Machine Learning Model Management and Inference (MLMMI)
07 LLM Serving

Dr. Arnab Phani
BIFOLD, TU Berlin
DEEM Lab

17.06.2026

Announcements

- **SIGMOD 2026 Updates**
- **Matthias Seeger's Guest Lecture**
  - Thanks for attending
  - How did you like it?
- **Project Updates**
  - Freedom of problem/dataset selection for projects 1, 2, 5
  - Include diverse problems: dataset sizes, domains, solution complexity, ML tasks, large search space
  - Robustness will be considered for grading
  - For project 4, implement various optimization opportunities: cache efficiency, memory layout, loop fusion, …
  - For project 3, combine easy and complex rewrites
  - Prioritize quality over quantity

BIFOLD in SIGMOD'26

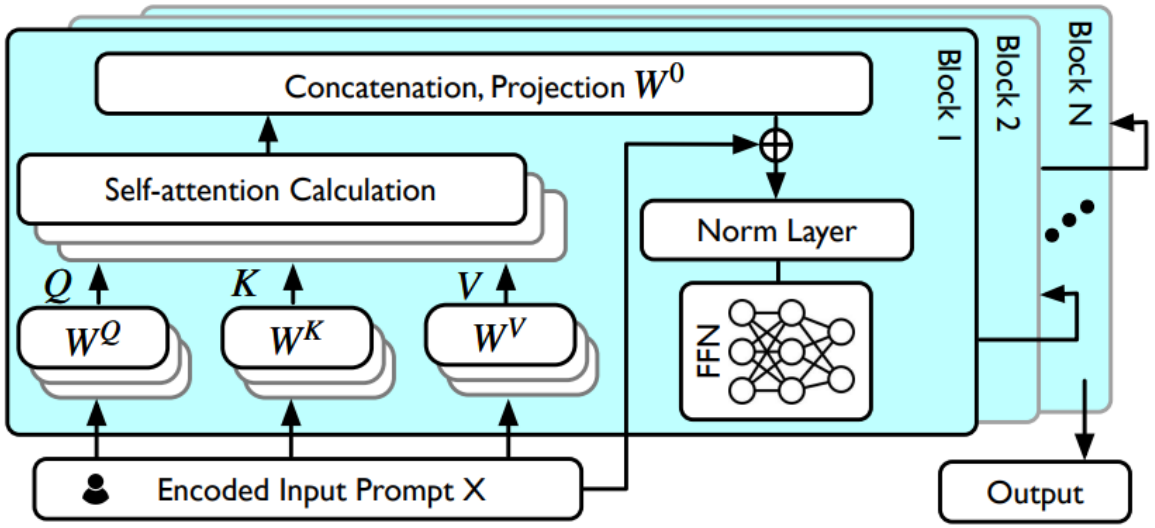
Project Presentations : July 15 (2-3 hours)
Project Report and Code: July 29

Agenda

- **LLM Inference Overview**
- **Traditional Model Serving Recap**
- **Iteration-Level Scheduling**
- **Prefill-Decode Disaggregation**
- **Local LLM Serving with Consumer-Grade GPUs**
- **LoRA LLM Serving**

LLM Inference Overview and Performance Analysis

Transformer-based LLM Architecture



Self-attention:

$$Q = XW^Q; K = XW^K; V = XW^V$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Attention head:

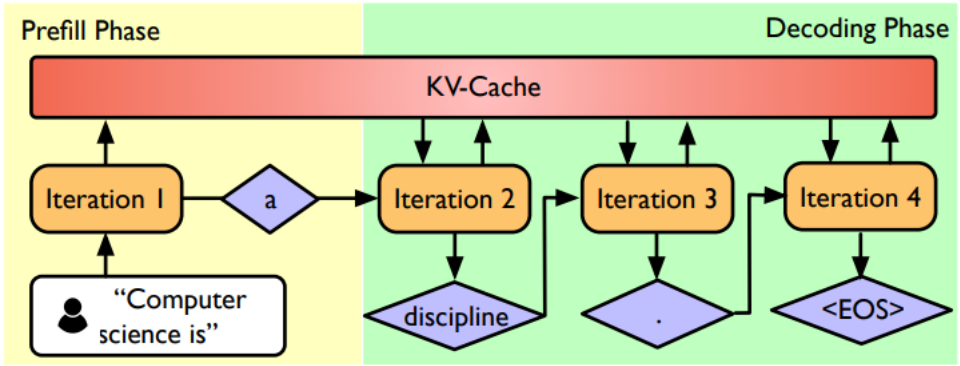
$$H_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^Q)$$

