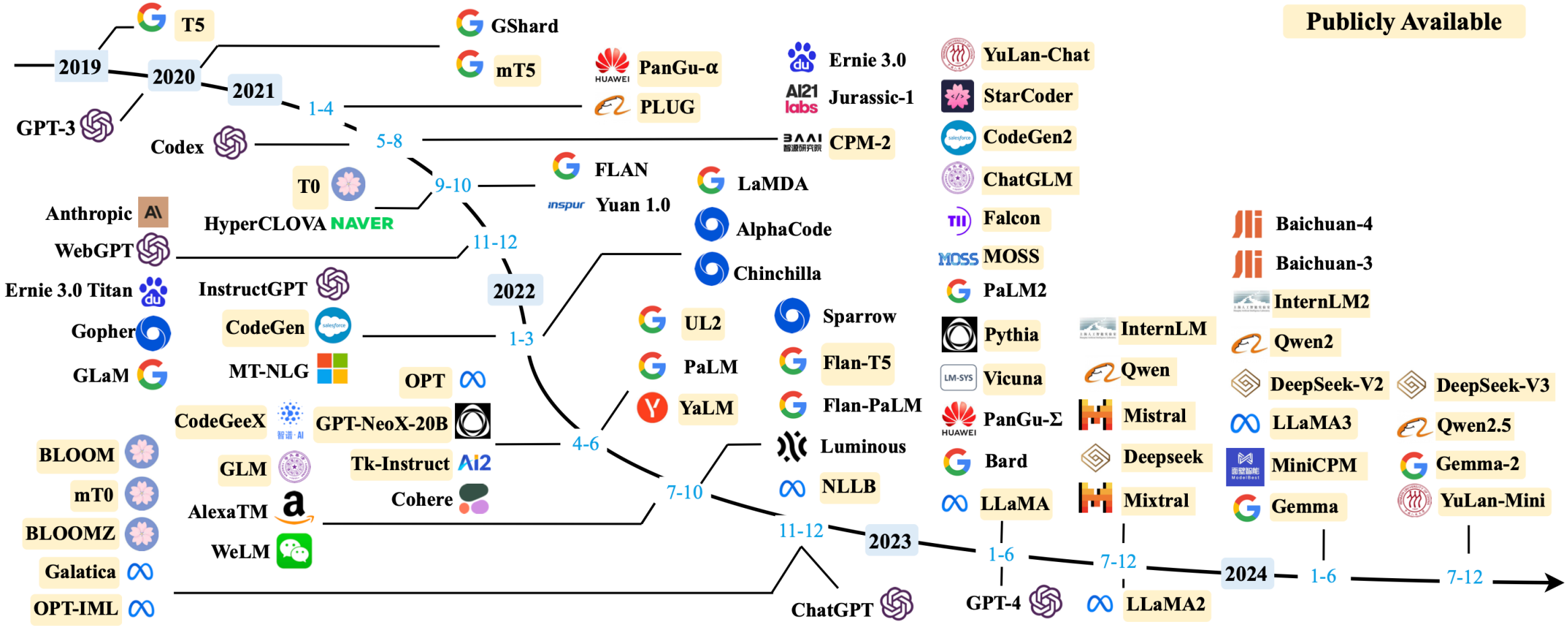
HydraServe: Minimizing Cold Start Latency for Serverless LLM Serving in Public Clouds

Chiheng Lou[1], Sheng Qi[1], Chao Jin[1], Dapeng Nie[2],
Haoran Yang[2], Yu Ding[2], Xuanzhe Liu[1], Xin Jin[1]

[1]School of Computer Science, Peking University [2]Alibaba Group

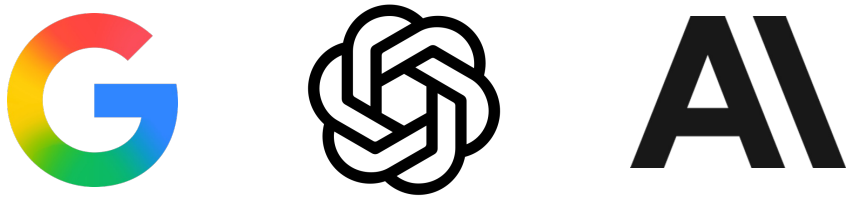
Proliferation of LLMs



Zhao et al., "A Survey of Large Language Models," arXiv:2303.18223.

LLM Serving in Public Cloud

- Models from anywhere
  - Closed-source models



  - Open-source models

  - Custom fine-tuned models

Hugging Face

Long-Tail Models

- Hugging Face hosts 2M+ models
- Top 0.01% models get ~50% downloads*

- Long-tail models in public clouds**
  - Limited requests
  - SLO-tolerant

*Ghosh et al., "State of Open Source on Hugging Face: Spring 2026".
**Duan et al., "Muxserve: Flexible Spatial-Temporal Multiplexing for Multiple LLM Serving," in ICML24.

Long-Tail Models

- Hugging Face hosts 2M+ models
- Top 0.01% models get ~50% downloads

- Long-tail models in public clouds
  - Limited requests
  - SLO-tolerant

- Serverless inference: **auto-scale** GPUs based on traffic
  - Pay only for active inference time
  - Ideal for long-tail models with sporadic traffic

Problem

- **Typical LLM Serving:**

Request Arrival ⟶ Inference ⟶ Response

Problem

- **Typical LLM Serving:**

Request Arrival ⟶ Inference ⟶ Response

- **Serverless:** reclaim instance during idle

Problem

- **Typical LLM Serving:**

Request Arrival ⟶ Inference ⟶ Response

- **Serverless:** reclaim instance during idle
  - **Cold Start:** create instance from scratch

Request Arrival ⟶ Instance Setup ⟶ Inference ⟶ Response

Problem

- **Typical LLM Serving:**

Request Arrival ⟶ Inference ⟶ Response

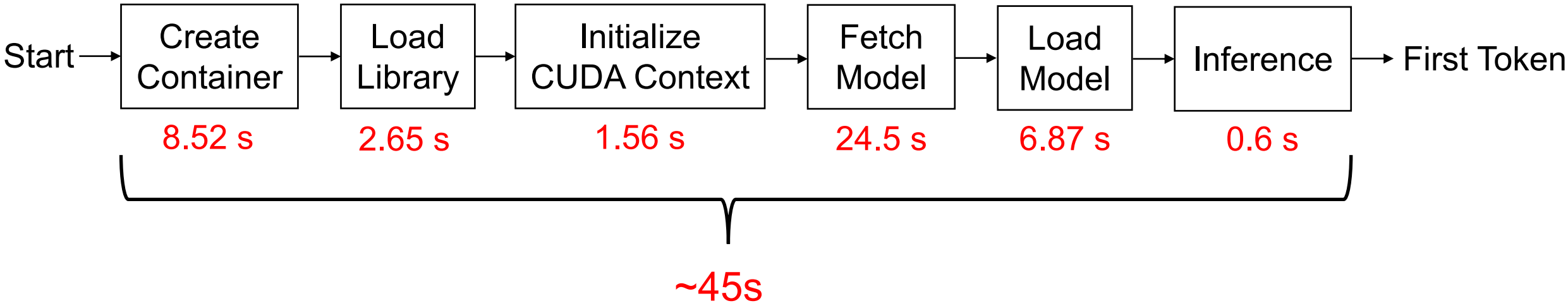
- **Serverless:** reclaim instance during idle
  - **Cold Start:** create instance from scratch

Request Arrival ⟶ Instance Setup ⟶ Inference ⟶ Response

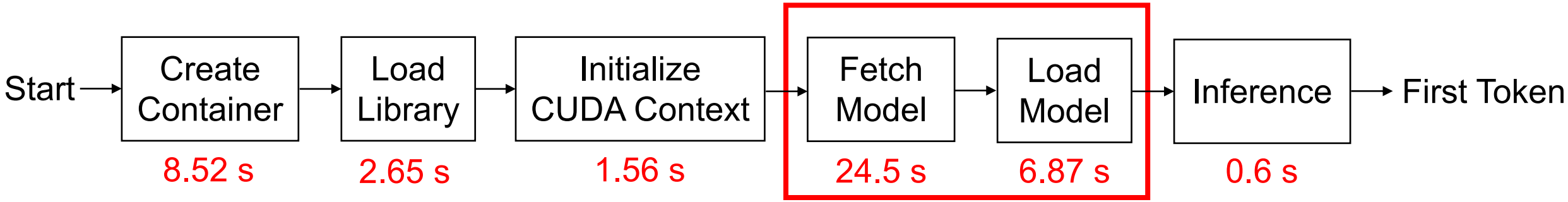
- **Extreme** cold-start latency (up to 45s)

Cold Start Breakdown

- vLLM + Llama2-7B on A10 GPU

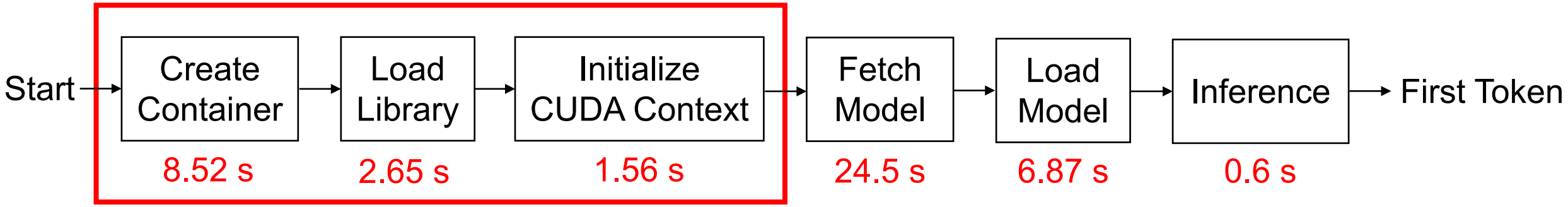


Cold Start Breakdown



- Challenge 1: Limited network bandwidth per model
  - Multiple users share GPU server
  - Serverless targets cost-efficient users ➡ Adopt economical setup

