ServerlessLLM: Low-Latency Serverless Inference for Large Language Models

Yao Fu[1] Leyang Xue[1] Yeqi Huang[1] Andrei-Octavian Brabete[1] Dmitrii Ustiugov[2]
Yuvraj Patel[1] Luo Mai[1]

[1]University of Edinburgh [2]NTU Singapore

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

in 2024 Fall Reading Group Meeting at USTC

Here I am

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Mingxuan Liu**（刘明轩）☺
- 1.5-year PhD student, Supervisor: Prof. **Jianhua Gu**（谷建华）and Dr. **Tianhai Zhao**（赵天海）
- School of Computer, Northwestern Polytechnical University (NPU) since 2015
- NPU HPC Center & Cloud Computing Lab (1 PhD Student + around 8 Master students)
  - **Cluster 1**: 10 CPU nodes + 3 GPU nodes each equipped wtih ***3 V100-32GB***, connected with ***100 Gbps Infiniband/RoCEv2***
  - **Cluster 2**: 4 CPU nodes with ***100Gbps/200Gbps DPU 2/3***, connected with ***100 Gbps P4 Programmable Switch***
  - **Cluster 3**: 5 CPU nodes + 4 GPU nodes, connected with **10 Gbps RoCEv2**
- Research Interests:
  - **Operating System**, **LSM-tree Storage**, **Container/Serverless**, **RDMA-based Disaggregated Memory, Rust for Linux, Programmable Network (SmartNIC/P4-Switch)**, **AI / LLM (Recently, since July, 2024)**
  - However, too fragmented to be in-depth! ☹ Prof. Cheng Li helped me gather and consolidate. ☺
- PhD thesis proposal: Research on ***Serverless Remote Elastic Auto-Scaling System*** Based on ***Programmable RDMA Network*** (Specifically for **AI / LLM scenarios**)

Outline

- **ServerlessLLM: Low-Latency Serverless Inference for Large Language Models**
- **Background**
- **Motivations**
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Designs**
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- **Evaluation**
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- **Discussion & Summary**

Outline

- **Background**
- **Motivations**
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Designs**
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- **Evaluation**
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- **Discussion & Summary**

Background: LLM Serverless inference

NORTHWESTERN POLYTECHNICAL UNIVERSITY

Booming demand for serving custom LLMs

- Open-source models ↑
- Fine-tuned models ↑
- Custom LLM services ↑

Llama-Family, MISTRAL AI_, Qwen, DeepSeek, Microsoft Phi-3, snowflake, Yi

Serverless as a cost-effective solution

Traditional Choices for Model Serving	
Buy a GPU server	Too expensive
Rent a GPU server	Underutilized
Use LLM-Service API	Usage limit & Cannot custom

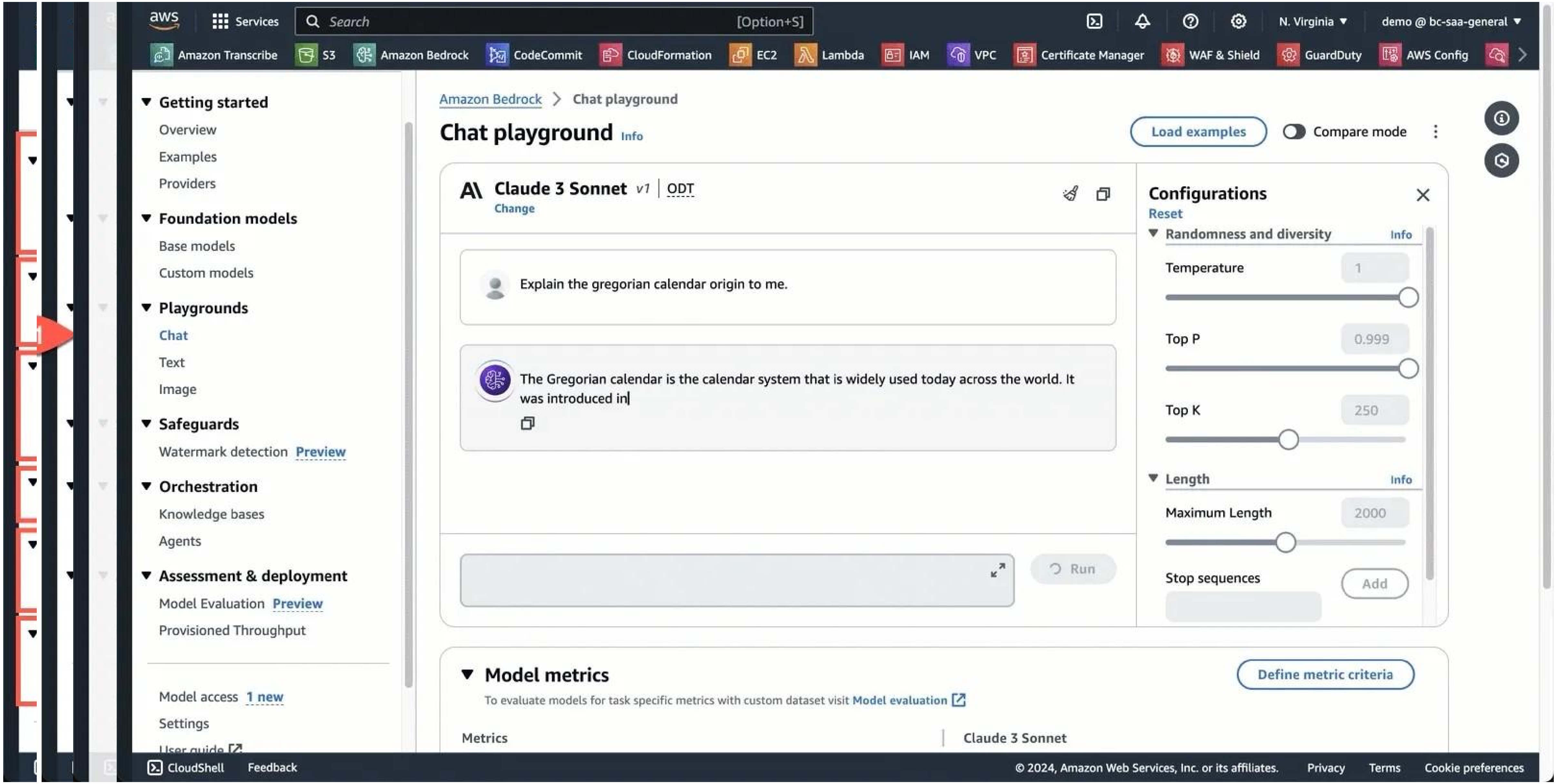
We need a **Pay-as-you-go** Model Serving Platform.

Huge interests from industry and academia

Hundreds competing to develop next-gen AI Serving Platform

Amazon Bedrock, Azure Machine Learning Studio, KServe, Ray Serve

Example: Different LLMs on Amazon Bedrock[3]



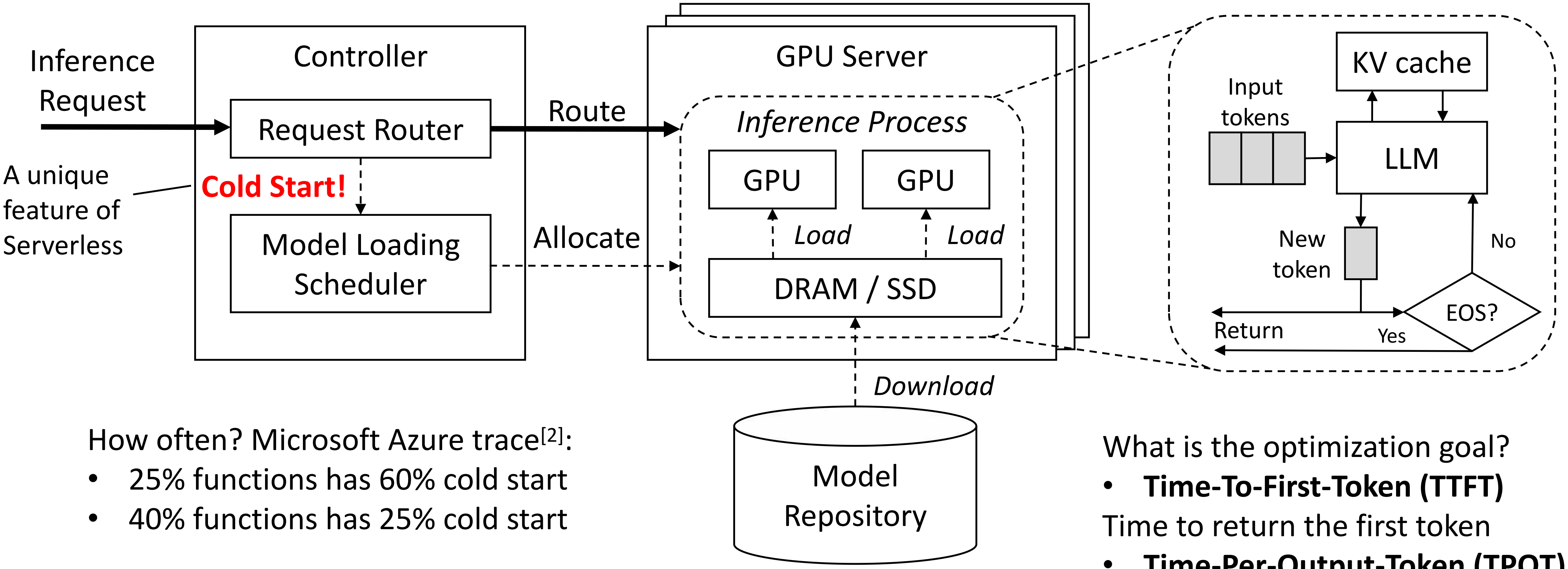
Background

- Suppose **you are a boss of cloud provider**, how to use limited resources to better meet user SLAs?
  - Who is the user? **vs** *The users of traditional LLM serving systems*
    - Companies that want to start a business using LLM
    - People who want to host their private LLM serving system in the cloud
  - What behaviors will users have?
    - Push their models into object storage
    - Run some models to serving for the business

What happens when the above **users deploy hundreds of models**, while **thousands of requests arrive**?

Background: System components in Serverless clusters

- Existing Serverless inference systems: Ray Serve, KServe (Kubernetes)



How often? Microsoft Azure trace[2]:

- 25% functions has 60% cold start
- 40% functions has 25% cold start

What is the optimization goal?

