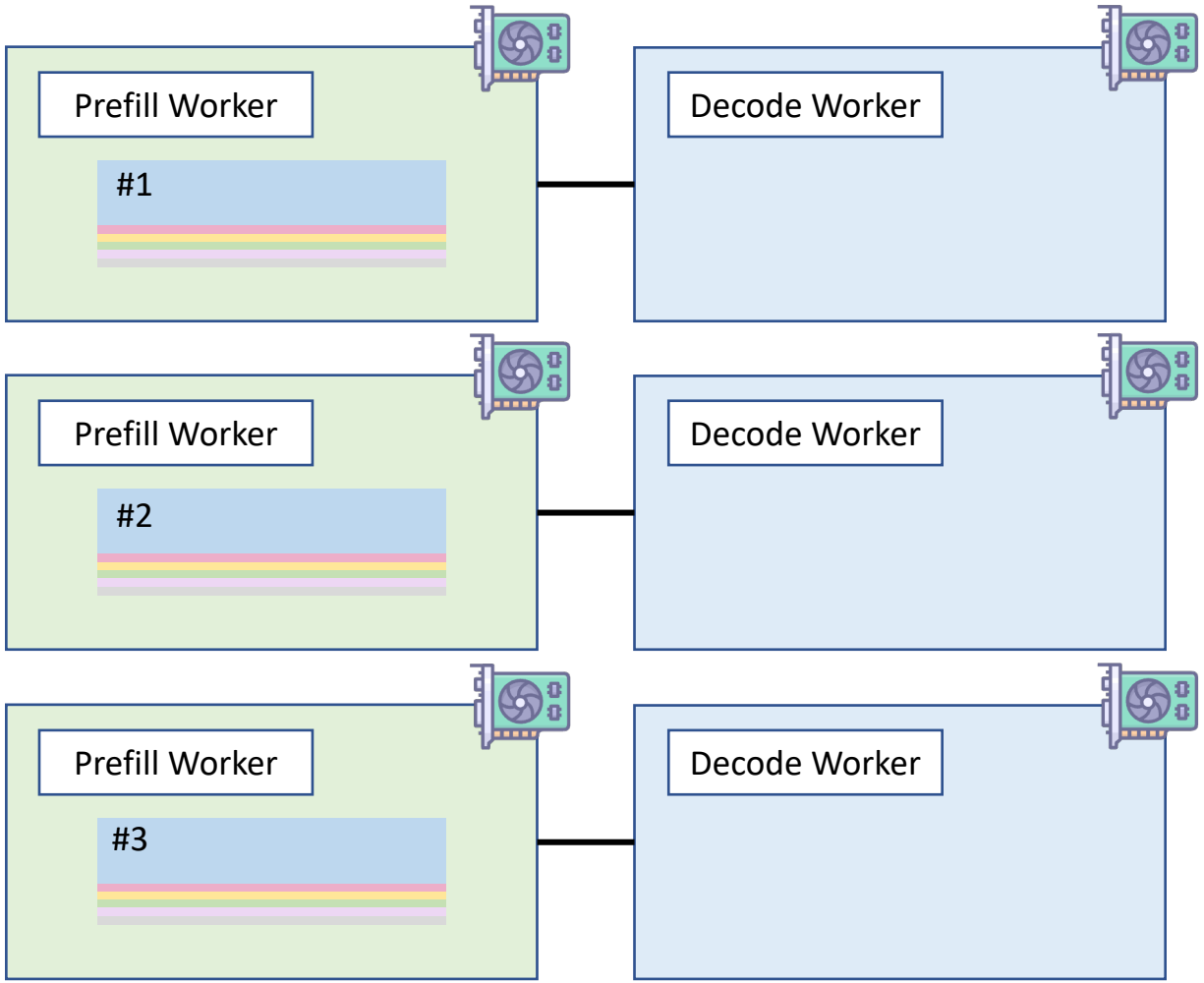
Method

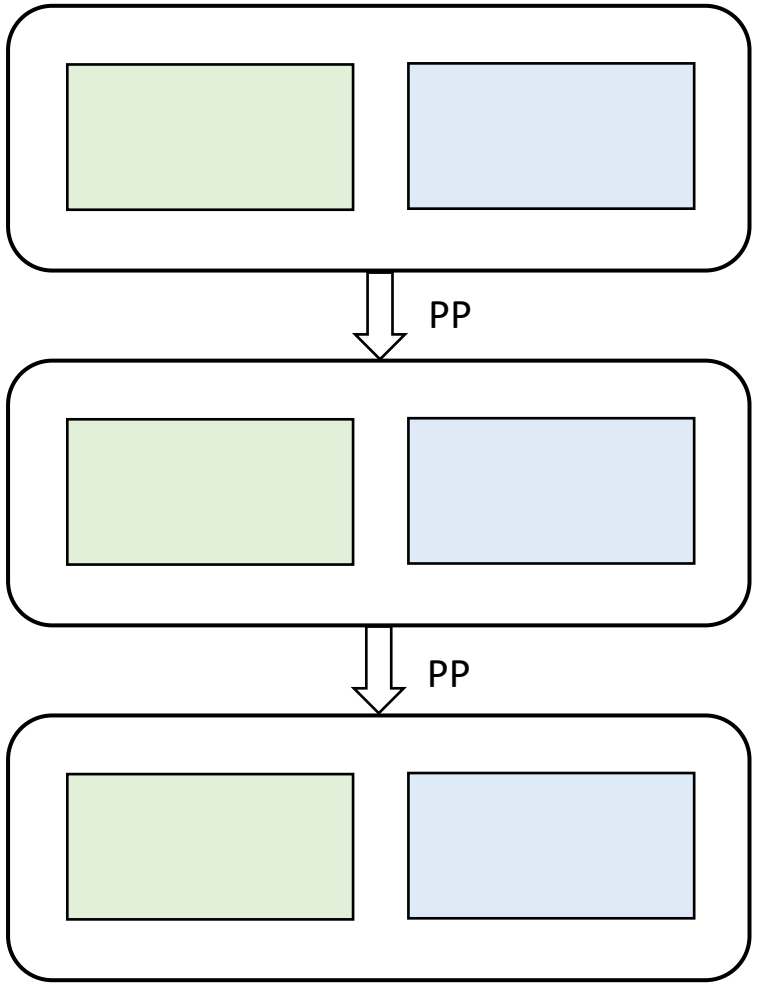
Intra-machine Search



Method

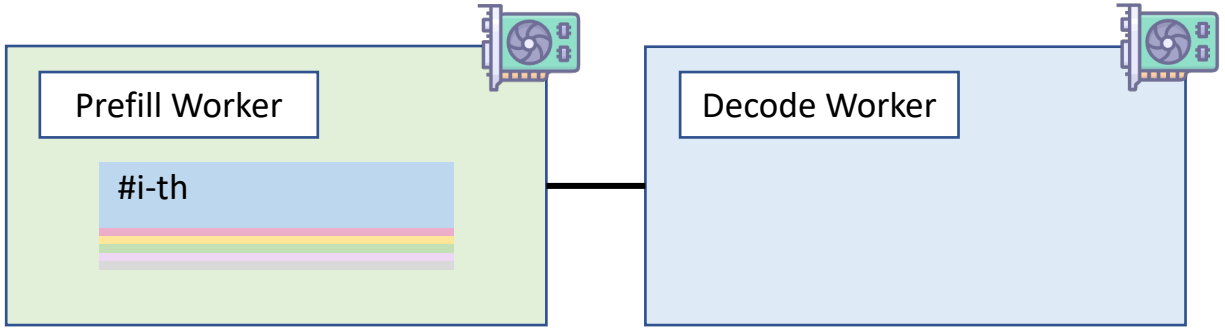
