

Nov 15–20, 2026, Chicago, USA







FluidGPU: Fine-Grained Kernel Disaggregation for Large Model Inference on Heterogeneous GPUs

Tiancheng Hu[1], Jin Qin[23], Lei Chen[23], Junhao Hu[1], Yuzheng Wang[1], Mingxing Zhang[4]
Chunwei Xia[5], Yizhou Shan[†], Huimin Cui[23], Ting Cao[4], Zheng Wang[5], Tao Xie[1], Chenxi Wang[23]

[1]Peking University [2]ICT, CAS [3]UCAS [4]Tsinghua University
[5]University of Leeds [†]Independent Researcher

北京大学
PEKING UNIVERSITY

Background & Motivation

GPU heterogeneity is a production reality in the cloud

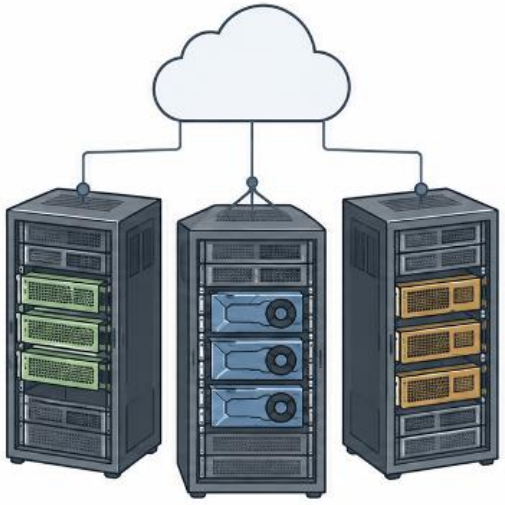
Major cloud providers offer hetegenous GPU from multiple generations and product lines.

AWS



A100, H100, B200
L4, L40s

Google Cloud



A100, H100, B200
L4, RTX PRO 6000

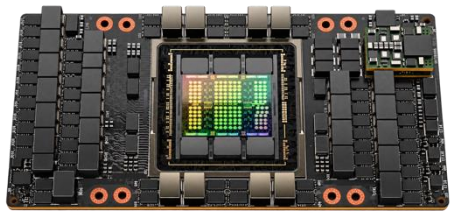
Cause of GPU heterogeneity:

- Mismatched retirement and production cycles
- Consideration of cost

GPU heterogeneity is a sustained production trend

Modular platforms sustain GPU heterogeneity

NVIDIA MGX supports reusable server designs across hardware generations.



H100



L40S



RTX PRO 6000

NVIDIA MGX

Modular server platform

Supermicro



GIGABYTE



GPU heterogeneity is a lasting systems challenge

GPU hardware heterogeneity

北京大学 PEKING UNIVERSITY

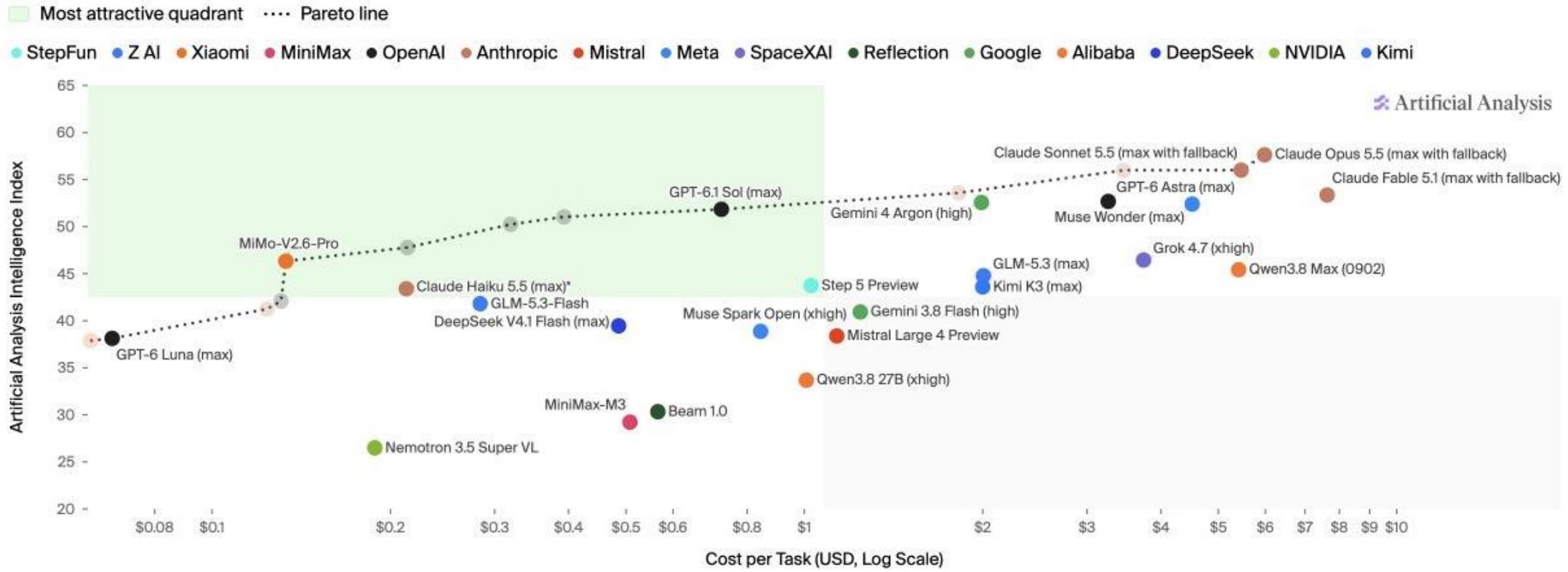
Different GPU emphasize different hardware resources.

Resource	A100	H100	B200	L40s	RTX Pro 6000
FP32 CUDA (TFLOPS)	19.5	67	80	91.6	120
BF16 Tensor (TFLOPS)	312	989	≈2,500	366.5	≈500
L2 cache (MB)	40	50	126	96	126
GPU memory (GB)	80	80	192	48	96
Mem. bandwidth (GB/s)	1,935	3,350	≈8,000	864	1,597
Relative rental cost	1.5	2.9	5.0	1.0	1.2

GPU hardwares make different trade-offs in compute, memory, and cost

Cost efficiency is a core advantage for AI models

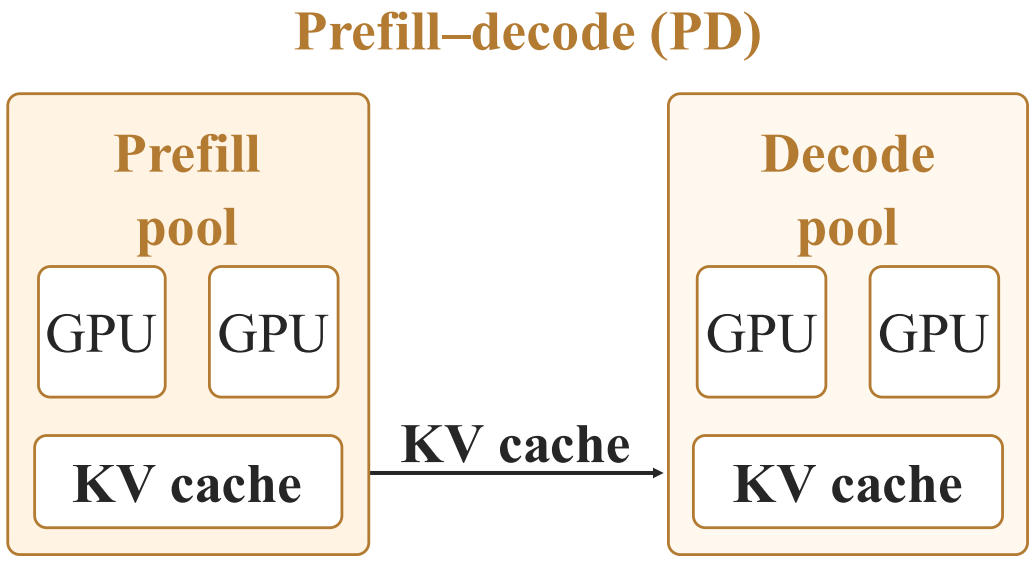
At comparable capability, a lower cost makes a model more competitive.



