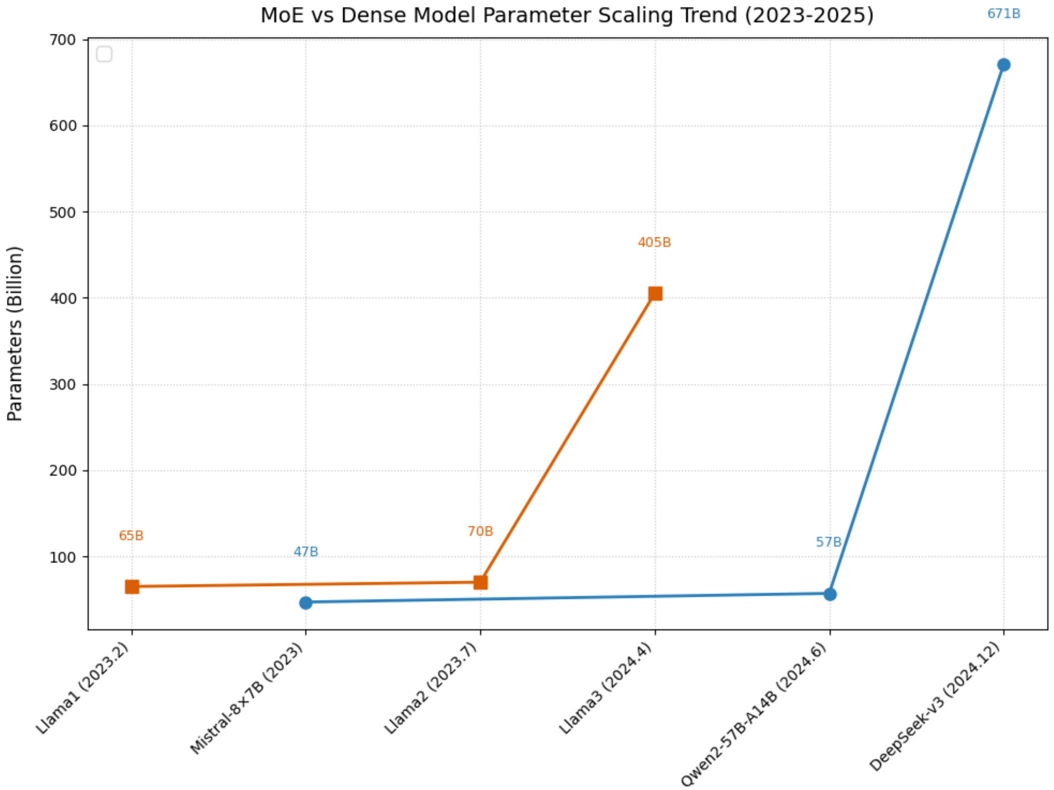
University of Science and Technology of China
Mohamed bin Zayed University of Artificial Intelligence

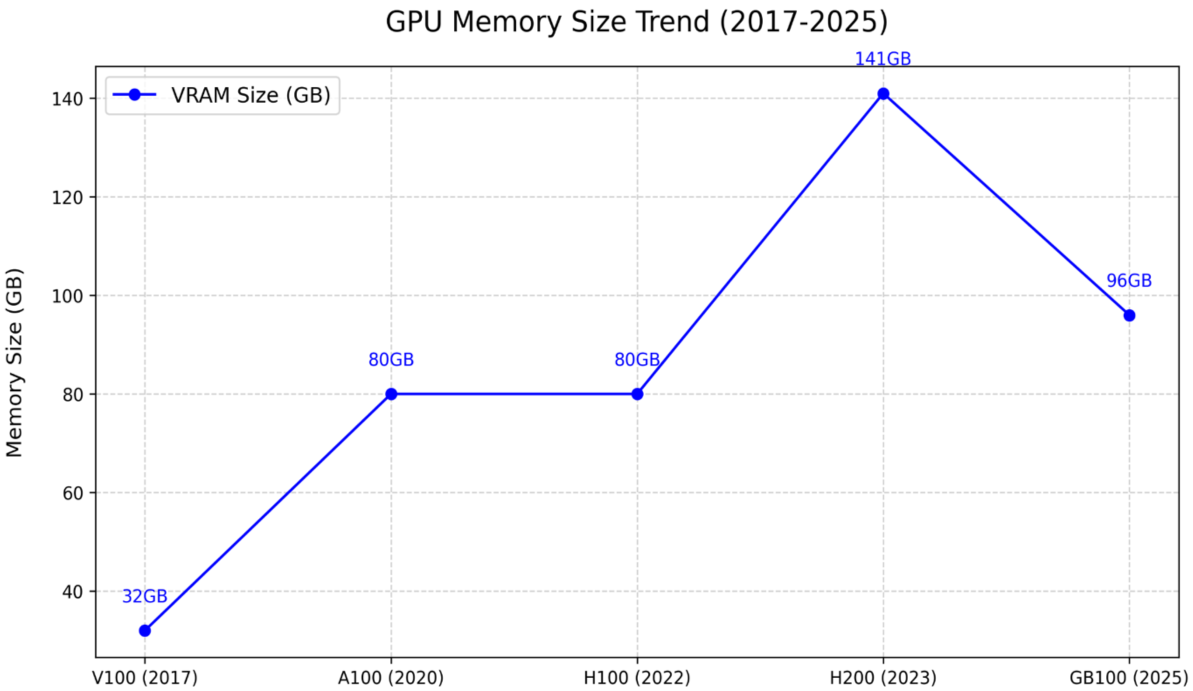
fMoE: Fine-Grained Expert Offloading for Large Mixture-of-Experts Serving