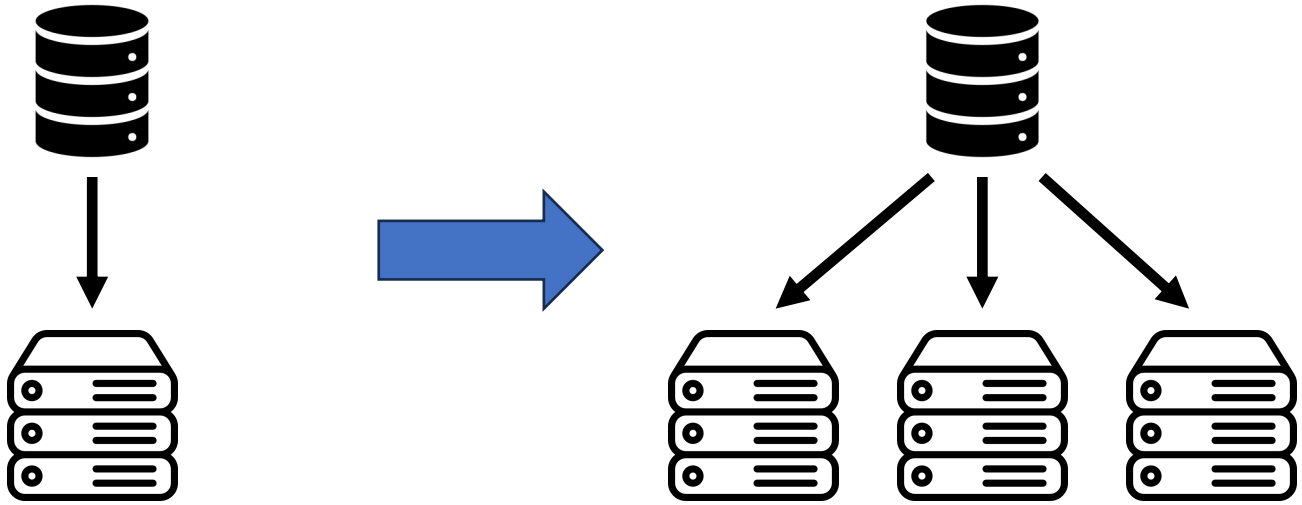
Cold Start Breakdown



- Challenge 2: Complex environment
  - Public clouds isolate users with containers
  - LLM serving framework requires diverse libraries

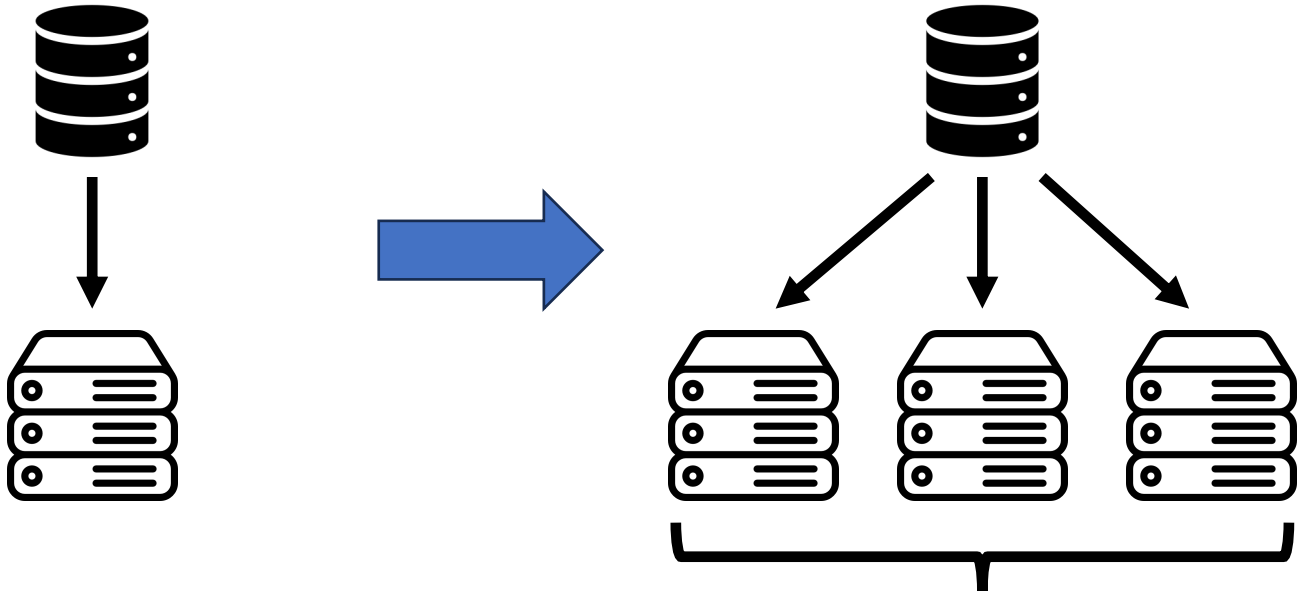
Insight

- **Insight 1: Aggregate bandwidth across servers**
  - Model fetching is bottlenecked at GPU server
  - Leverage multiple servers: each fetches a fraction of the model in parallel



Insight

- **Insight 1: Aggregate bandwidth across servers**
  - Model fetching is bottlenecked at GPU server
  - Leverage multiple servers: each fetches a fraction of the model in parallel



Pipeline Parallelism

Why Pipeline Parallelism

Why Pipeline Parallelism

Minimal inter-node comm. cost

Why Pipeline Parallelism

Minimal inter-node comm. cost

Easy to adopt for various model arch.

Why Pipeline Parallelism


Minimal inter-node comm. cost


Easy to adopt for various model arch.


Increased inference latency

Why Pipeline Parallelism


Minimal inter-node comm. cost


Easy to adopt for various model arch.