$$\text{Multi-Head Attention} = \text{Concat}(H_1, H_2, ..., H_h)W^O$$

- **Overview**
  - LLMs are built on multiple transformer blocks; each with multi-head attentions and FFNs
  - Attention computes a weighted average of tokens of interest
  - Initially, the transformer applies three weight matrices (W^Q, W^K, W^V) to the input X (encoded representation of input tokens) to compute queries Q, keys K, and values V

* LLM Inference Serving: Survey of Recent Advances and Opportunities. arXiv. 2024.

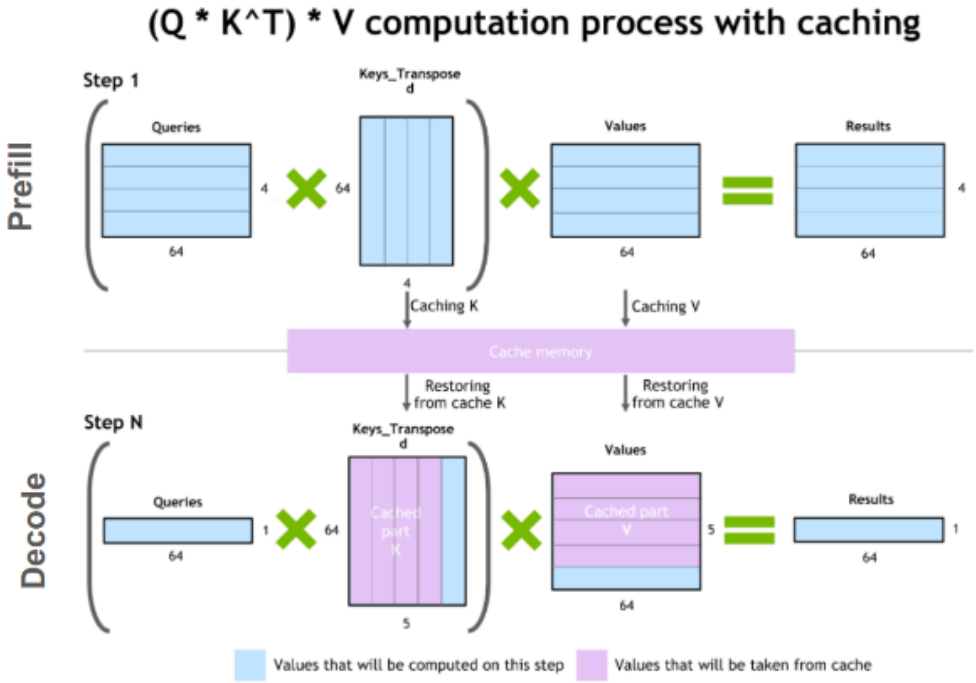
LLM Inference

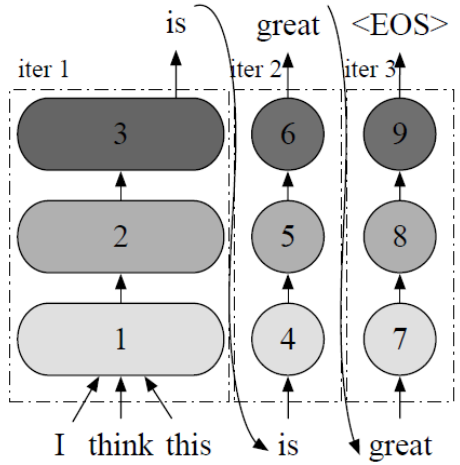


- **Inference Process of LLMs**
  - Prefill Stage: processing model input (all in one forward pass); compute-bound
  - Decode Stage: generating output token, one at a time; memory-bound
  - Iterative: each forward pass generates a single token
  - Autoregressive: generation consumes prompt tokens + previously generated tokens; end with a EoS token

* LLM Inference Serving: Survey of Recent Advances and Opportunities. ArXiv. 2024.