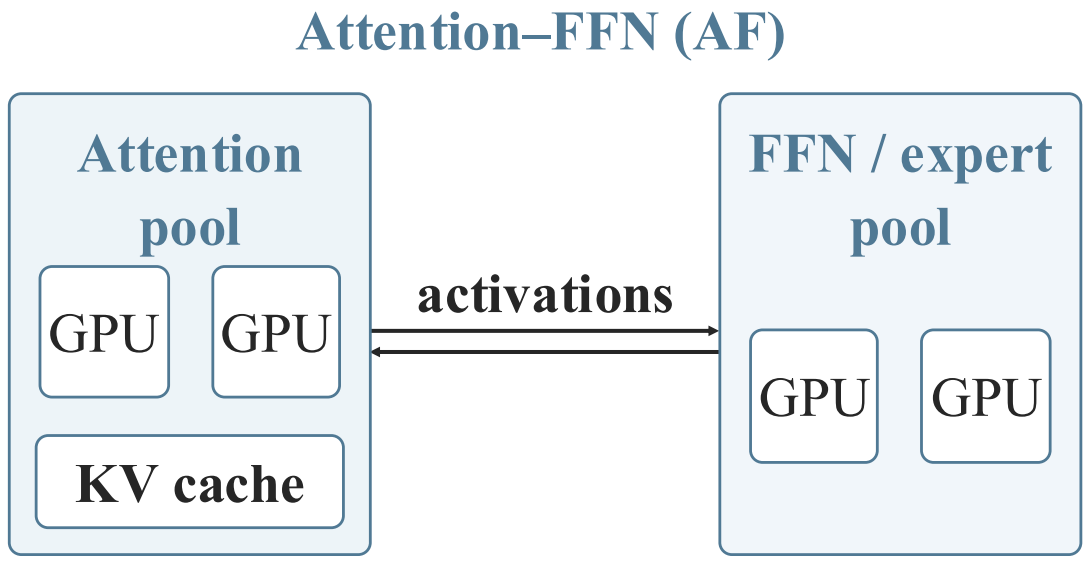
It is essential to improve throughput and cost efficiency on heterogeneous GPUs

Existing approaches utilize heterogeneous GPUs by disaggregation

State-of-the-art disaggregation methods include PD disaggregation and AF disaggregation



Prefill processes the prompt and transfers the KV cache and decode then generates tokens in another GPU pool.



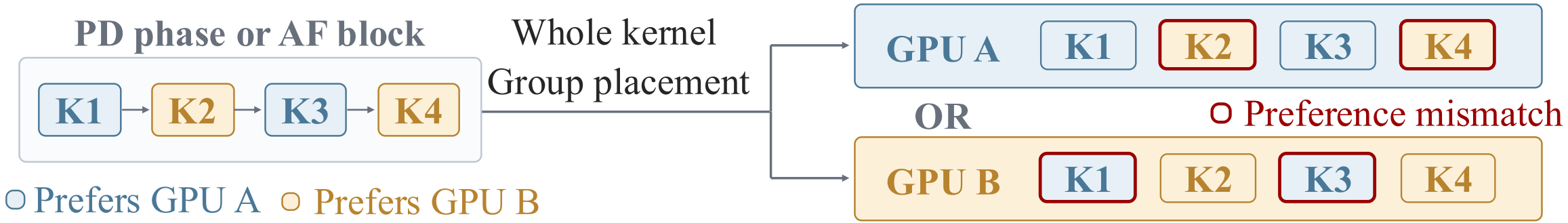
Separate GPU pools run attention and FFN computation. Each layer exchanges intermediate activations

Limitations of existing disaggregation approaches

These disaggregation approaches have two main limitations:

1 Coarse granularity leaves application–GPU mismatches

The coarse disaggregation granularity of existing approaches can not fully match applications' requirements with heterogenous hardware features.

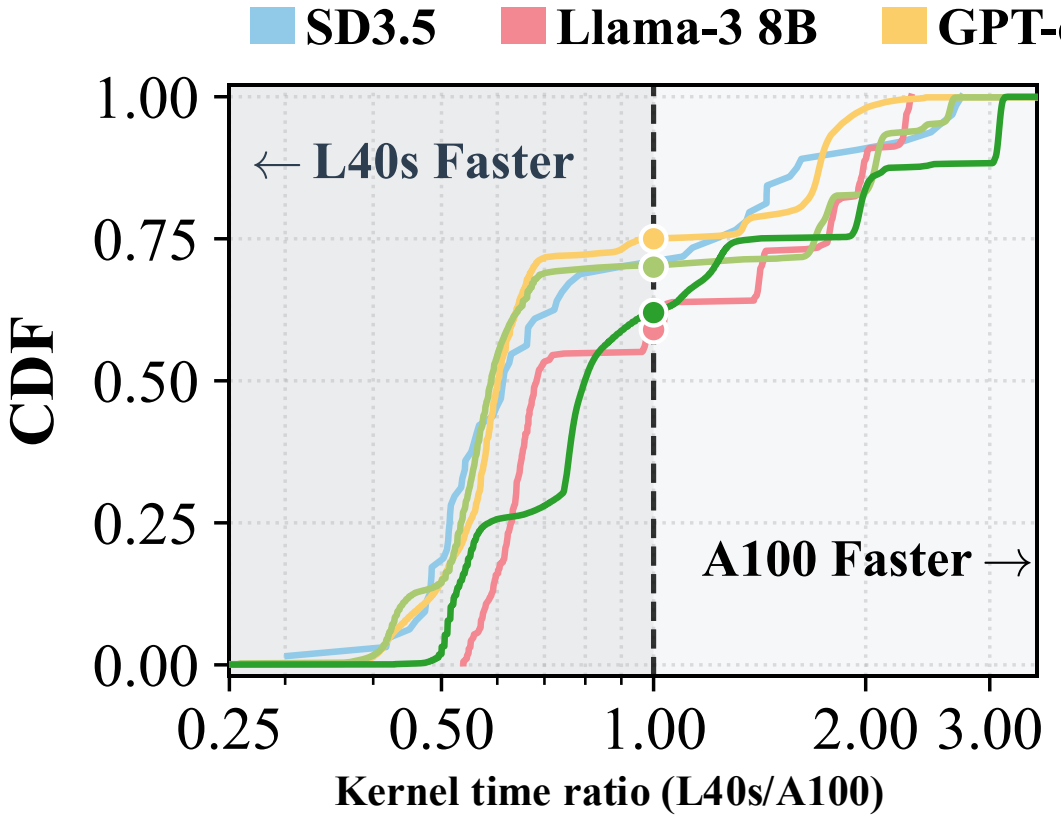


2 Poor generality across model architectures

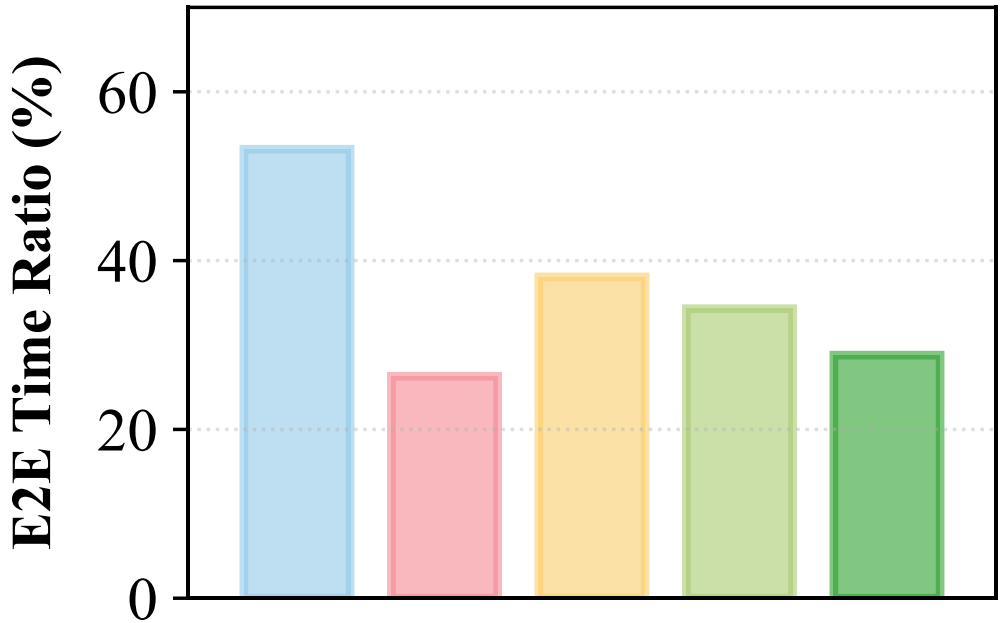
Either PD disaggregation or AF disaggregation is only suitable with limited model architectures.

Kernel heterogeneity of different models

We profile kernels on A100 and L40s using five models spanning LLMs, SSMs, MLLMs and diffusion.



