dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving

Author: Bingyang Wu[1], Ruidong Zhu[1], Zili Zhang[1], Peng Sun[2], Xuanzhe Liu[1] and Xin Jin [1]

[1] Peking University [2] Shanghai AI Lab

OSDI 2024

Presented by Chizheng Fang

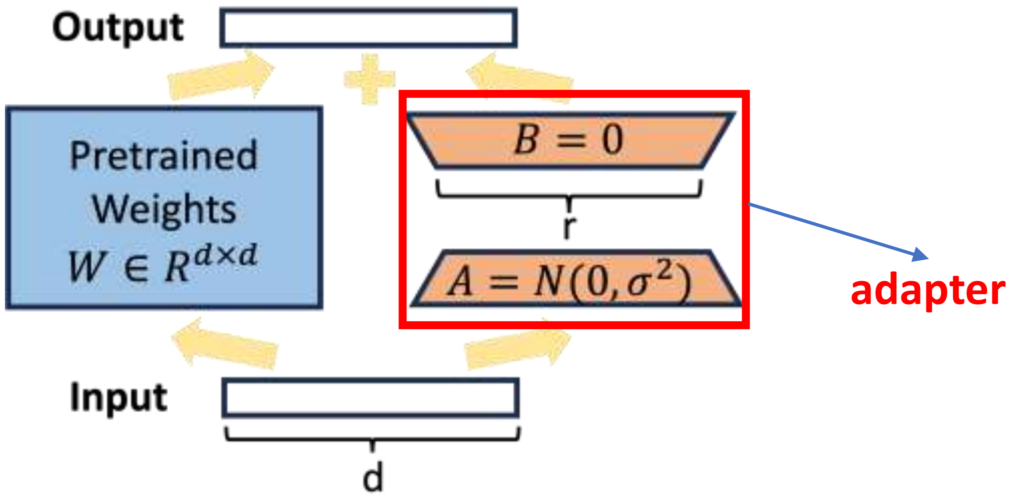
USTC, CHINA
ADSLAB
先进数据系统实验室

Outline

- Background
- Challenges
- Design
- Implementation & Evaluation
- Summary

Background

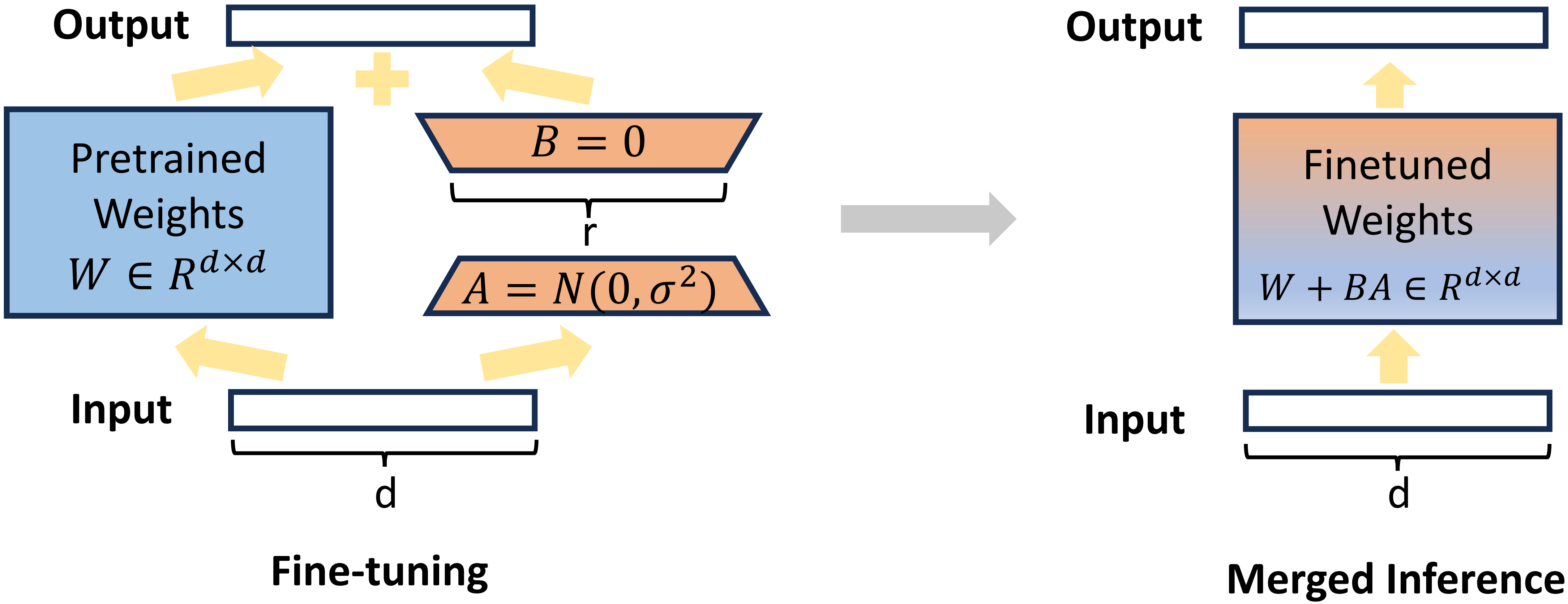
- LoRA (Low-Rank Adaptation): A popular approach to fine-tune LLMs
  - h = $Wx + BA$
  - Compared to fully fine-tuning GPT-3 175B, LoRA can reduce the number of trainable parameters by 10,000x and the GPU consumption by 3x



Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "LoRA: Low-Rank Adaptation of Large Language Models." ICLR (2022)