KV Cache

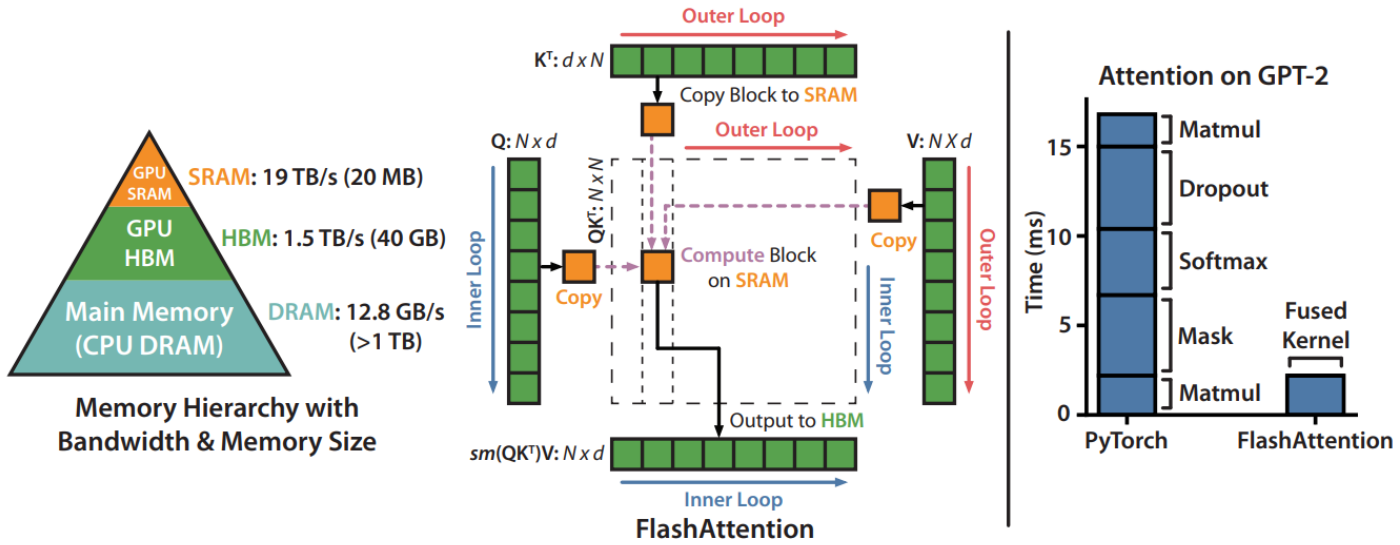




- **Overview**
  - KV cache stores Key (K) and Value (V) tensors computed at each layer to avoid recomputation
  - Prefill stores computed KVs of input sequence in KV cache
  - In each iteration in decode phase each new token only needs to attend to cached KV states + the latest token

* https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization

Attention is Memory-Bound



- **Size of LLM Parameters**
  - Llama2 70B ~ 130GB to store float16 parameters
- **Memory IO vs Throughput**
  - For a single token, have to read 130 GB to compute cores
  - CPU memory b/w ~= 10-50 GB/s; GPU memory b/w ~= 2000 GB/s (A100 80GB)
  - Hight throughput requires many FLOPS

L1 Latency: 1 ns
L2 Latency: 2.5 ns
L3 Latency: 10 ns
RAM Latency: 50 ns

(per core)
L1 Bandwidth: 210 GB/s
L2 Bandwidth: 80 GB/s
L3 Bandwidth: 60 GB/s

(whole system)
RAM Bandwidth: 45 GB/s

Numbers Every Programmer Should Know

* FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness. NeurIPS 2022.

Compute-Bound vs Memory-Bound Operations Recap

- **Back-of-the-Envelope Calculation**
  - Count FLOPs, memory traffic, and compute; then compare with machine balance

$$Arithmetic\ Intensity = \frac{Operations}{Bytes\ moved}$$

Example GPU:
Peak compute = 80 TFLOP/s, Peak memory = 2 TB/s
Machine balance: 80 / 2 = 40
AI < 40 => memory-bound
AI > 40 => compute-bound

  - Example 1: Matrix addition (FP 32) => $\mathrm{AI} = 1 / 12 = 0.083$ => memory-bound
  - Example 2: Matrix multiplication => $\mathrm{AI} = 2\mathrm{N}^3 / 12\mathrm{N}^2 = 170 \text{ for } \mathrm{N} = 1024$ => compute-bound