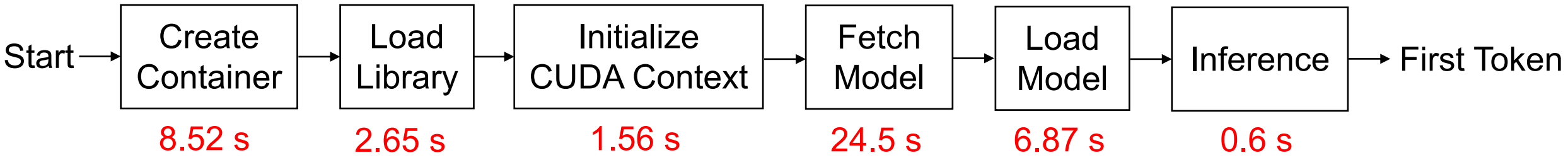

Increased inference latency

Pipeline Consolidation

*Merge pipeline stages **in background** after cold start*

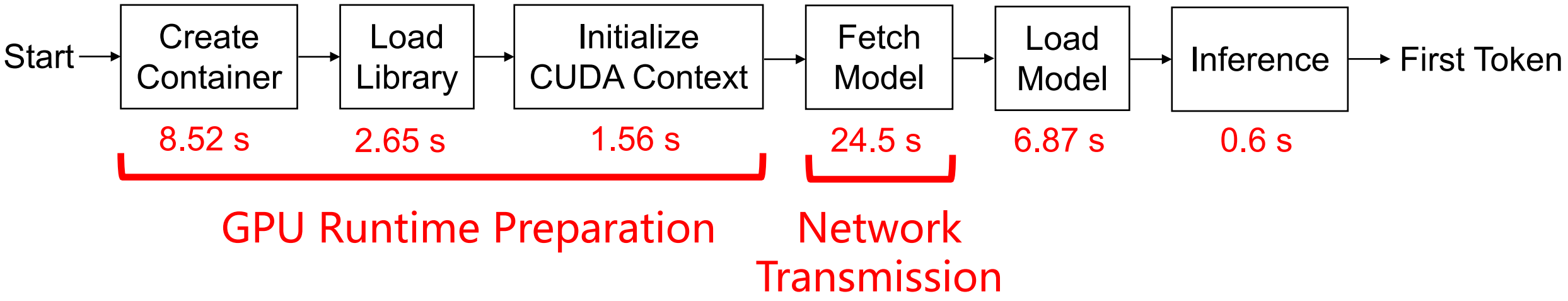
Insight

- **Insight 2: Overlap cold-start stages**



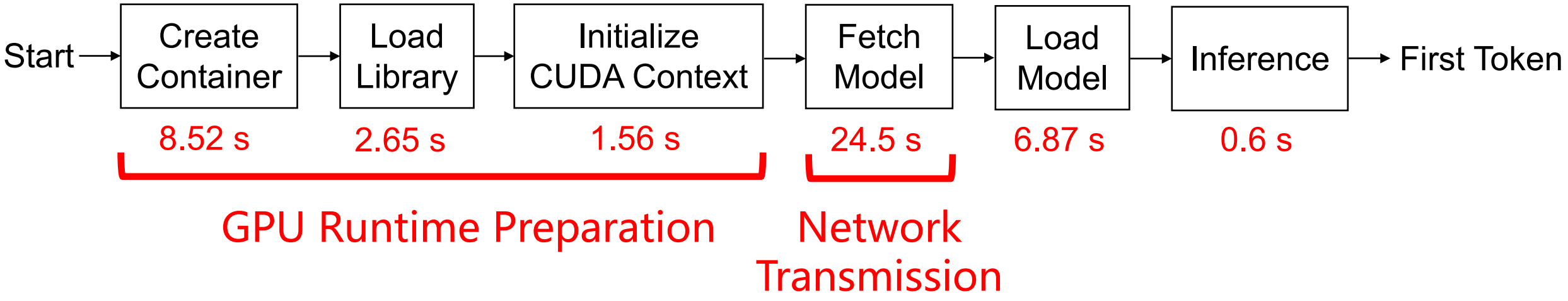
Insight

- **Insight 2: Overlap cold-start stages**



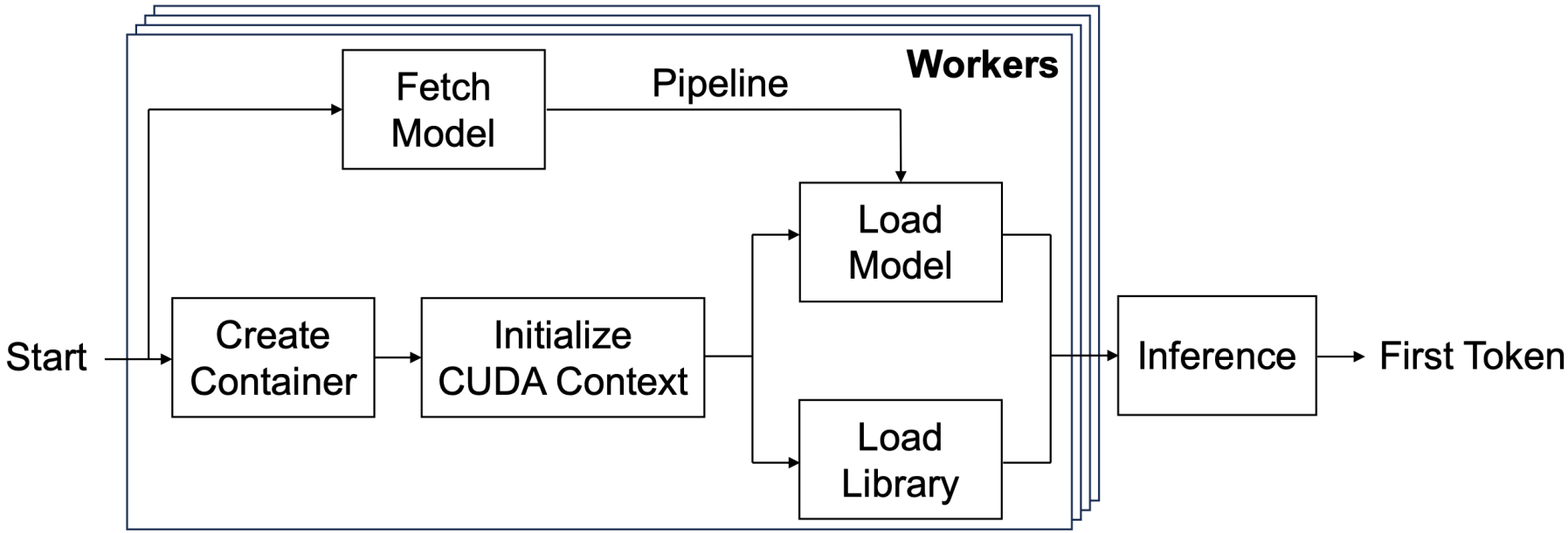
Insight

- **Insight 2: Overlap cold-start stages**
  - Cold-start stages have weak dependencies

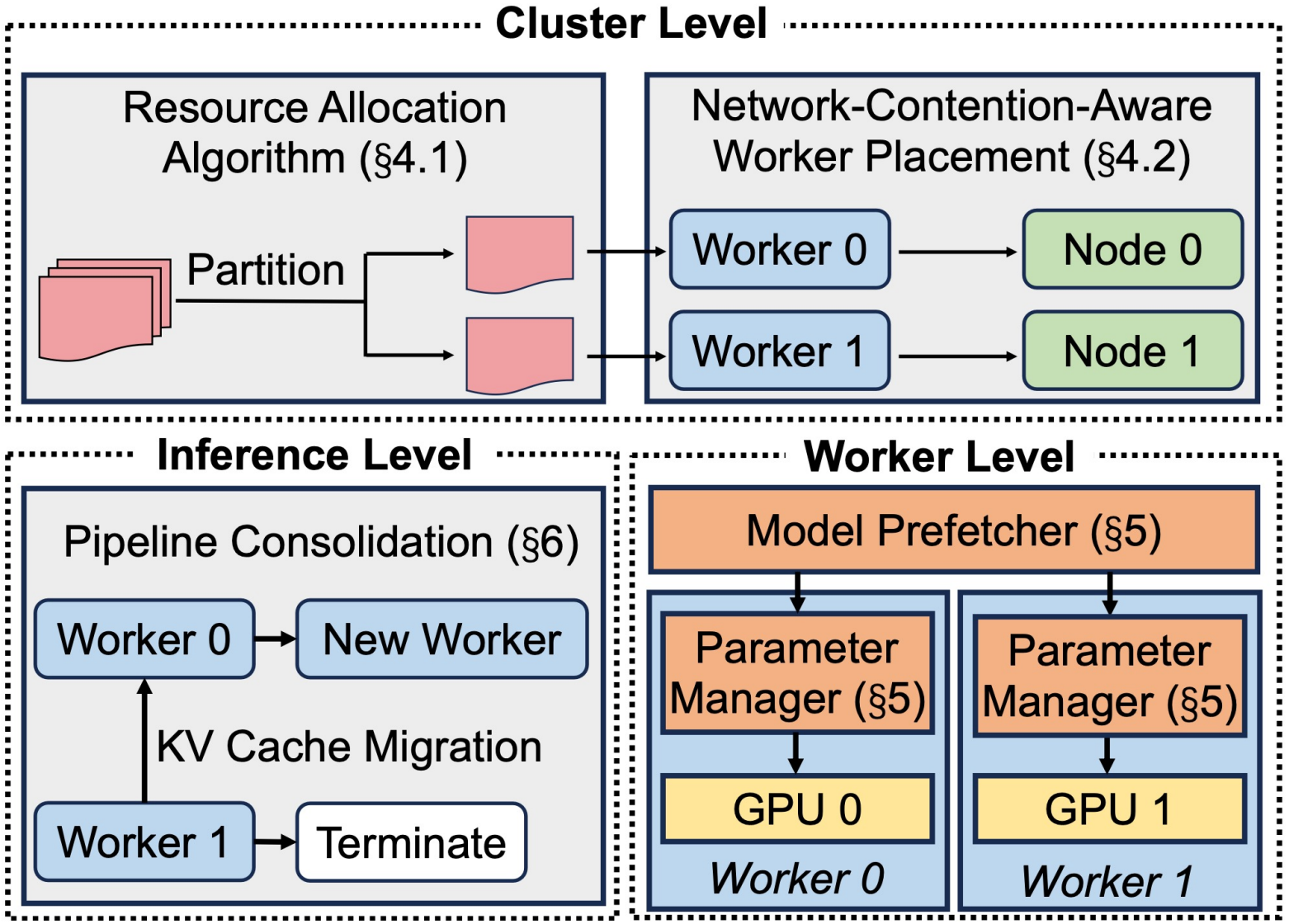


Insight

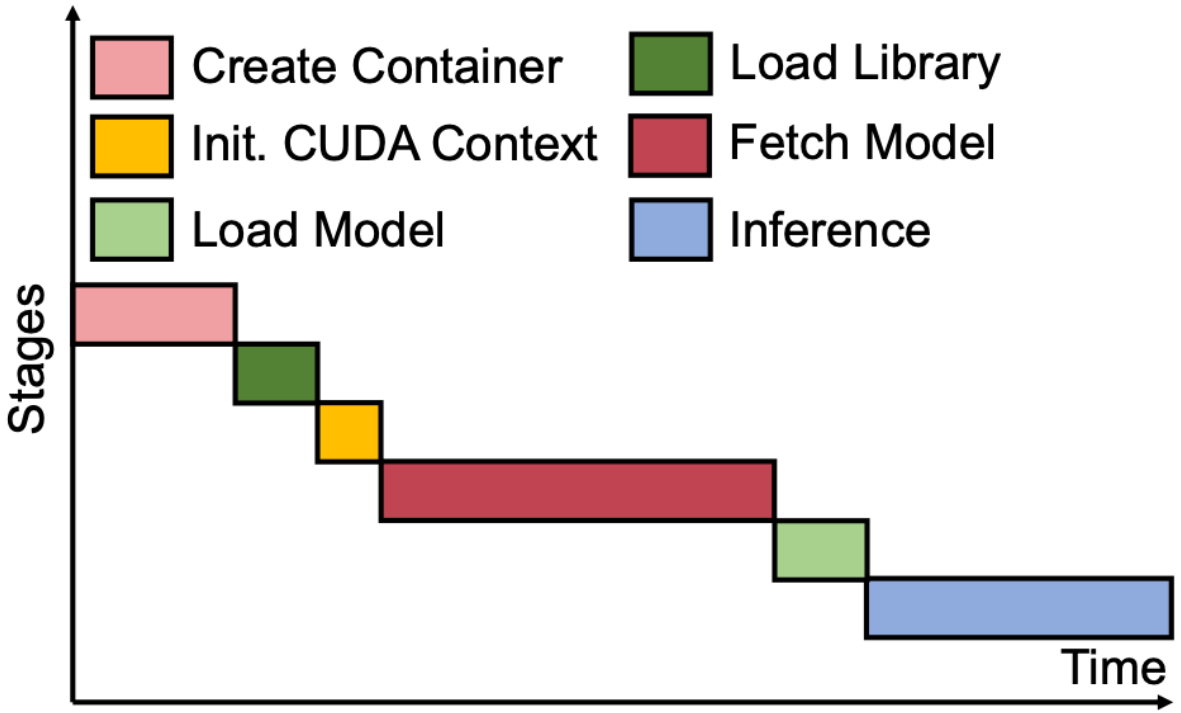
- **Insight 2: Overlap cold-start stages**
  - Cold-start stages have weak dependencies
  - Perform best-effort overlapping