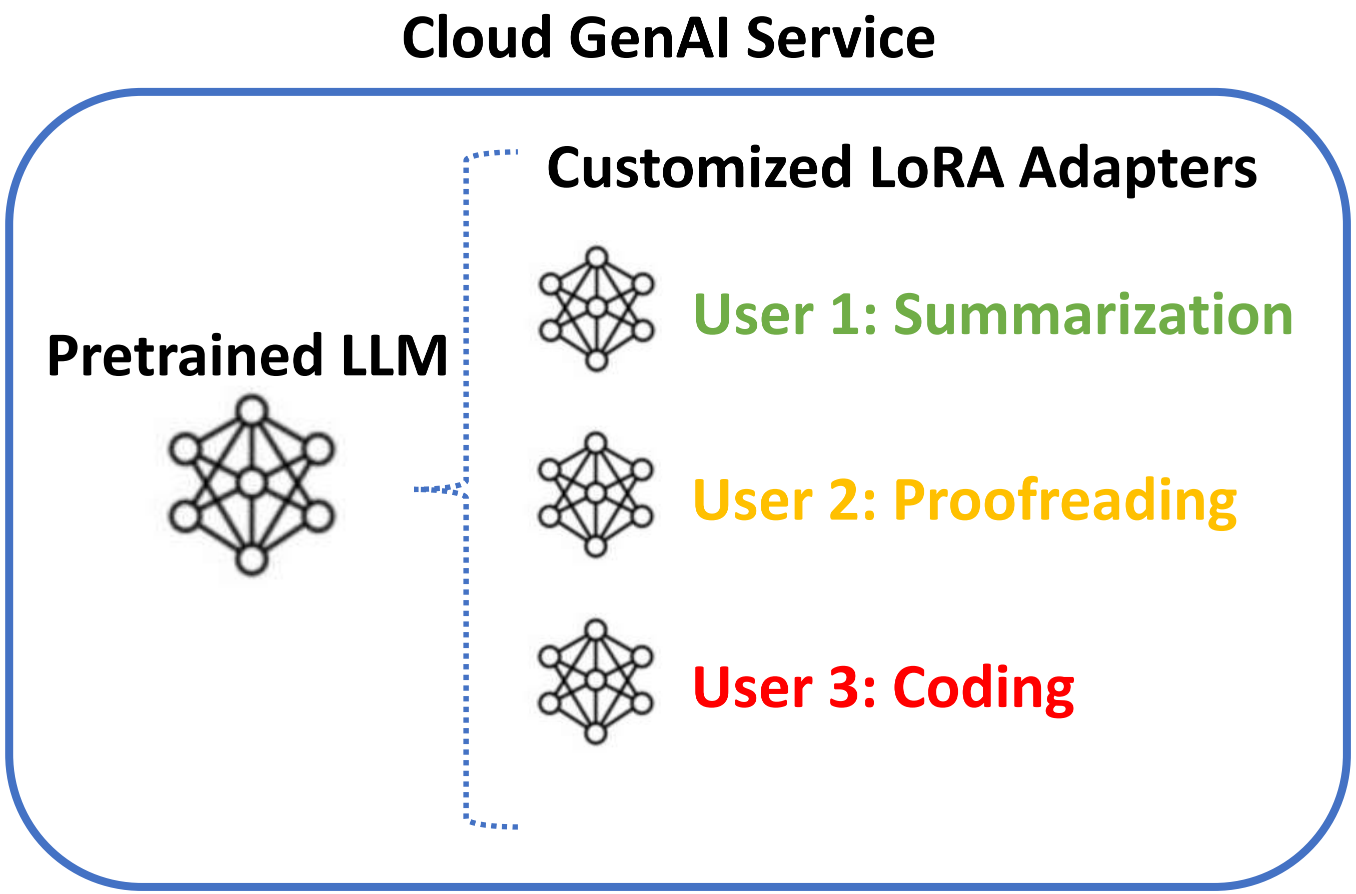
Background

- LoRA introduces no inference overhead when serving a single LoRA LLM



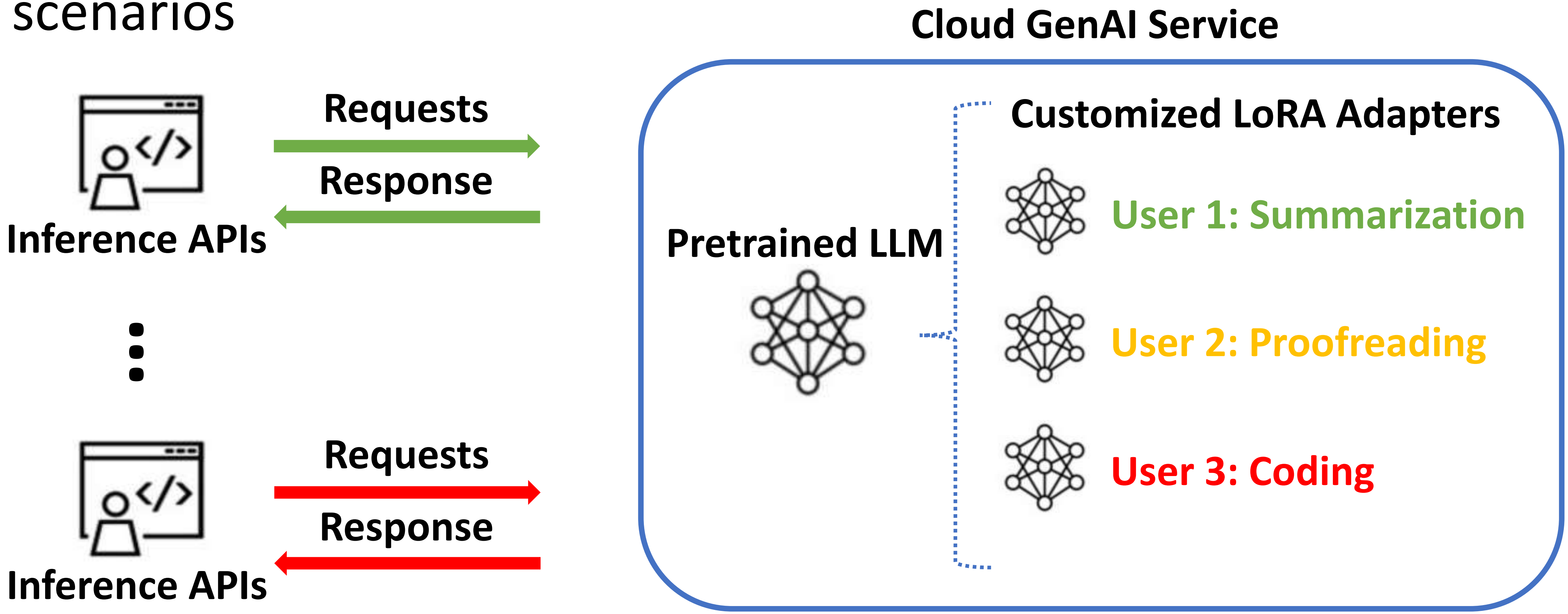
Background

- Cloud providers may host many adapters for a LLM



Background

- Cloud providers may host many adapters for a LLM
- Different users may use different adapters for different scenarios

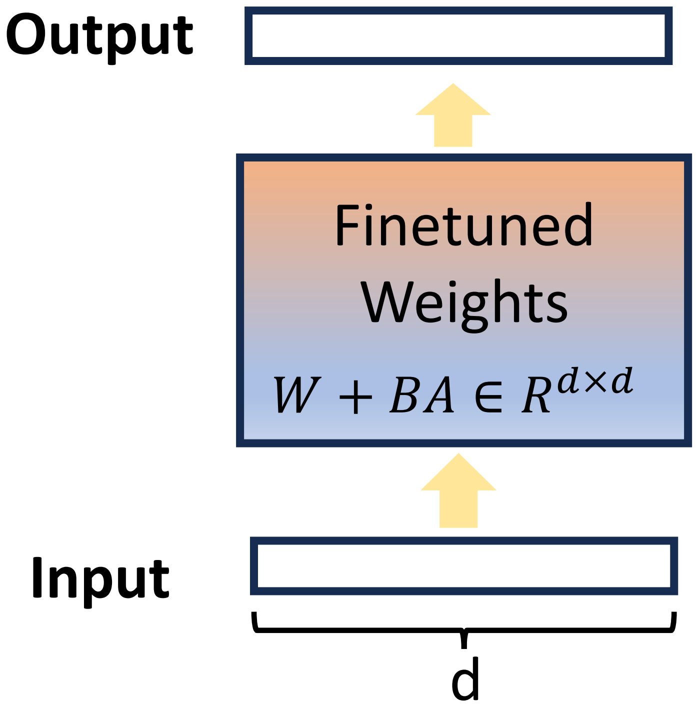


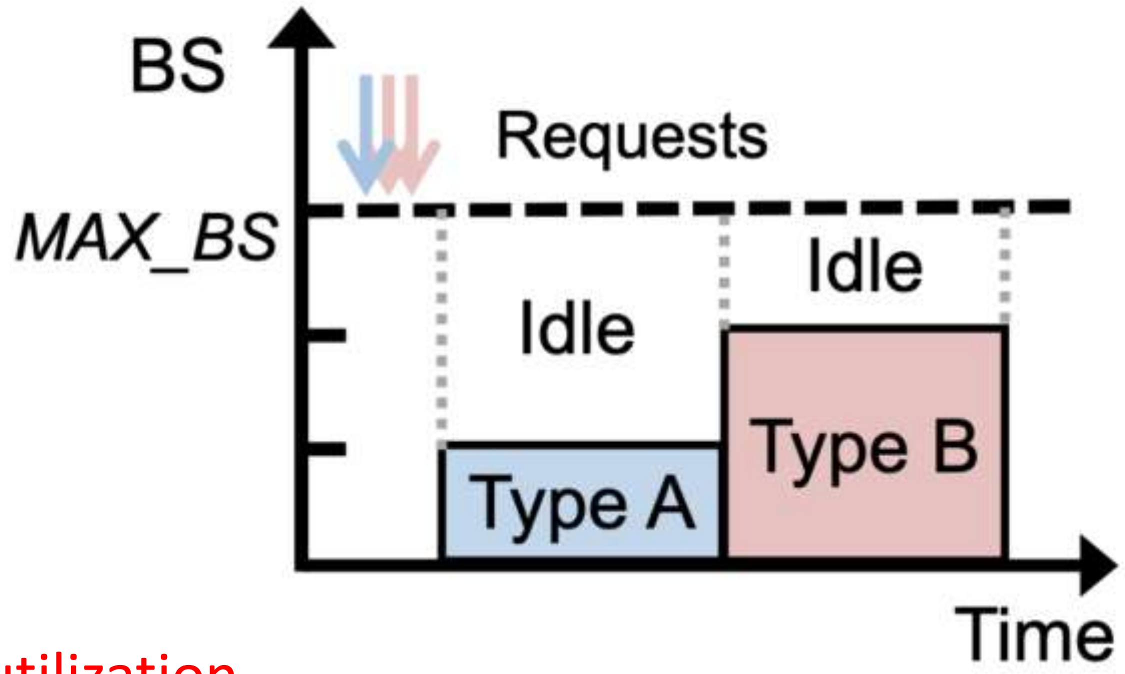
Outline

- Background
- Challenges
- Design
- Implementation & Evaluation
- Summary

Challenge(1): Intra-replica

● **Merged inference**: Former LoRA serving system forces other types of requests to wait until the completion of the current batch.

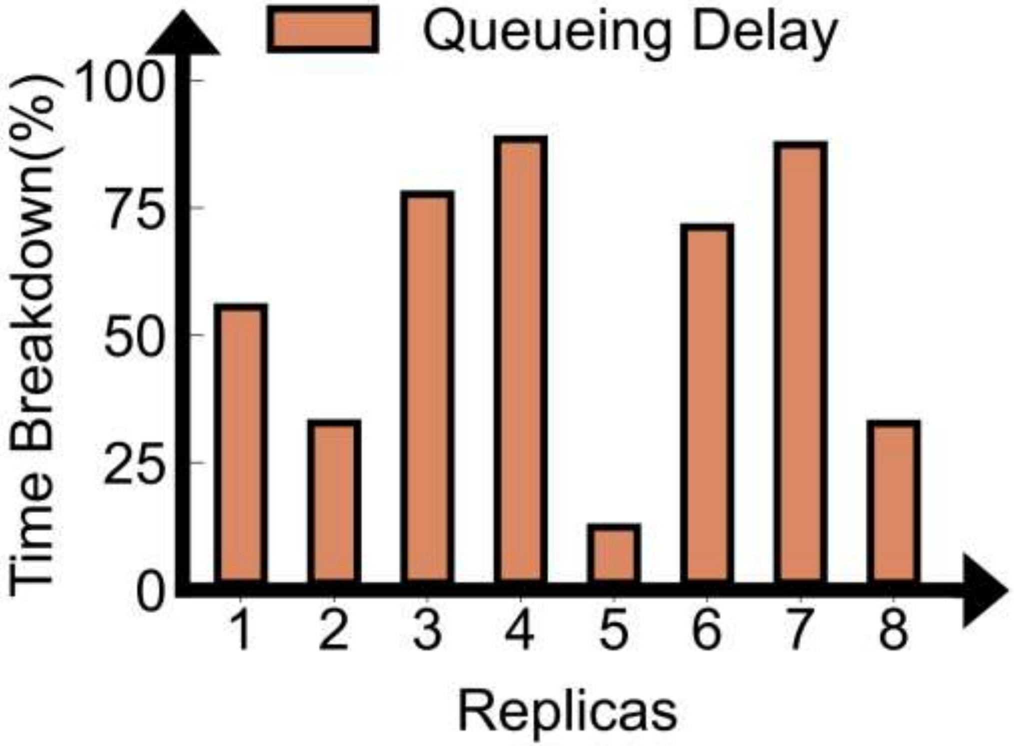




Low GPU utilization

Challenge(2): Inter-replica

- The burst of variable requests leads to severe load imbalance under static LoRA placement
- Input and output lengths of requests are highly variable