- **How to Increase Arithmetic Intensity?**
  - Increase AI by increasing compute per byte: cache locality, tiling, vectorization, operator fusion, pipelining
  - Reduce memory traffic: compression, quantization, sparsity exploitation
  - Trade memory for compute: recomputation (GPU + DRAM, CPU + disk), activation checkpointing

Performance Metrics for LLM Serving

- **Time To First Token (TTFT):** Queueing time + prefill time: How quickly users start seeing a response, i.e., response time
- **Time Per Output Token (TPOT):** Time to generate each additional output token. Average TPOT x output token length is the time users see all the response after seeing the first token
- **Latency:** The overall request time it takes for the model to generate the full response for a user. latency = (TTFT) + (TPOT) * (the number of tokens to be generated)
- **Throughput:** The number of output tokens per second an inference server can generate across all users and requests

LLM Serving Systems and Optimizations

Traditional Model Serving Recap

- **Serving Constraints**
  - Low latency; user-facing applications
  - High-throughput
  - Scalability during high loads

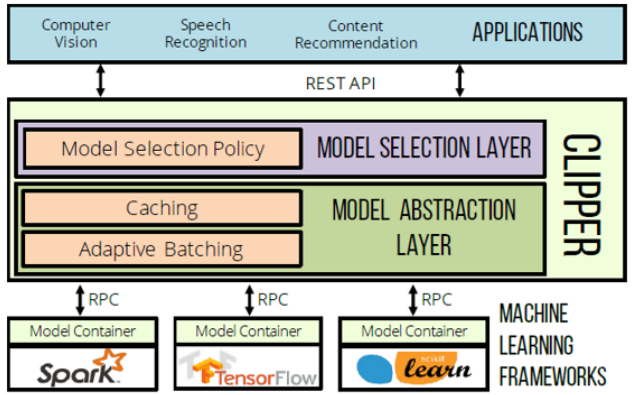
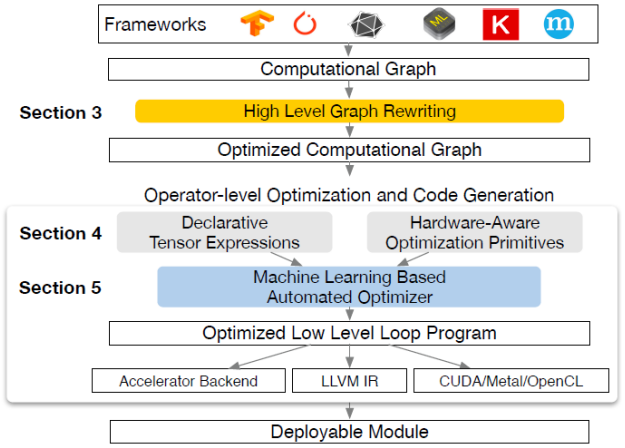
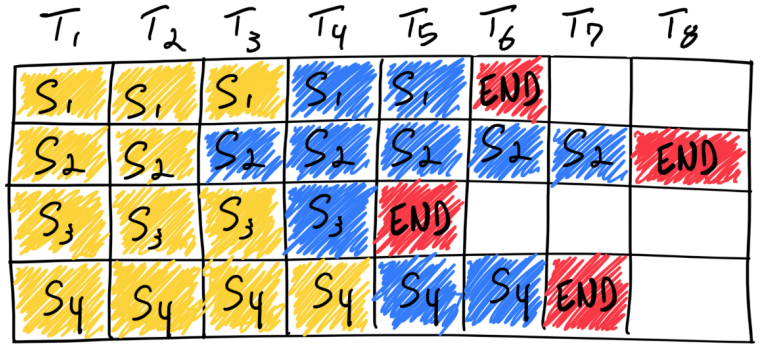


Figure 1: The Clipper Architecture.

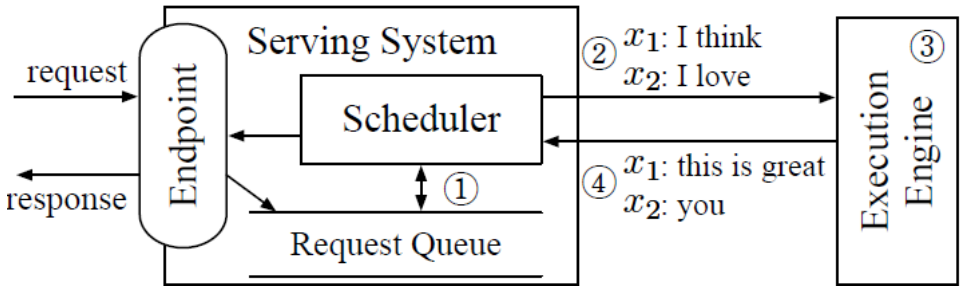
- **Serving Optimizations**
  - Batching improves throughput and utilize HW efficiently
  - Caching: popular items are requested frequently; reuse inference
  - Compilation of entire graph; operator fusion
  - Quantization of model weights
  - Serverless inference