- **Time-To-First-Token (TTFT)**

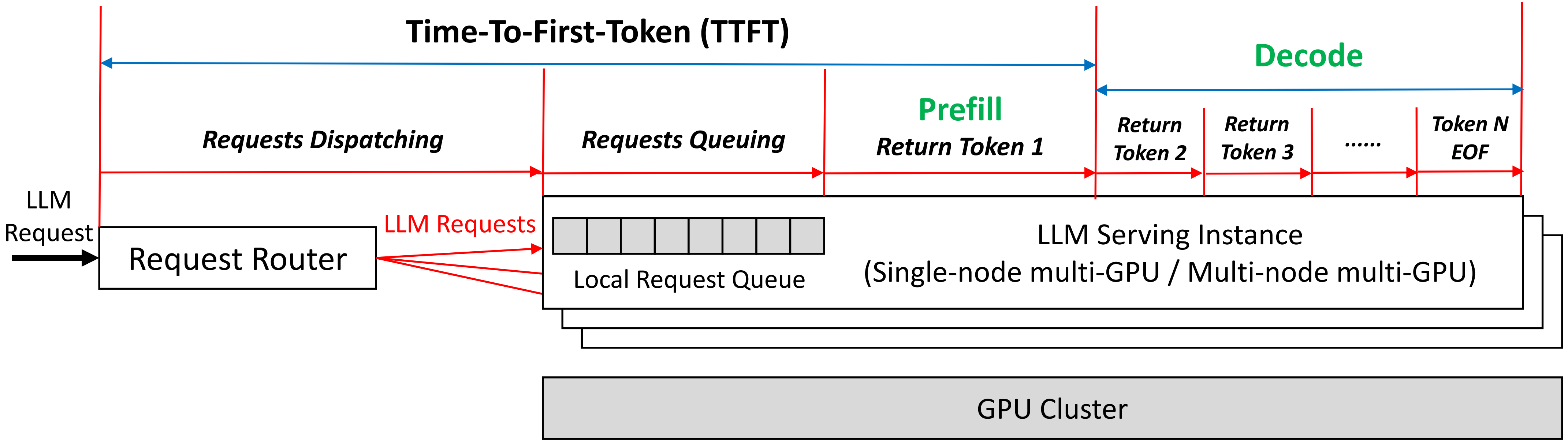
Time to return the first token

- **Time-Per-Output-Token (TPOT)**

Time between each token response

Background: LLM Inference Serving Design Goals

- LLM Inference Cluster Performance Optimization Goal: **Maximize the Token Generation Rate**
- Constraints (***X***, ***Y***, ***M*** are defined according to the scenario):
  - **TTFT** < ***X*** seconds
  - During the decode phase, at least ***M*** tokens must be returned within a window of ***Y*** seconds.



Outline

- Background
- **Motivations**
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Designs
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- Evaluation
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- Discussion & Summary

Motivation

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **(Common) Challenges** in Serverless LLM
- Existing Solutions
- Design Intuitions (to optimize on Existing Solutions)
- **(Special) Challenges** in Optimization beyond Existing Solutions

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

Challenges within Serverless LLM

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Measurement setup: 10Gbps network, A5000 with PCIe 4.0, NVMe SSD

Measured latency (s) of each cold-start step

	Download	Load	Generate 1st token	End-to-end
LLaMA-2-7B	10.8	4.8	0.8	20.1
LLaMA-2-13B	21.0	9.5	0.9	34.5
LLaMA-2-70B	111.9	48.0	8.3	173.7

20X

☺

78%-92% of total TTFT (Time To First Token) latency ☹

Common Challenges | Existing Solutions | Design Intuitions | Special Challenges

Challenges within Serverless LLM

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Cold-start latency !

- (***Remote -> Local***) LLM ckpts are large, prolonging downloads.
  - Example: LLaMA-2-70B (130GB), from S3 takes **26s+** using a fast commodity 5GB/s network
  - Grok-1 -> 600 GB, DBRX -> 250GB, and Mixtral-8x22B -> 280GB
- (***Local Storage -> GPU***) Loading LLM ckpts incurs a lengthy process (even though PCIe-4.0 NVMe SSD).
  - Average **30.27s** (Pytorch) / **16.95s** (Safetensors) between 10 different models
  - Example 1: OPT-30B model into 4 GPUs requires **34s** using PyTorch
  - Example 2: Loading LLaMA-2-70B into 8 GPUs takes **84s** using PyTorch
- The goal of LLM serving system: **TTFT** (Time To First Token) < **100ms !**

Common Challenges → Existing Solutions → Design Intuitions → Special Challenges

Existing Solutions

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

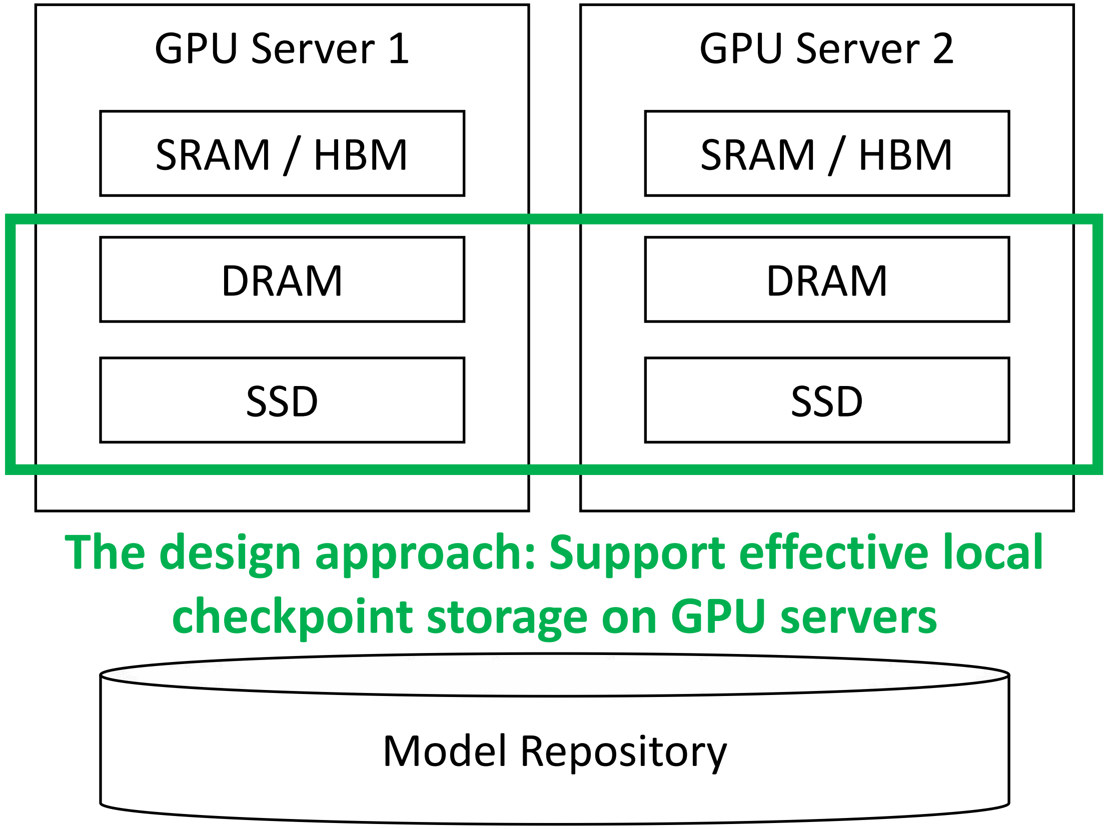
- **Over-subscribing GPUs** -> Expensive (> 5X oversubscription)
  - Maintains **warm GPU instances** to ***bypass*** model download and loading
  - *AWS Serverless Inference*, *Infless@ASPLOS'22*[4] -> only test for small models
  - **Weakness**: smaller models (ResNet, BERT...) is ok, LLM is so EXPENSIVE!
- **Caching checkpoints in host memory** -> Limited capacity (600 GB Grok-1?)
  - *Clockwork@OSDI20*[5], *DeepPlan@EuroSys23*[6] -> only test for small models
  - **Weakness**: smaller models (up to **a few GBs**) is ok, LLM significantly cache misses
- **Deploying additional storage servers** -> Expensive ($16/H for 200 Gb capacity)
  - **Weakness 1**: Slow. Still **20s+** model downloading, even connected to local commodity storage servers equipped with a 100 Gbps NIC
  - **Weakness 2**: Cost.
    - *AWS ElasticCache servers* to support 70B Model, Cost doubled
    - ***cache.c7gn.16xlarge servers*** (210 GB Mem with 200 Gbps Network) $16.3/h (= one ***8-GPU g5.48xlarge server***)

Existing Solutions only efficient for conventional smaller models (up to **a few GBs** is ok!)

Design Intuitions (to optimize on Existing Solutions)

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Facing GPU Cluster with Multi-Tier Storage:
  - **Observation 1: Capacity.** A significant portion of the host memory and storage devices in GPU servers remains **underutilized**.
  - **Observation 2: Bandwidth.** An 8-GPU server utilizing PCIe 5.0 technology **can achieve**:
    - an aggregated bandwidth of 512 GB/s between the host memory and GPUs.
    - around 60 GB/s from NVMe SSDs (RAID 0) to host memory.
    - However, this bandwidth is **not fully utilized**.