Severe load imbalance

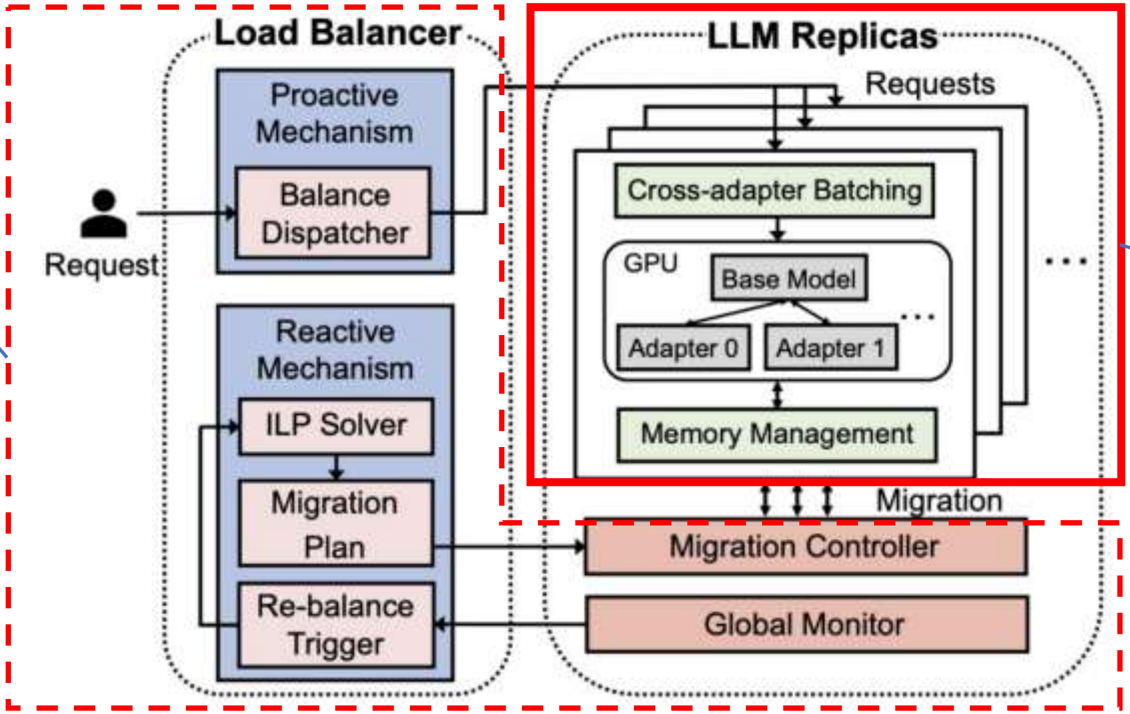
Outline

- Background
- Challenges
- Design
- Implementation & Evaluation
- Summary

dLoRA Overview

- Insights: dynamically orchestrate requests and LoRA adapters
- Methods:
  - **Intra-replica**: dynamic batching + memory management
  - **Inter-replica**: proactive dispatching + reactive migration

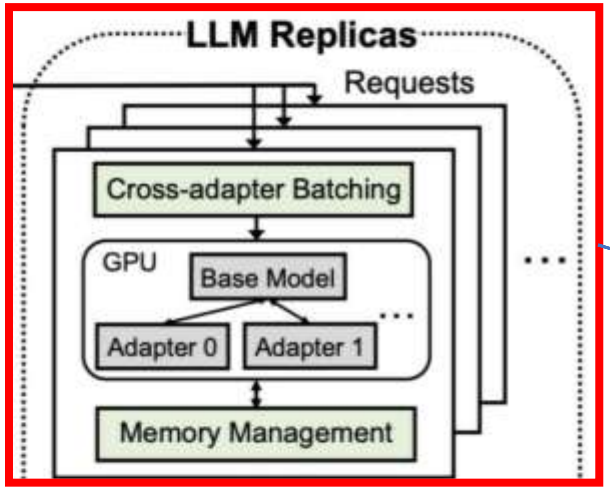
Inter-replica



Intra-replica

dLoRA Overview

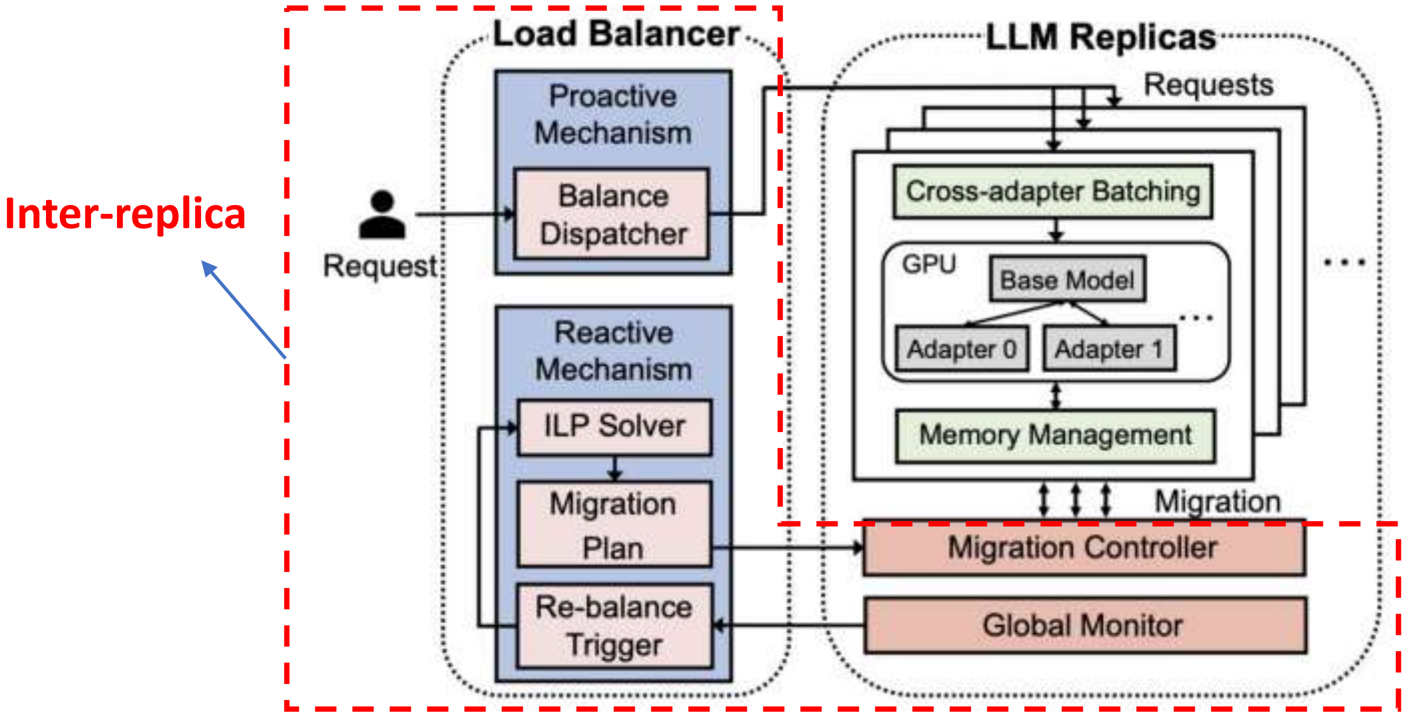
- Insights: dynamically orchestrate requests and LoRA adapters
- Methods :
  - **Intra-replica**: dynamic batching + memory management



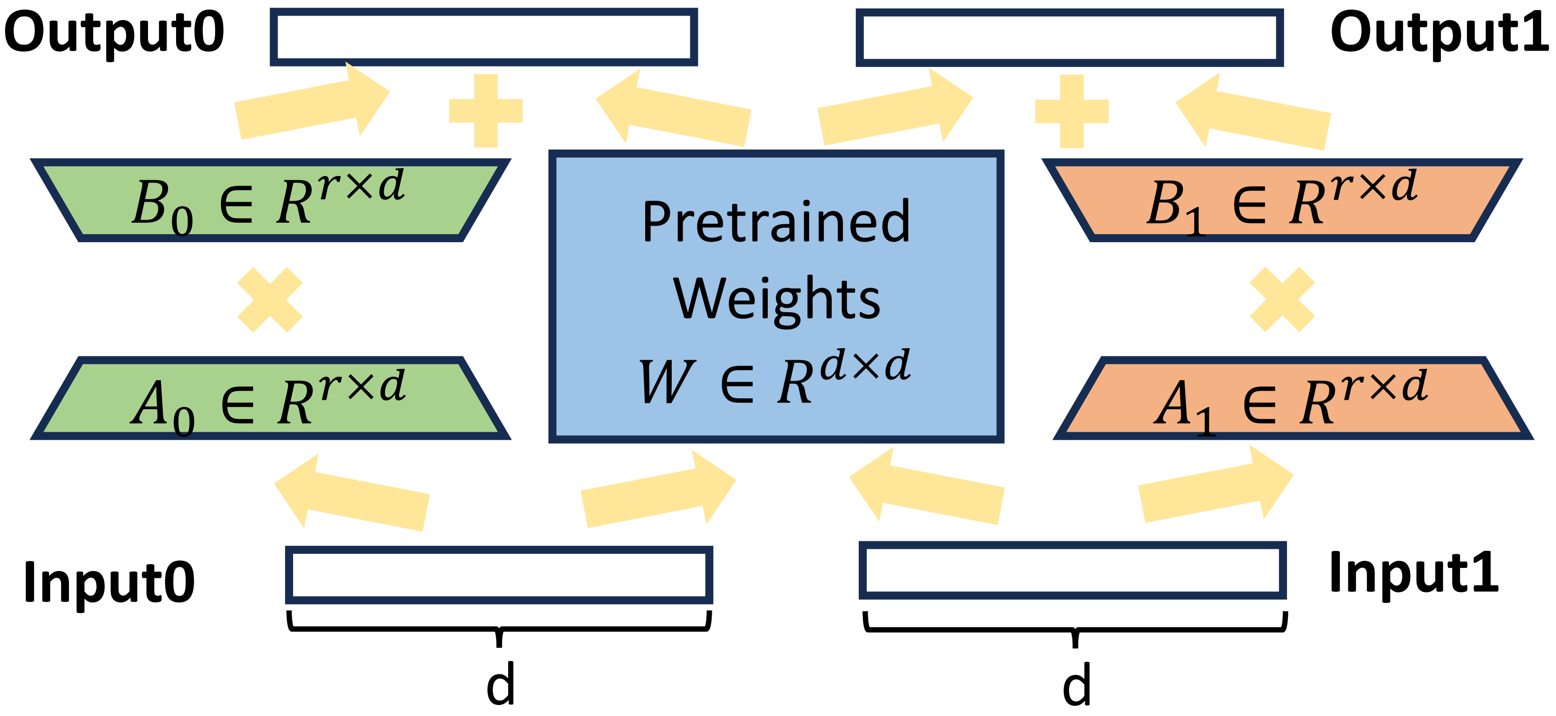
Intra-replica

dLoRA Overview

- Insights: dynamically orchestrate requests and LoRA adapters
- Methods:
  - **Intra-replica**: dynamic batching + memory management
  - **Inter-replica**: proactive dispatching + reactive migration