Static Batching

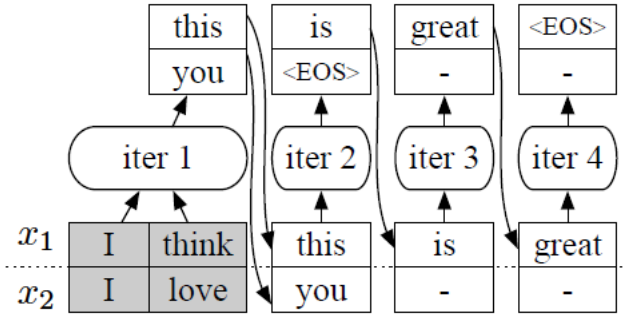


Static Caching



Overall Workflow

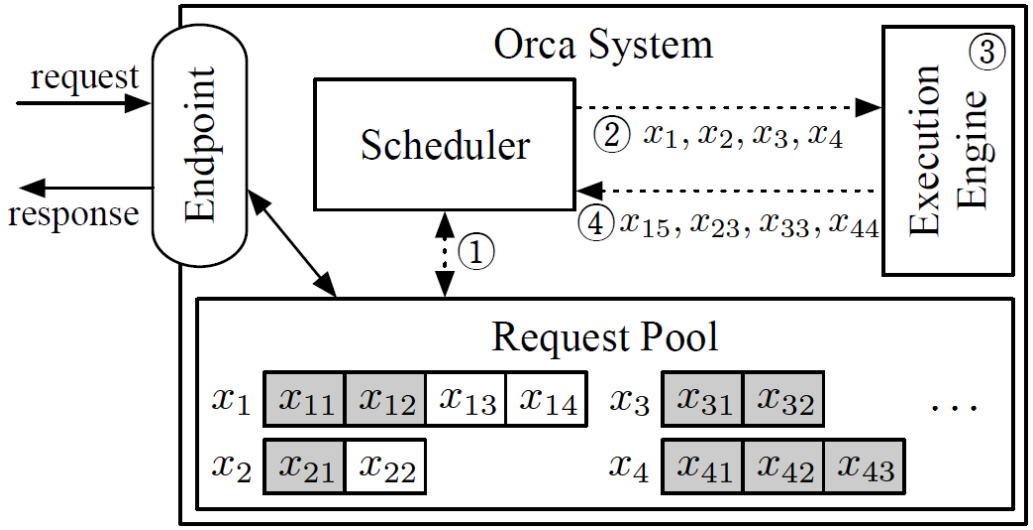
- **Overview**
  - Batch multiple user requests, execute, and return generated texts back
- **Problems**
  - Each request may require different number of iterations
  - Prevent early return of the finished requests
  - Delay new request until current batch finished
  - Extra computation for inactive requests



Extra Computation

* Orca: A Distributed Serving System for Transformer-Based Generative Models. OSDI 2022.