Intra-machine Search





Method

Intra-machine Search



PP

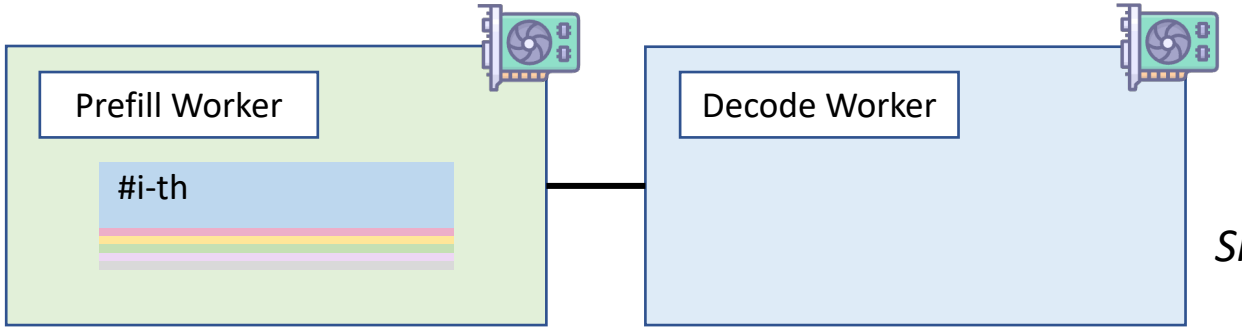
Algorithm 2 Low Node-Affinity Placement Algorithm