(a) Kernel count statistics

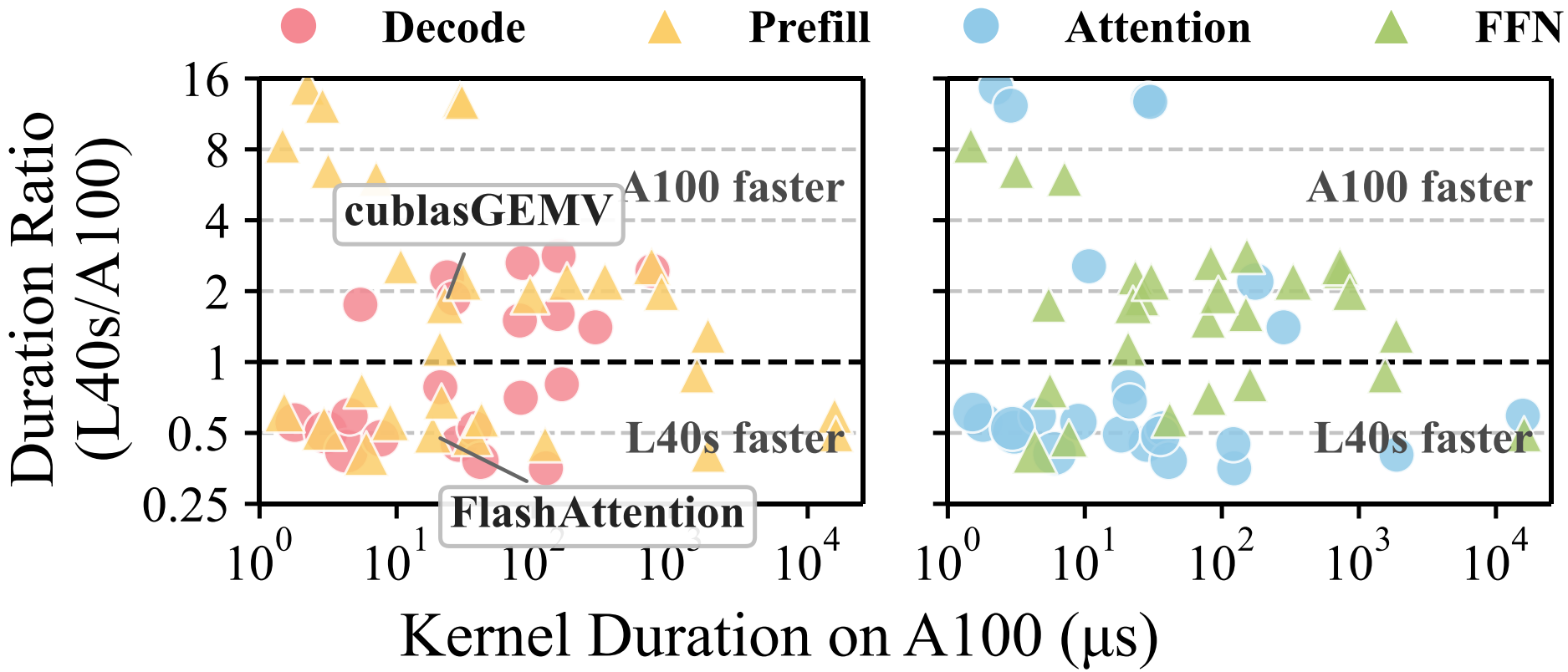


(b) E2E A100 time share of L40s-faster kernels

The heterogeneity of applications is, in essence, the heterogeneity of kernels.

Coarse granularity leaves application–GPU mismatches

We profile GPT-oss 20B kernels on A100 and L40s, grouping them by prefill/decode phases and attention/FFN blocks.



Existing approaches cannot fully align kernel demands with GPU strengths

Lmited generality in the diverse models

PD and AF disaggregation have limited generality because they rely on model-specific boundaries, but kernel are the common user-level GPU scheduling unit across model architectures

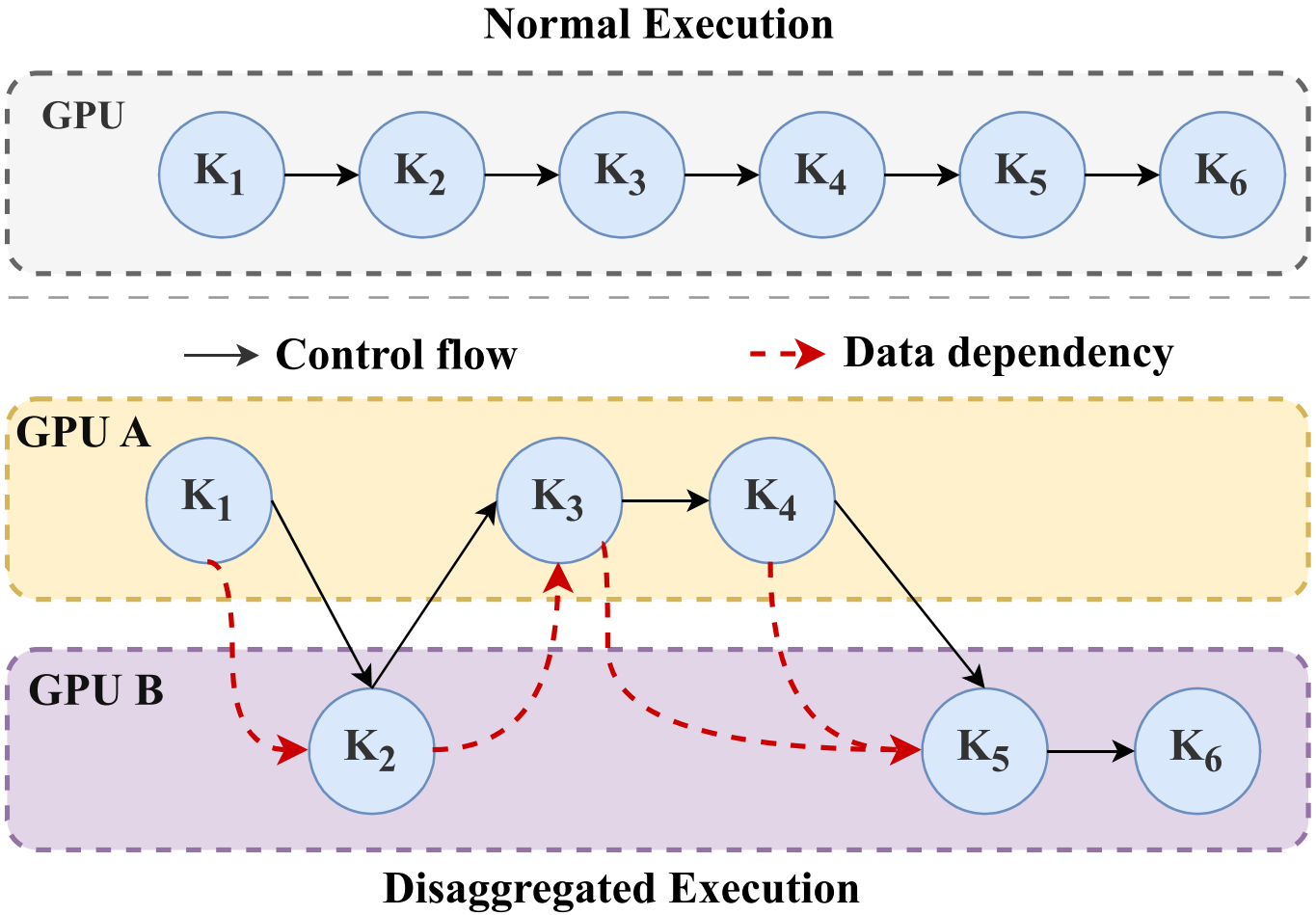
Evaluated model family	PD Dis.	AF Dis.	FluidGPU
LLM / MLLM	✅	✅	✅
SSM (Mamba)	✅	❌	✅
Diffusion (SD3.5)	❌	❌	✅

Key insight:

Align hardware heterogeneity with application heterogeneity at kernel granularity

Kernel disaggregation across heterogeneous GPUs

Place individual kernels on different GPUs while preserving their dependencies.