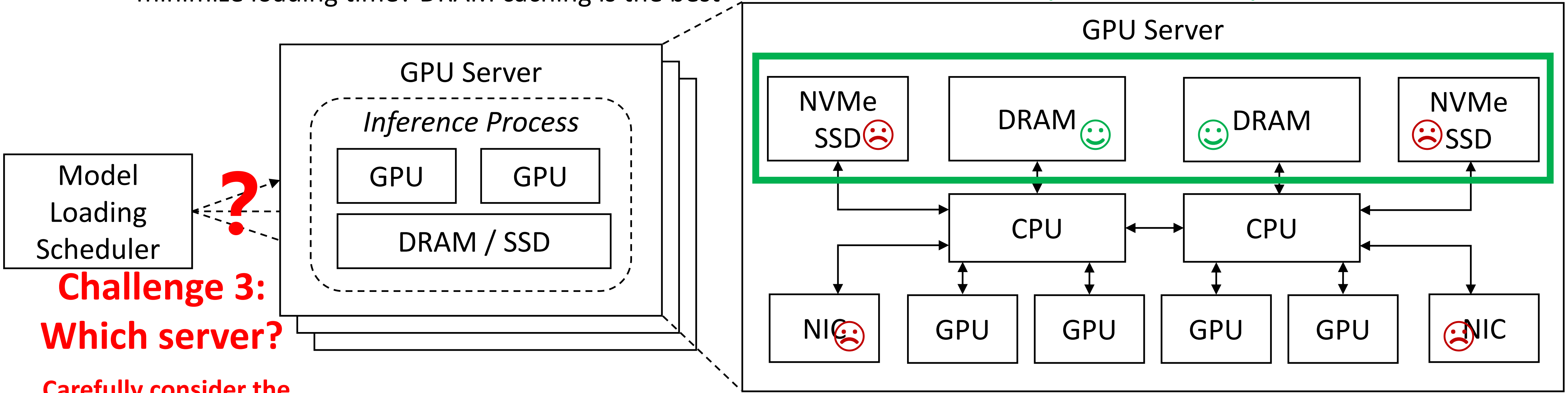
Common Challenges > Existing Solutions > **Design Intuitions** > Special Challenges

Challenges/Optimization beyond Existing Solutions

- How can we fully harness the bandwidth (at **each level** of the Storage Hierarchy) on GPU servers?
- How to use **Locality-Principle (!!)** to select servers to
  - avoid downloading time? SSD caching is better
  - minimize loading time? DRAM caching is the best

Challenge 2: Locality-driven inference

Schedule requests onto GPU servers with locally stored checkpoints (DRAM is the best)



Challenge 3: Which server?

Carefully consider the checkpoint's locality in the entire cluster

Challenge 1: Storage hierarchy is complex

Fully harness the bandwidth at each level of the Storage Hierarchy

Challenges/Optimization beyond Existing Solutions

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Goal: Reduce cold-start latency -> minimize model loading time**
- **For Challenge 1: Support complex multi-tiered storage hierarchy (Capacity & BW)**
    - PyTorch/TensorFlow/ONNX Runtime are primarily designed to enhance the training and debugging, not optimized for read performance.
    - Safetensors can enhance loading performance, but still fail to fully leverage the capabilities of a multi-tiered storage hierarchy.
  - => Need to **fully harness bandwidth** on GPU server. How to do?
- **For Challenge 2: Strong (More Effifient) locality-driven inference**
    - *ClockWork@OSDI20*[5] depend on accurate predictions of model inference time.
    - *Shepherd@NSDI23*[7] **preempt (!!)** current inferences, causing redundant computations.
  - => Workload is interactive and unpredictable durations & preemption-based **locality-driven inference** lead to redundant computations. How to do?
- **For Challenge 3: Scheduling models for optimized startup time**
  - => Need accurately **estimate the startup times** considering the cluster's checkpoint locality. How to do?

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

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Background
- Motivations
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Designs**
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- Evaluation
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- Discussion & Summary

Design 1: Why a new checkpoint design?

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Existing focus

Training scenario

- Persist many, **load few**

Cold-start scenario

- Persist once, **load many**

Mismatch!

PyTorch:

- **34s to load a 40B model.**
- **84s to load a 70B model.**

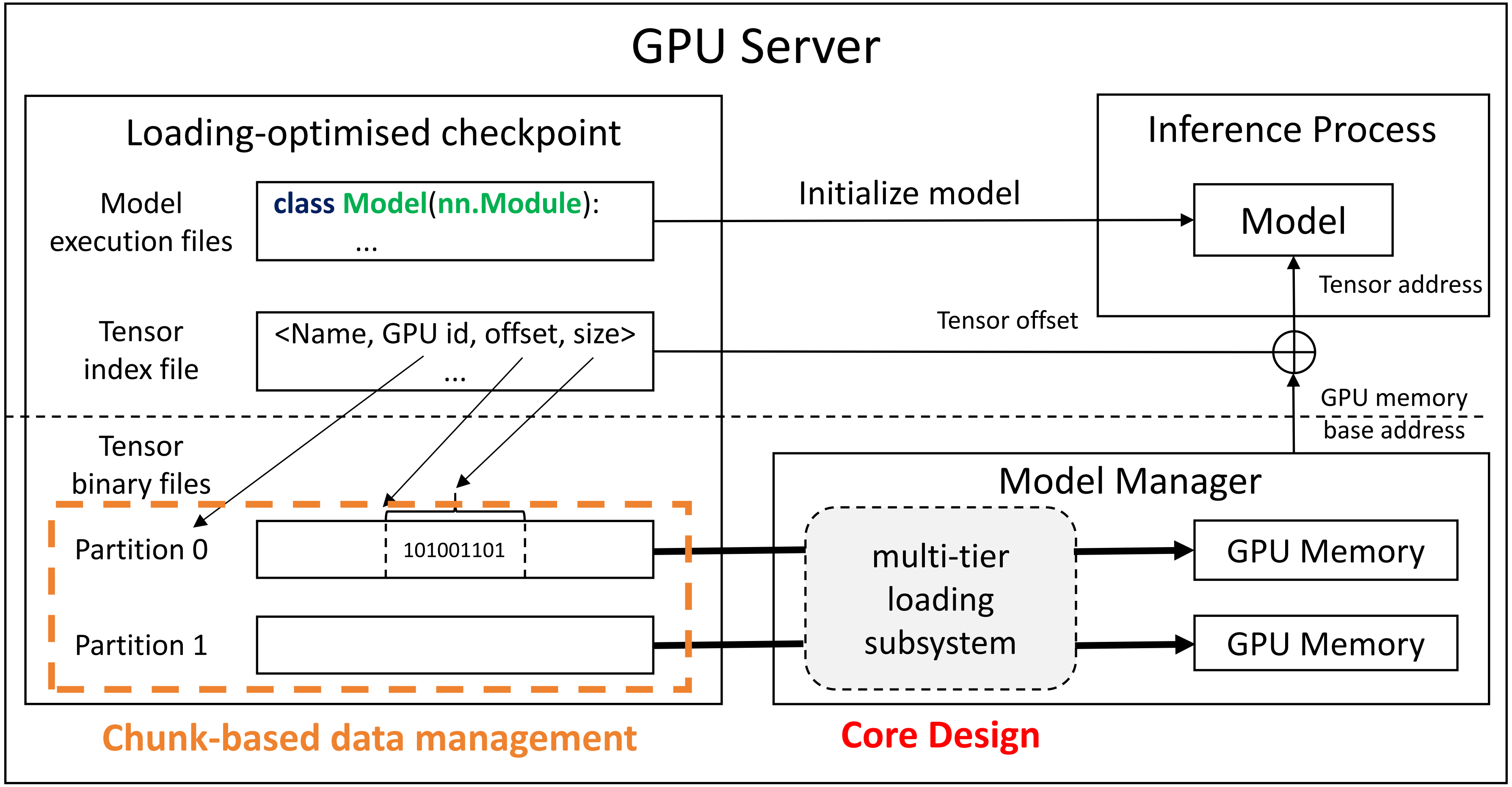
Design 1: Cold-start-friendly checkpoint loading

Decouple model initialization & checkpoint loading

- **Overlapping**
- **Independent**

Avoid blocking GPUs

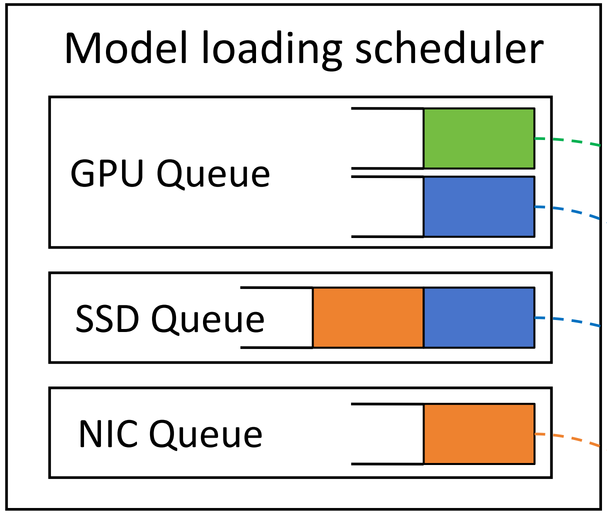
- **GPU-side sequential read**
- **Direct I/O**