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*

Method

Intra-machine Search



PP

Simulate

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**

          $config^* \leftarrow config$

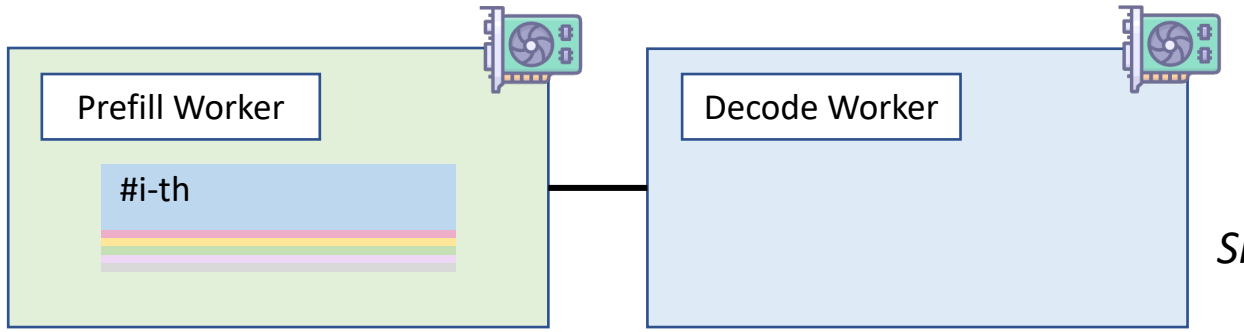
$n \leftarrow \lceil \frac{R}{config.*goodput} \rceil$

$best\_plm \leftarrow (n, config^*)$

return *best_plm*

Method

Intra-machine Search



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** ← PP
  $\mathcal{P} \leftarrow$ 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**
        $config \leftarrow (inter\_op, P_p, P_d)$
        $\hat{G}_p, \hat{G}_d \leftarrow$ parallel$(G, config)$ ← Simulate
        $config.goodput \leftarrow$ simulate$(\hat{G}_p, \hat{G}_d, W)$ ← Simulate
        if $\frac{config.^*goodput}{config.^*num\_gpus} < \frac{config.goodput}{config.num\_gpus}$ **then**
          $config^* \leftarrow config$
$n \leftarrow \lceil \frac{R}{config.^*goodput} \rceil$ ← Solution
$best\_plm \leftarrow (n, config^*)$ ← Solution
return *best_plm*

Contents

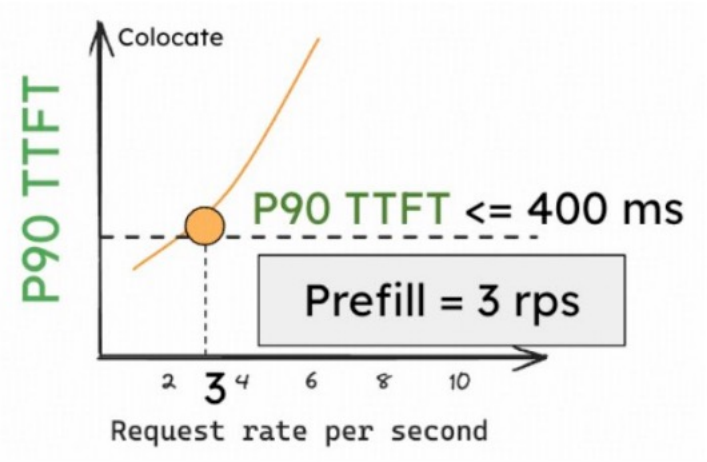
- Background
- Problem
- Opportunity
- Challenge
- Method
- **Evaluation**
- Thinking

