OSDI'24 ServerlessLLM: Locality-Enhanced Serverless Inference for Large Language Models

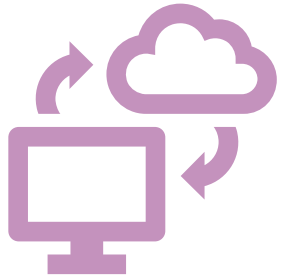
Haodong Tian

May 29, 2024

Why LLM x Serverless?

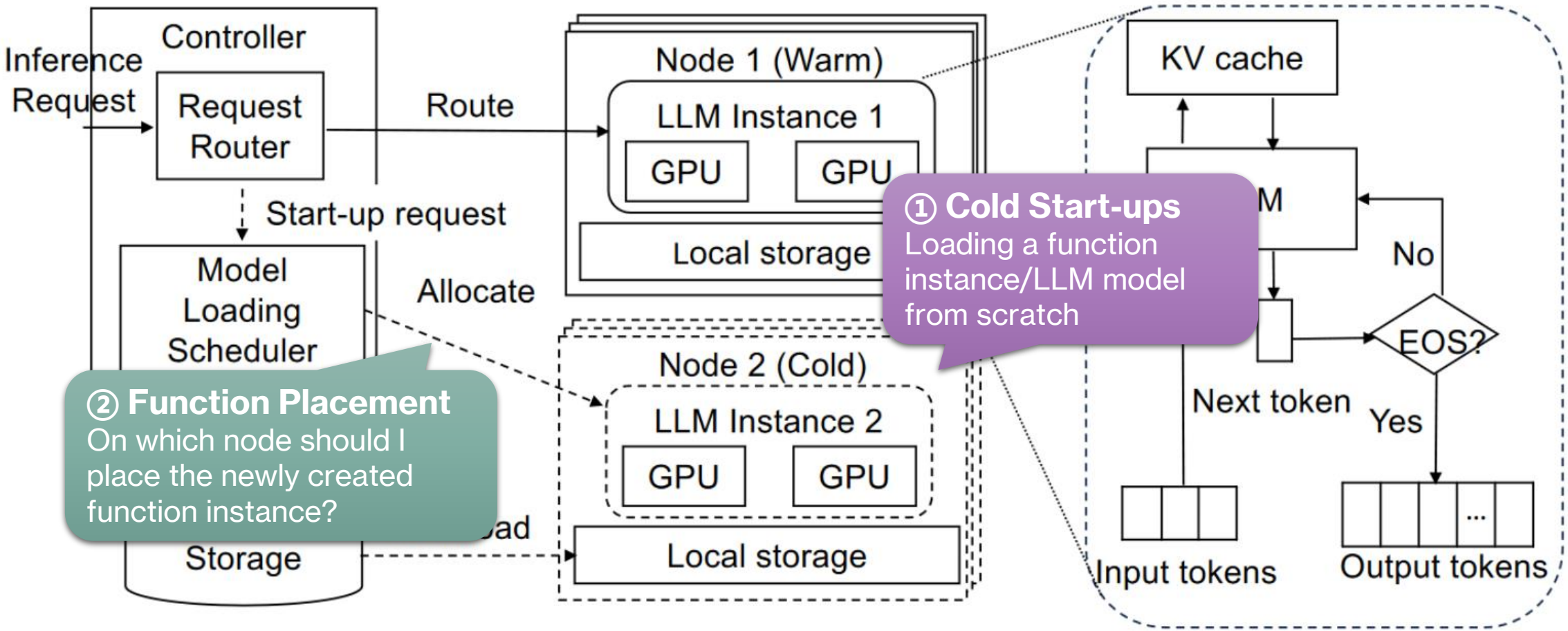
LLM Inference Characteristics

Large memory footprint
Facing highly dynamic, bursty traffic



Serverless Advantages

Pay-as-to-go pricing policy
Scalable nature



Overview of LLM serverless inference systems

Problems with LLM serverless inference

1. Costly **checkpoint download** from model repositories
2. Costly **checkpoint loading** from storage devices

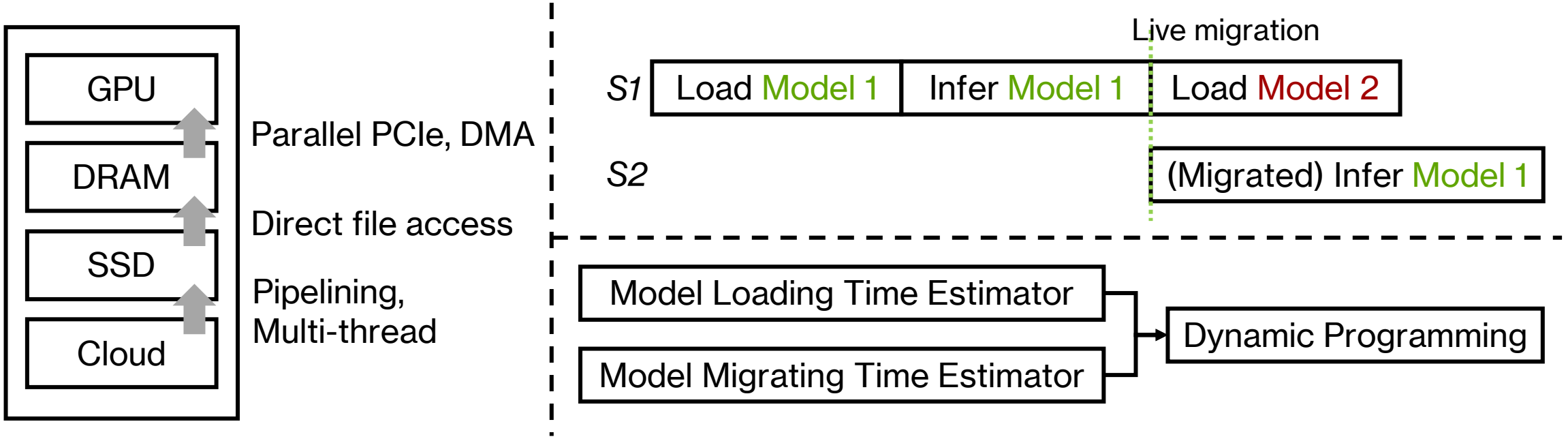
Current solutions:

- **Over-subscribing model instances** – LLM requires significantly higher demand on computing resources
- **Caching models in host memory** – LLM checkpoints are too large, so cache misses are common
- **Deploying additional storage servers** – Still unbearable downloading overhead

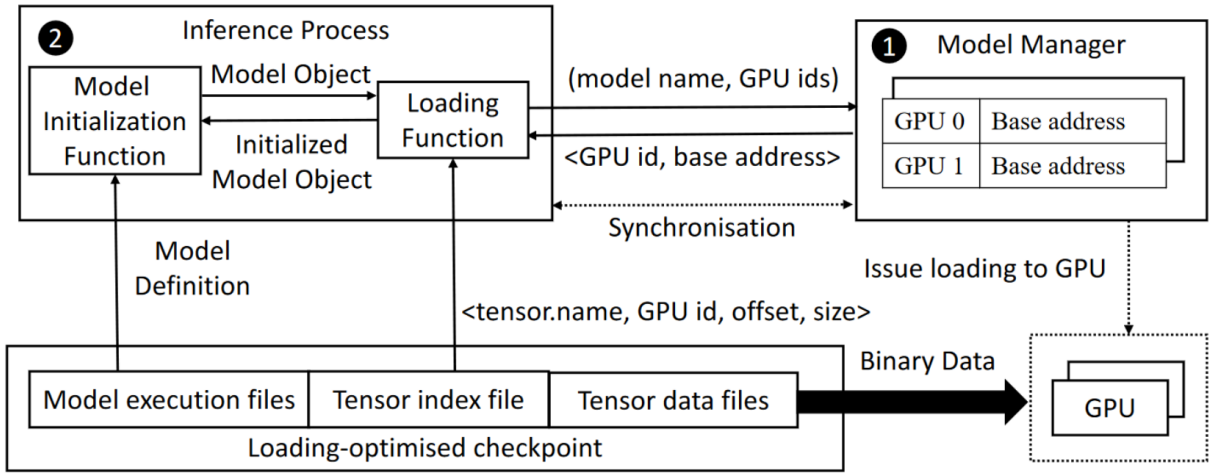
- **Locality**

Main Contributions

1. **Fast LLM checkpoint loading** → Tackle cold start-ups
2. **Live migration of LLM inference** → Further enhance locality
3. **Locality-aware server allocation** → Function placement



1. Fast LLM checkpoint loading



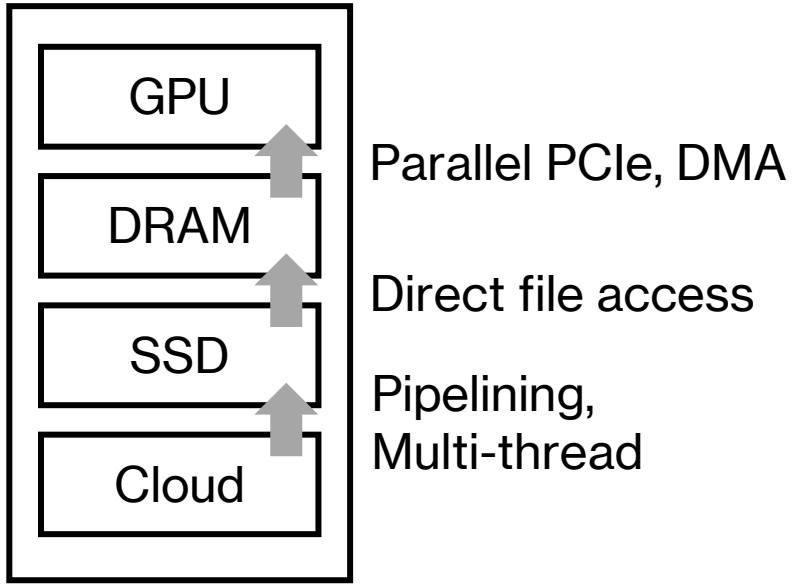
PyTorch/TF checkpoints:
- Debug/Training friendly
- Not loading friendly