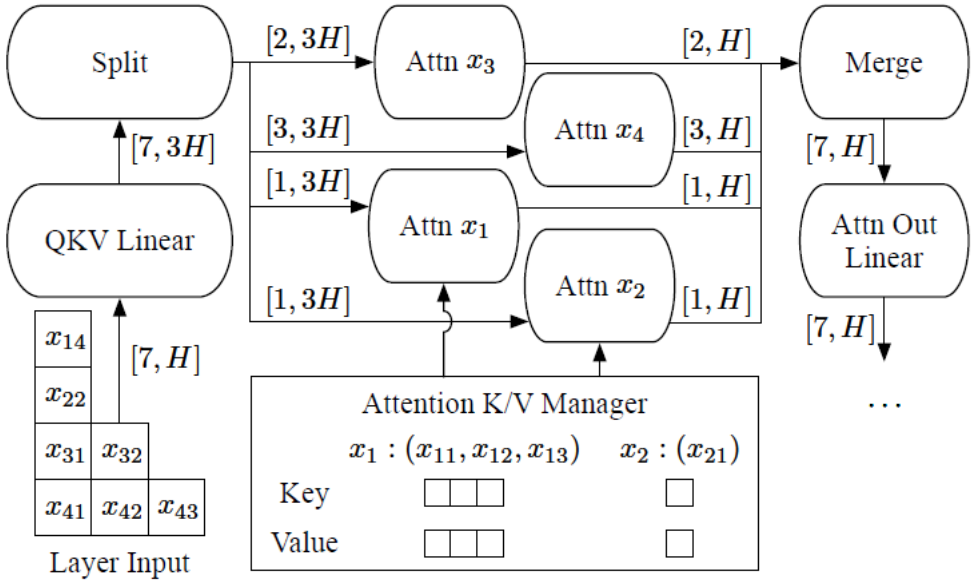
Iteration-level Scheduling



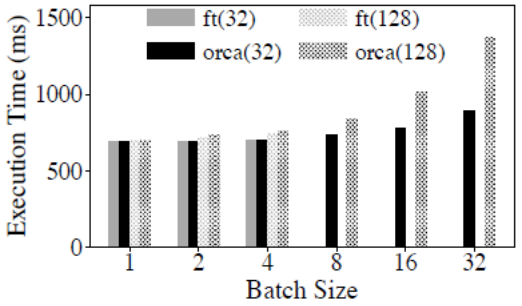
- **Overview**
  - Schedule execution at the granularity of iteration; immediate return completed requests
- **Problem: Batching Arbitrary Set of Iterations**
  - Cannot batch two requests if (1) both are in prefill and have different # tokens,
  - (2) both in decode but processing a token at different index,
  - (3) they are in different phases (prefill / decode)
  - Irregularly shaped input tensors cannot be batched

* Orca: A Distributed Serving System for Transformer-Based Generative Models. OSDI 2022.

Selective Batching



ORCA Execution Engine



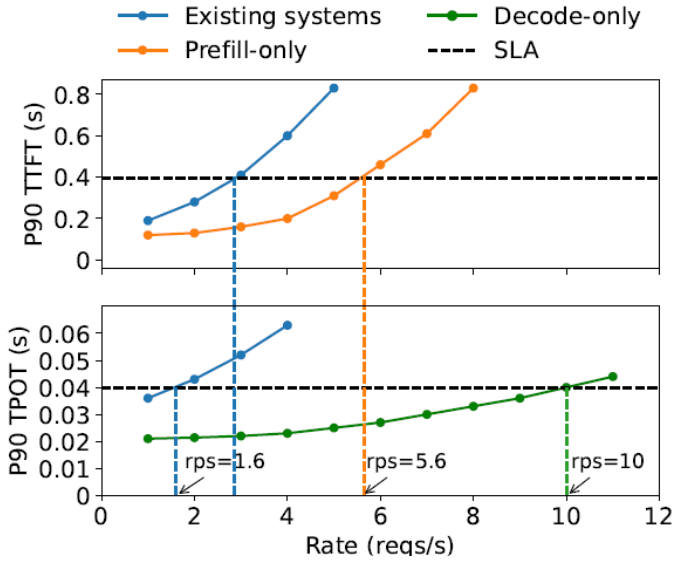
(a) 13B model, 1 GPU.

Small overhead of not batching attentions

- **Overview**
  - Selectively apply batching only to certain operations
  - Non-attention matmul and layer normalization can be made to work on irregular tensors by flattening
  - Process attention operation token-wise, batch other operations
  - Maintain KV cache for each request separately
  - ORCA proposes inter-, intra-operator parallelism, pipeline parallelism and specialized distributed scheduler

* Orca: A Distributed Serving System for Transformer-Based Generative Models. OSDI 2022.