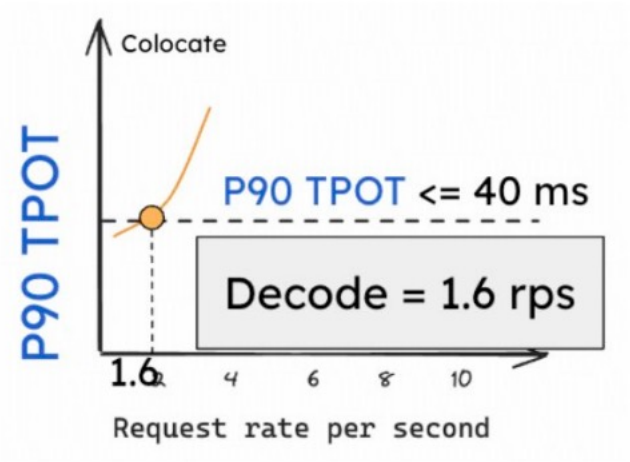
Evaluation

LLM App	Dataset	TTFT	TPOT
Chatbot	shareGPT	Tight	Medium
Code completion	HumanEval	Tight	Tight
Summarization	LongBench	Loose	Medium

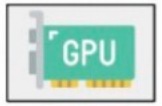
Model	Dataset	Prefill		Decoding	
		TP	PP	TP	PP
OPT-13B	ShareGPT	2	1	1	1
OPT-66B	ShareGPT	4	1	2	2
OPT-66B	LongBench	4	1	2	2
OPT-66B	HumanEval	4	1	2	2
OPT-175B	ShareGPT	3	3	4	3

Evaluation





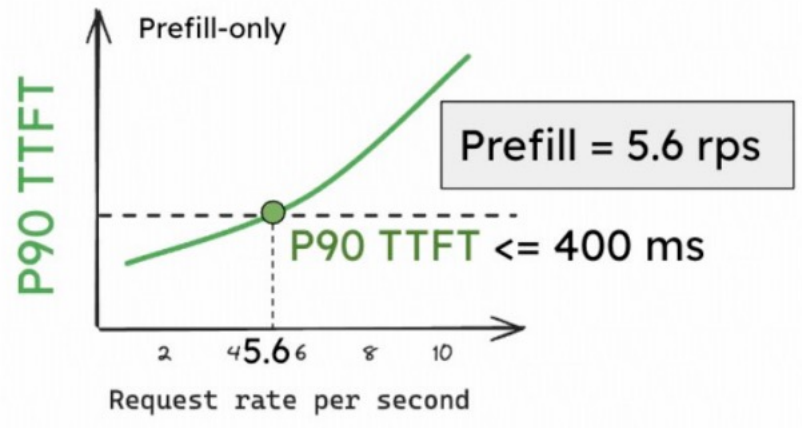
Colocation

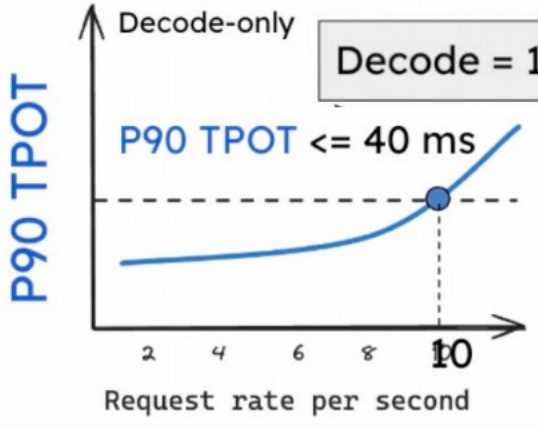


Max System rps

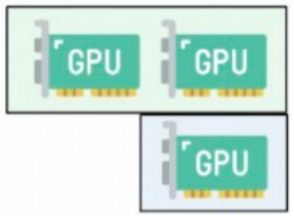
= Min(Prefill, Decode)

= 1.6 rps / GPU





Disaggregation (2P1D)