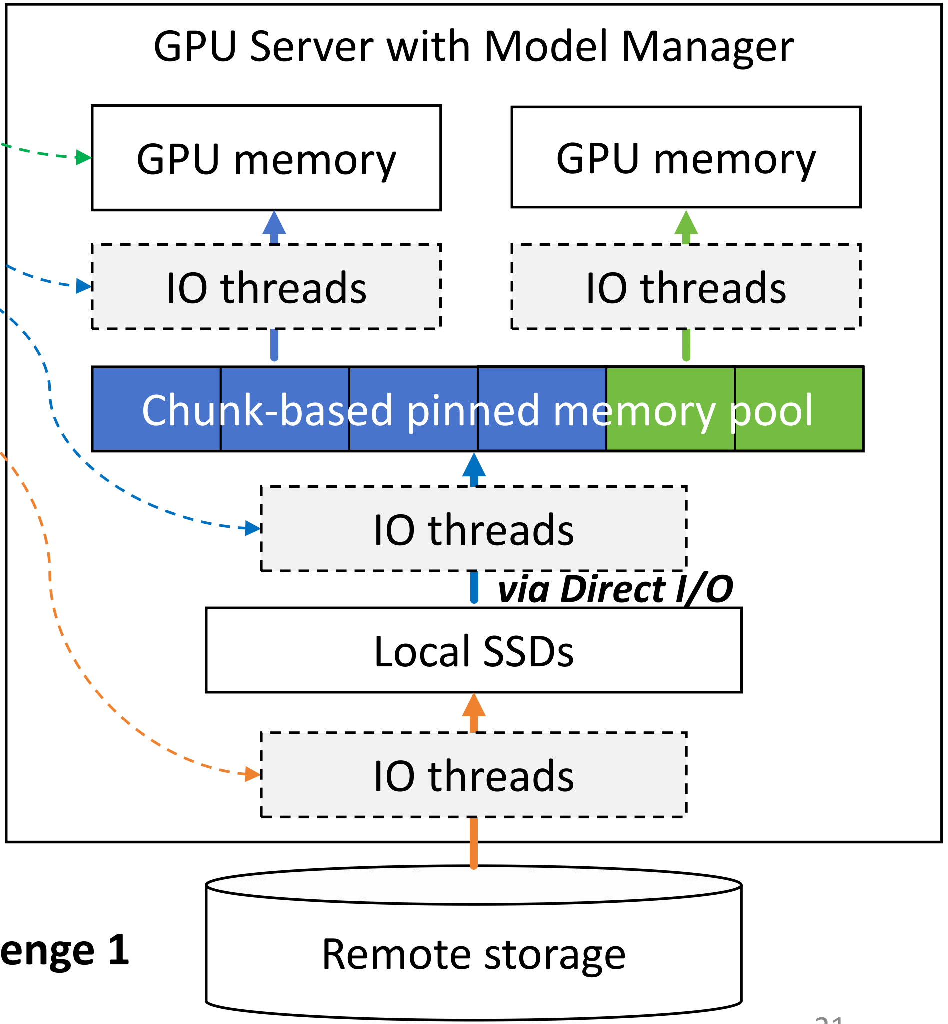


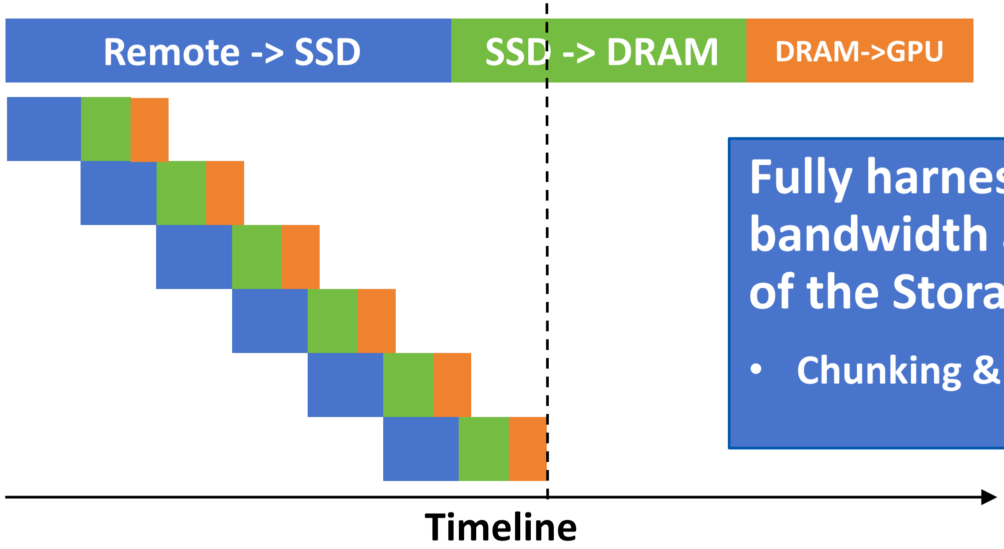
Design 1: Multi-tier Loading Subsystem

Design and benefits:

- Multi-tier pipeline
- IO threads
- Direct I/O
  - open("example.ckpt", **O_DIRECT**)
- Pinned Memory
  - cudaMallocHost







Fully harness the bandwidth at each level of the Storage Hierarchy

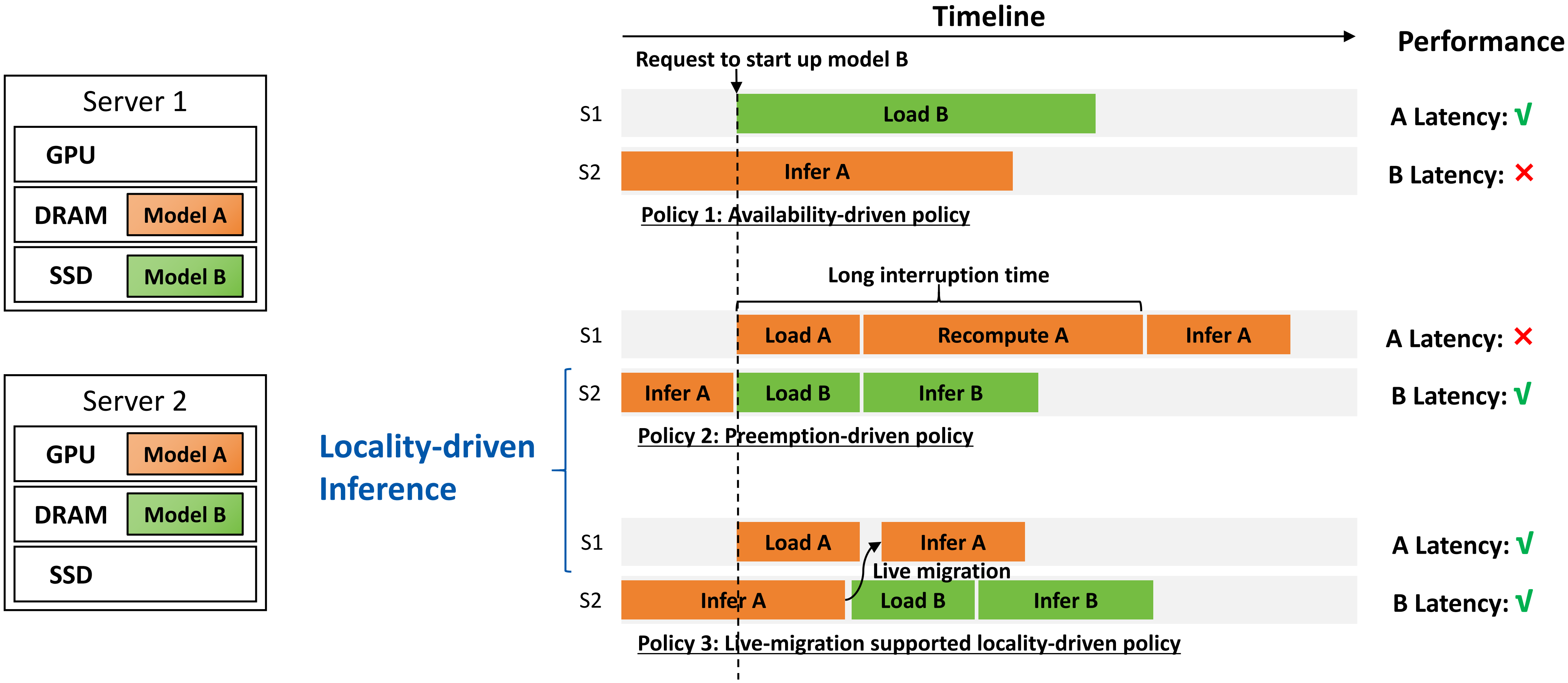
- **Chunking & Overlaping**

√ Solved Challenge 1

Design 2: Locality-driven Inference - Migration is Better

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Example: There are Server 1 and Server 2, suppose there is a request to load Model B, how to do?



Design 2: Live Migration of LLM Inference

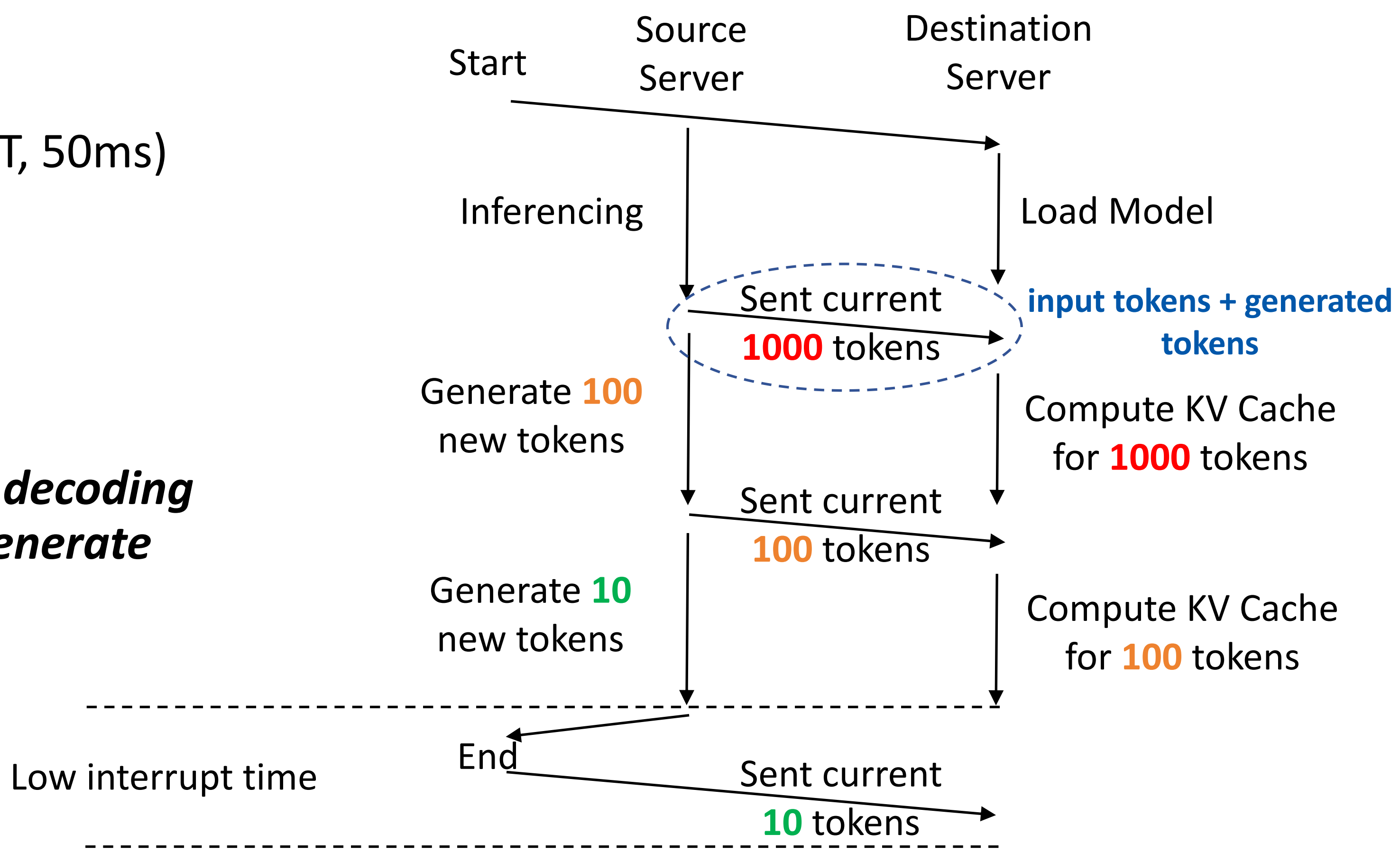
西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