Place individual kernels

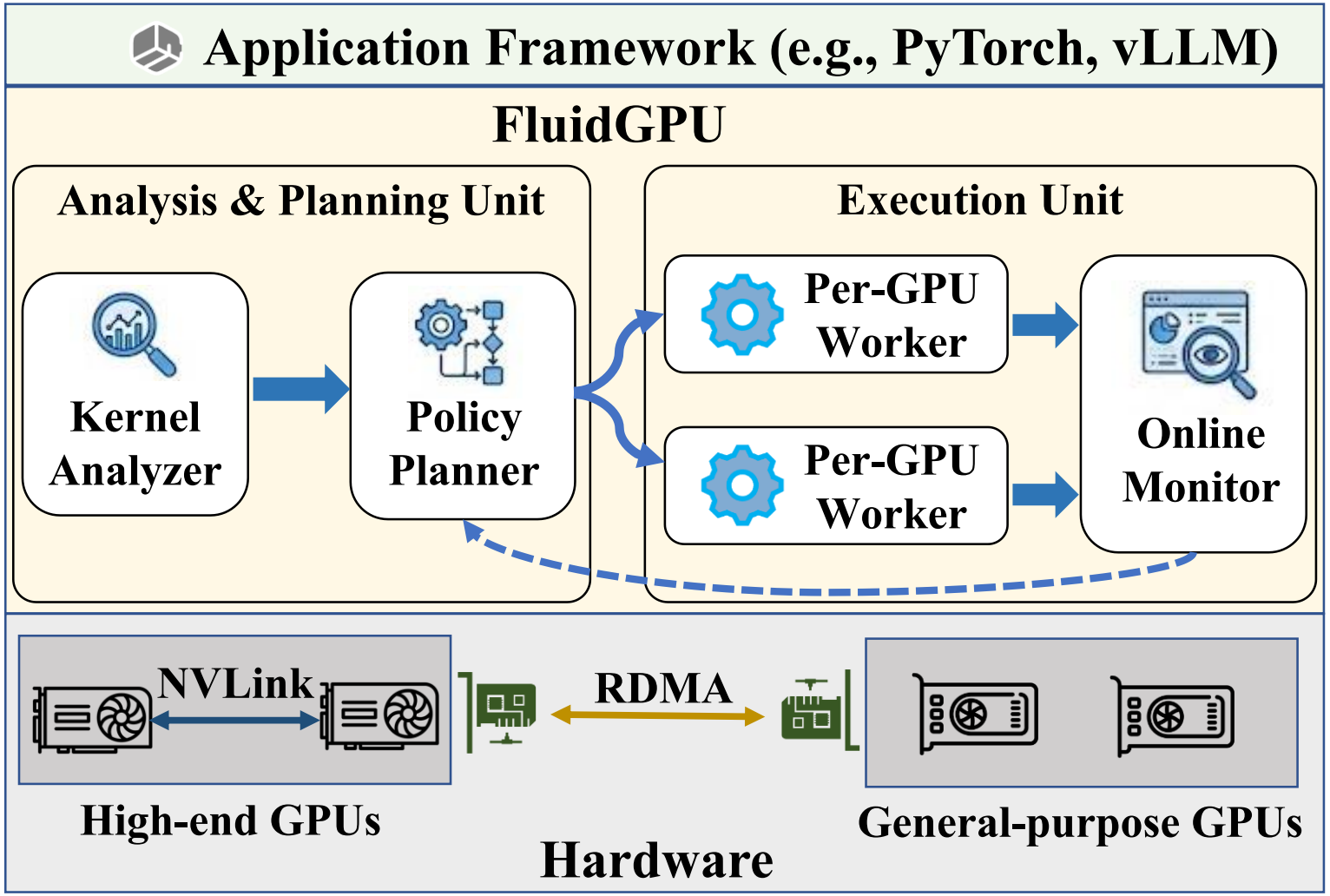
GPU A runs K1, K3 and K4.
GPU B runs K2, K5 and K6.

Preserve data dependencies

K2 reads K1's output.
K3 reads K2's output.
K5 reads outputs of K3 and K4.

Design

FluidGPU system overview



1 Kernel Analyzer

Capture memory access, inter-kernel data dependencies, and kernel latency

2 Policy Planner

Generate kernel scheduling policy

3 GPU Workers

Executes the scheduling policy and orchestrates inter-GPU communication

4 Online Monitor

Collects runtime statistics to support dynamic policy adaptation

Analysis of kernel memory access

北京大学 PEKING UNIVERSITY

Two types of kernels in large model inference

Standard library kernels

```
cublasStatus_t cublasSgemm(
  cublasHandle_t handle,
  cublasOperation_t transa,
  cublasOperation_t transb,
  int m, int n, int k,
  const float *alpha,
  const float *A, int lda,
  const float *B, int ldb,
  const float *beta,
  float *C, int ldc);
```

This kind of kernel's data dependency can be acquired from the API semantics directly

Opaque custom / JIT kernels

Original PTX

```
ld.global.f32 %x, [%addr];
```

↓

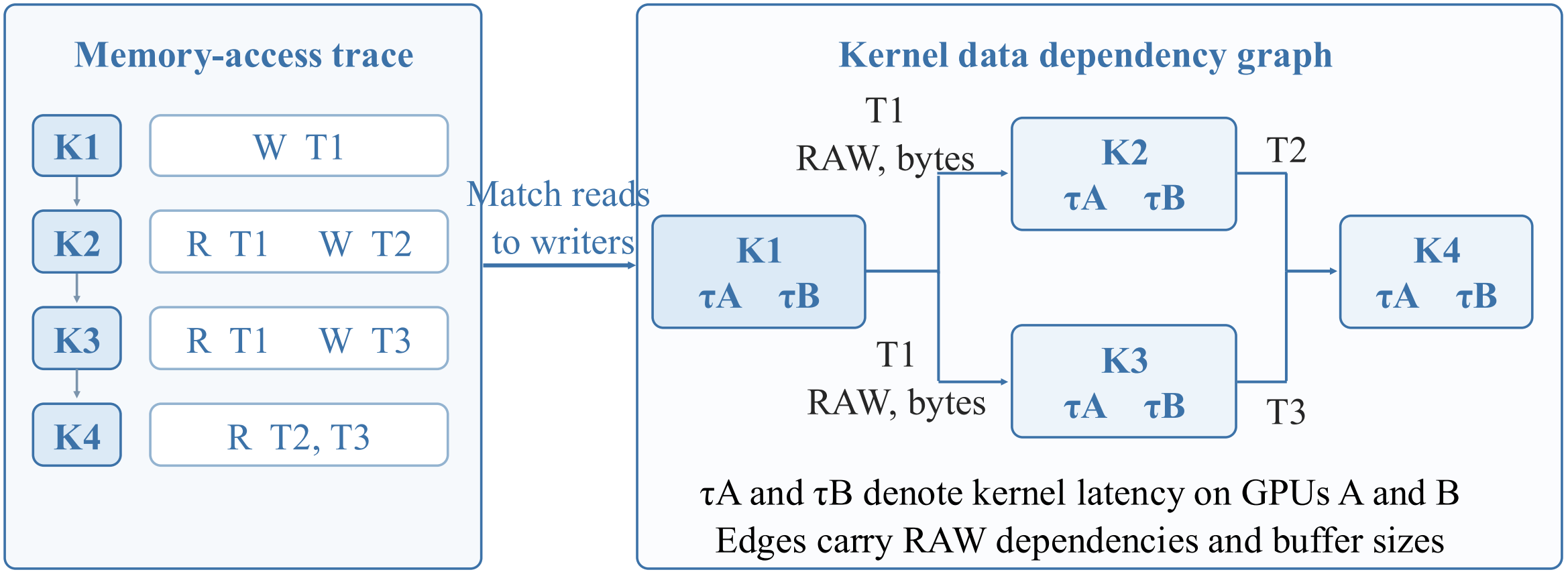
PTX with injected address probes

```
atom.global.min.u64 %t, [%min], %addr;
atom.global.max.u64 %t, [%max], %addr;
ld.global.f32 %x, [%addr];
```

This kind of kernel's data dependency can be acquired by instrumenting opaque-kernel PTX.

Building the kernel data dependency graph

For every read, find the latest preceding writer of the same buffer.



Communication-aware policy planning

Generate a kernel scheduling policy using measured kernel latencies and cross-GPU transfer costs while balancing GPU load.