Prefill-Decode Disaggregation



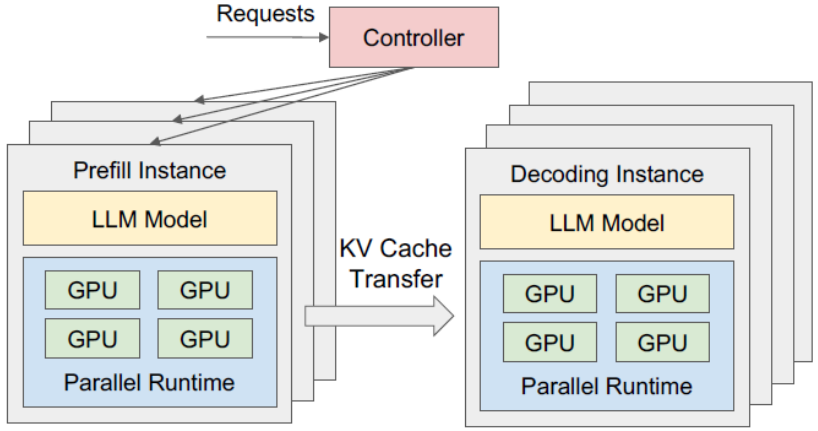
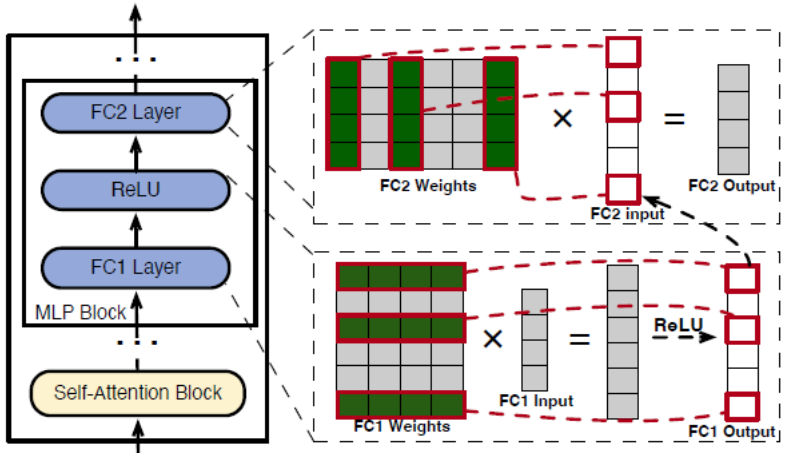


Figure 6: DistServe Runtime System Architecture

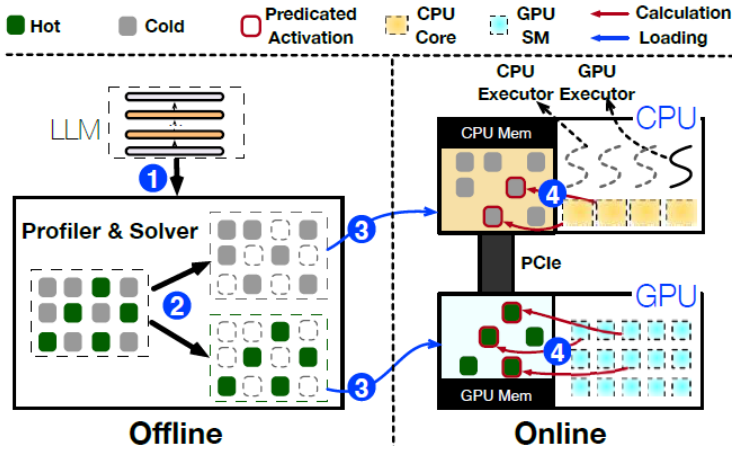
- **Problems**
  - Prefill and decode have distinct characteristics (compute, memory, parallelism potentials, metrics, SLOs)
  - Batching prefill and decode together cause interference; prefill can slow down decode;
  - Colocation prevents independent parallelism tuning; hard to achieve optimal performance for both
- **DistServe**
  - Disaggregate prefill and decode; assign to separate GPUs; additional (small) communication overhead
  - Eliminate prefill-decode interference; enable independent tuning and scaling
  - Tailored parallelism strategies, pipelining, efficient KV cache transfer

* DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. OSDI 2024.