Challenges:

- Large KV Cache (up tp 10s GBs)
- Strict time-per-output-token (TPOT, 50ms)

- **Token is smaller than KV cache**
  - **(8B vs. 100s KB)**

- **Observation: *Prefill* is faster than *decoding* (*Compute KV Cache* is faster than *generate tokens*)**

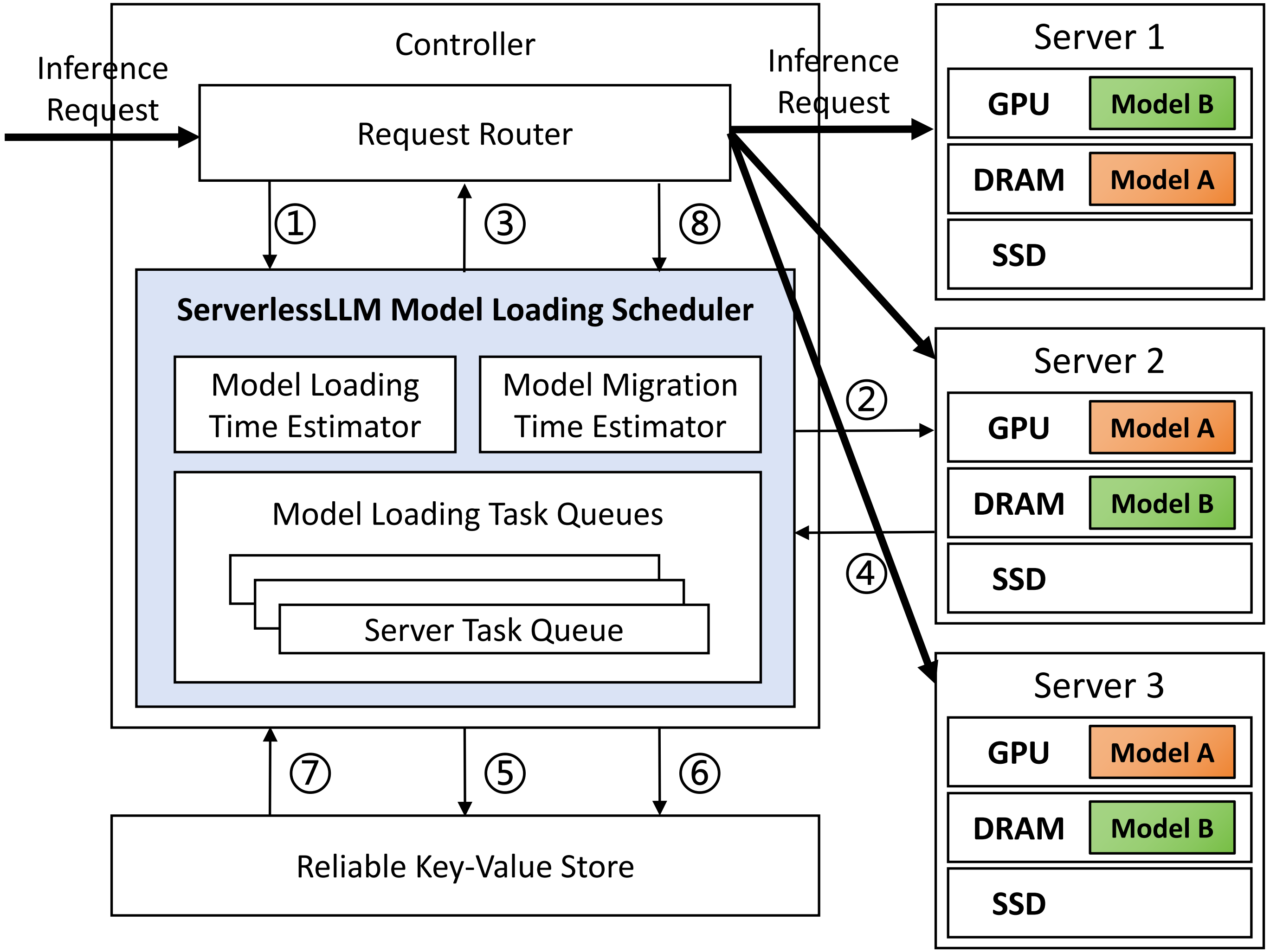


Replace preemption policy with migration-based locality-driven inference

- **Overlaping & Only migrate tokens**

√ Solved Challenge 2

Design 3: ServerlessLLM Model Loading Scheduler



Notify to load Model

① Trigger ***Scheduler*** to select Server for the user-selected Model

② Notify the Server to load the Model (, then ***IO threads*** in server execute tasks from ***Server Task Queue***)

③ Notify ***Request Router*** start to route requests

Monitoring server metrics

④ Collecting server metrics (GPU/DRAM/SSD metrics, local request queue metrics...)

⑤ PUT GPU metrics to KVS

⑥ PUT DRAM/SSD metrics to KVS

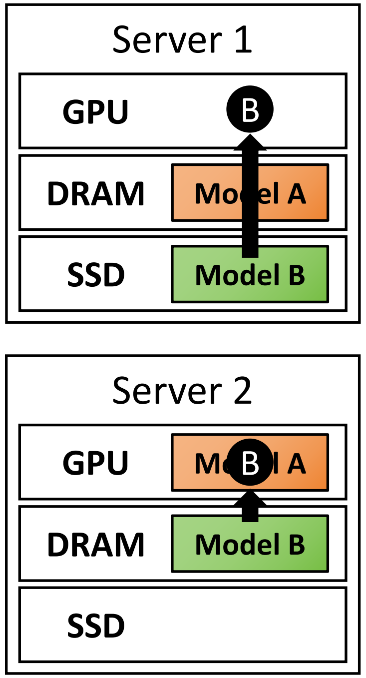
Estimators get metrics

⑦ Estimators GET server metrics

⑧ Estimators GET real-time output tokens

Design 3: Startup-Time-Optimized Model Scheduling

- Example: There are Server 1 and Server 2, suppose there is a request to load Model B, how to do with with **migration-based locality-driven inference**?



OPTION 1: load from SSD

$$T(\text{Startup}) = \boxed{T(\text{SSD Load Model B})}$$

$$\text{T(Storage Load Model)} = \frac{\text{Size(Model)}}{\text{Storage\_bindwidth}}$$

From NVMe SSD >> From Fast Pinned Memory

Model **Loading** Time Estimator

OPTION 2: load from DRAM, migrate A away

$$T(\text{Startup}) = \boxed{T(\text{DRAM Load Model A})} + \boxed{T(\text{Migrate Model A})} + \boxed{T(\text{DRAM Load Model B})}$$

Model **Migration** Time Estimator

$$\text{T(Migrate Model)} = \text{f (num\_of\_input\_tokens, num\_of\_output\_tokens)}$$

Accurately estimate the startup times

- **Monitoring & Two estimator**

√ Solved Challenge 3

Outline

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Background**
- **Motivations**
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- **Designs**
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- **Evaluation**
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- **Discussion & Summary**

Evaluation: Setup

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- **Test bed (i)**: one GPU server
  - 8 NVIDIA A5000 GPUs (24 GB), 1TB DDR4 memory, 2 AMD EPYC 7453 CPUs
  - 2 PCIe 4.0 NVMe 4TB SSDs (in RAID 0), 2 SATA 3.0 4TB SSDs (in RAID 0)
  - 1 Remote MinIO with 1Gbps network
- **Test bed (ii)**: 4 GPU servers connected with 10 Gbps Ethernet, each server:
  - 4 A40 GPUs (48 GB), 512 GB DDR4 memory, 2 Intel Xeon Silver 4314 CPUs
  - 1 PCIe 4.0 NVMe 2TB SSD
- **Models**:
  - OPTs (2.7B, 6.7B, 13B, 30B and 66B), LLaMAs (7B, 13B, 70B), Falcon (7B, 40B)
  - For cluster evaluation on test bed (ii):
    - replicate OPT-6.7B/OPT-13B/OPT-30B models for 32/16/8 instances respectively that are treated as **different models**, thus total 32+16+8=56 type of models.
    - replicate each model and distribute them across **nodes' SSDs** using round-robin placement until **the total cluster-wide storage limit** is reached.