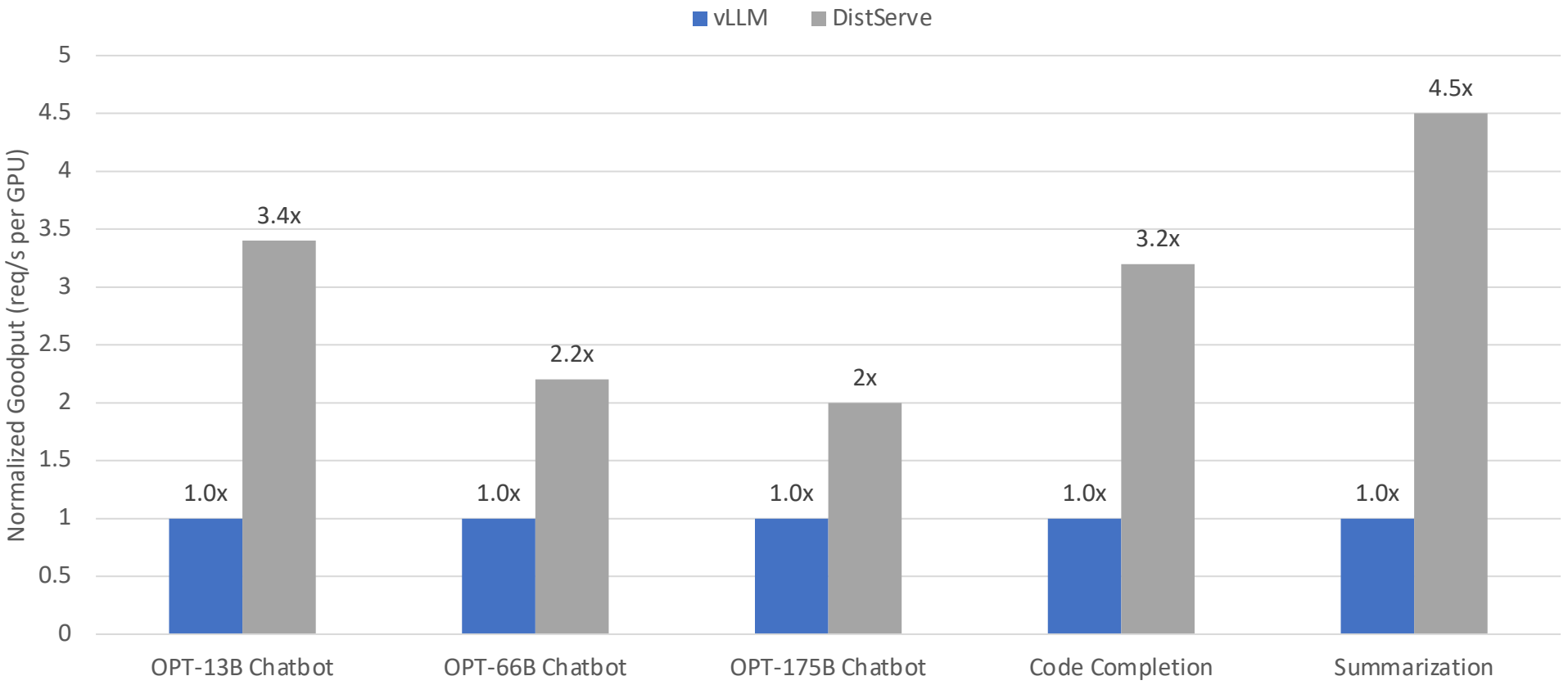
Max System rps

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

= 3.3 rps / GPU

13

Evaluation


vLLM
DistServe
Normalized Goodput (req/s per GPU)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1.0x
3.4x
1.0x
2.2x
1.0x
2x
1.0x
3.2x
1.0x
4.5x
OPT-13B Chatbot
OPT-66B Chatbot
OPT-175B Chatbot
Code Completion
Summarization

Contents

- Background
- Problem
- Opportunity
- Challenge
- Method
- Evaluation
- **Thinking**

Thinking

- **文章有什么问题，基于这个 paper 还能做什么**
  - P/D分离服务架构，会导致可维护性下降，整个引擎耦合性变高
  - KV transfer 可以增量更新，但实际上传输开销并不大，这种方式是否有益还需要进行 trade-off 判断

- **Paper 的 idea 能不能应用在自己的工作上面**
  - 重新考虑不同应用的 ”goodput”，针对不同的 SLO 侧重不同的策略

- **这个 paper 能不能泛化**
  - 根据模型各层/各阶段计算性质不同，对资源的偏好不同，应用不同的推理手段。例如端侧的 NPU 适合做 prefill 性质的计算，需要重新回顾下 powerinfer2（个人认为是在端侧做PD 分离的一种尝试）

Q & A

Presenter: Yitao Wang

2024.3.7