Presented by Jia He, Jiaqi Ruan

Contents

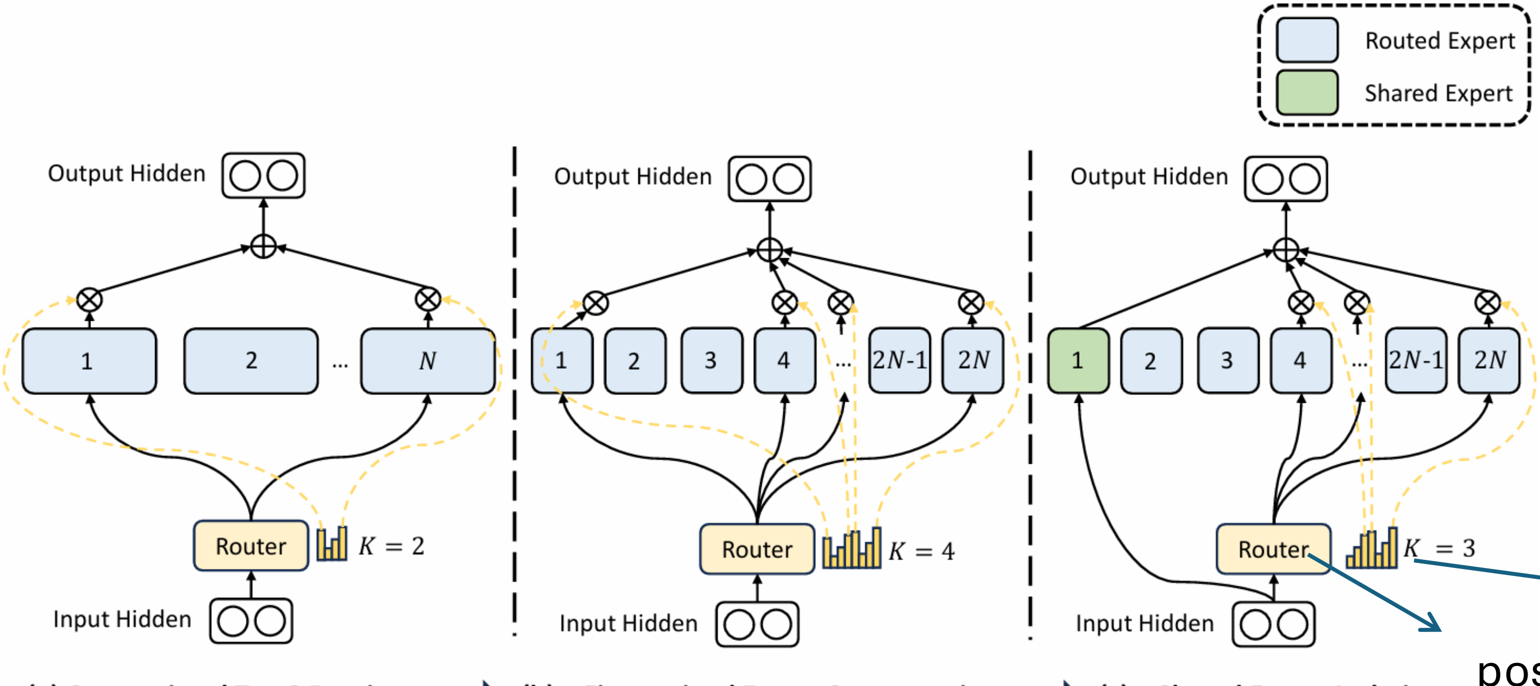
Background





The growth rate of large model parameters far exceeds the current growth rate of GPU memory.