Evaluation: Setup

NORTHWESTERN POLYTECHNICAL UNIVERSITY

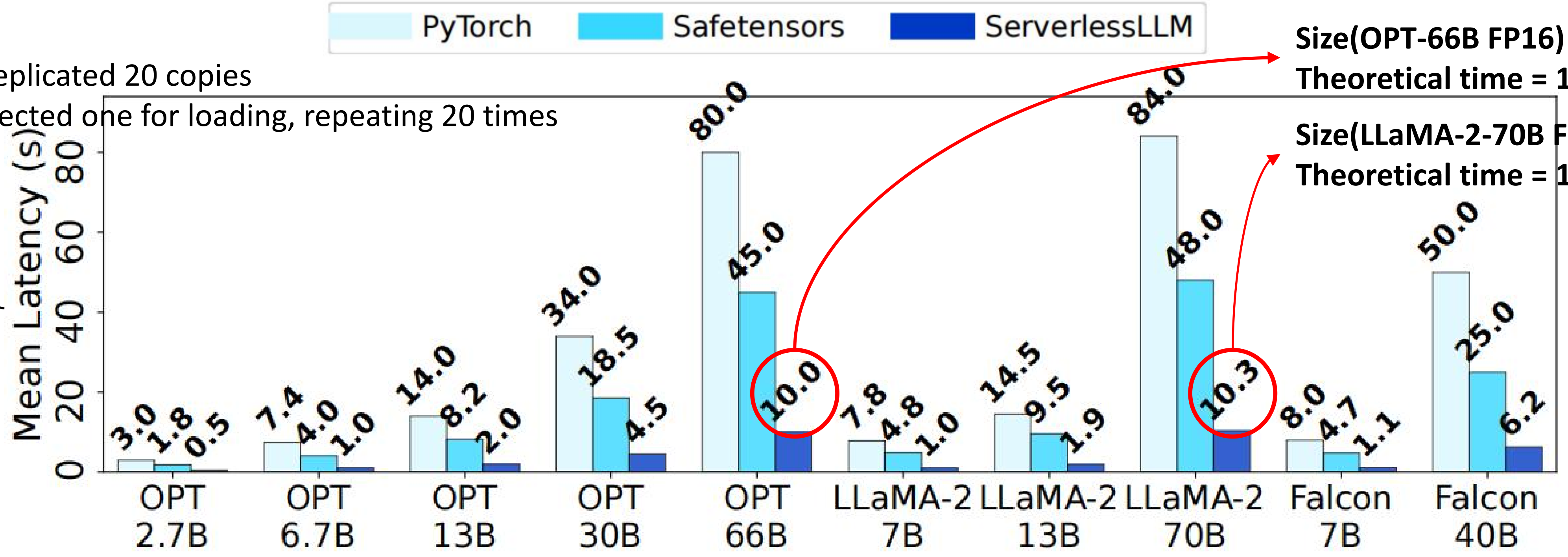
- **Datasets**:
  - GSM8K - contains problems created by human problem writers
  - ShareGPT - contains multilanguage chat from GPT4
- **Workloads**: (for cluster evaluation on **test bed (ii)**)
  - Real-world Trace: *AzureFunctionsInvocationTrace2021@SOSP21*[8]
    - This is a trace of function invocations for *two weeks starting on 2021-01-31*, containing invocation arrival and departure (or compeletion) times, with the followiwng schema:
      - app: application id (encrypted)
      - func: function id (encrypted), and unique only within an application
      - end_timestamp: function invocation end timestamp in millisecond
      - duration: duration of function invocation in millisecond
  - Use Gamma distribution (CV=8) to generate **the desired RPS**

Evaluation 1-1: ServerlessLLM Checkpoint Loading

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Test bed (i): a GPU server with 8 NVIDIA A5000 GPUs (24 GB)
- Load all types of models in **FP16** from **RAID0-NVMe** (Thpt = 12 GB/s).

Each model replicated 20 copies
Randomly selected one for loading, repeating 20 times



Size(OPT-66B FP16) = 132 GB
Theoretical time = 132/12 = 11s

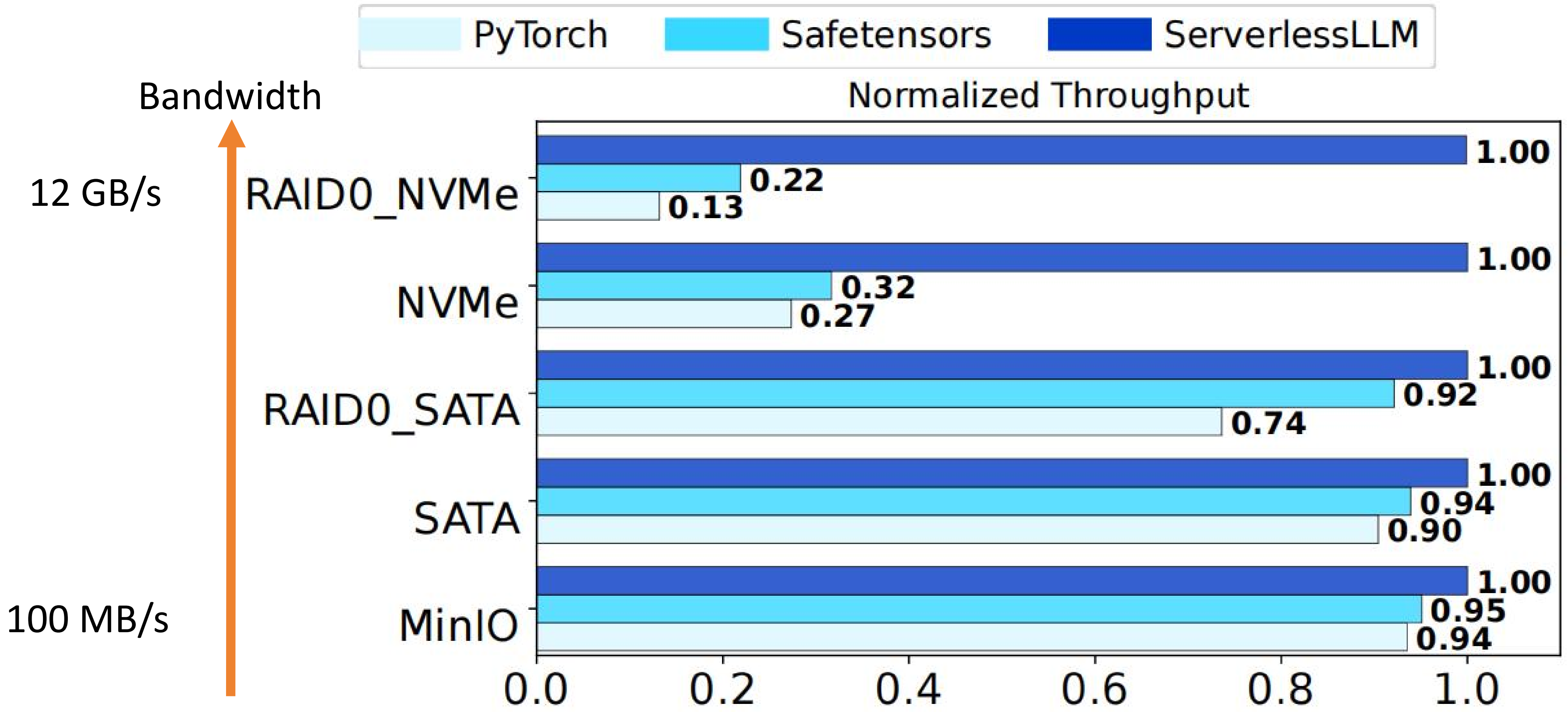
Size(LLaMA-2-70B FP16) = 140 GB
Theoretical time = 140/12 = 11.67s

- The average loading time of ServerlessLLM: 3.85s
- Smallest model (OPT-2.7B): 6X and 3.6X faster than PyTorch and Safetensors, respectively.
- Largest model (LLaMA-2-70B): 8.2X and 4.7X faster than PyTorch and Safetensors, respectively.
- The loading performance is agnostic to the type of the model. OPT-13B and LLaMA-2-13B is similar.

Evaluation 1-2: ServerlessLLM Checkpoint Loading

西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- Test bed (i): a GPU server with 8 NVIDIA A5000 GPUs (24 GB), loading **LLaMA-2-7B** from different storage media

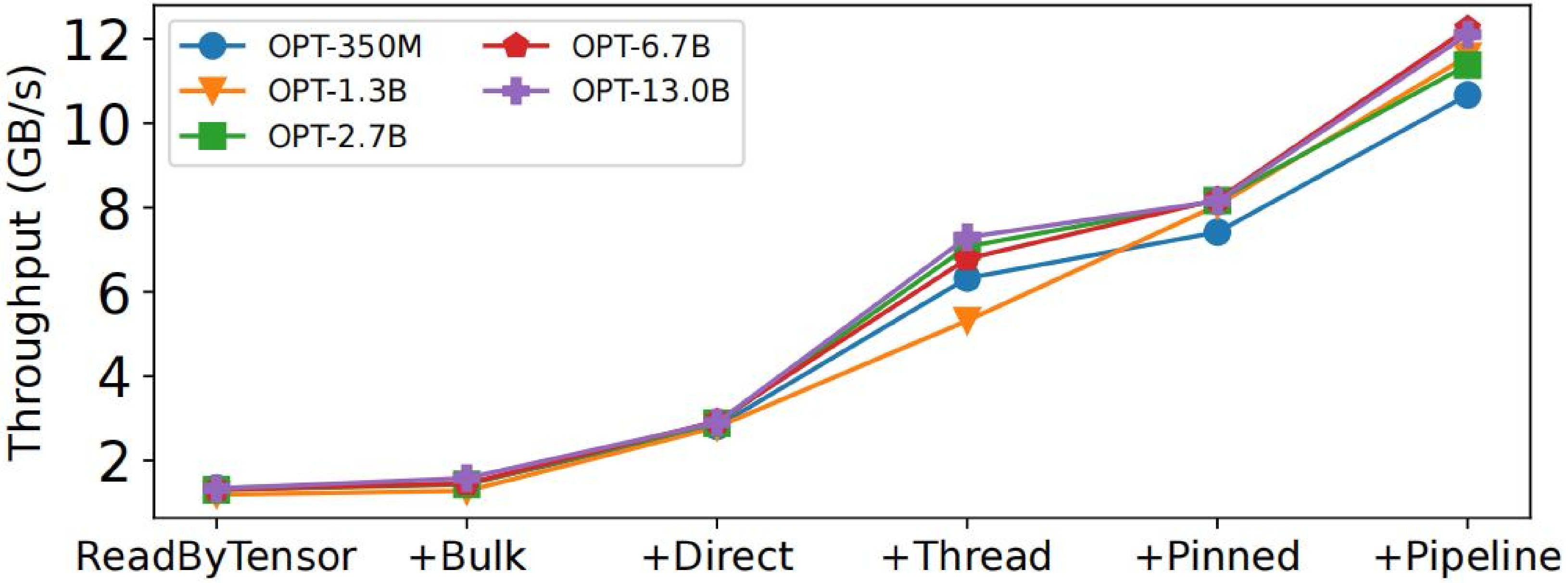


- Baseline 1: **Thpt** of storage device. Use FIO with asynchronous 4M direct sequential read (depth = 32).
- Baseline 2: **Thpt** of MinIO. Use the official MinIO benchmark.
- ServerlessLLM harnesses different storage mediums and saturating entire bandwidth.