- **Loading-Optimized Checkpoints**
  - Support sequential **chunk-based reading**
  - Support efficient **tensor addressing**
- *Components:*
  - **Model execution files**: defines model architecture, model parallelism plan
  - ***Tensor index file**: efficient tensor addressing
  - **Tensor data files**: model parameters

1. Fast LLM checkpoint loading

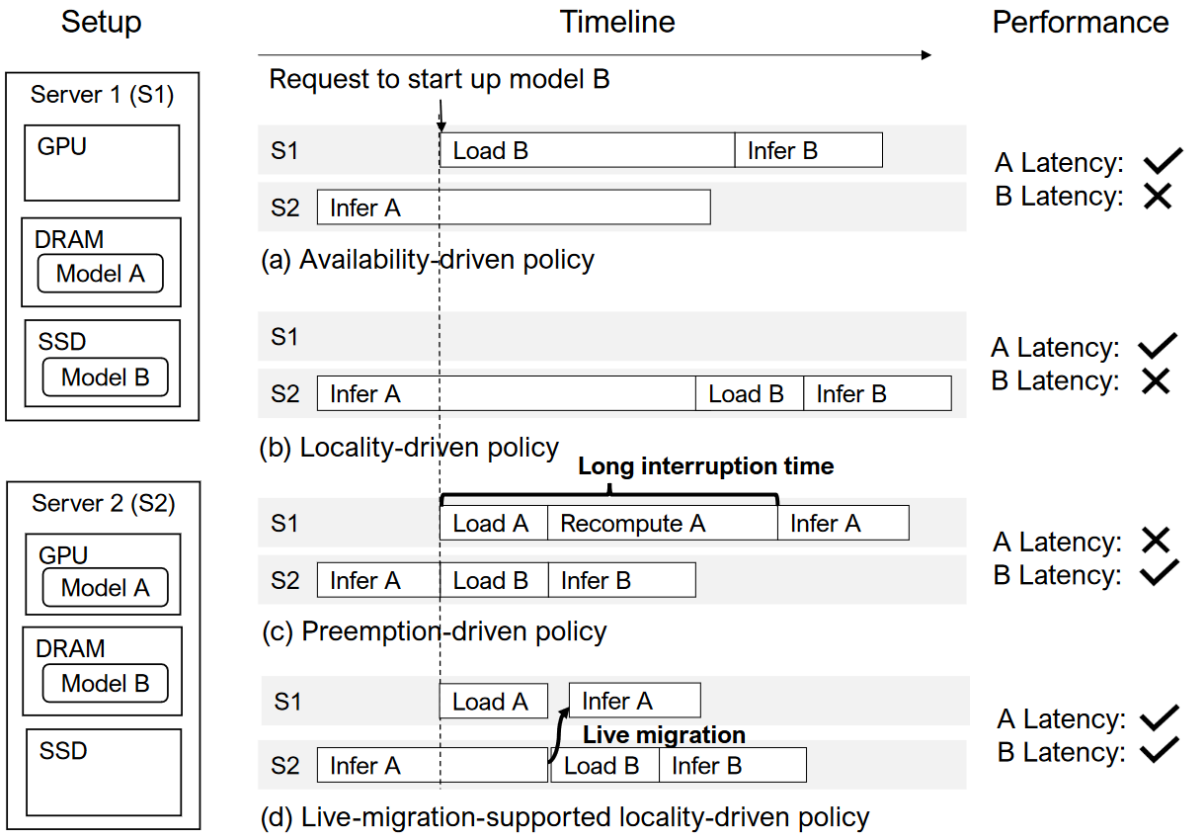
- **Efficient Multi-Tier Checkpoint Loading Subsystem**

1. In-memory data chunk pool
   - Utilizing parallel PCIe links → parallel DRAM-to-GPU PCIe
   - Supporting application-specific controls → API for allocation and deallocation of memory
   - Mitigating memory fragmentation → fixed-size memory chunks
2. Efficient data path
   - Exploiting direct file access → O_DIRECT in Linux
   - Exploiting pinned memory → DMA from mem to GPU
3. Multi-stage data loading pipeline
   - Optimization for intra-stage throughput → multiple threads
   - Efficient inter-stage coordination → task-queue based pipeline