1 Estimate per-GPU workloads

Each kernel maps to exactly one GPU

Compute $T_g = \sum_k t_{k,g}\, x_{k,g}$

$x \in \{0, 1\}$ whether to place kernel k on GPU g
t = measured kernel latency

Transfer $M_g = \sum_{e \in cut(g)} c_e$

cut(g) contains incoming cross-GPU DDG edges

2 Identify the pipeline bottleneck

Compute and communication overlap in steady state

$$W_g = max(T_g, M_g)$$

The slower component limits each GPU stage

$$\textbf{\textit{Throughput} } \propto 1 / W$$

W is the largest stage time across all GPUs

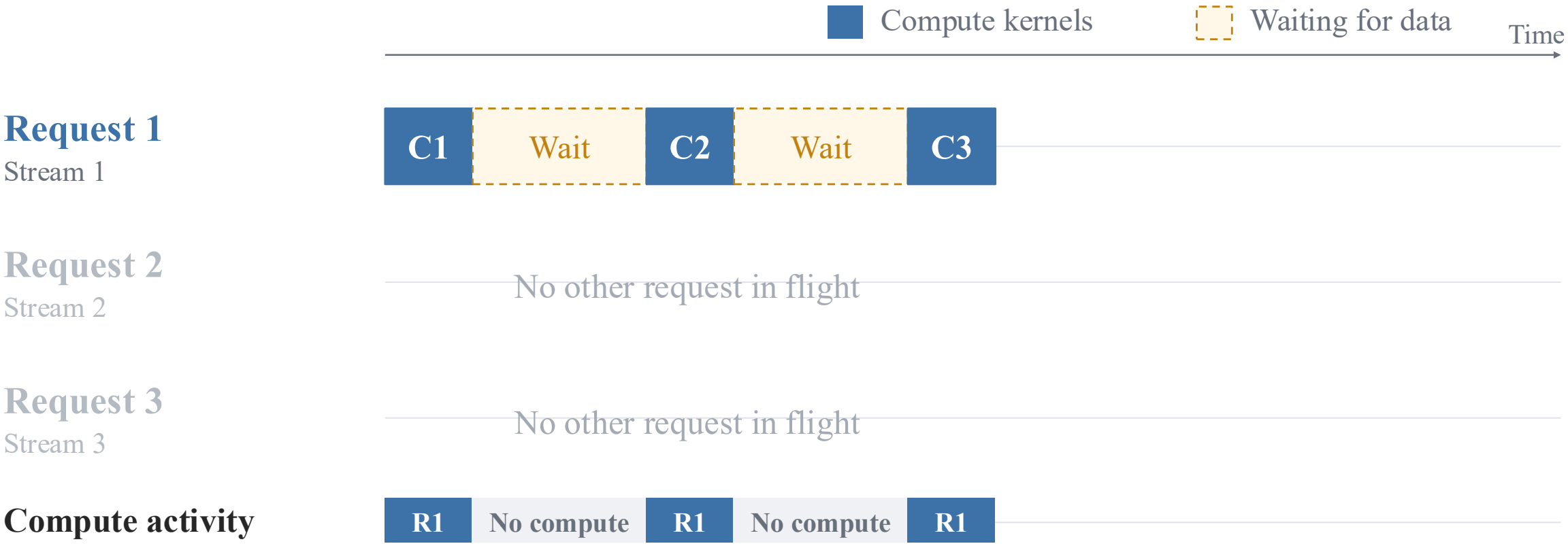
3 Minimize the bottleneck with MILP

$\textbf{\textit{min } W}$ subject to $W \geq T_g$, $W \geq M_g$ for every GPU g

The MILP finds the throughput-optimal placement under the profiled cost model for a fixed hardware configuration.

GPU worker ---- Asynchronous request pipelining

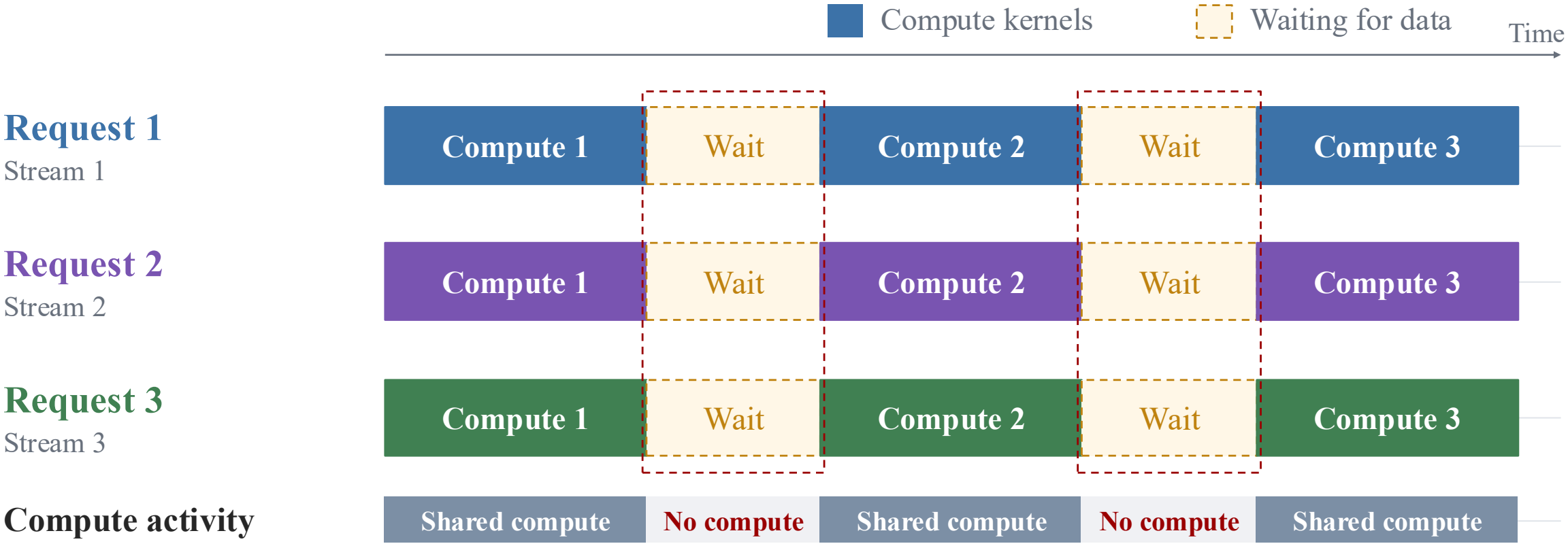
One GPU worker with one active request running on one compute stream.



Without other ready kernels, synchronized stalls leave compute idle

GPU worker ---- Asynchronous request pipelining

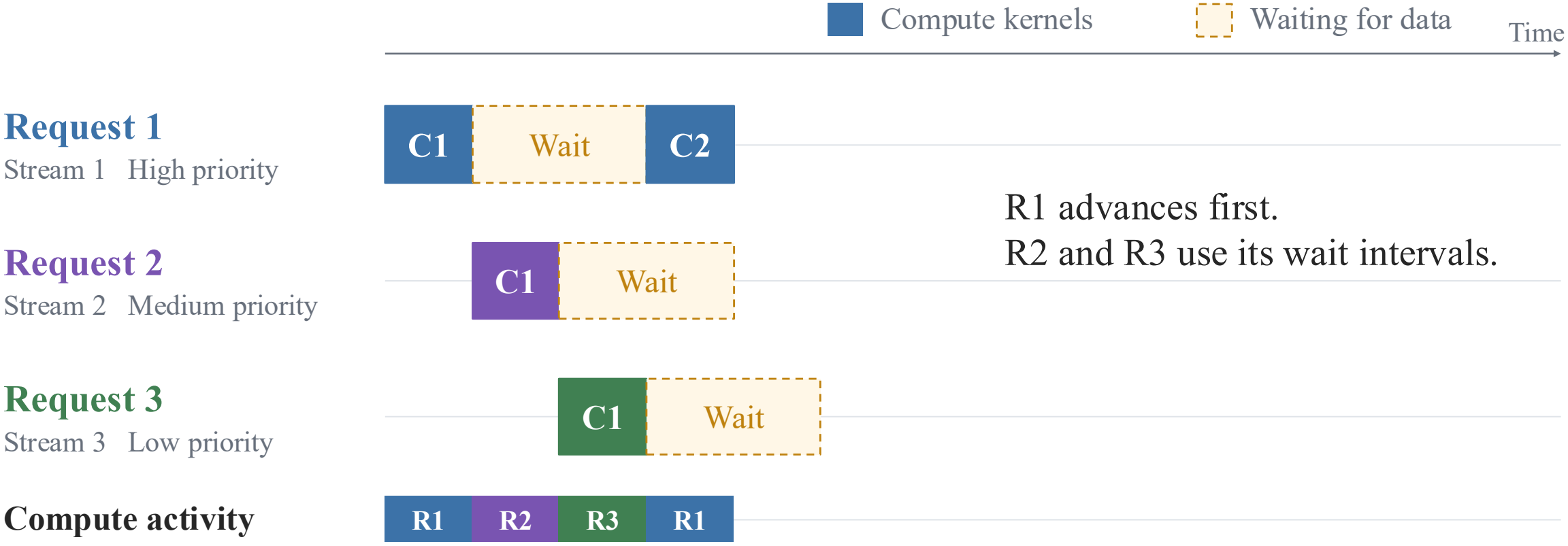
Three concurrent requests on one GPU worker. One compute stream per request with equal priority.