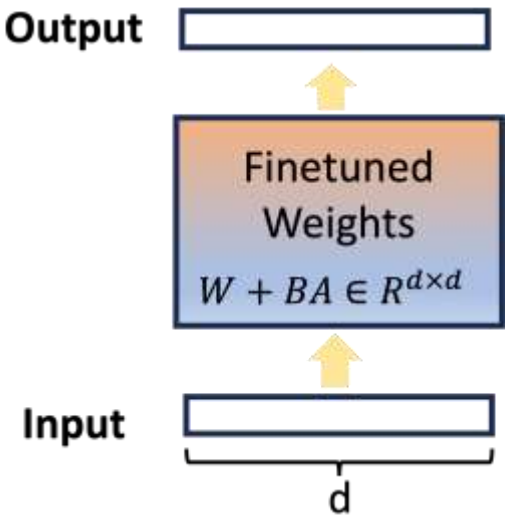


Dynamic Batching

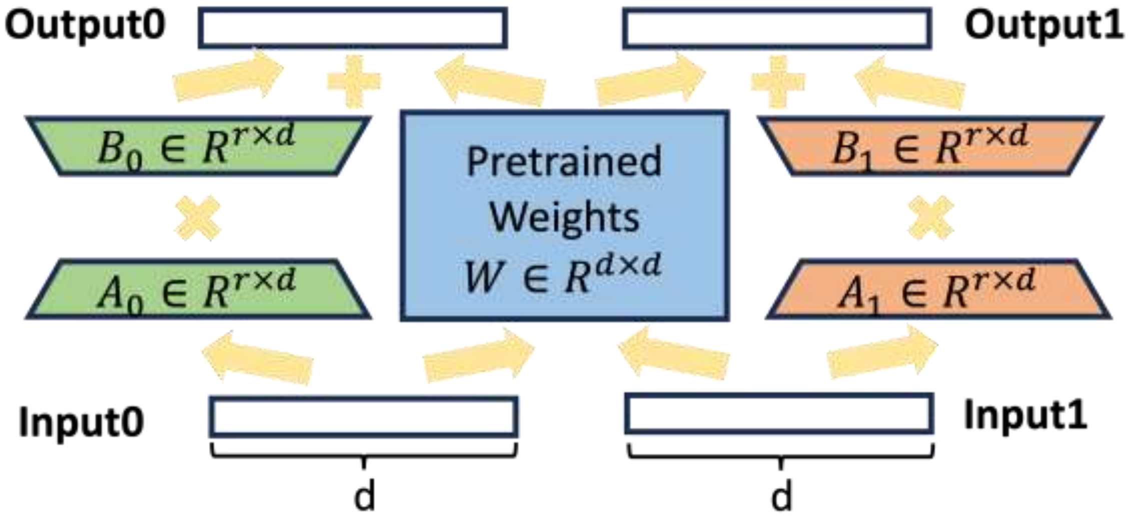


Dynamic Batching

● **Unmerged Inference**: share the same computation among different requests



Merged Inference

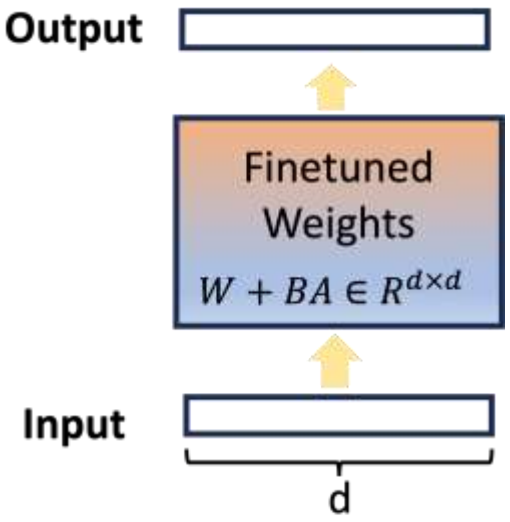


Unmerged Inference

Dynamic Batching

- **Unmerged Inference**: share the same computation among different requests

$$W' = W + BA$$
$$y = W'x$$

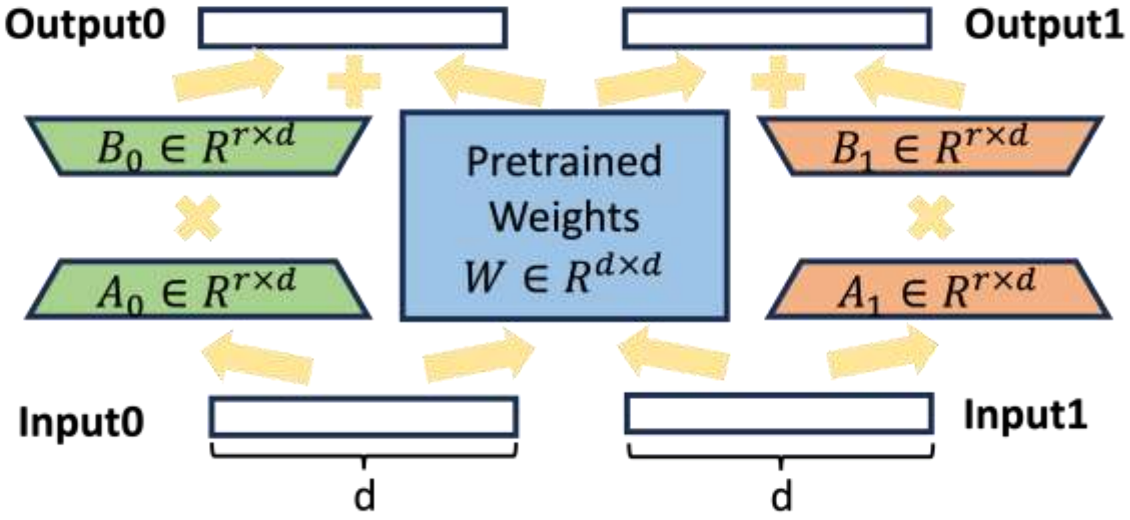


Merged Inference

Batched as $W \times [x_0, x_1]$

$$\begin{aligned} y_0 &= \boxed{Wx_0} + \boxed{B_0A_0x_0} \\ y_1 &= \boxed{Wx_1} + \boxed{B_1A_1x_1} \end{aligned}$$

Computed in parallel

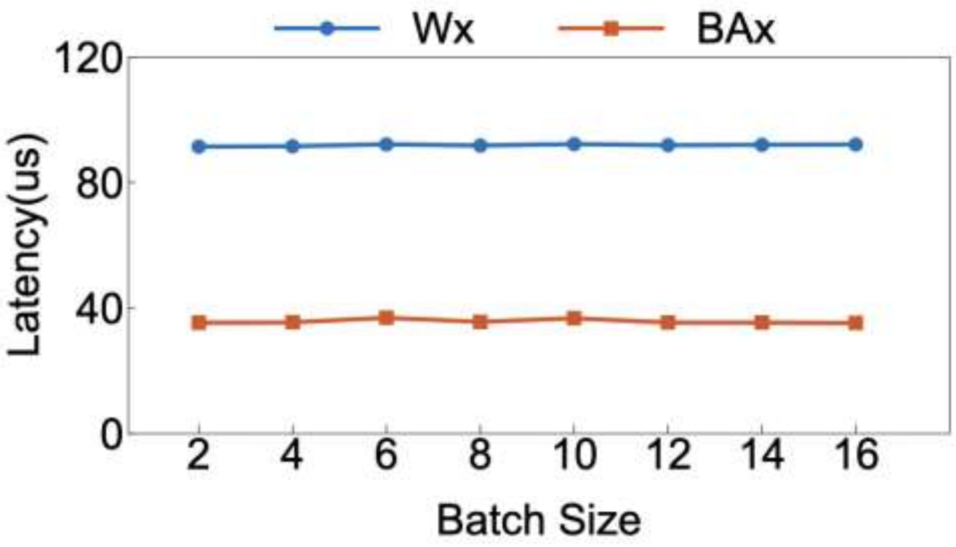


Unmerged Inference

Dynamic Batching

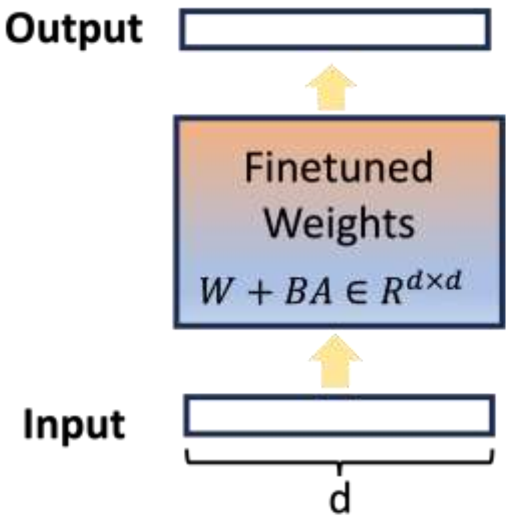
- **Merged Inference:** $y = W'x$
- **Unmerged Inference:** $y_0 = Wx_0 + B_0A_0x_0$
  - introduces two additional matrix multiplications and one additional matrix addition in each layer.
  - computation BAx is 38.9% of computation Wx.