Background



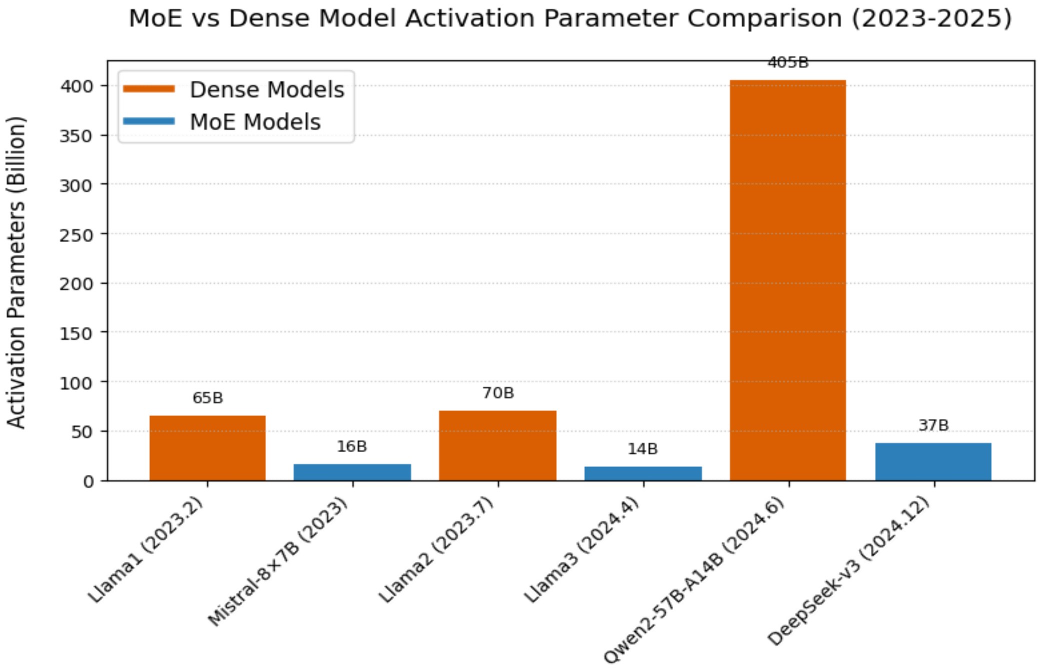
(a) Conventional Top-2 Routing → (b) + Fine-grained Expert Segmentation → (c) + Shared Expert Isolation (DeepSeekMoE)

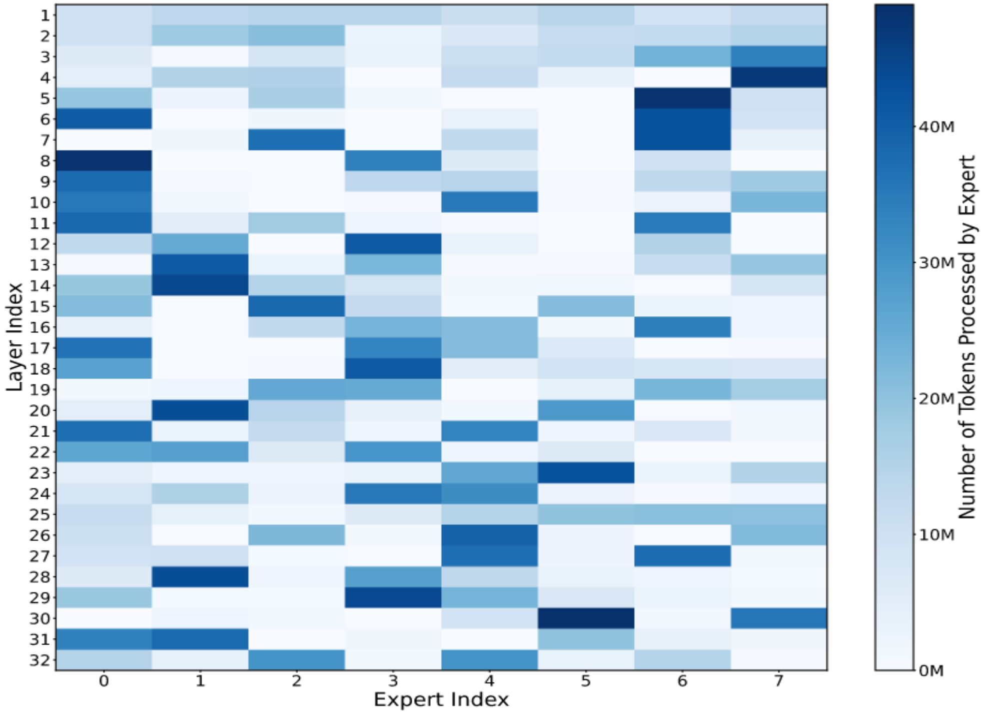
According to the routing
Activate partial parameters.