More streams alone do not prevent synchronized stalls

GPU worker ---- Asynchronous request pipelining

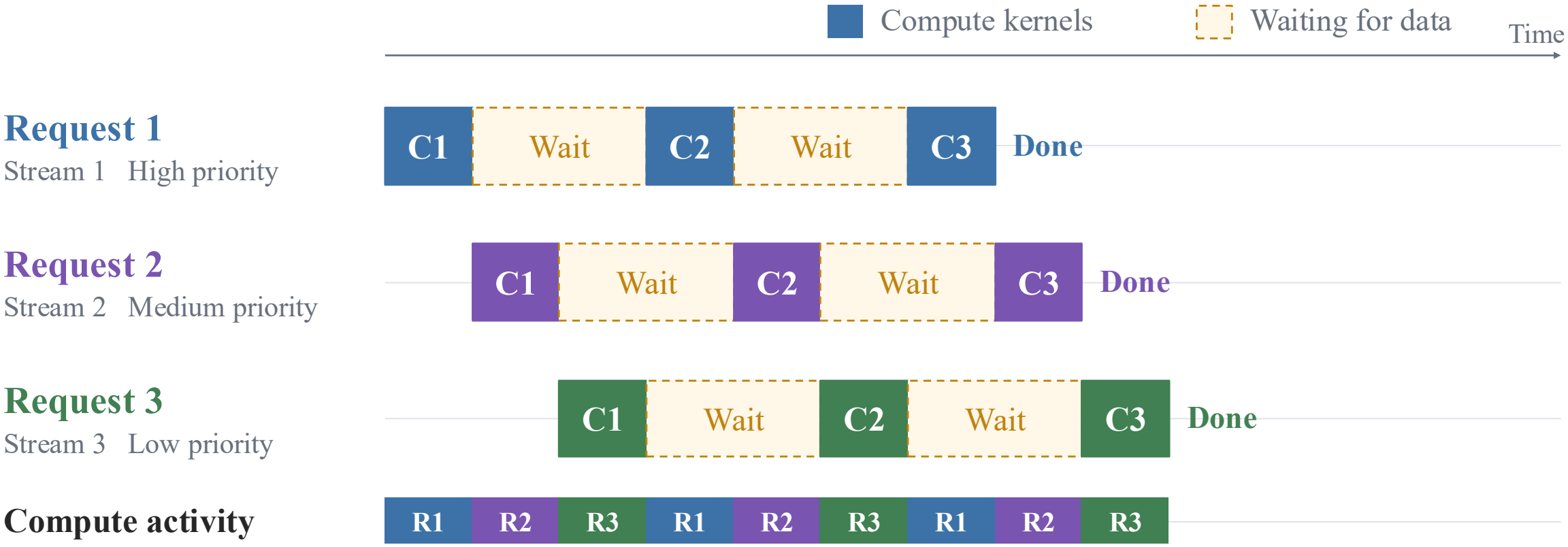
Three concurrent requests on one GPU worker. One compute stream per request with different priorities.



Stream priorities help stagger communication waits across requests

GPU worker ---- Asynchronous request pipelining

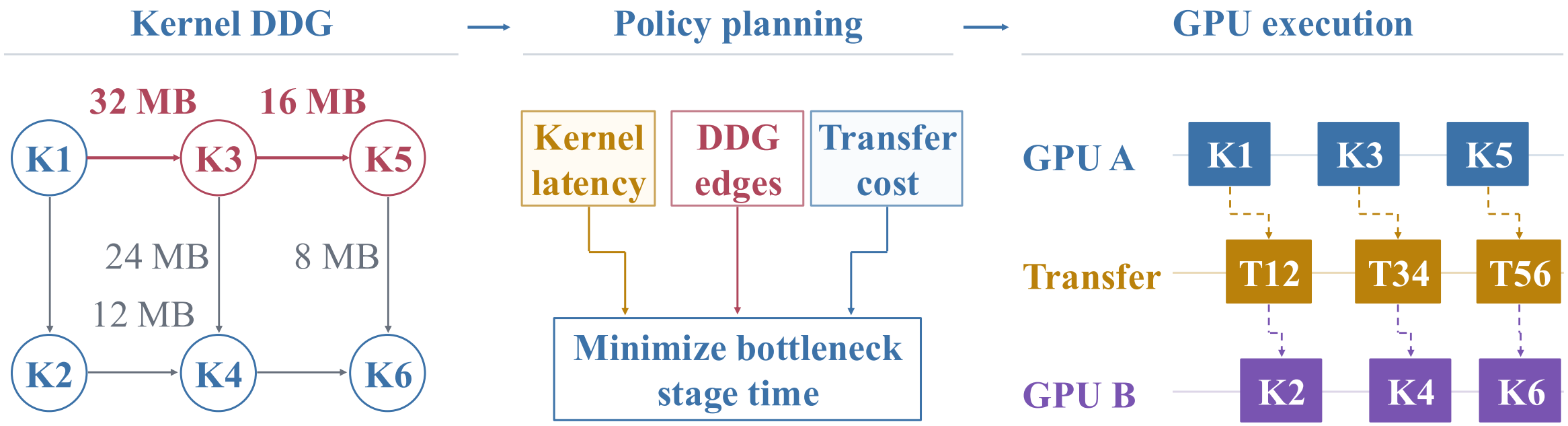
A steady state of GPU worker under asynchronous request pipelining.



When one request waits for remote data, another request runs ready kernels

FluidGPU workflow

FluidGPU builds a DDG from kernel analysis and then computes communication-aware scheduling policy with the consideration of workload balance. GPU workers execute this policy with asynchronous request pipelining.



Evaluation

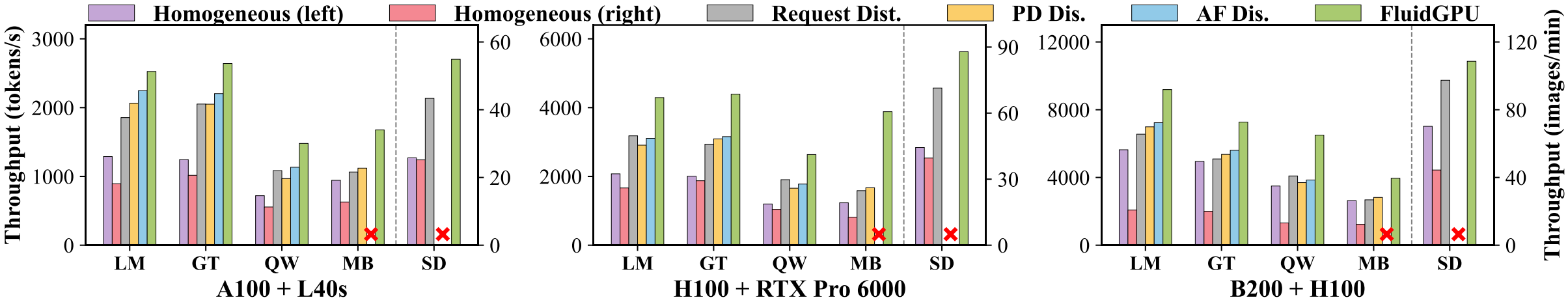
Evaluation setup

北京大学 PEKING UNIVERSITY

Dimension	Configurations
Local GPU pairs	A100 + L40s, H100 + RTX Pro 6000, B200 + H100
Model families	LLM, SSM, multimodal LLM, diffusion
Representative models	Llama-3 8B, GPT-oss 20B, Mamba-Codestral 7B Qwen2.5-VL 7B, Stable Diffusion 3.5
Baselines	Single GPU, PD disaggregation, AF disaggregation Independent request replicas with request distribution
Stack and network	vLLM / PyTorch, NCCL, RDMA