Require a combine approach

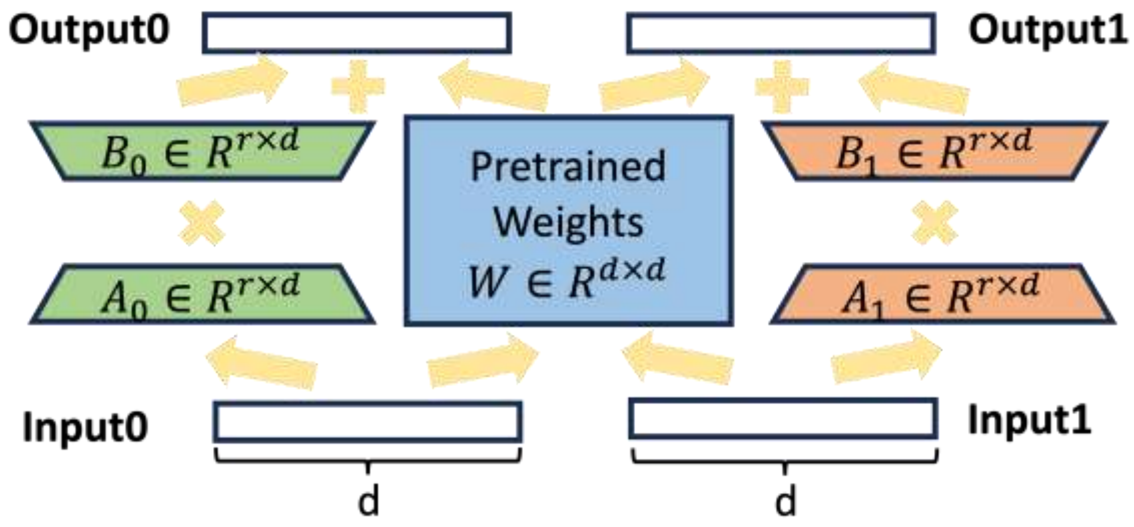


Dynamic Batching

- Executed at iteration granularity
  - Iteration: output a token
- Assume current state is unmerged
- Calculate their ratio of the throughput of merged and unmerged
  - If ratio > α_{switch}, switch to merged
  - Otherwise, remain unmerged



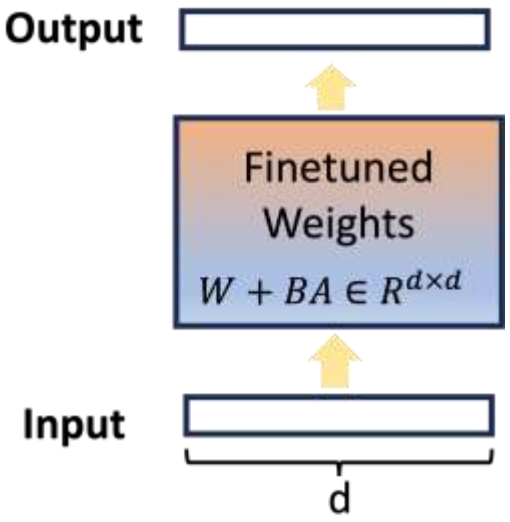
Merged Inference



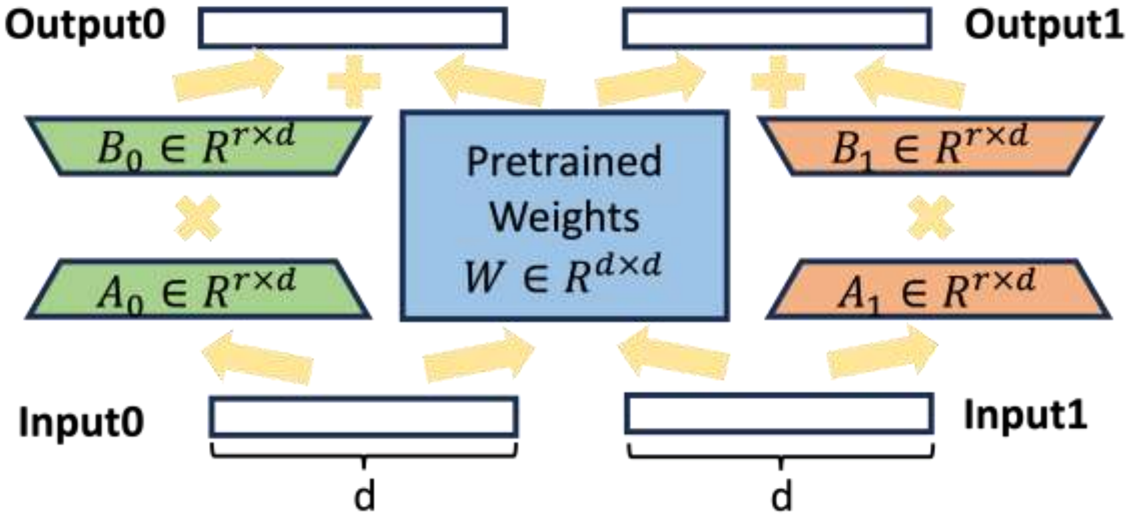
Unmerged Inference

Dynamic Batching

- Executed at iteration granularity
  - Iteration: output a token
- Assume current state is merged
- Calculate their ratio of the throughput of merged and unmerged
  - If ratio < β_{switch}, switch to unmerged
  - Otherwise, remain merged



Merged Inference

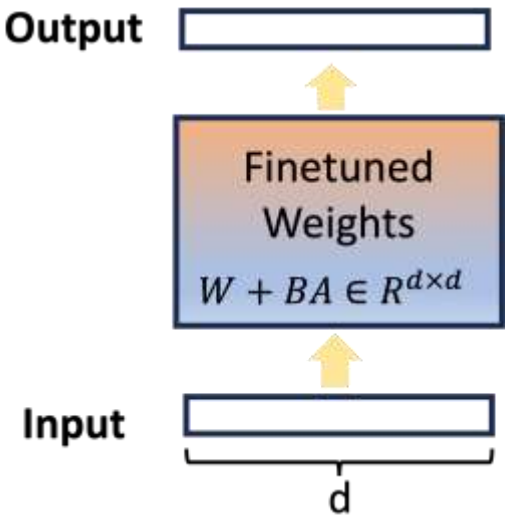


Unmerged Inference

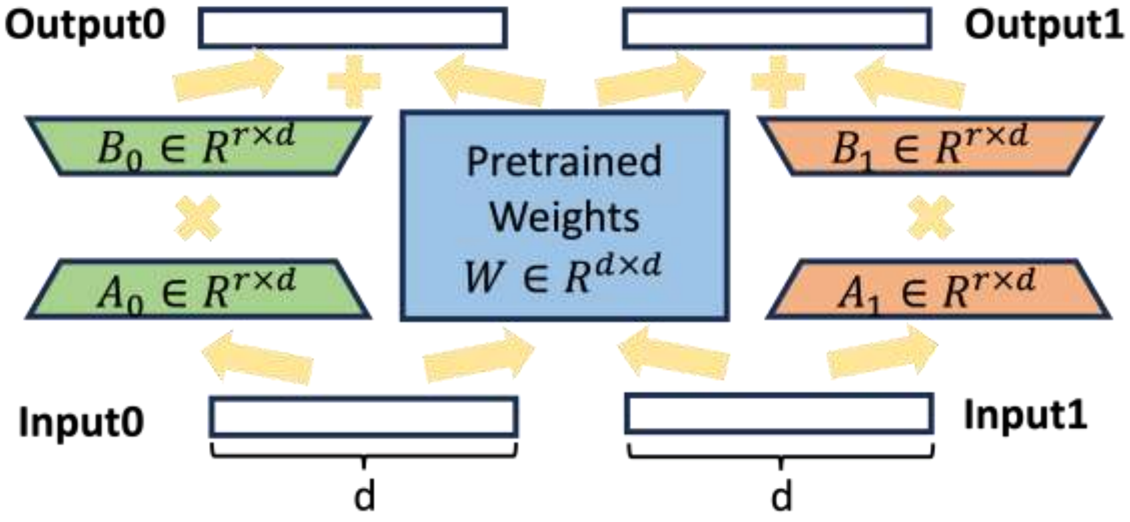
Dynamic Batching

- Executed at iteration granularity
  - Iteration: output a token
- Assume current state is merged
- Calculate their ratio of the throughput of merged and unmerged
  - If ratio < β_{switch}, switch to unmerged
  - Otherwise, remain merged

α_{switch} and β_{switch} are key parameters



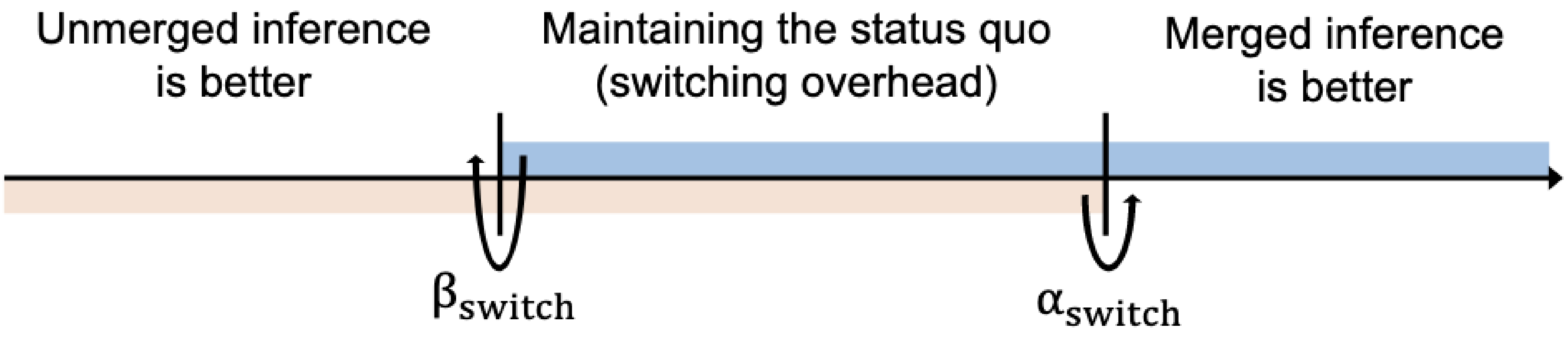
Merged Inference



Unmerged Inference

Dynamic Batching

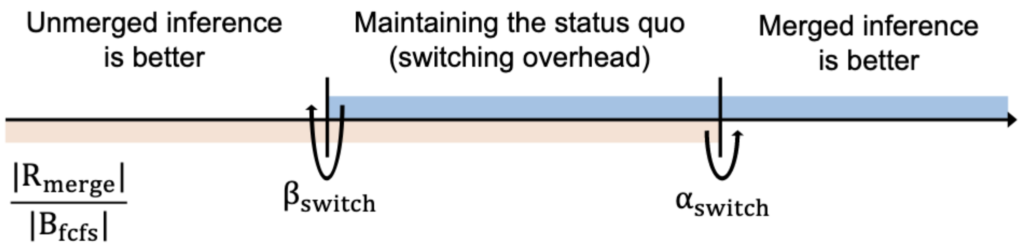
- How to choose α_{switch} and β_{switch}
- Insights:
  - Switching overhead can be amortized across multiple future iterations.
  - Despite the unavailability of future knowledge, leveraging historical retrospection is possible.