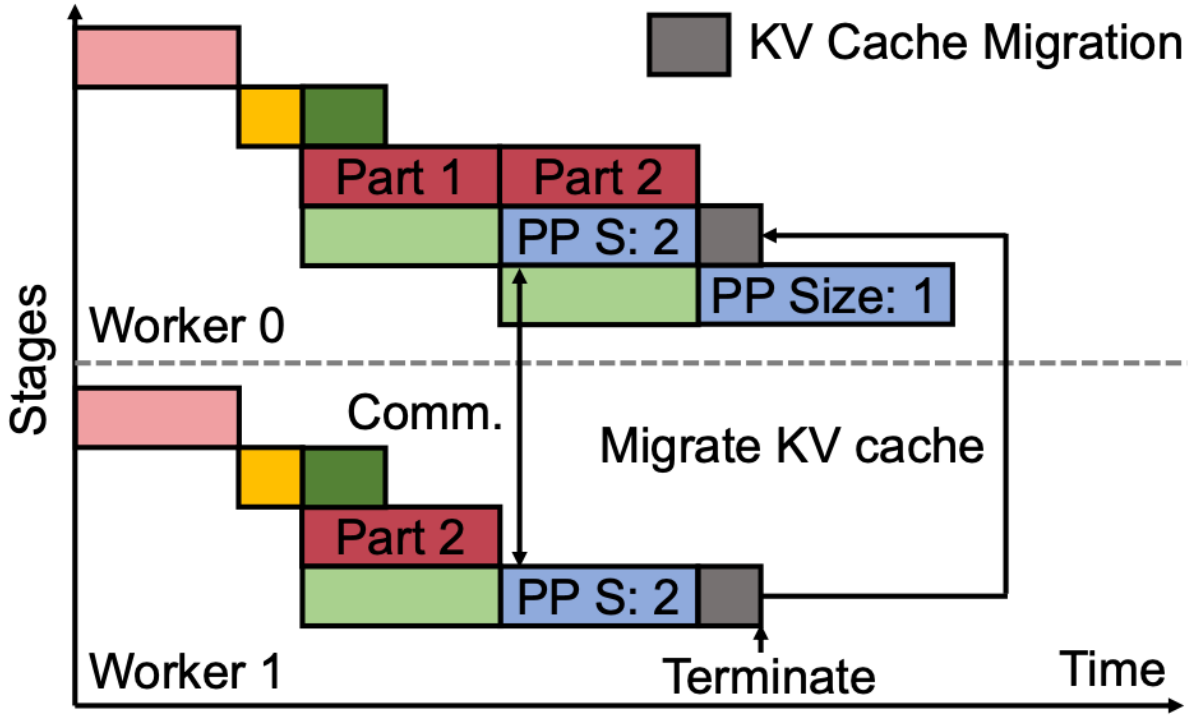
HydraServe Architecture



Introduce Pipeline Parallelism

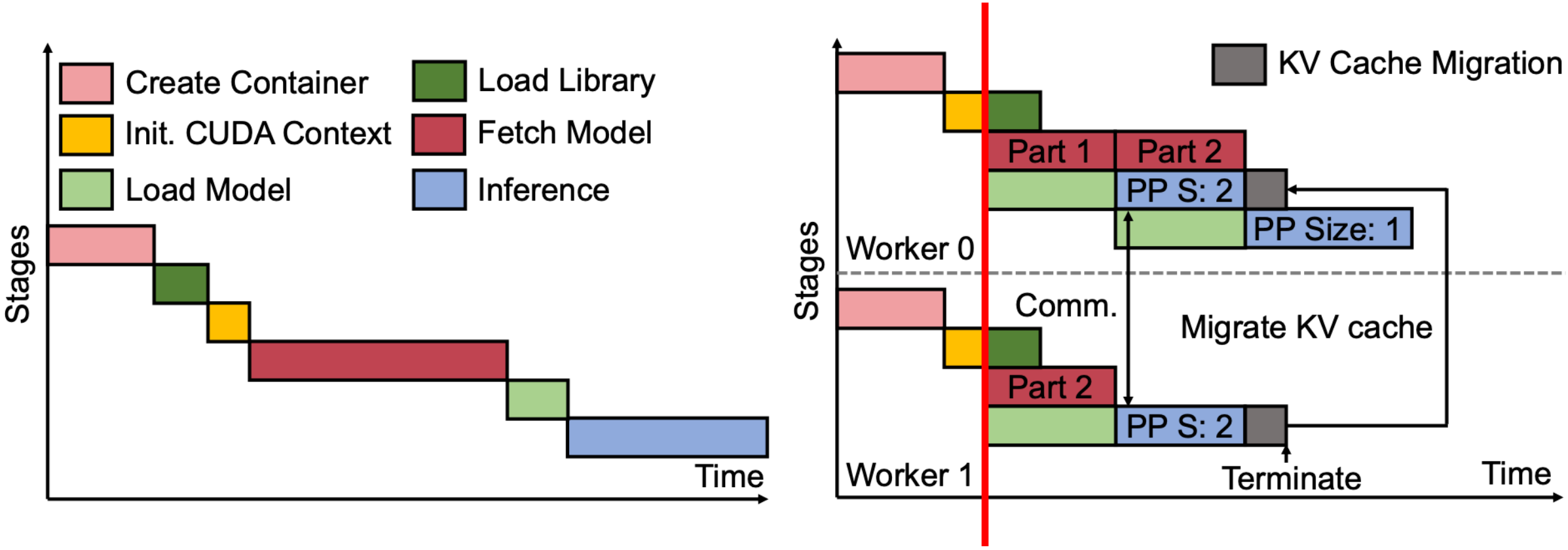


Normal Cold Start



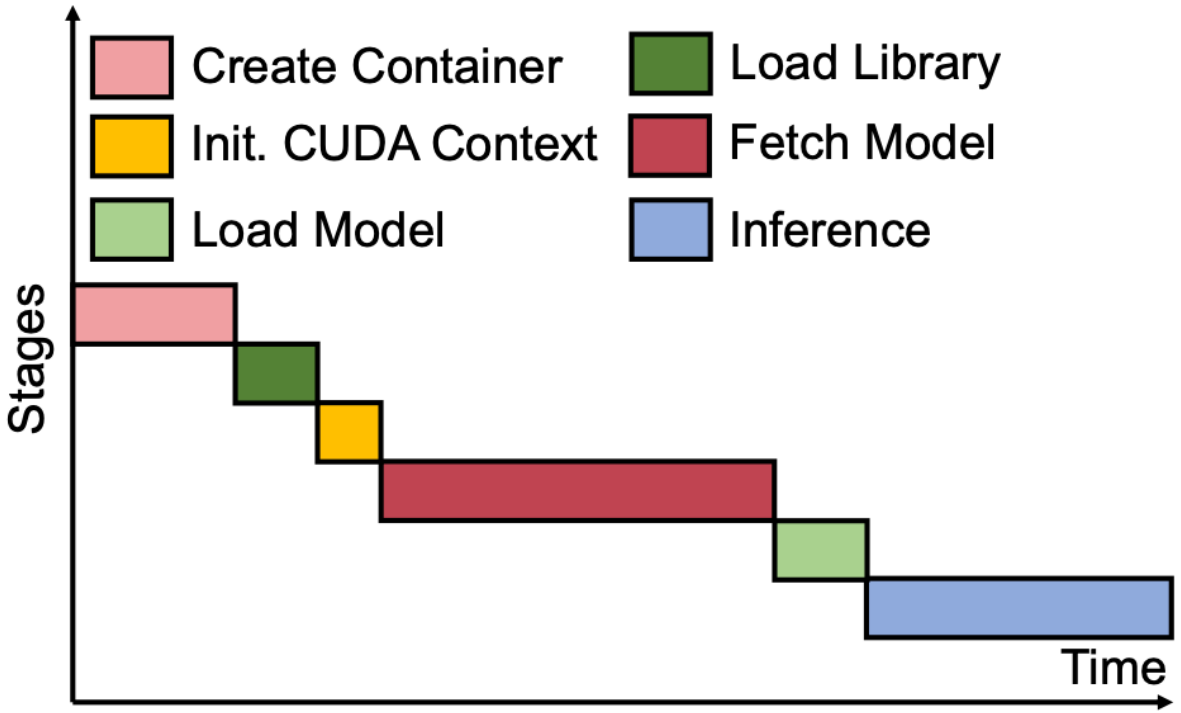
Cold Start with Pipeline Parallelism

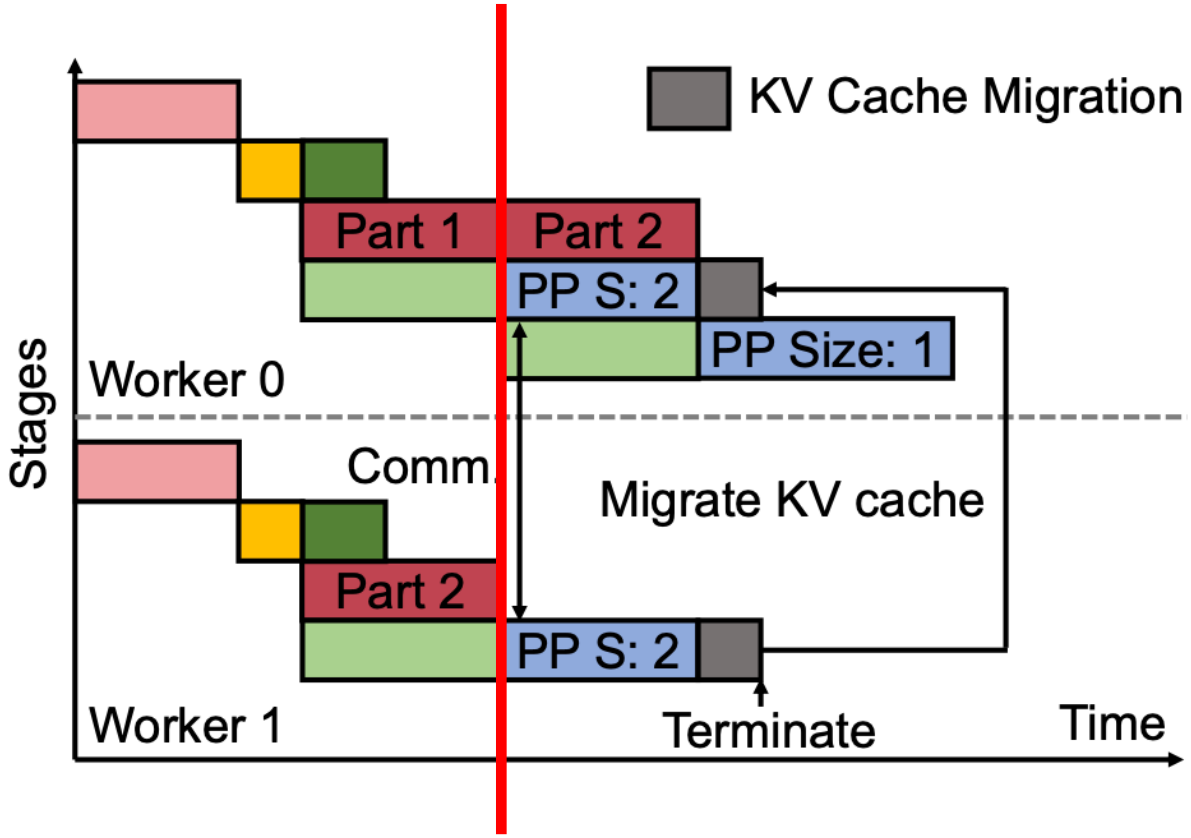
Introduce Pipeline Parallelism



Fetch models in parallel

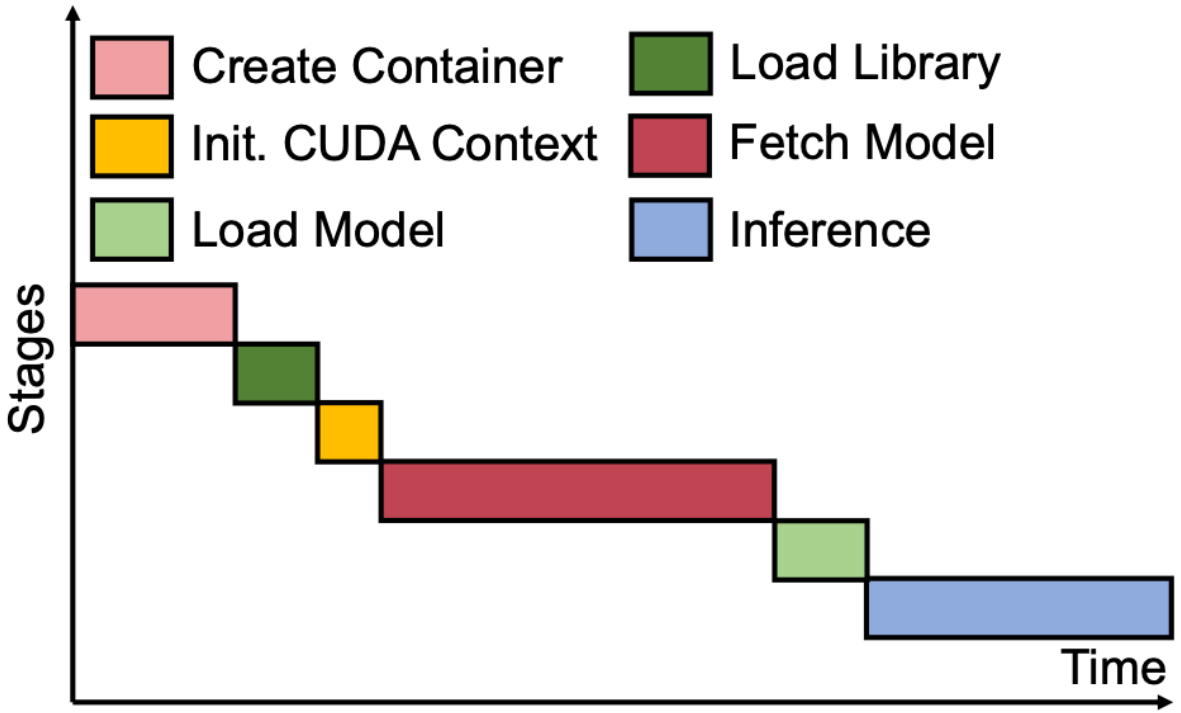