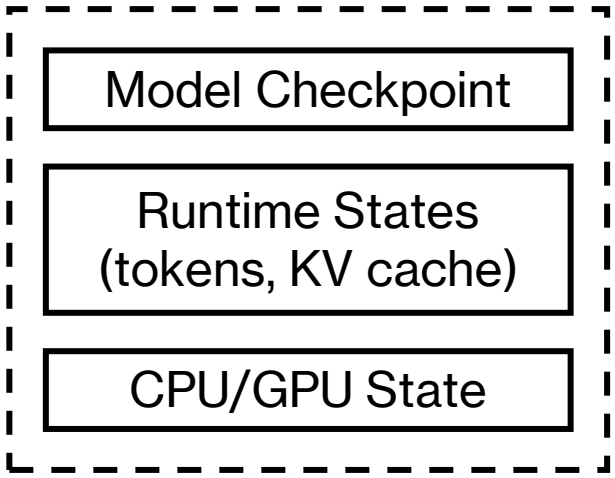
2. Live migration of LLM inference

- Case study: Need for Live Migration

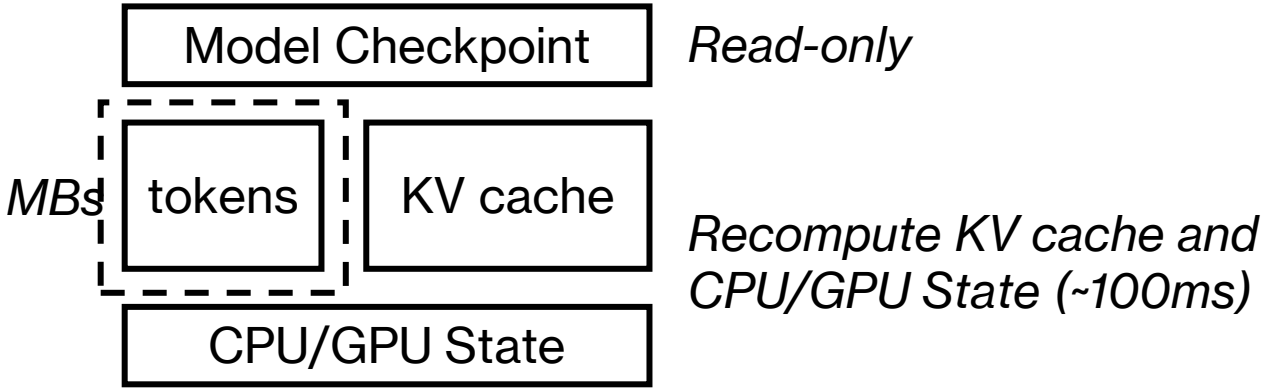


2. Live migration of LLM inference

- Live Migration Process



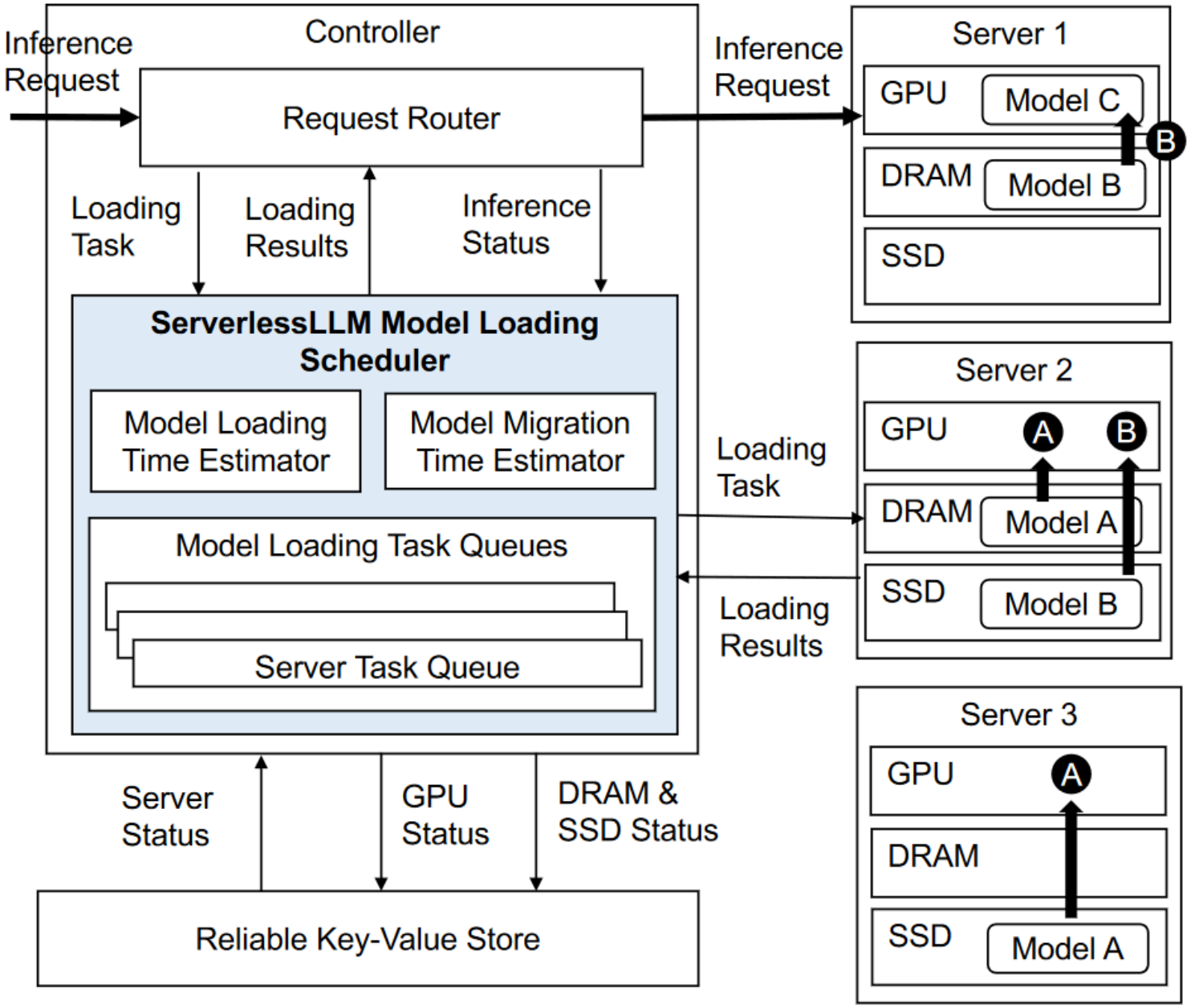
Snapshot creation and transfer overhead



Token-based migration

3. Locality-aware server allocation

Model loading scheduler design



3. Locality-aware server allocation

- **Estimating Model Loading Time**

$$q + n/b$$

$$(queuing\ time) + (model\ size)/(bandwidth)$$

  - Sequential model loading per server
  - Use the slowest bandwidth for estimation (pipeline)
  - Continuously monitoring the latency