Evaluation 1-3: ServerlessLLM Checkpoint Loading

西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

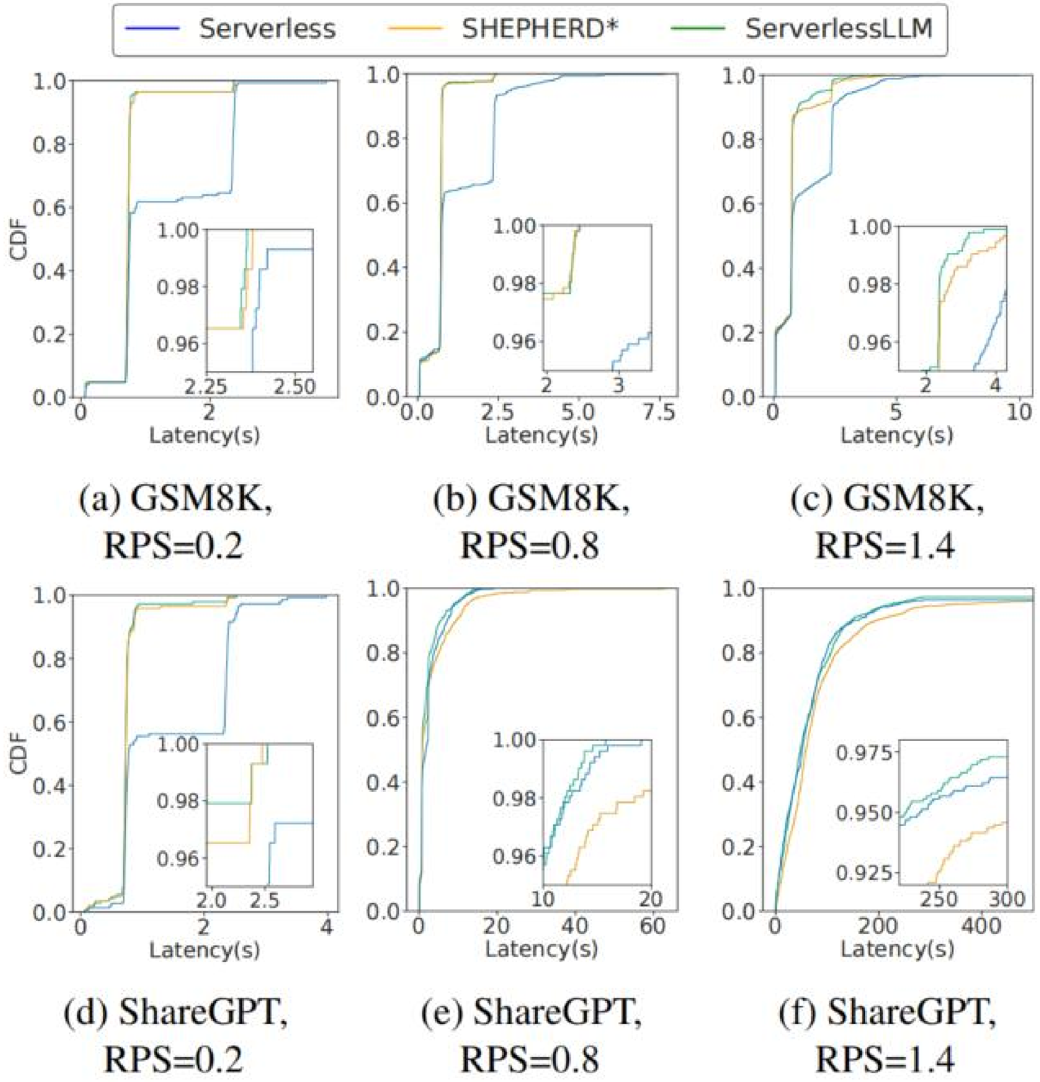
- Test bed (i): a GPU server with 8 NVIDIA A5000 GPUs (24 GB) and **RAID0-NVMe (**Thpt = **12 GB/s)**
- Run ServerlessLLM in a container, limit **4 CPU cores**, Chunk size = 16MB, Pinned mem size = ?



- **Bulk reading** improves 1.2x throughput, mitigating the throughput degradation from reading small tensors (on average one-third of the tensors in the model are less than 1MB).
- **Direct IO** improves 2.1x throughput, bypassing cache and data copy in the kernel.
- **Multi-thread** improves 2.3x throughput, as multiple channels within the SSD can be concurrently accessed.
- **Pinned memory** provides a further 1.4x throughput, bypassing the CPU with GPU DMA.
- **Pipeline** provides a final 1.5x improvement in throughput, helping to avoid synchronization for all data on each storage tier.

Evaluation 2-1: ServerlessLLM Model Scheduler

- Test bed (ii): 4 GPU servers connected with 10 Gbps Ethernet, scheduling **OPT-6.7B model**



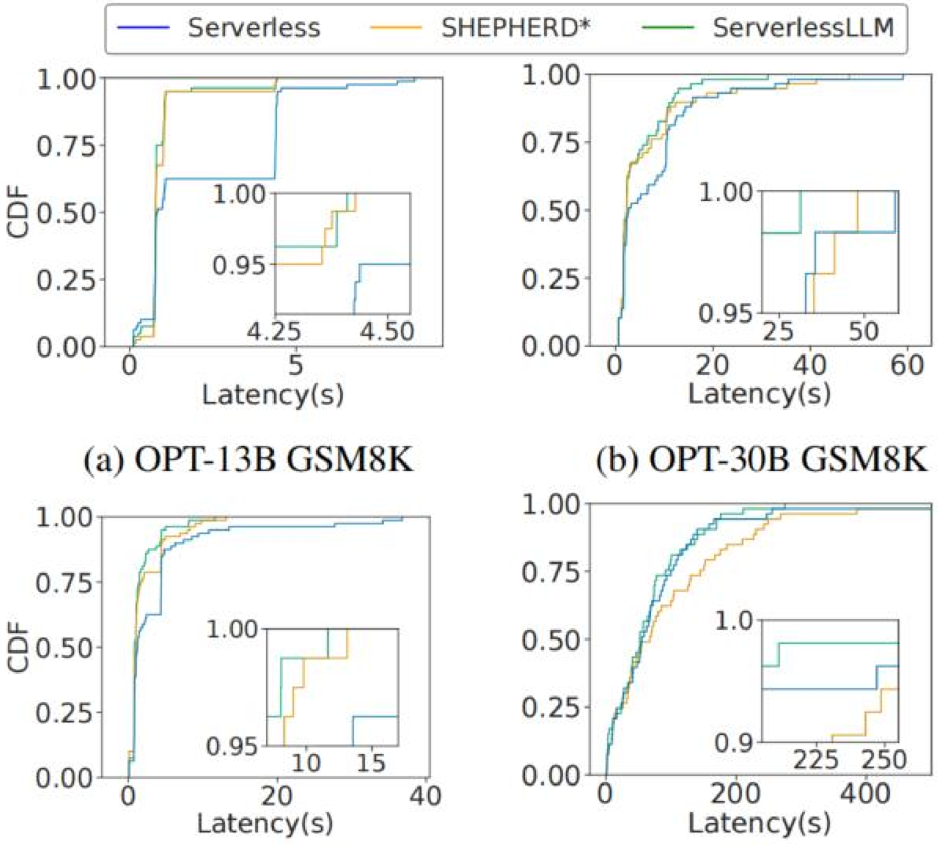
Baseline 1: Serverless scheduler (w/o any optimization for loading & randomly chooses any GPU available) -> **Available-driven**

Baseline 2: Shepherd rely on **preemption** (while ServerlessLLM will rely on live migration) + ServerlessLLM's loading time estimation strategy -> **Locality-driven** (Any optimization for loading? not metioned)

- (a)/(b)/(d): No migration or preemption, similar with Shepherd
- (e): Shepherd **2X higher** P99 latency due to preemption.
  - **114** migrations/40 preemptions of 513 total requests
- (c): Shepherd **1.27X higher** P99 latency due to preemption.
  - **53** migrations/9 preemptions of 925 total requests
  - **2X times** read from SSD than ServerlessLLM
- (f): Shepherd **1.5X higher** P99 latency due to preemption.
  - **64** migrations/166 preemptions of 925 total requests
  - GPU occupancy reaches 100% for all three

Evaluation 2-2: ServerlessLLM Model Scheduler

- Test bed (ii): 4 GPU servers connected with 10 Gbps Ethernet, scheduling **OPT-13B/30B model** (RPS = ?, not metioned)



(a) OPT-13B GSM8K (b) OPT-30B GSM8K

(c) OPT-13B ShareGPT (d) OPT-30B ShareGPT

- locality-aware scheduling is more important for **larger models** as caching them in host memory
- (a)/(b)/(c): Serverless Scheduler, 35-40% times wasting in loading from SSD
- (d) For the OPT-30B ShareGPT, the model size is 66 GB. Hence, only **two models** can be stored in the GPU memory (4 A40 48GB GPUs, 4×48=192GB)
- Even in this extreme case, ServerlessLLM still achieves 35% and 45% lower P99 latency compared to Serverless and Shepherd

Evaluation 3: Entire ServerlessLLM in Action

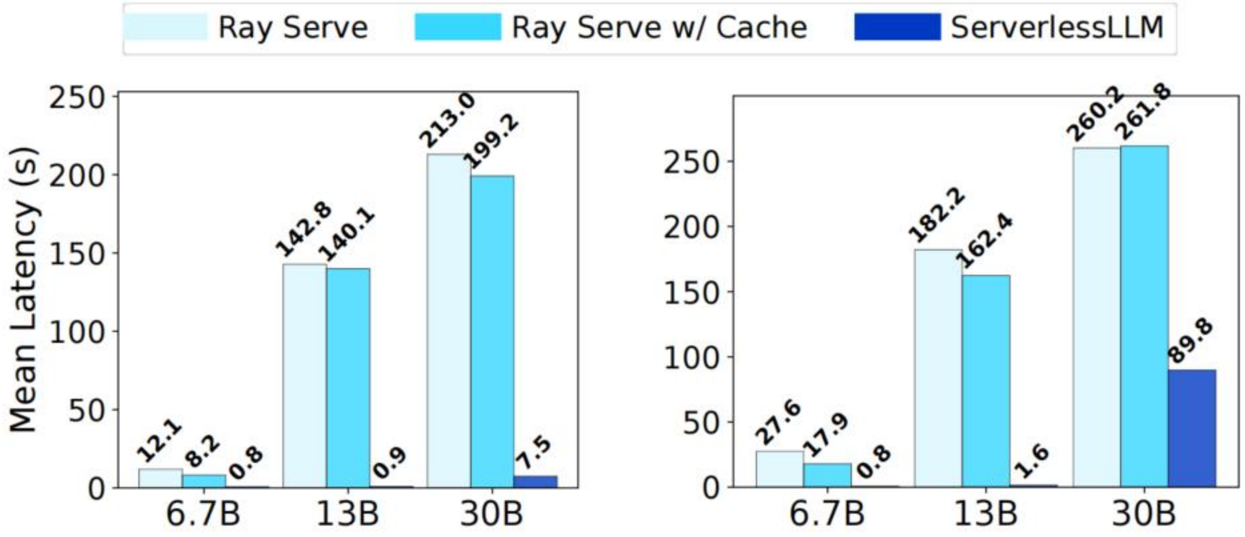
西北工業大學 NORTHWESTERN POLYTECHNICAL UNIVERSITY