Dynamic Batching

- Iteration granularity breakpoints, such as replica switching, changes in R_{merge}, or after processing a set number of iterations.
- Based on the data collected from the preceding period.

N_I : number of the iterations in the previous period
B_i : $R_{merged}[:\ maxbs]$ in i_{th} iteration
B_i': B_{fcfs} in i_{th} iteration
t_M: switching overhead



Algorithm 2 Adaptive Threshold Tuning

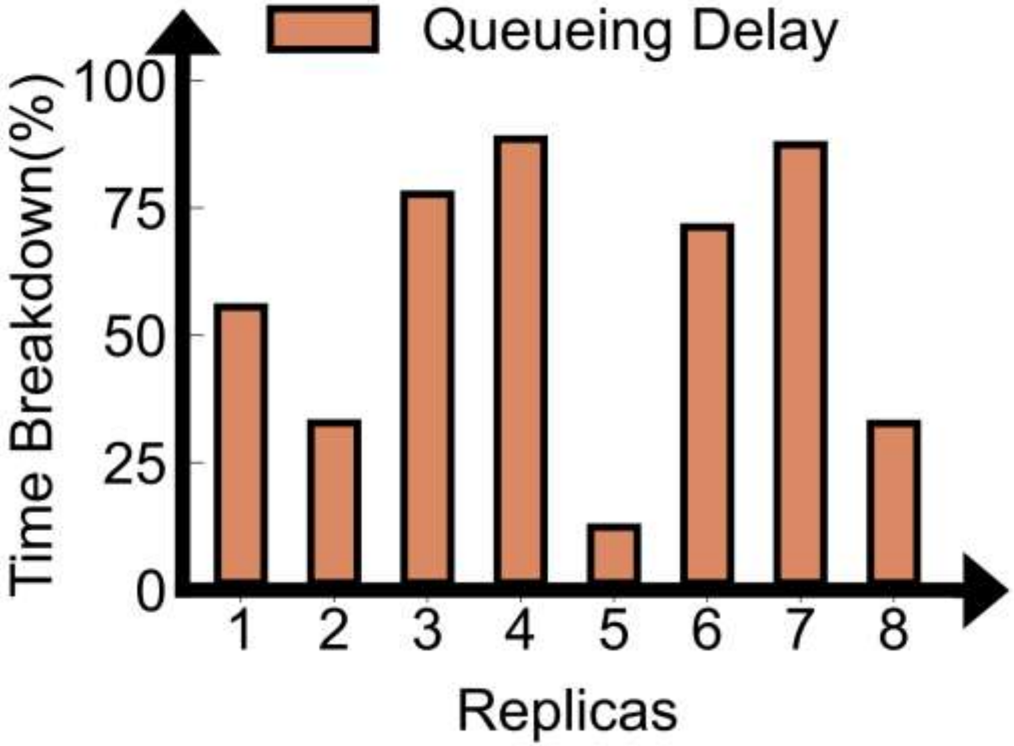
Input: Candidate period N_I, Merged batches $B_1, B_2, \ldots, B_I$, Switching overhead t_M, Current switching threshold α_{switch}
Output: New switching threshold α_{switch}
function ADAPTIVETUNING(N_I, $\{B_i\}$, t_M, α_{switch})
  $T_{merge} = \frac{\sum_{i=1}^{N_I} |B_i|}{\sum_{i=1}^{N_I} IterationTime(B_i) + t_M}$
  $T_{unmerge} = \frac{\sum_{i=1}^{N_I} |B_i'|}{\sum_{i=1}^{N_I} IterationTime(B_i')}$
  if $T_{merge} > T_{unmerge}$ **then**
    $\alpha_{switch} = \alpha_{switch} - \gamma_{dec}$
  else
    $\alpha_{switch} = \alpha_{switch} \times \gamma_{mul}$
  return α_{switch}

Dynamic Batching

- **Starvation prevention:**
  - Allocating a credit to each LoRA adapter, transferred to any preempted adapter.
  - When the credits of certain adapters exceed a threshold, prioritizing processing requests with these adapters.
- **Memory management**
  - Employing a swapping mechanism that swaps LoRA adapters and KV cache between GPU and host memory
  - Could be overlapped with execution using prefetching techniques.

Challenge(2): Inter-replica

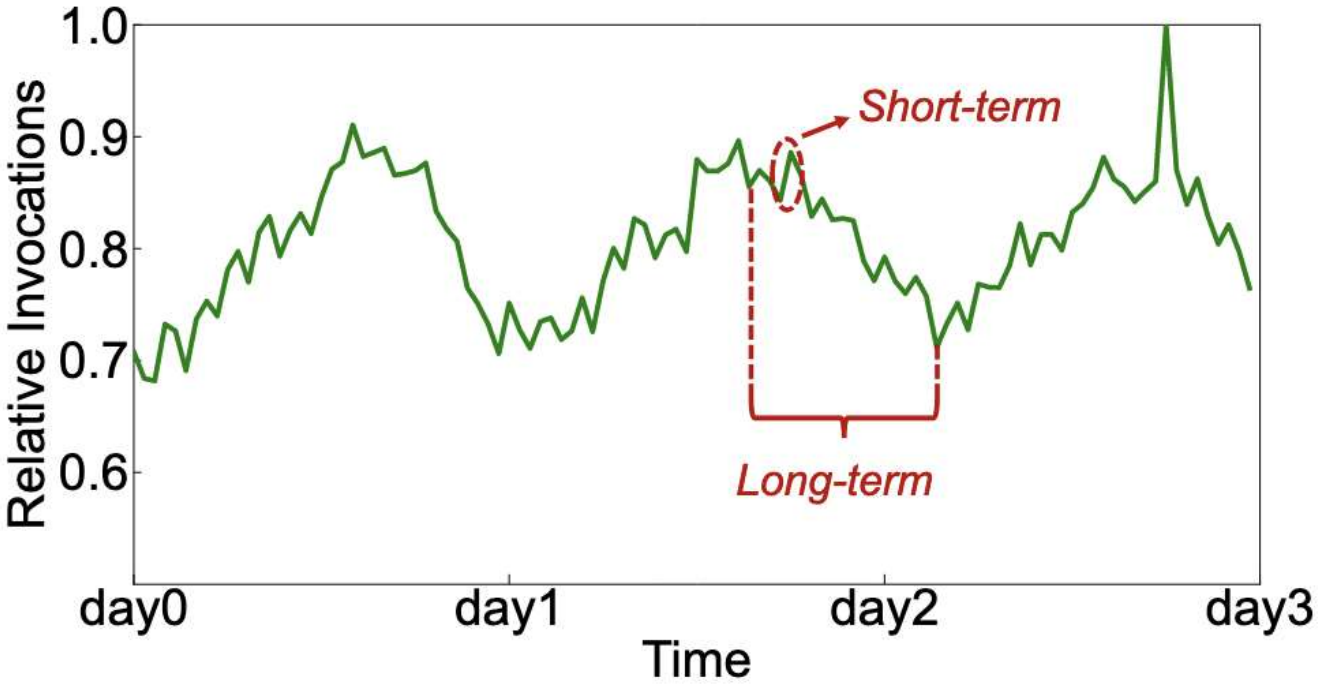
- The burst of variable requests leads to severe load imbalance under static LoRA placement
- Input and output lengths of requests are highly variable



Severe load imbalance

Dynamic Load Balancing

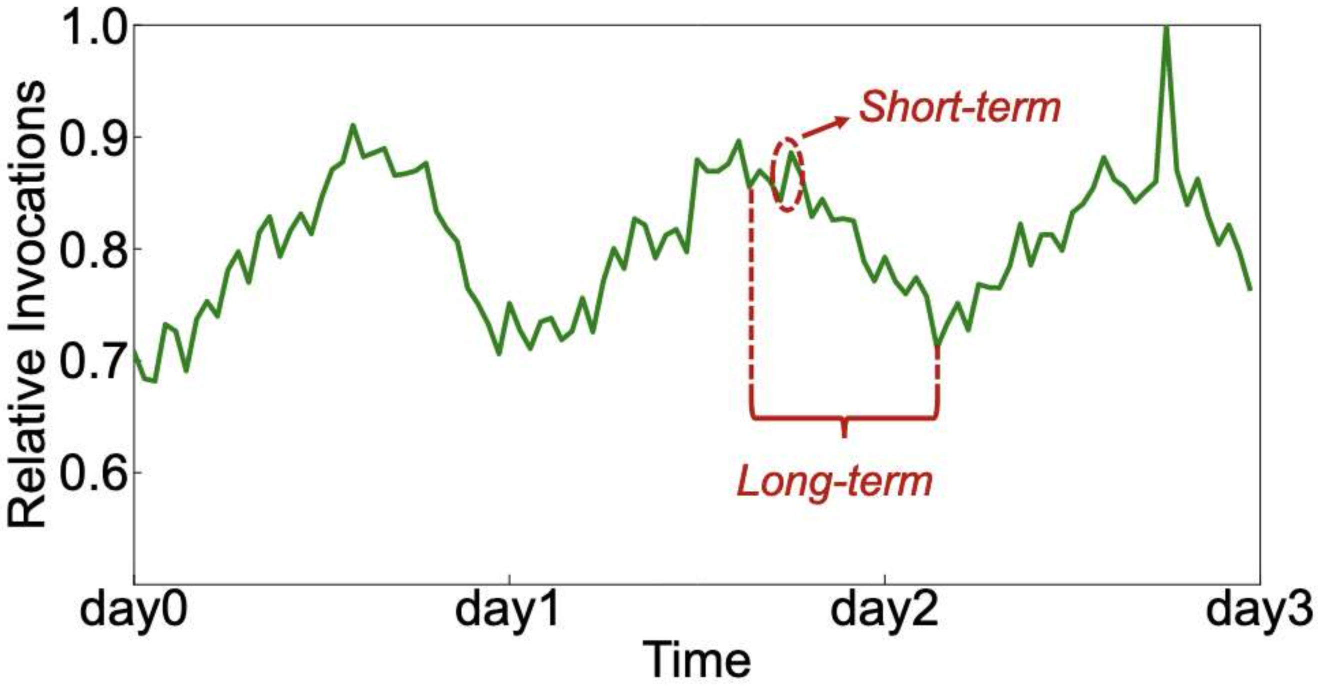
- **Proactive Mechanism**
  - In the long term, the pattern exhibits predictability and periodicity
  - In the short term, the pattern is marked by unpredictability and burstiness.