3. Locality-aware server allocation

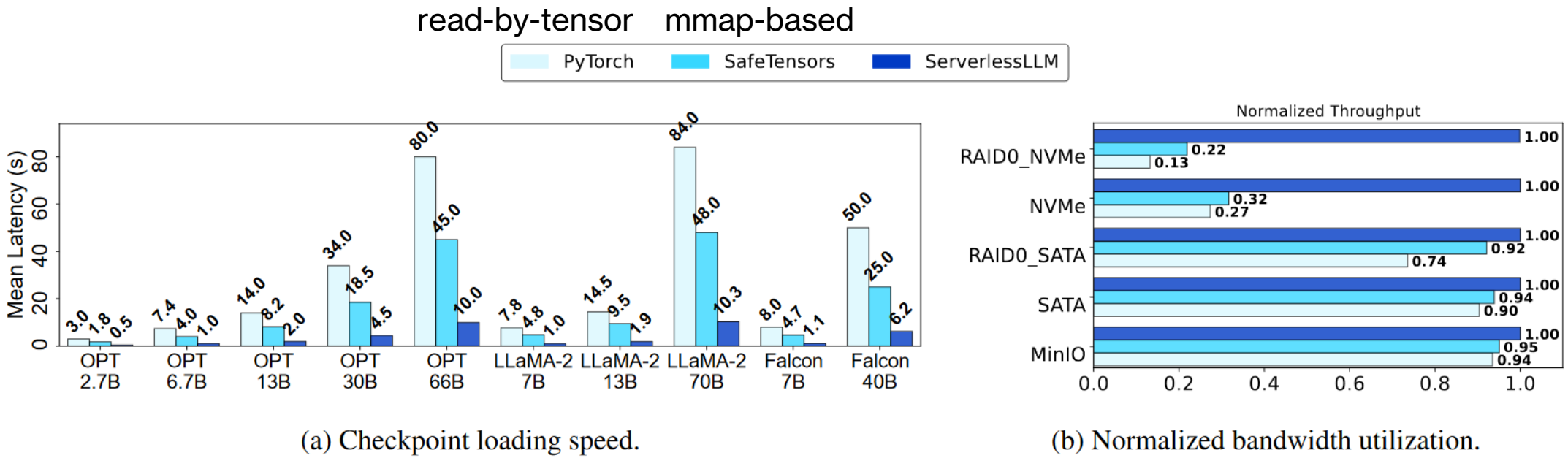
- **Estimating Model Migration Time**

$$a \times (t_{in} + t_{out}) + b$$

$$a \times (input\ tokens + output\ tokens) + b$$

  - Model-specific parameters (a and b) vary with each LLM's batch sizes and other factors
  - Obtaining real-time t_{out} can lead to bottlenecks, so use the inference duration d and average time to produce a token t to approximate $t_{out} = d/t$
  - For selecting the optimal server, ServerlessLLM employs a *dynamic programming* approach

Evaluation – Checkpoint Loading



(a) Checkpoint loading speed.

(b) Normalized bandwidth utilization.

Figure 6: Checkpoint loading performance.