- For cluster evaluation on test bed (ii)
- **Baseline**:
  - ***Ray Serve*** (Version 2.7.0) (Always download from Reomte Storage) + Safetensors
  - ***Ray Serve w/ Cache*** (adopt a local SSD cache utilizing the LRU policy to avoid costly model downloads) + Safetensors
  - ***KServe*** (Version 0.10.2), the SOTA serverless inference system designed for Kubernetes clusters
- **For best performance:**
  - ***Ray Serve*** and ***Ray Serve w/ Cache*** are both **storing model checkpoints on local SSDs** before testing
  - Assuming exclusive **10 Gbps** network to estimate download latency
  - Set the maximum concurrency to one (only **one request** is processed at a time)
  - Launch parallel LLM inference **clients** to generate various workloads
  - Each request has a timeout threshold of 300 seconds

Evaluation 3-1: Loading-optimized checkpoints

- Test bed (ii): 4 servers connected with 10 Gbps Ethernet, processing the request loading **OPT-6.7B/13B/30B.**

The average latency per **start-up (loading)** in **a complete serverless workload** (Azure Trace)

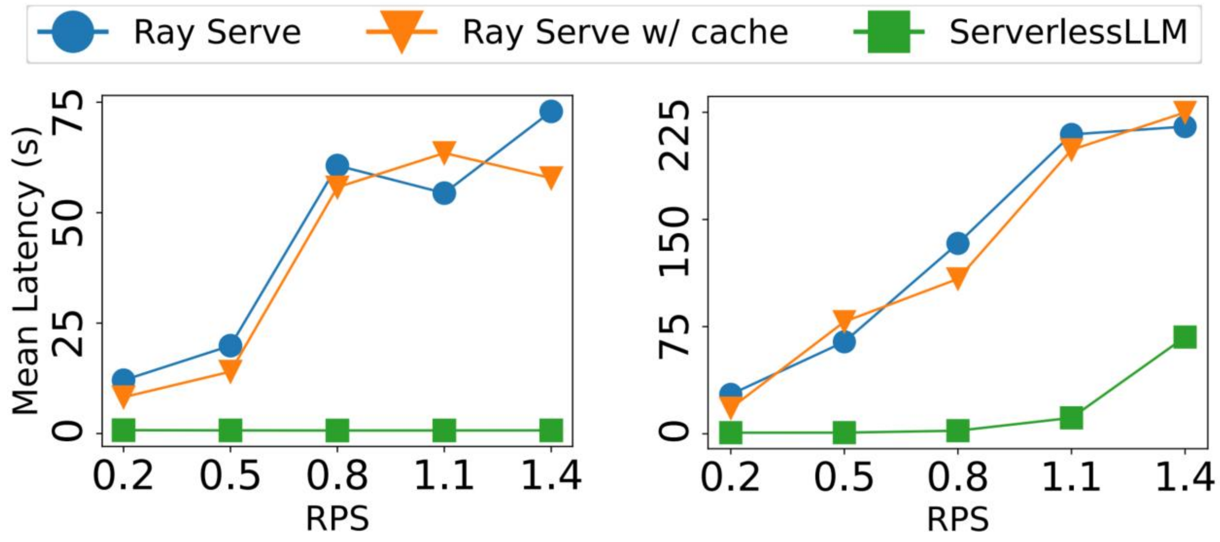


(a) GSM8K (b) ShareGPT

- GSM8K: ServerlessLLM can fulfill **89%** of requests within a 300-second timeout with OPT-30B, whereas Ray Serve with Cache manages only **26%**.
- ShareGPT: When utilizing OPT-30B, ServerlessLLM begins to confront GPU limitations (with **all GPUs occupied** and **migration unable find more resources**), leading to an increased latency of **89.9s**.

Evaluation 3-2: Live Migration & Loading Scheduler

- Test bed (ii): 4 servers connected with 10 Gbps Ethernet
- Replicate OPT-6.7B/13B/30B models for 32/16/8, simulating **56 different models**

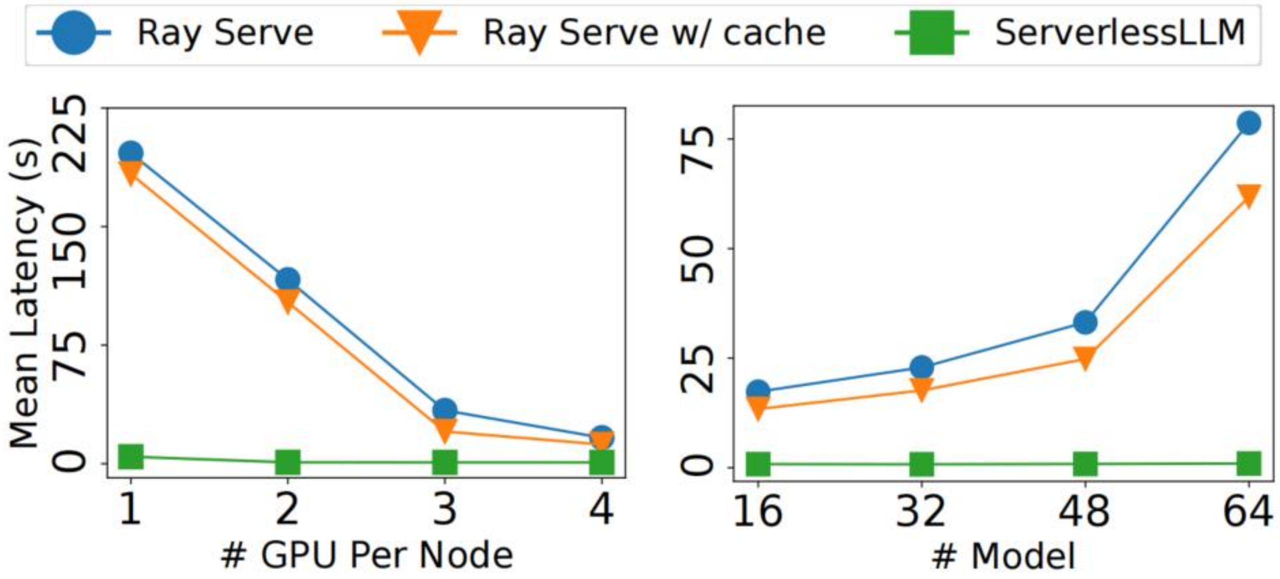


(a) GSM8k (b) ShareGPT

- GSM8K: ServerlessLLM consistently maintains low latency, approximately **1 second**
- ShareGPT: ServerlessLLM maintains performance improvements up to **212X**. At an RPS of 1.4, ServerlessLLM's latency begins to rise. Despite **live migration** and **optimized server scheduling**, the limited GPU resources eventually impact performance.

Evaluation 3-3: Resource Utilization

- Test bed (ii): 4 servers connected with 10 Gbps Ethernet
- Replicate OPT-6.7B/13B/30B models for 32/16/8, simulating **56 different models**. (RPS = ?, not metioned)



(a) Impacts of # GPUs per node (b) Impacts of # models

- ServerlessLLM scales well with elastic resources.
- As the number of models grows, the performance gap widens, showcasing ServerlessLLM's potential suitability for largescale serverless platforms.

Outline

- Background
- Motivations
  - (Common) Challenges in Serverless LLM
  - Existing Solutions
  - Design Intuitions (to optimize on Existing Solutions)
  - (Special) Challenges in Optimization beyond Existing Solutions
- Designs
  - Multi-Tier Checkpoint Loading
  - Live Migration of LLM Inference
  - Startup-Time-Optimized Model Scheduling
- Evaluation
  - Test on one GPU Server with 8 A8000 GPUs
  - Test on GPU Cluster, each GPU Server with 4 A40 GPUs
- **Discussion & Summary**

Discussion & Summary

- Pros
  - Solve the challenge of two hottest areas: Serverless and LLMs.
  - Low Latency and Efficient Resource Utilization
  - Scalability and Cost Efficiency
- Cons
  - Treat Ray Serve as a serverless platform (as a baseline for evaluation). Maybe Ray Serve over k8s is more comfortable.
  - Not discuss the impact of the size of the KV Cache in Live-Migration scenario
  - It would be better to provide a Scheduler algorithm.
  - Assume the case where the model can be completely put into the GPU memory of a Node. What about larger models? How to parallelize models in the Serverless scenario?
  - The Implementation section is missing.

Reference

[1] Fu, Yao, et al. "{ServerlessLLM}:{Low-Latency} Serverless Inference for Large Language Models." 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). 2024.

[2] Shahrad, Mohammad, et al. "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider." 2020 USENIX annual technical conference (USENIX ATC 20). 2020.

[3] Brandon Carroll, "Getting started with different LLMs on Amazon Bedrock", https://community.aws/content/2fVW67K1gRKNVzP5xyZ4ADIcFEf/getting-started-with-different-llms-on-amazon-bedrock?lang=en

[4] Yang, Yanan, et al. "INFless: a native serverless system for low-latency, high-throughput inference." Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022.

[5] Gujarati, Arpan, et al. "Serving {DNNs} like clockwork: Performance predictability from the bottom up." 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). 2020.

[6] Jeong, Jinwoo, Seungsu Baek, and Jeongseob Ahn. "Fast and efficient model serving using multi-GPUs with direct-host-access." Proceedings of the Eighteenth European Conference on Computer Systems. 2023.

[7] Zhang, Hong, et al. "{SHEPHERD}: Serving {DNNs} in the wild." 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

[8] Zhang, Yanqi, et al. "Faster and cheaper serverless computing on harvested resources." Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021.

Thank you!