Local LLM Serving with Consumer-Grade GPUs

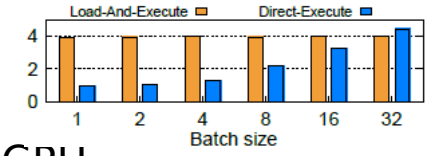


Sparsely-activated Neurons



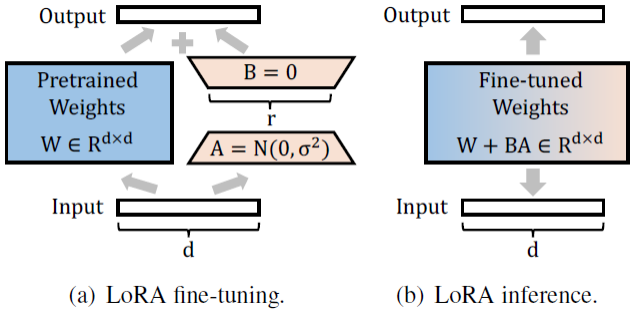
PowerInfer Overview

- **PowerInfer**
  - LLM inference exhibits high data locality; small subset of neurons contributes to the majority of activations
  - PowerInfer exploits this locality; assigns hot neurons to GPU and cold neurons to CPU
  - Offline phase: preselects and preloads hot-activated neurons onto the offline
  - Online phase: online predictors during runtime to identify activated neurons
  - For small batch sizes, computation in CPU is faster than transferring the weights to GPU

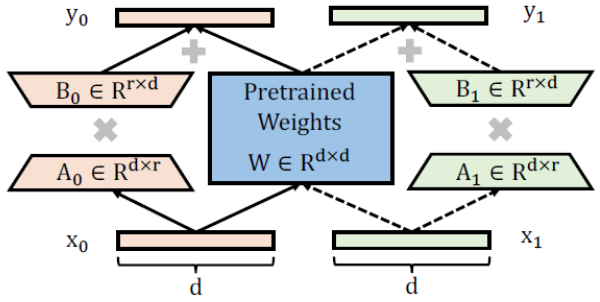


* PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. SOSP 2024.

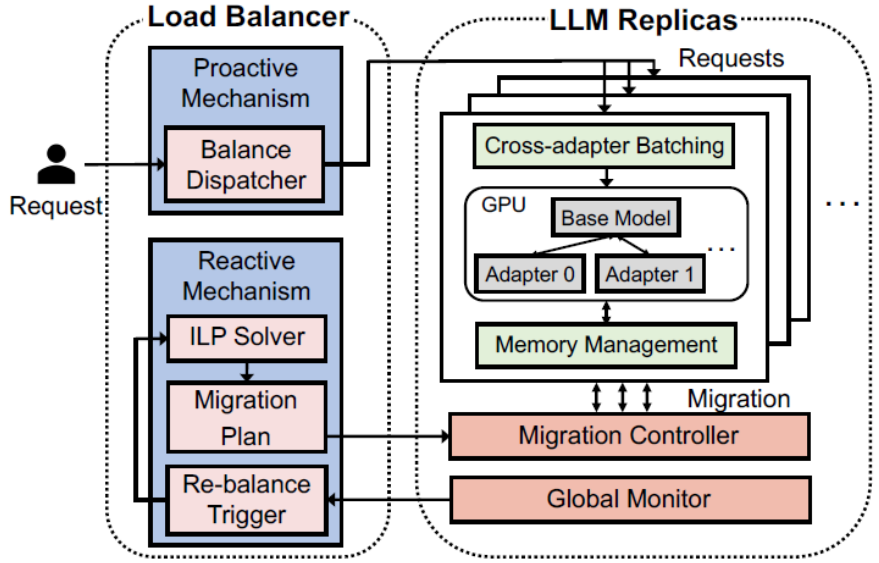
LoRA LLM Serving



LoRA Optimization



Unmerged Inference



dLoRA Architecture

- **dLoRA**
  - LoRA adaptation is parameter-efficient fine-tuning method; fine-tune a small # of parameters (adapter)
  - Serving challenge: existing systems are for single model serving; cannot share base models; many replicas
  - One request at-a-time for low-frequent types; many adapters share few GPUs lead to load imbalance
  - dLoRA deploys multiple replicas, each with a subset of adapters and one base model on several GPUs
  - Unmerged inference allows batching different types but incurs additional compute
  - Merged inference improves compute but adds queuing delays and low GPU efficiency
  - Dynamic merging/unmerging; dynamic migrate requests and adapters between replicas

* dLoRA: Dynamically Orchestrating Requests and Adapters for LoRA LLM Serving. OSDI 2024.

Summary

- LLMs serving requires tailored techniques
- Typical static batching is not sufficient; selective batching is necessary
- Prefill and decode have different characteristics and need dedicated optimizations
- An important new research direction is to support local LLM serving (quality vs. latency & cost)
- Serving multiple fine-tuned LLMs via LoRA requires dedicated scheduling techniques

Please read the recommended papers