Harness full bandwidth of storage devices

Evaluation – Checkpoint Loading

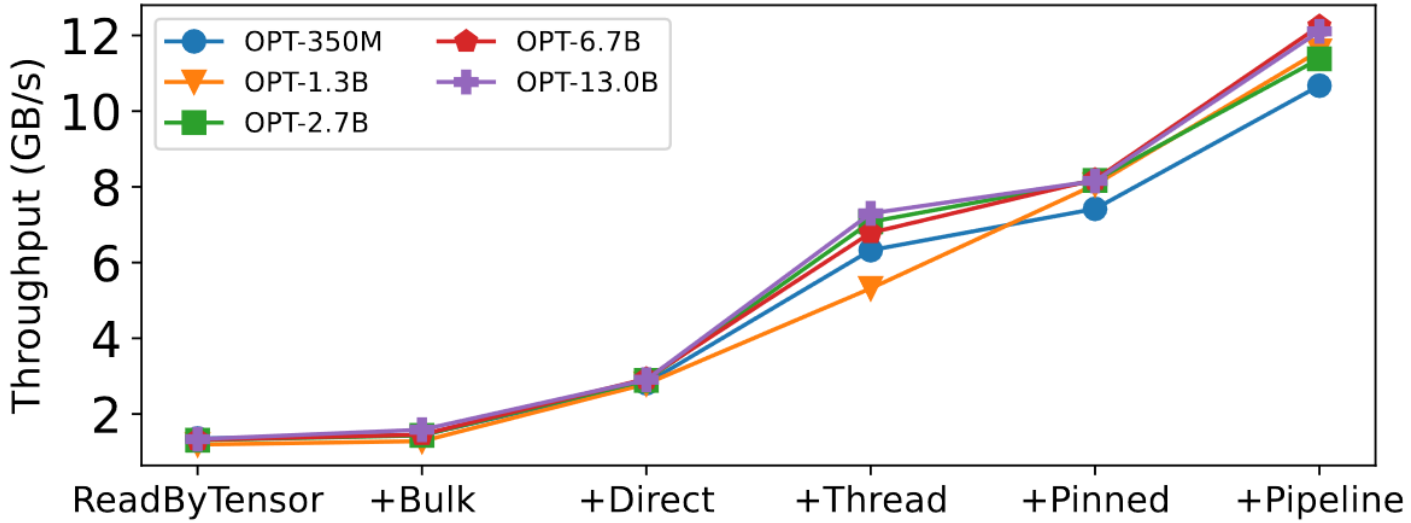


Figure 7: Performance breakdown of checkpoint loaders.

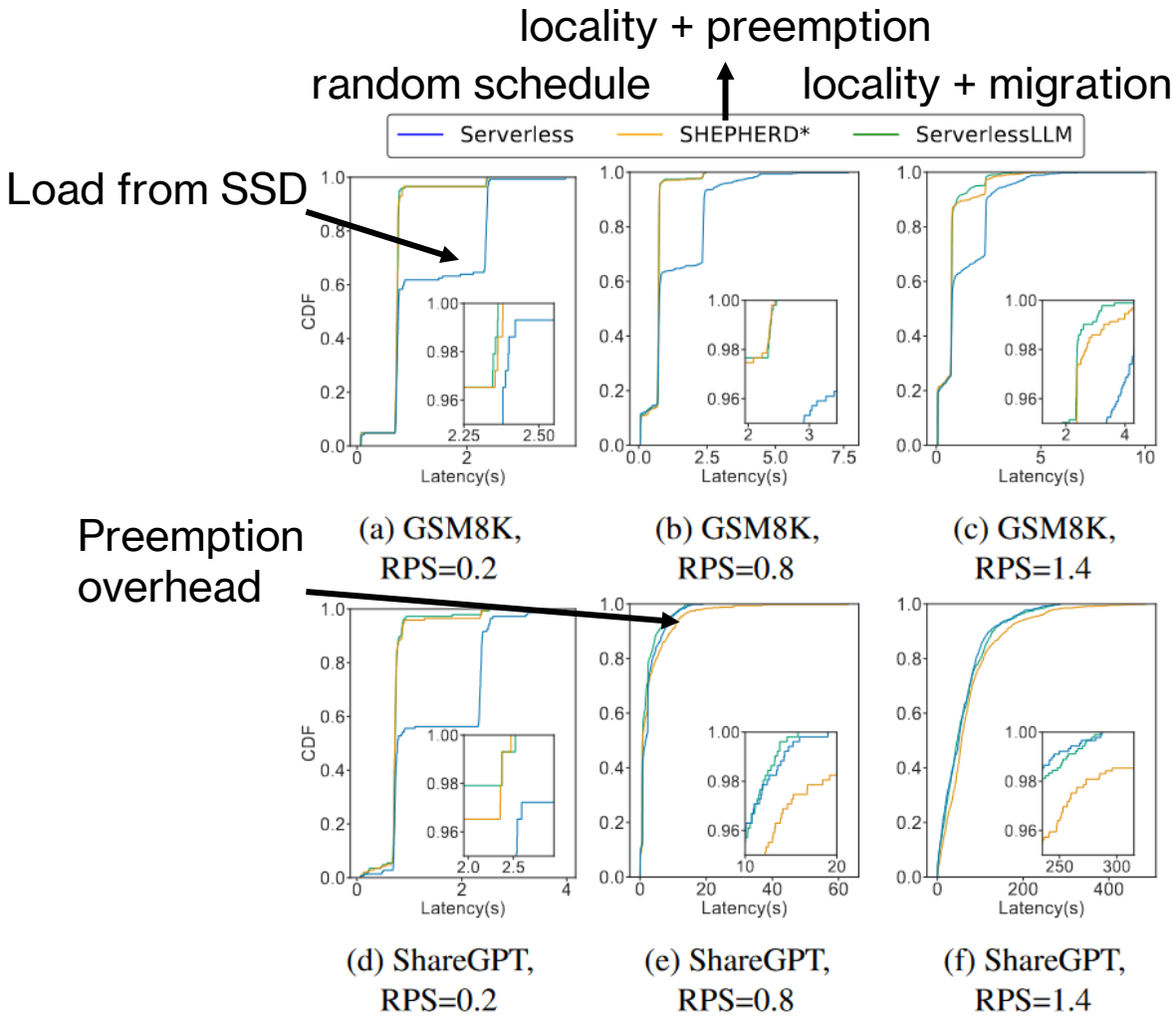


Figure 8: Impacts of RPS on model loading schedulers.