Introduce Pipeline Parallelism

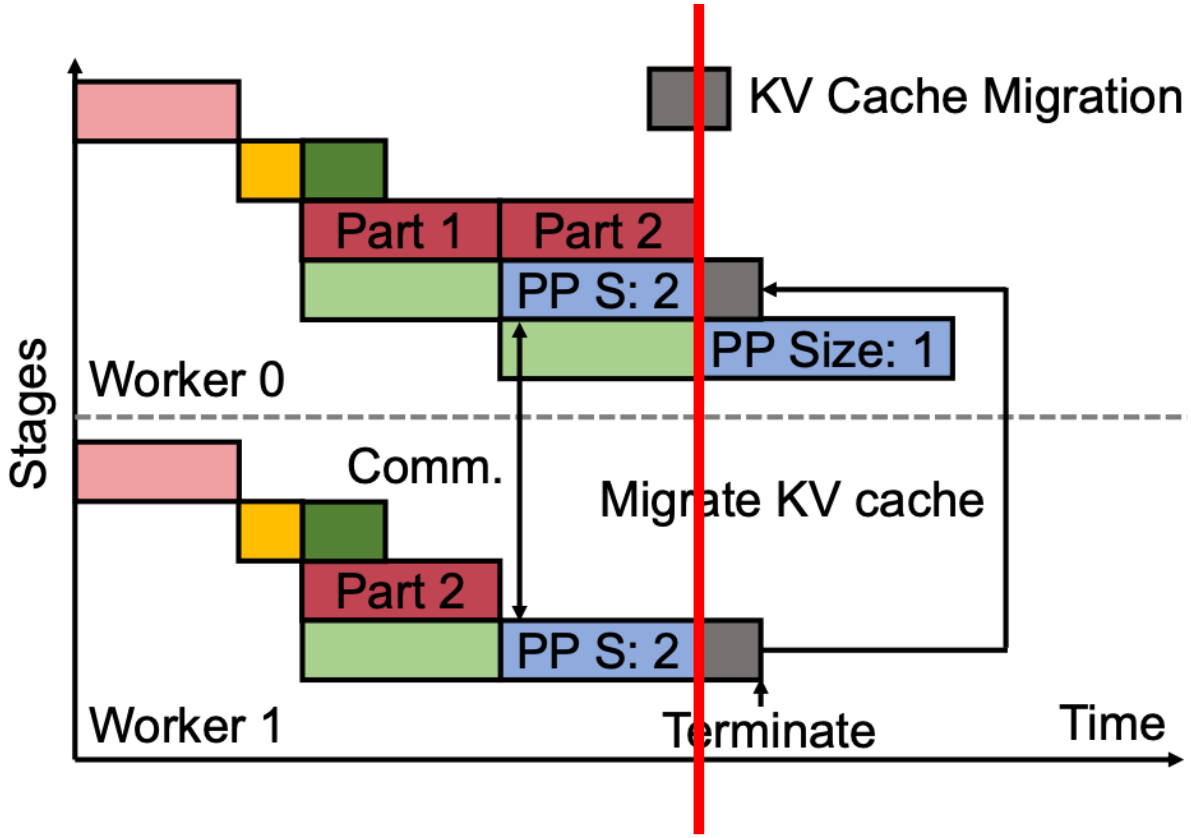




Worker 0 fetch model part 2

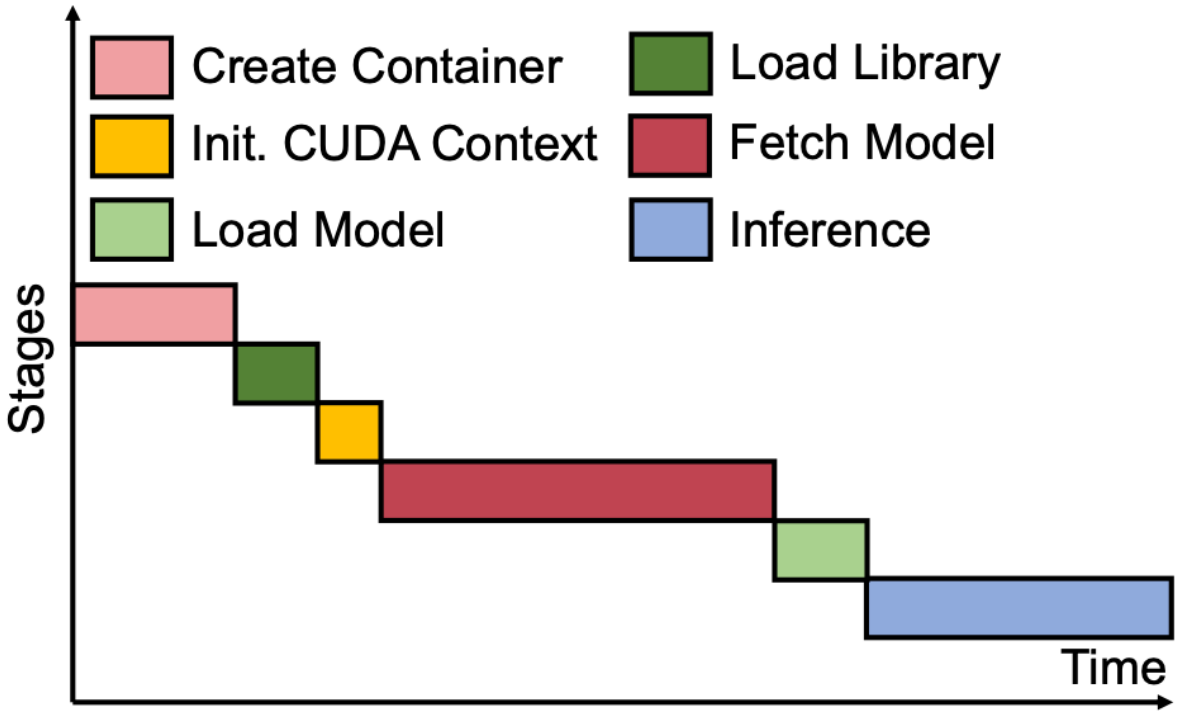
Introduce Pipeline Parallelism

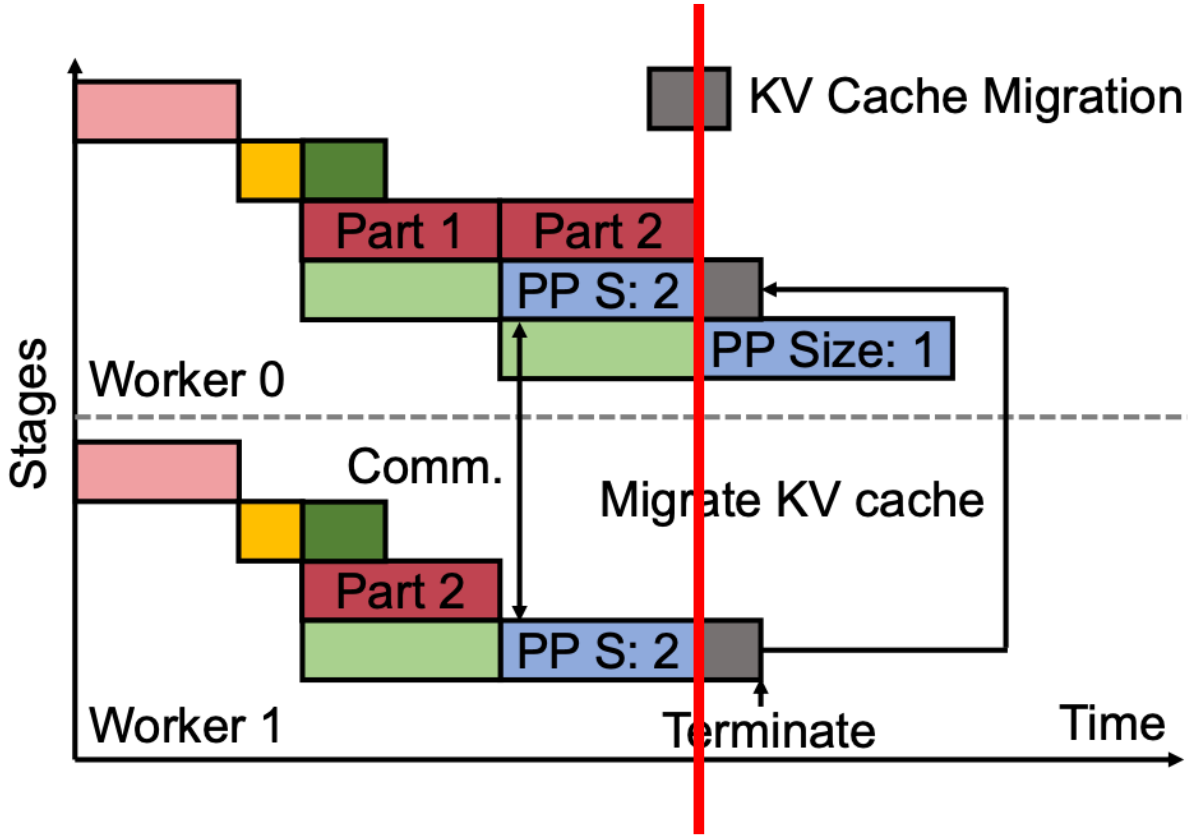




Migrate KV Cache from other workers

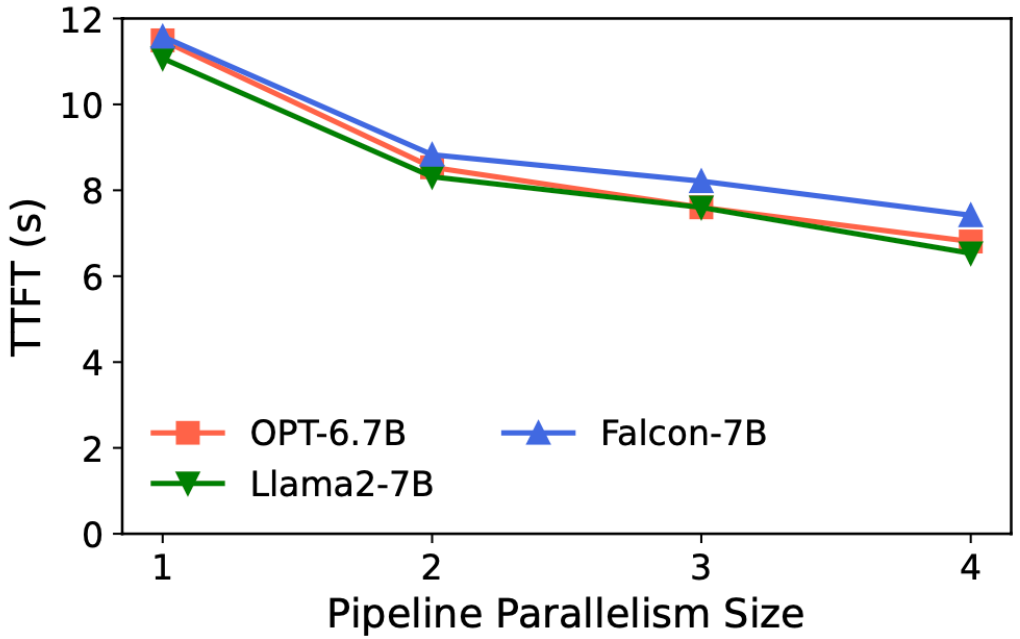
Introduce Pipeline Parallelism

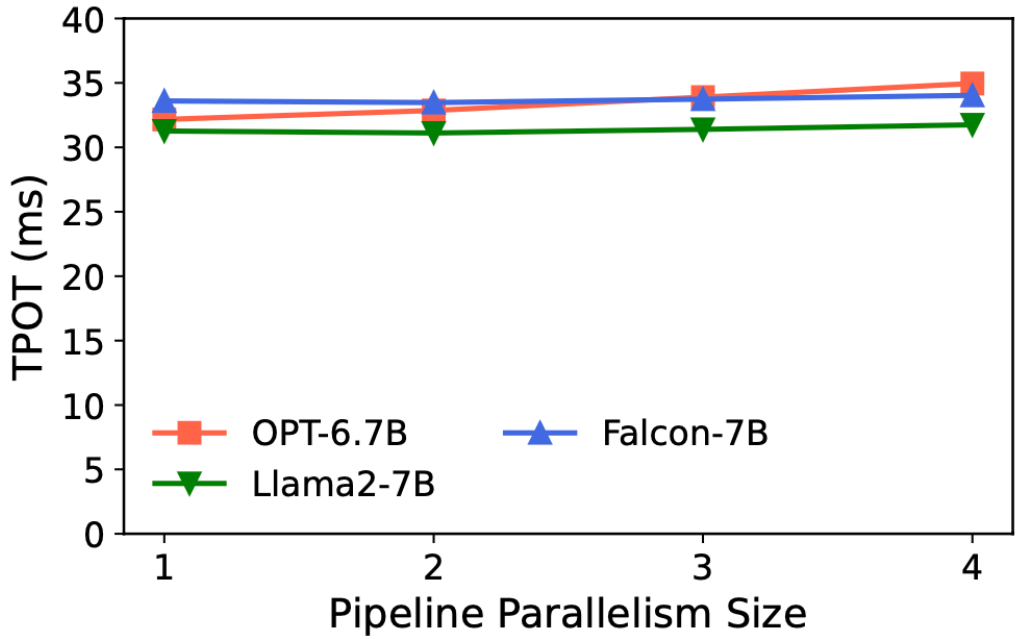




Fall back to single-stage inference