Offline throughput improves over coarse disaggregation

Five workloads across three heterogeneous GPU pairs, throughput higher is better



LM Llama-3 8B GT GPT-oss 20B QW Qwen2.5-VL 7B MB Mamba 7B SD SD3.5

1.5× over PD **1.35× over AF**

Mean throughput across applicable configurations

28.7% more throughput

than independent replicas
GPT-oss 20B on A100 + L40s

Cost efficiency

北京大学 PEKING UNIVERSITY

Normalized Perf/$ under the paper's GPU rental-cost model, higher is better

	A100+L40s	H100+RTX Pro 6000	B200+H100
Homo. (left)	1.00	1.00	1.00
Homo. (right)	1.18	2.01	0.78
PD Dis.	0.87	1.00	0.70
AF Dis.	1.02	1.07	0.74
FluidGPU	**1.21**	**1.64**	**1.01**

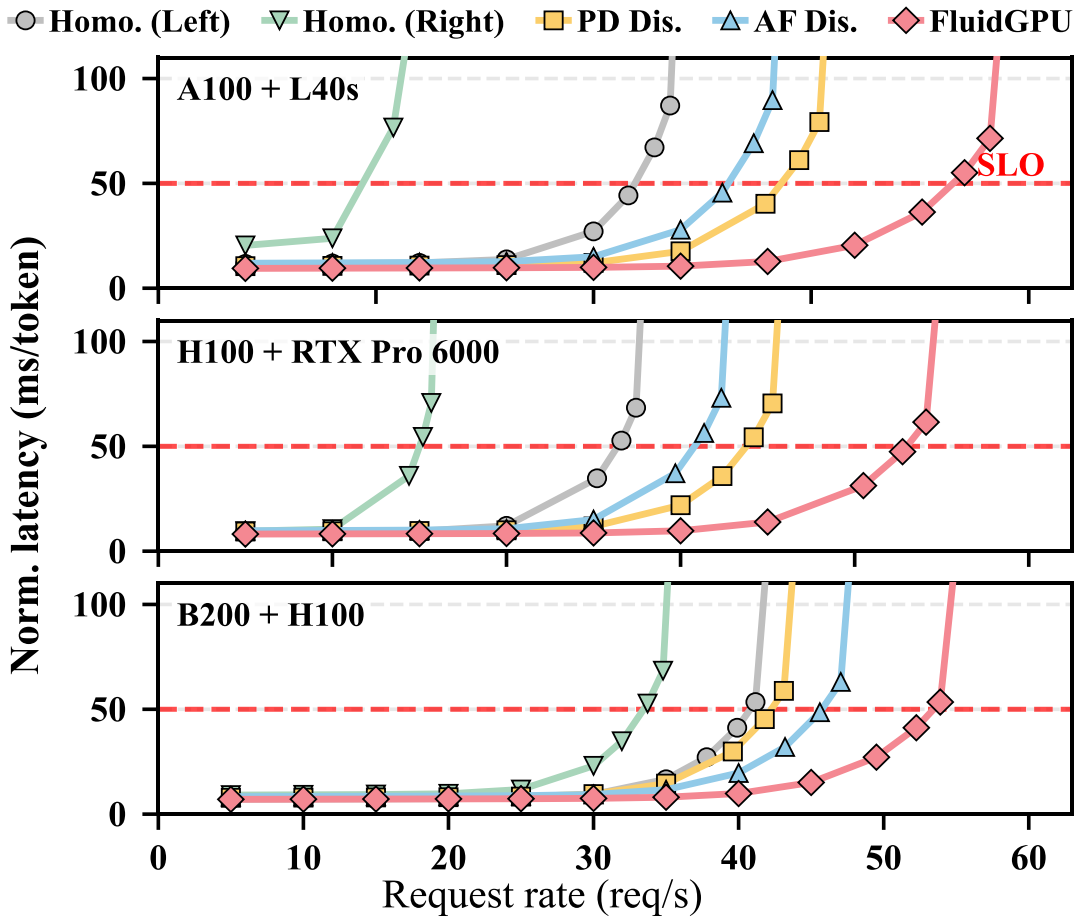
Up to 1.6× over PD

Up to 1.5× over AF

Highest cost efficiency among the evaluated disaggregation baselines

Online serving sustains a higher request rate

Online serving under varied request rate with poisson distribution, latency lower is better



Up to 1.3×

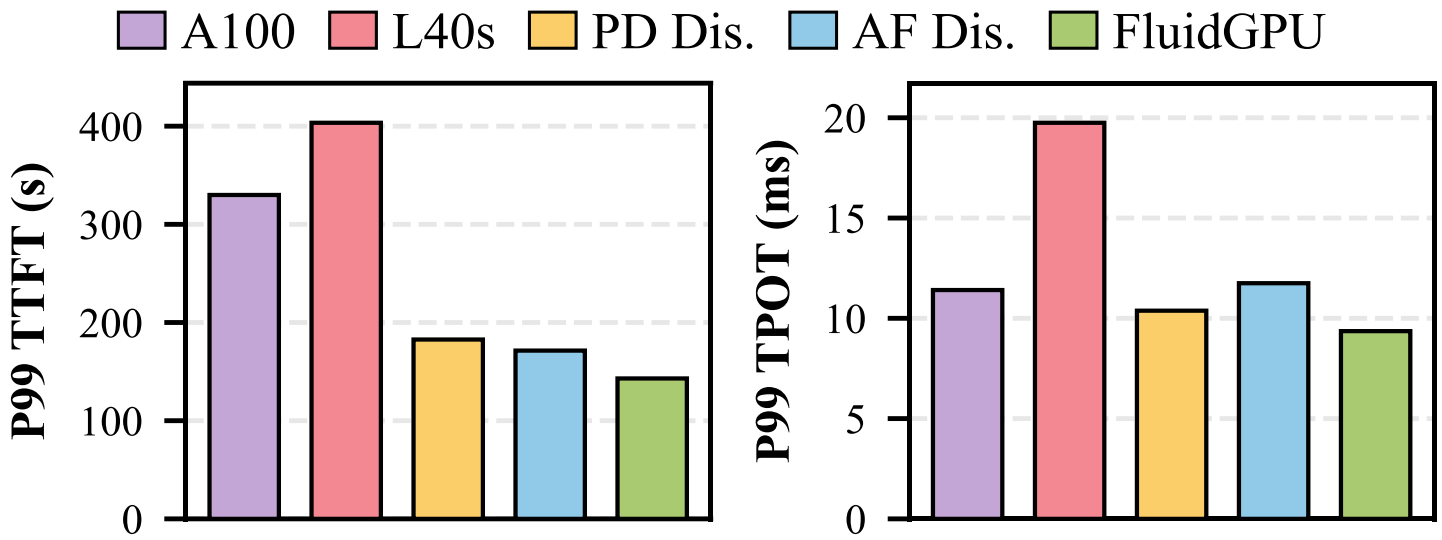
the best baseline's request rate
at the same 50 ms/token SLO

At low request rates

PD latency / FluidGPU: 1.3×
AF latency / FluidGPU: 1.2×

Tail latency

TPOT and TTFT of GPT-oss 20B on A100+L40s, lower is better



Baseline / FluidGPU

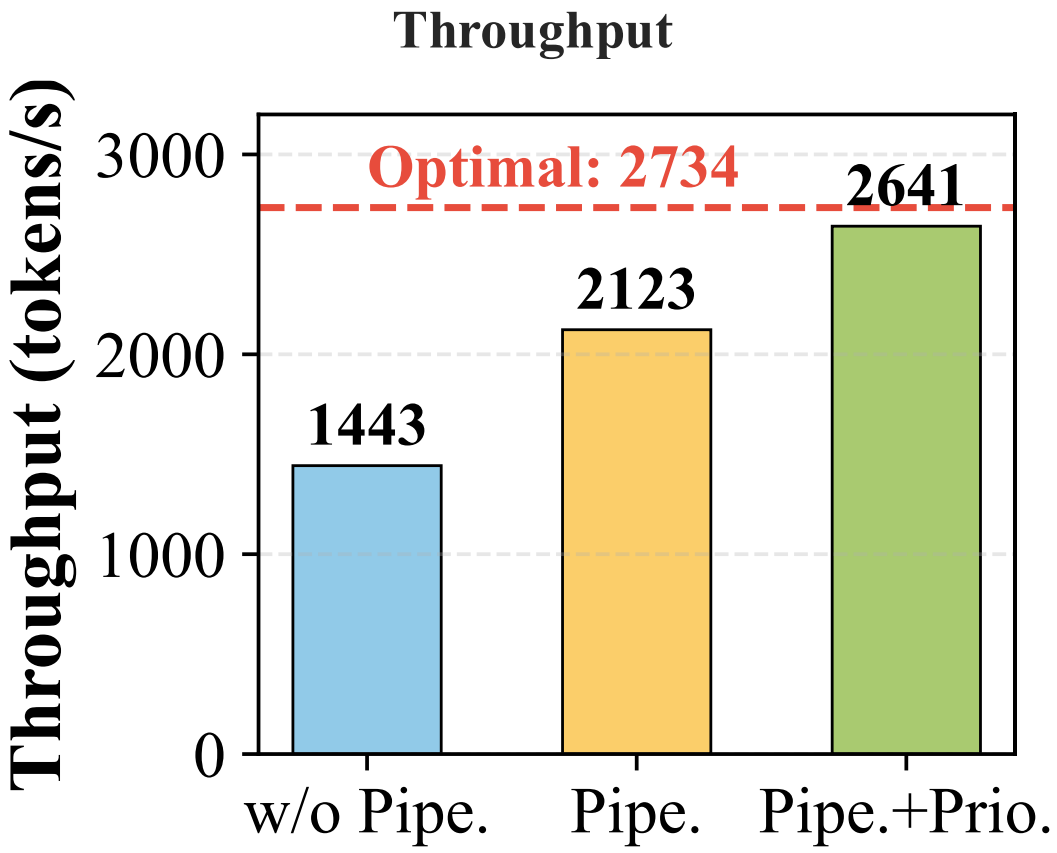
Baseline	P99 TTFT	P99 TPOT
PD	1.28×	1.11×
AF	1.20×	1.26×