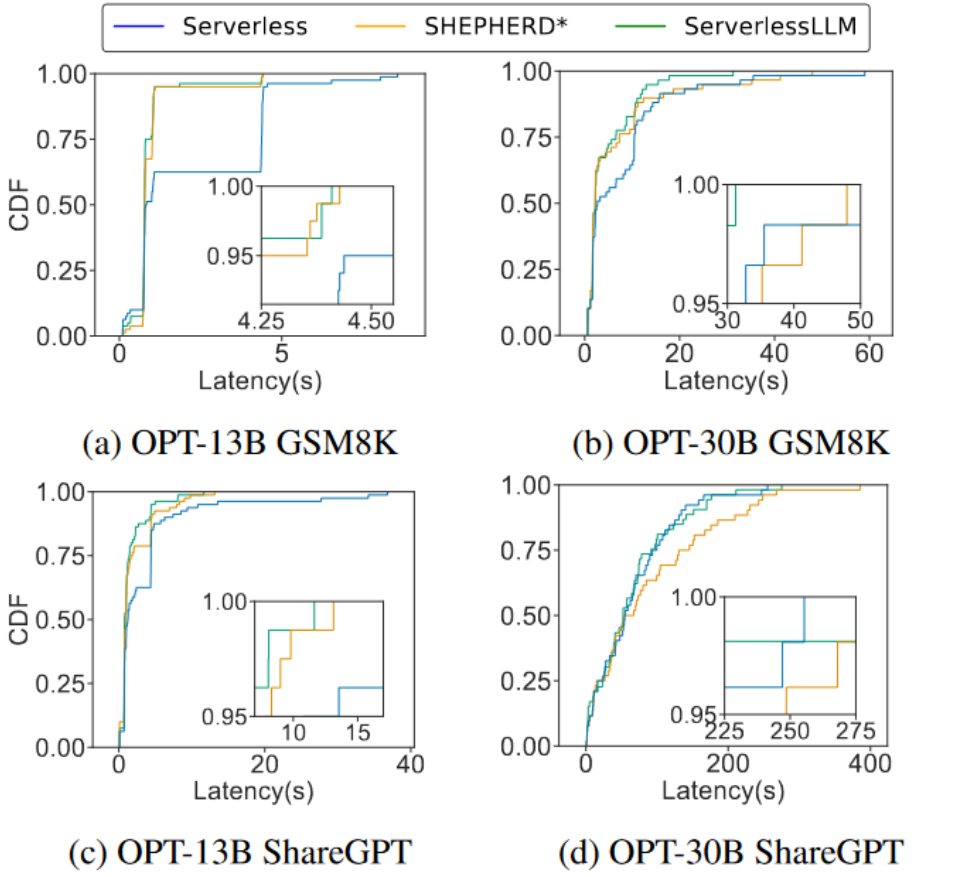


Figure 9: Impacts of datasets and models on model loading schedulers.