Tradeoff Analysis





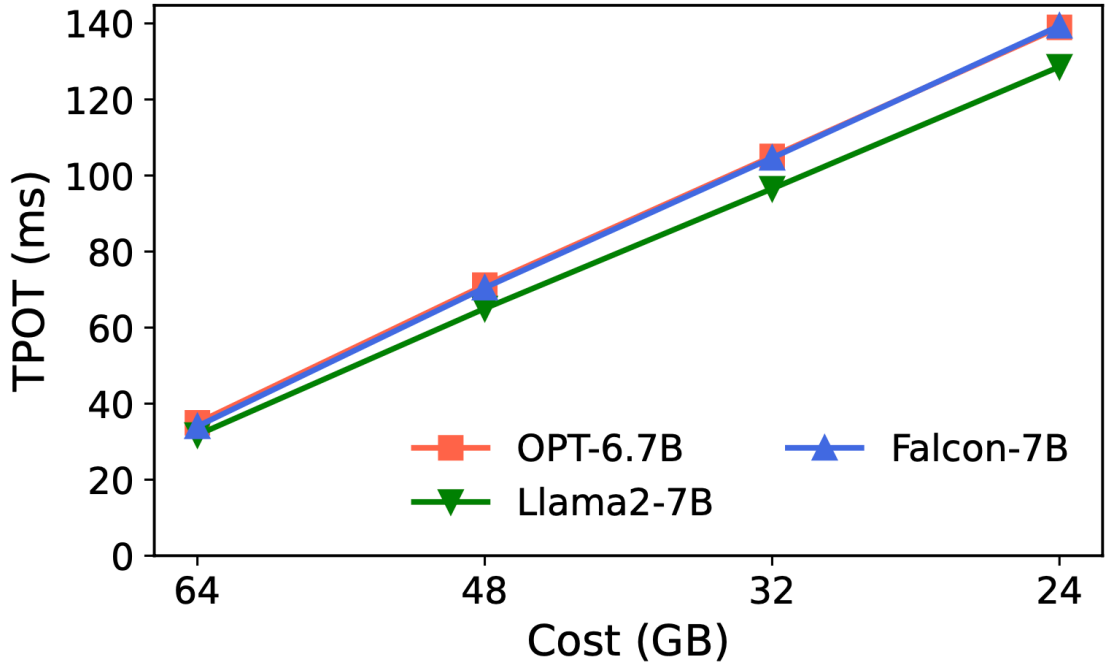
- More pipeline stages bring lower TTFT
  - Marginal gain diminishes due to other cold-start stages
- Slight TPOT increase; proportional cost

Tradeoff Analysis

- Option: reducing cost by allocating less GPU mem per worker
  - Single-worker case: 1 worker * 16 GB memory
  - Pipelined case: 4 workers * 8 GB memory