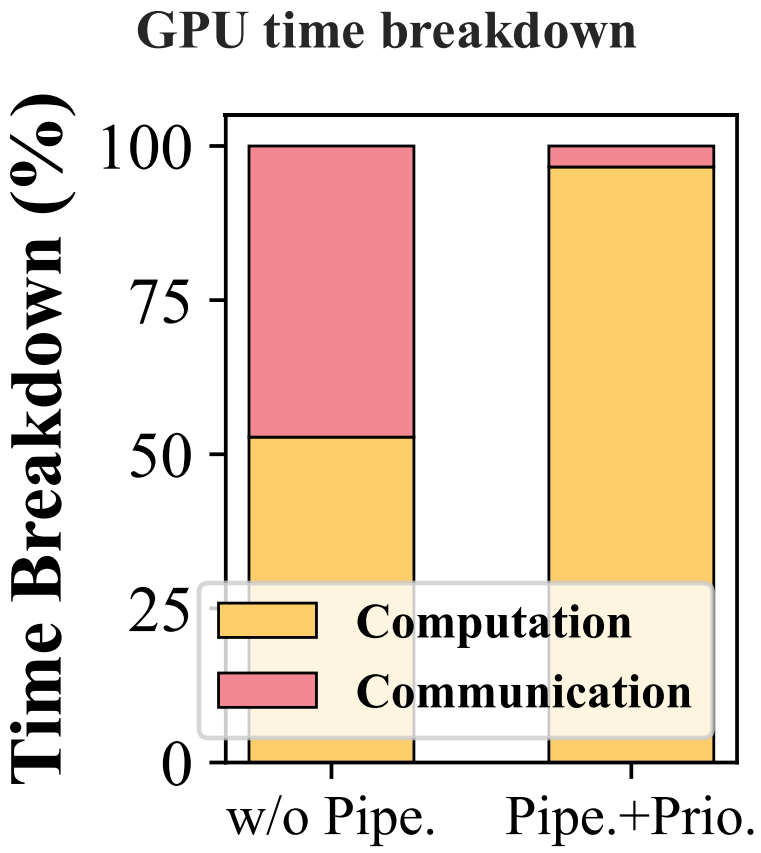
TTFT: higher throughput reduces queueing delay.

TPOT: kernel placement reduces request execution time.

Pipelining and priorities realize the placement gains

Ablation study of asynchronous pipelining of GPT-oss 20B on A100 + L40s





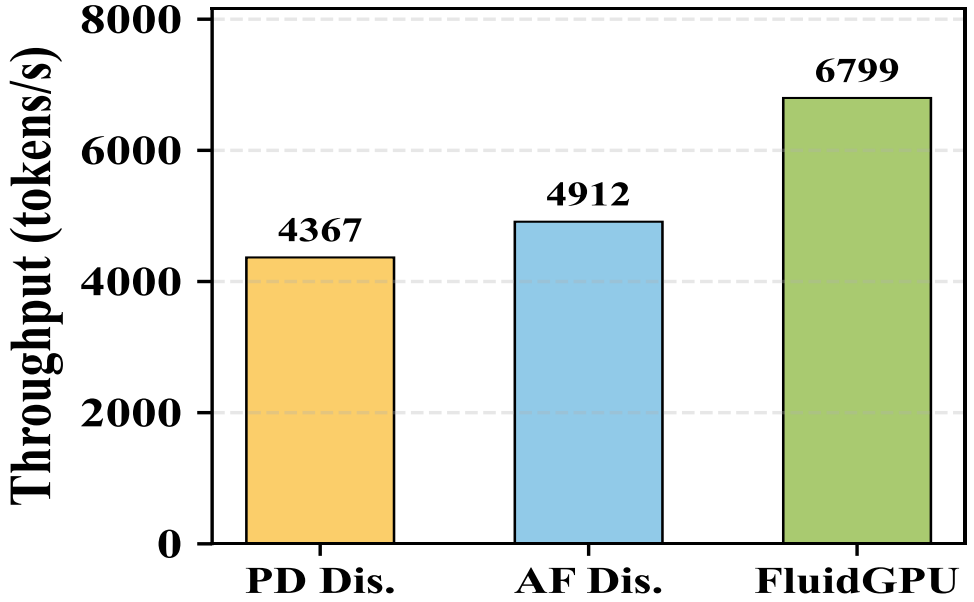
Pipelining with priorities reaches 96.6% throughput of the optimal

Scalability and network sensitivity

北京大学 PEKING UNIVERSITY

Scalability

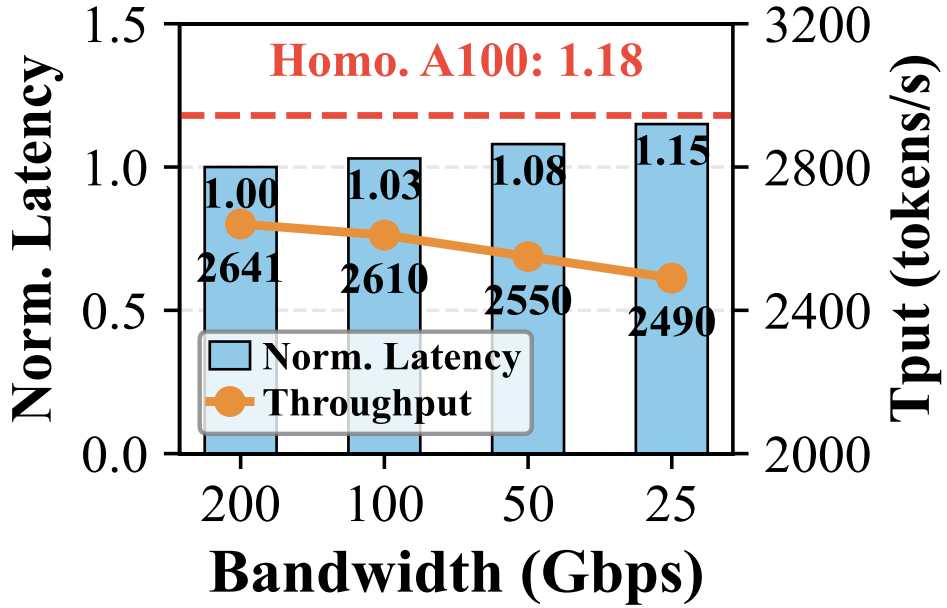
Qwen3 235B, 8 B200 + 8 H100
TP = 8, cross-node RDMA



Kernel disaggregation composes with TP on 16 GPUs

Network sensitivity

GPT-oss 20B, A100 + L40s



At 25 Gbps, less than 6% offline throughput loss

Key takeaways



北京大学
PEKING UNIVERSITY

😟 **Motivation** Coarse-grained disaggregation approaches fail to match application heterogeneity with GPU heterogeneity and limit generality.

💡 **Key Insight** Aligning application demands with heterogeneous GPU capabilities at the kernel-level granularity.

⚙️ **Design** Automatic API/PTX dependency analysis, communication-aware MILP placement, and priority-aware asynchronous request pipelining.

📊 **Evaluation** Up to 2.3× throughput and 1.6× Perf/$ improvement compared with the state-of-the-art baselines.

github.com/ICTPLSys/FluidGPU ✉ tiancheng.hu@stu.pku.edu.cn