Evaluation – Model Loading Scheduler

Evaluation – End-to-end performance

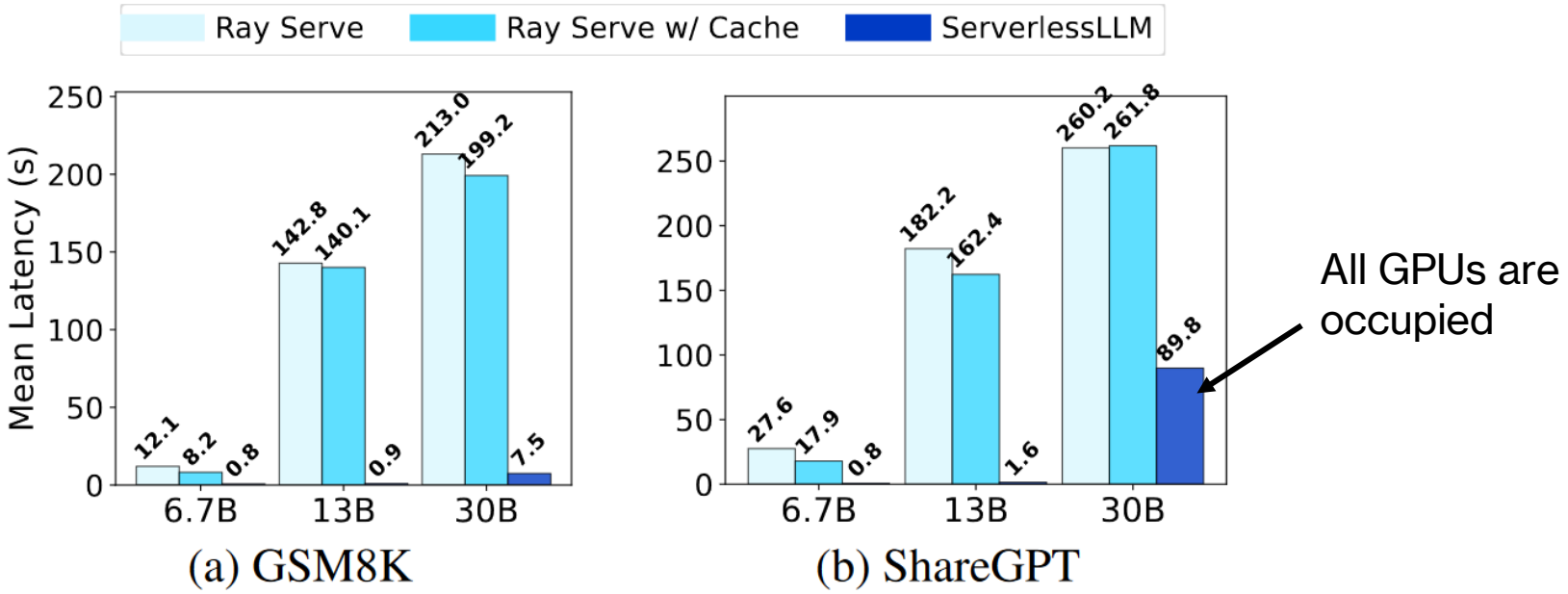
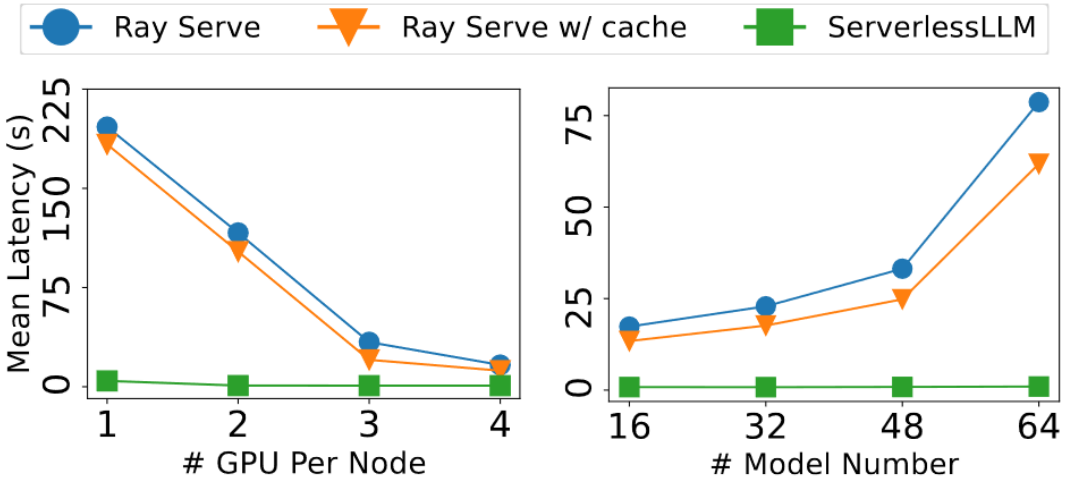


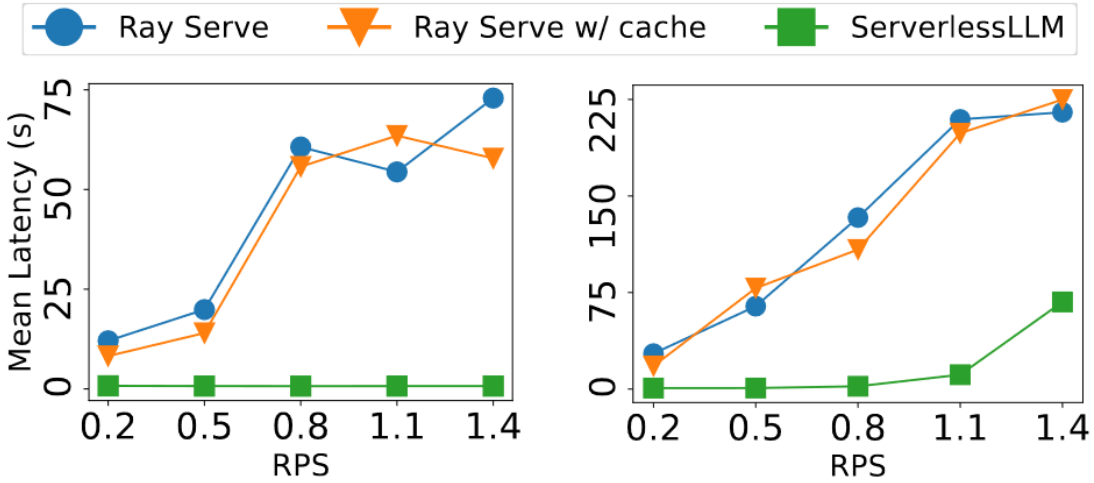
Figure 10: Impacts of datasets and models on overall serving systems.

Evaluation – End-to-end performance



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

Figure 12: System scalability and resource efficiency.



(a) GSM8k (b) ShareGPT

Figure 11: Impacts of RPS on overall serving systems.

Discussion

- 传统 Serverless 尺度 << LLM 尺度
  - 冷启动问题 << LLM 模型加载问题
  - Serverless 调度问题 << LLM 模型局部性问题
  - Live Migration 问题
- 只考虑了 Server 级别的资源分配，没有考虑容器级别的资源分配
  - 有些违背 Serverless 的初衷，是否能做到更细粒度的调度?