Tradeoff Analysis

- Option: reducing cost by allocating less GPU mem per worker
  - Single-worker case: 1 worker * 16 GB memory
  - Pipelined case: 4 workers * 8 GB memory
- Computation power is proportional to allocated memory

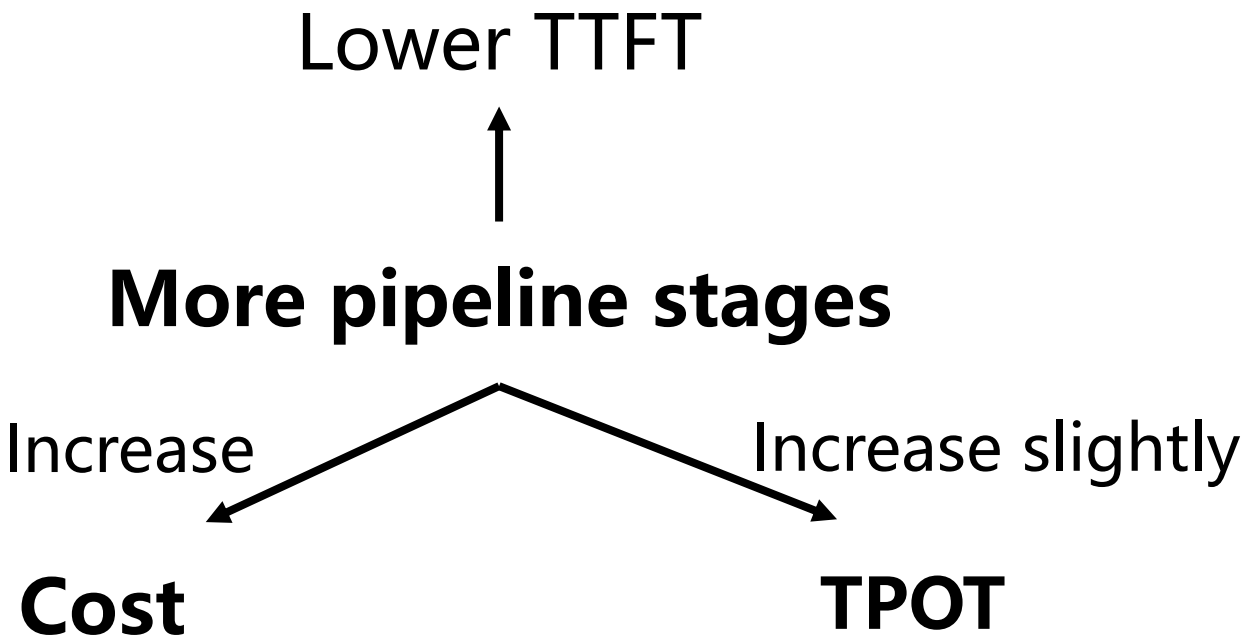


Tradeoff Summary

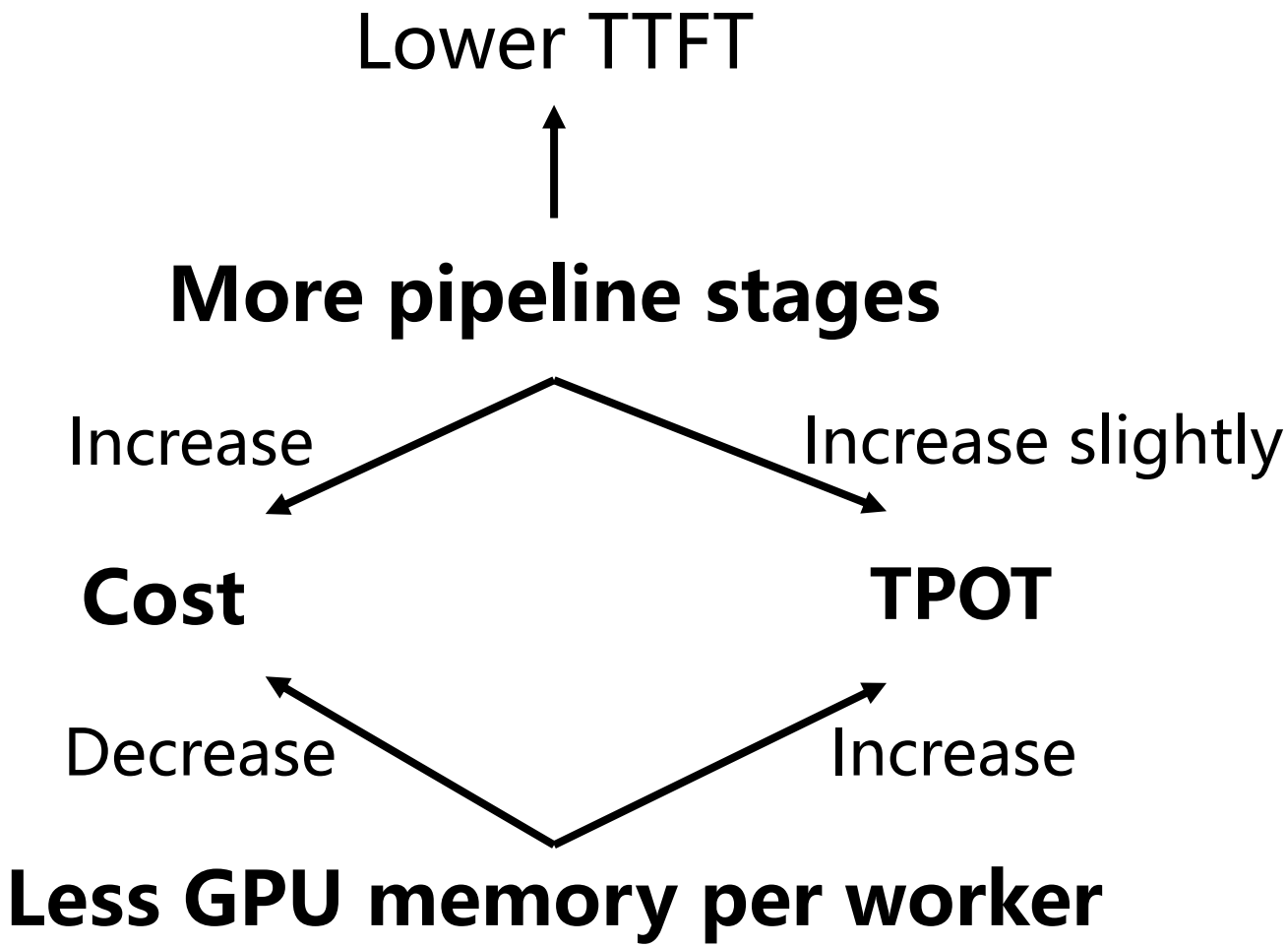
Lower TTFT

More pipeline stages

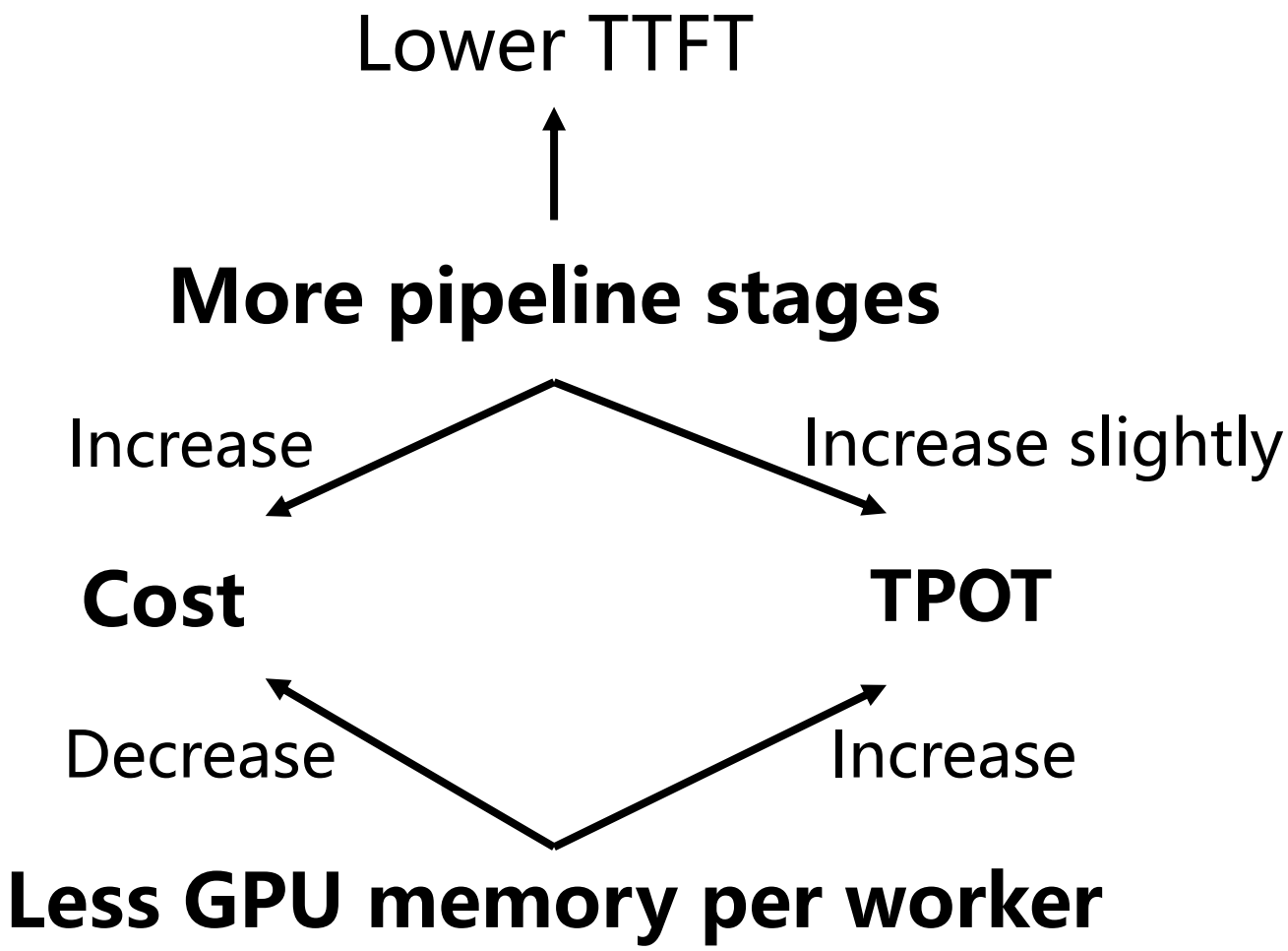
Tradeoff Summary



Tradeoff Summary



Tradeoff Summary



SLO-Aware
Resource Allocation

Overlap Cold-Start Stages

- **Model Prefetcher**

Fetch model to host memory before container creation

- **Parameter Manager**

Load model to GPU before python library loaded

Overlap Cold-Start Stages

- **Model Prefetcher**

Fetch model to host memory before container creation

- **Parameter Manager**

Load model to GPU before python library loaded

All stages hidden behind fetching and loading model to GPU

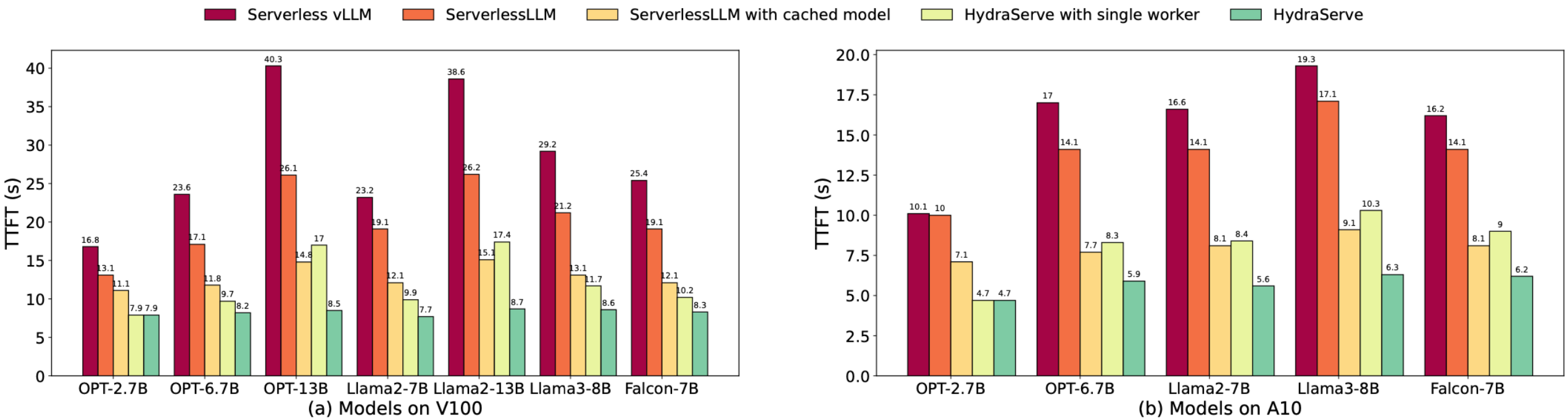
Evaluation – Setup

- Hybrid cluster with A10 and V100 GPUs
- Compare with vLLM* and ServerlessLLM**