Proactive Mechanism

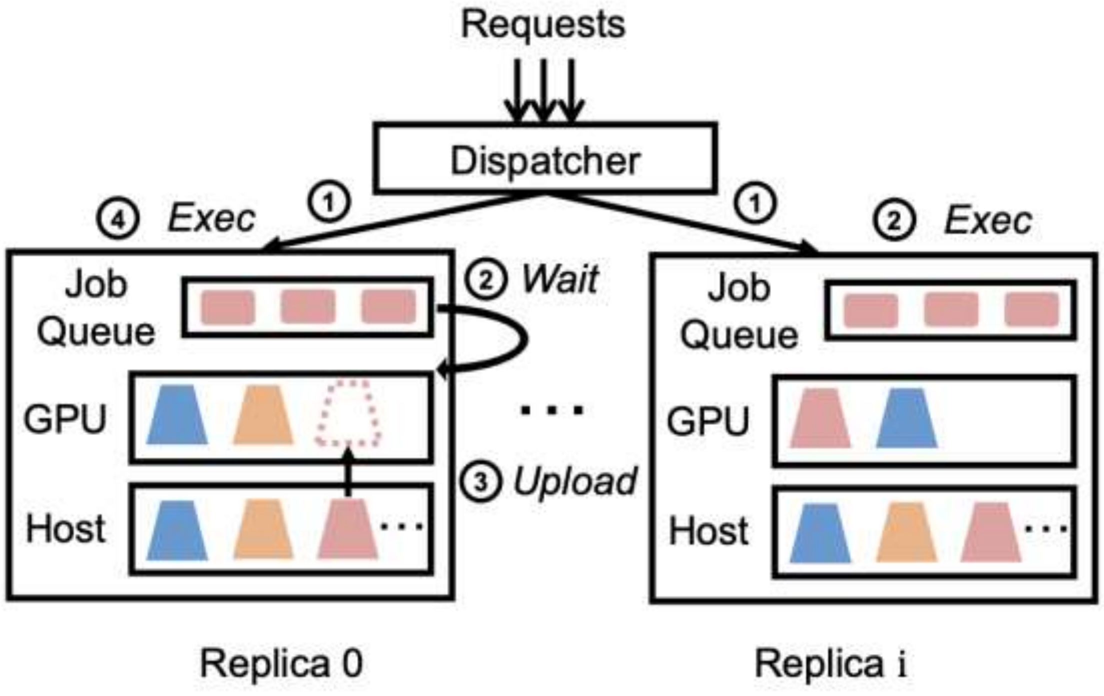
- **Long term strategy**
  - Preload adapters with lowest burst tolerance to maximize the minimum burst tolerance.

$$bt(i) = \frac{\max number\ of\ requests}{average\ load}$$



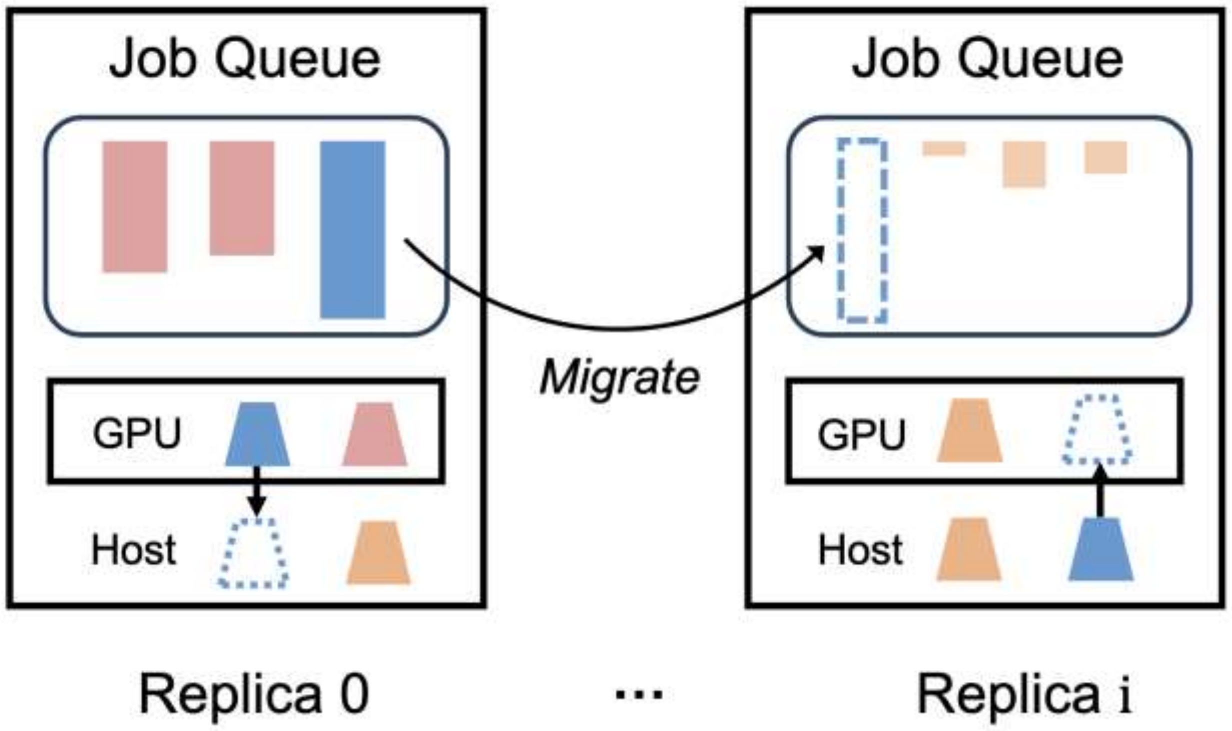
Proactive Mechanism

- **Short term strategy**
  - Estimate pending time for each replica, including adapter load time(if not loaded) and queueing time.
  - Dispatch the request to the replica with the lowest estimate.



Reactive Migration

● Due to the variable input and output lengths of LLM requests, load imbalance still exists

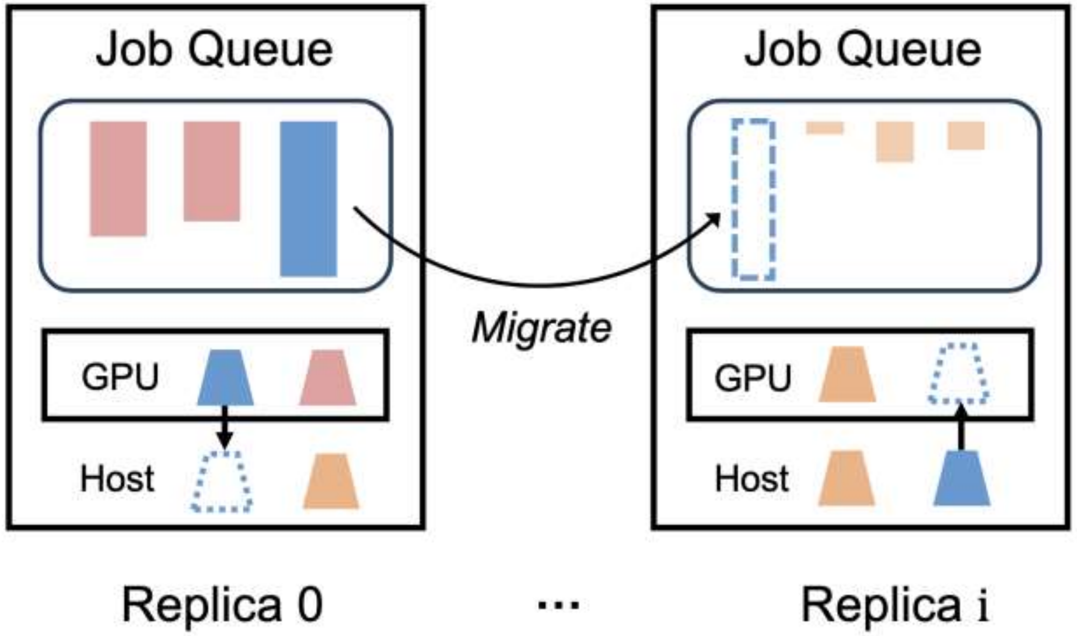


Reactive Migration

● Use ILP to decide how to migrate

● Only triggers when the available GPU memory beyond memory threshold or queuing delay beyond computation threshold.

● Only considers migration between top K overloaded replicas and top K underloaded replicas.