*Kwon et al., “Efficient memory management for large language model serving with pagedattention,” in SOSP23.

**Fu et al., “Serverlessllm: Low-latency serverless inference for large language models,” in OSDI24.

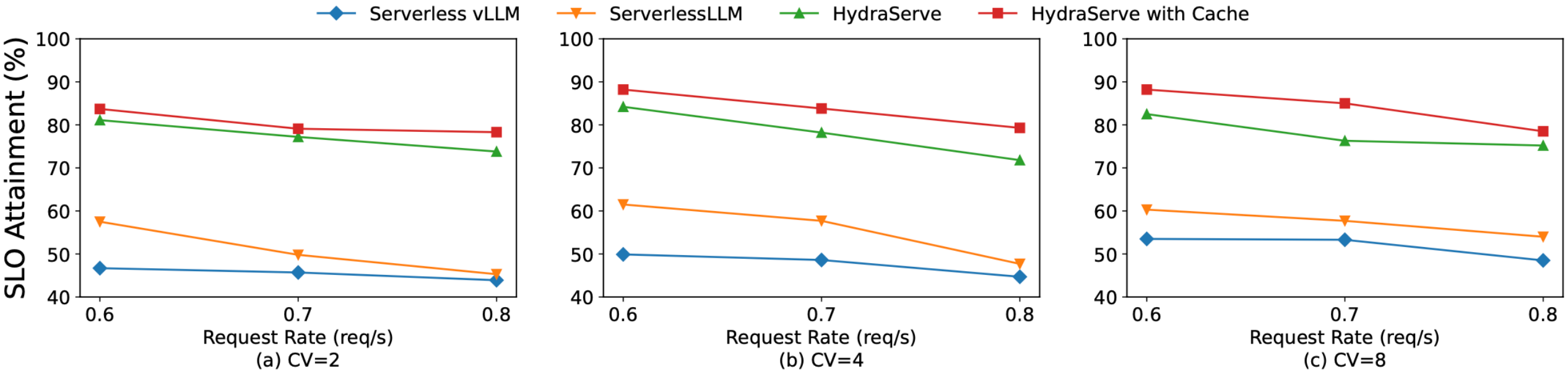
Evaluation – Cold Start Latency



HydraServe vs. serverless vLLM: **2.1×– 4.7×** TTFT reduction

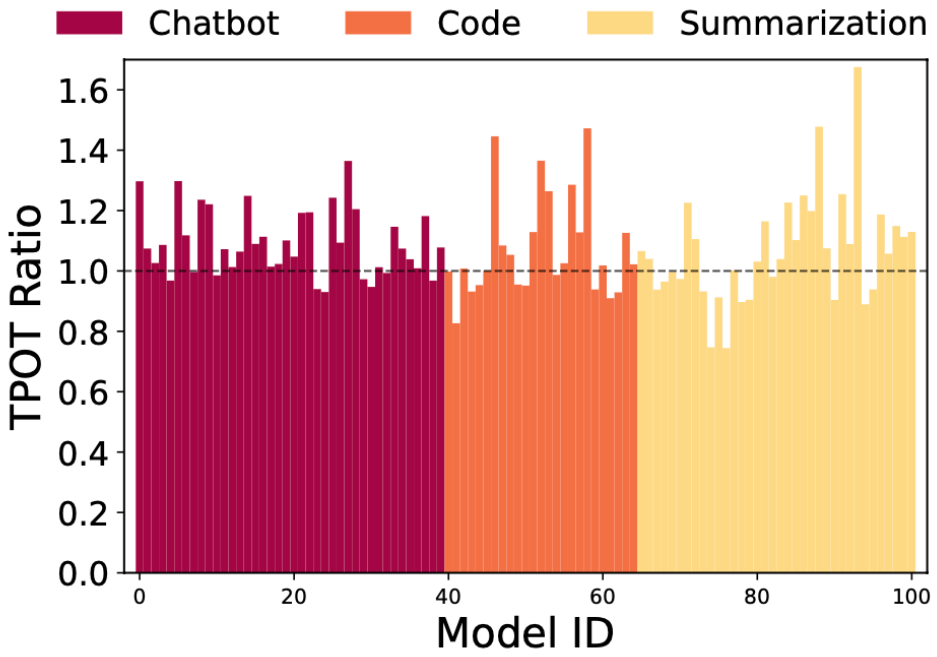
HydraServe vs. ServerlessLLM: **1.7×– 3.1×** TTFT reduction

Evaluation – End-to-End SLO Attainment

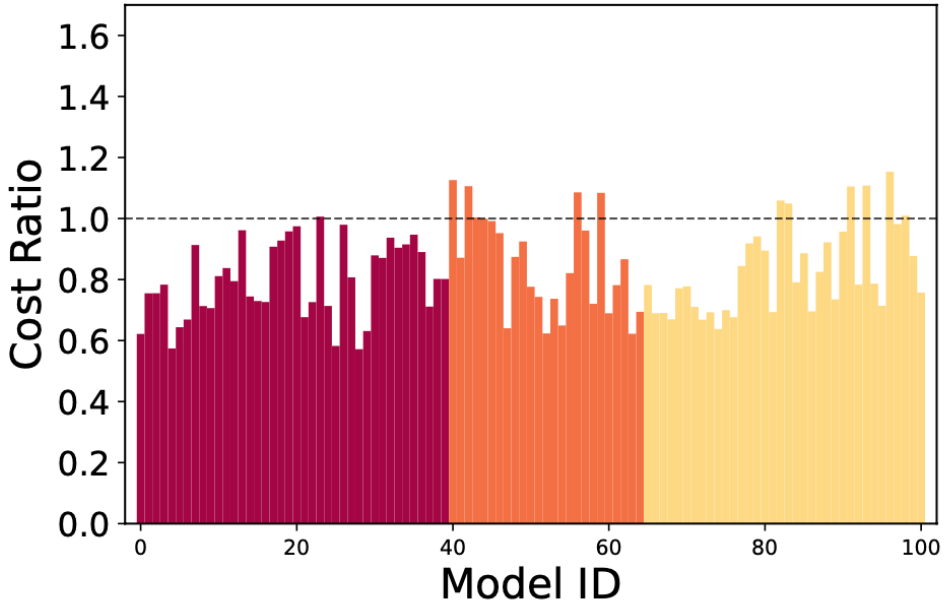


HydraServe achieves 1.43×–1.74× TTFT SLO attainment improvement

Evaluation – TPOT and Cost Analysis



(a) TPOT ratios for different models.



(b) Cost ratios for different models.

1.06× TPOT Increase.

1.12× Cost Reduction

Conclusion

- HydraServe leverages **pipeline parallelism** and **stage overlapping** to minimize cold start latency
- HydraServe achieves a **1.7×– 4.7×** TTFT reduction compared to existing approaches

https://github.com/LLMServe/hydraserve

louchiheng23@stu.pku.edu.cn