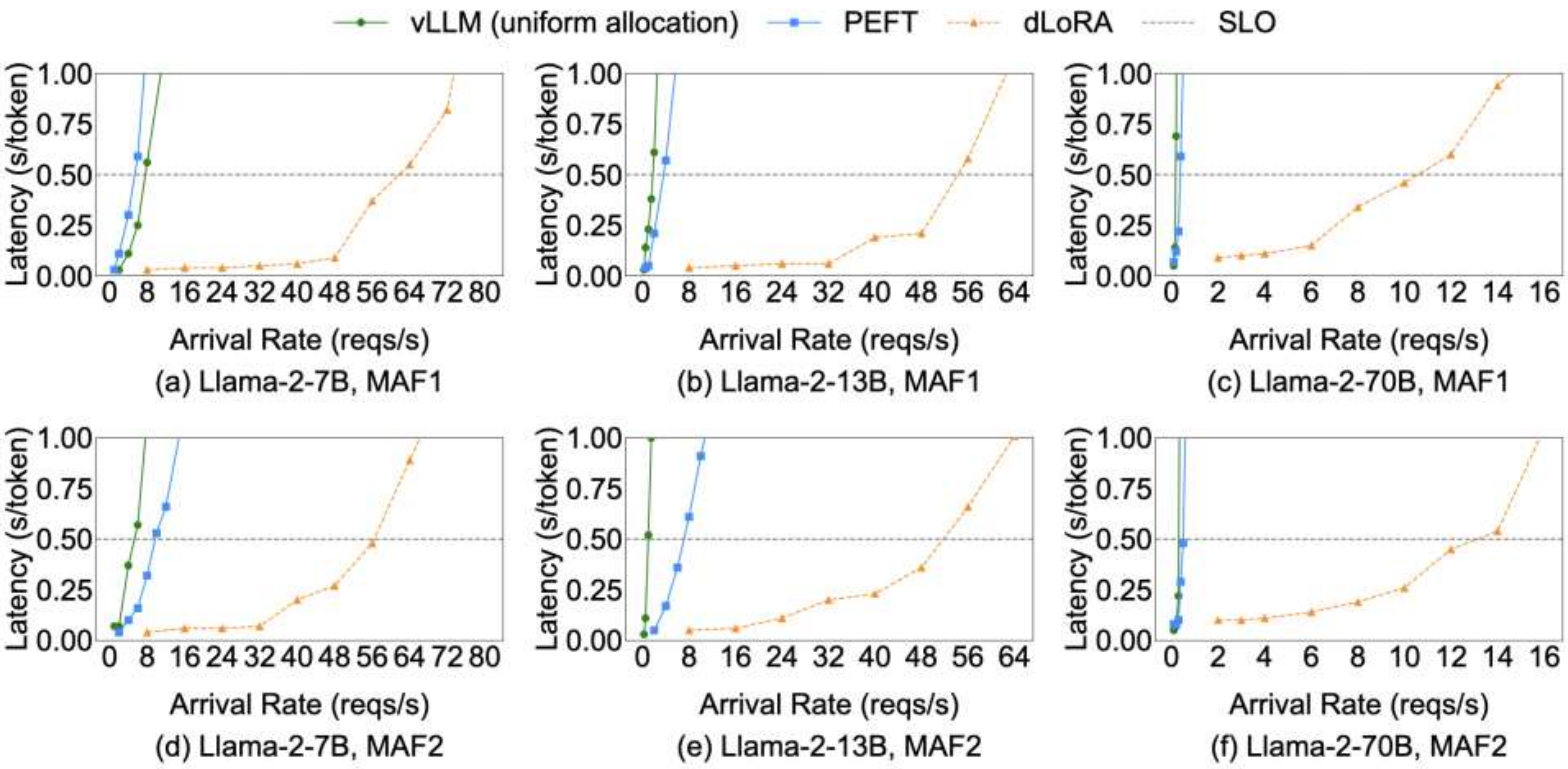
Outline

- Background
- Challenges
- Design
- Implementation & Evaluation
- Summary

Evaluation

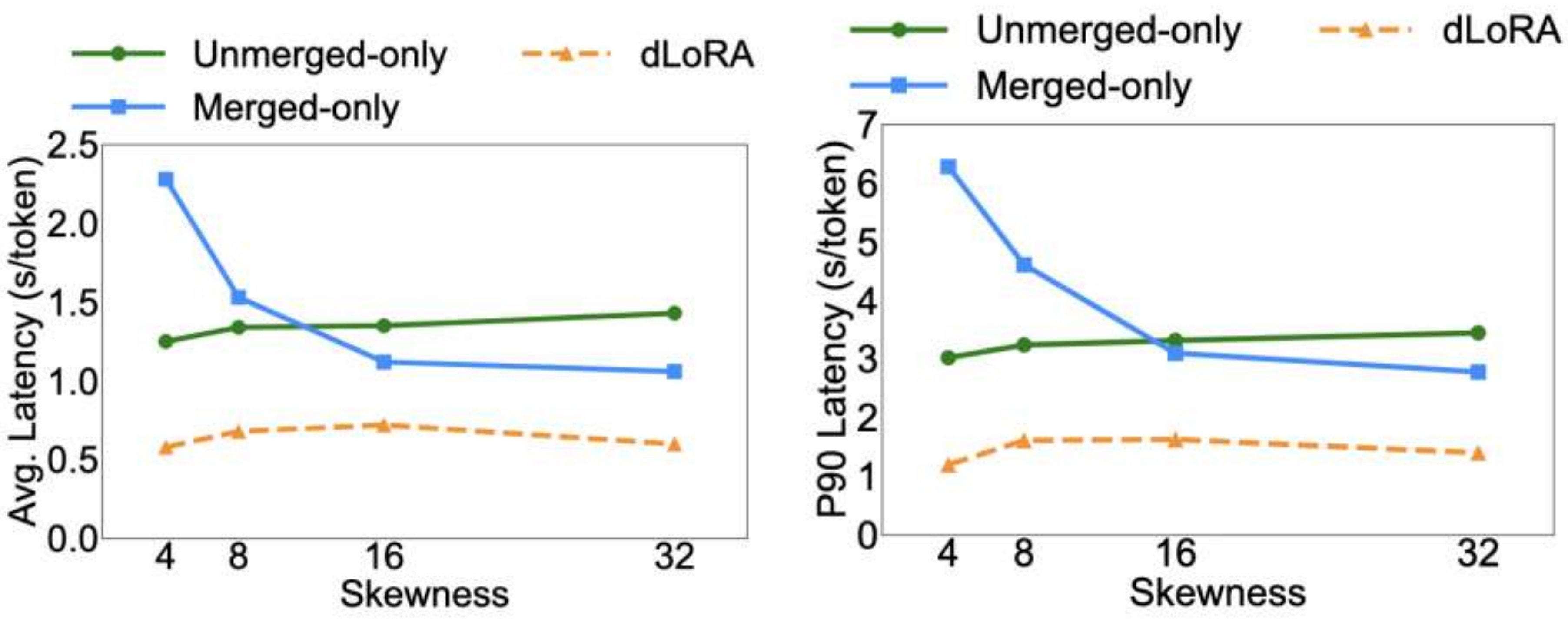
- Implementation
  - Base on based on vLLM
- Experimental Setup
  - Testbed: 4 nodes * 8 NVIDIA A800-80GB GPUs
  - Models: LLaMA-2 (7B, 13B, 70B) + 128 LoRA adapters
  - Dataset: ShareGPT
  - Trace: Microsoft Azure function trace 2019 (MAF1) and 2021 (MAF2)
- Baselines:
  - vLLM
  - HuggingFace PEFT

End-to-end performance



dLoRA improves the throughput by up to 57.9× compared to **vLLM** and up to 26.0× compared to **PEFT** under the SLO requirement.

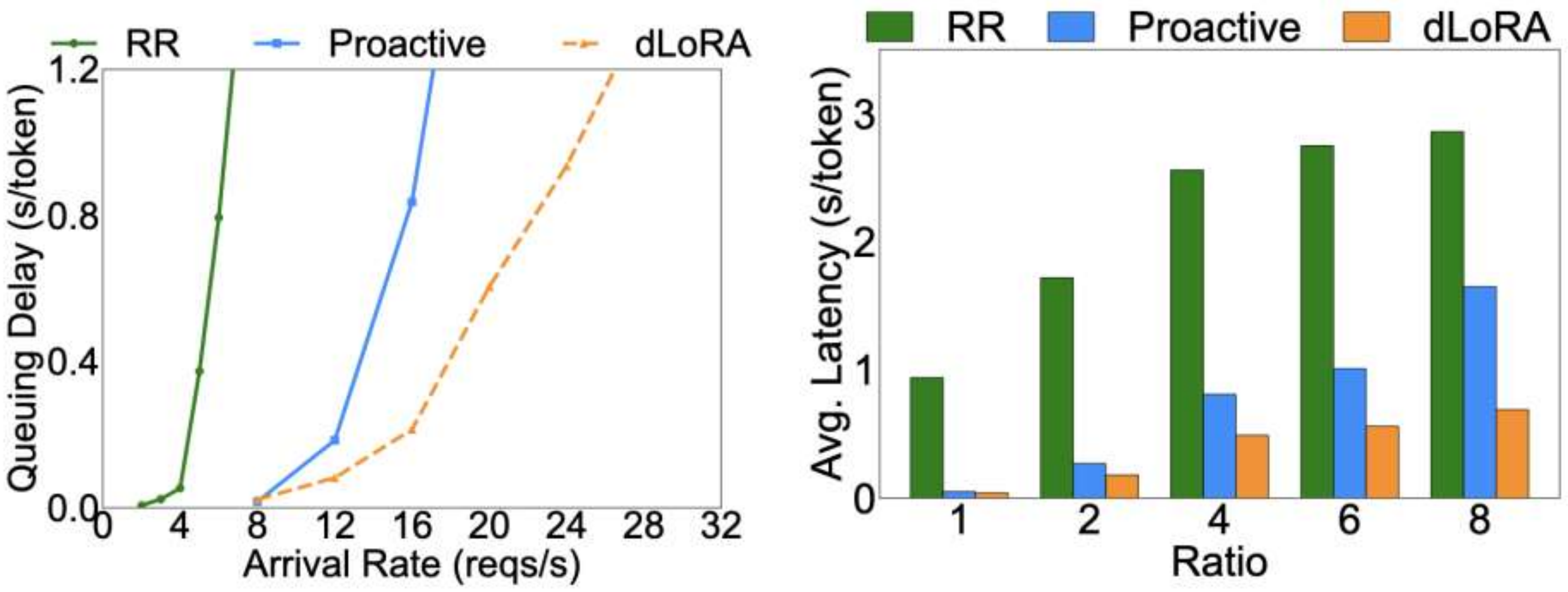
Effectiveness of dynamic batching



(a) Average Latency. (b) P90 Latency.

dLoRA improves the latency by up to 3.9× compared to **Merged-only** and up to 2.4× compared to **Unmerged-only.**

Effectiveness of dynamic load balancing



(a) Reduction in Queuing Delay. (b) Stability under Different Ratios.

dLoRA reduces queueing delay by up to 3.6× compared to **RR** and 1.4× compared to **Proactive Dispatch** under the SLO requirement.

dLoRA reduces average latency by up to 23.5× compared to **RR** and 2.39× compared to **Proactive Dispatch** under the SLO requirement.

Outline

- Background
- Challenges
- Design
- Implementation & Evaluation
- Summary

Summary

- **dLoRA**: an efficient serving system for multi-LoRA LLMs
  - Intra-replica: dynamically merges and unmerges adapters
  - Inter-replica: dynamically migrates both requests and adapters

Thanks!