top-k expert based possibility

possibility

Background





For Token

Only a small fraction of parameters are required for inference on a single token.

For Sequence

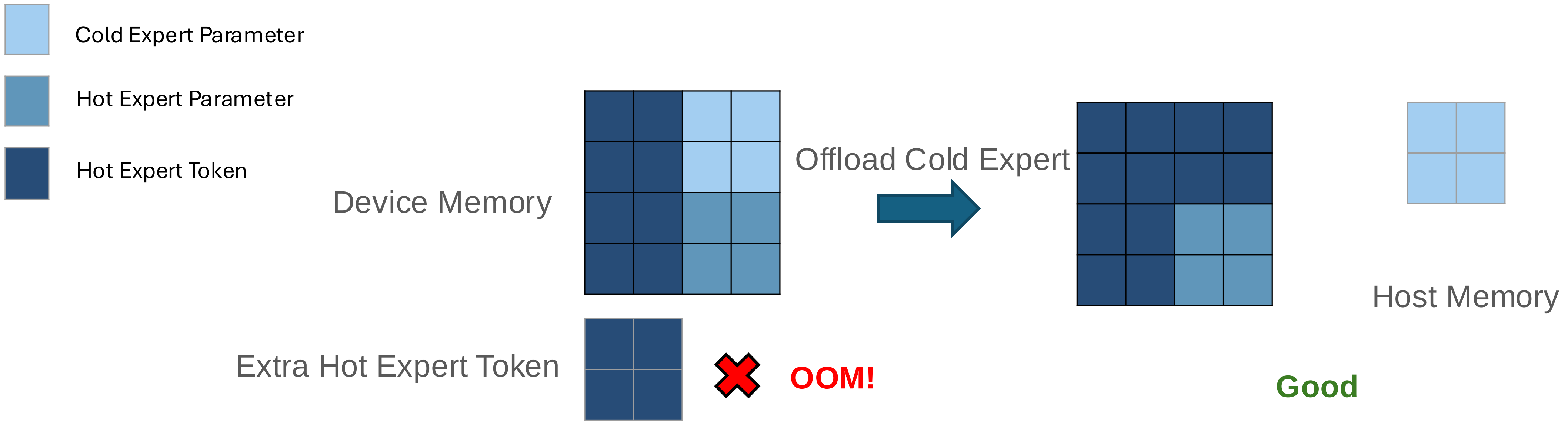
A large number of tokens are routed to specific devices, increasing their memory pressure.

Background

For DeepSeekV3

Token [1,7168]   Expert Parameter [2048,7168] & [4096,7168]

The memory usage of Expert Parameters is equivalent to 6k tokens.（Without considering the buffer）



Contents

Related Work

System	Type	Granularity	MoE Offload Strategy
FlexGen [1]	Dense	-	-
DeepSpeed Inference	MoE	Naive	LRU Cache + Dependency Prefetch.
Mixtral-Offloading	MoE	Activation	LRU Cache + Speculative Prediction.
BrainStorm [2]	MoE	Dataset	LRU Cache + Model Count Prediction.
ProMoE [3]	MoE	Activation	Training a predictor to prefetch experts based on predictions.
MoE-Infinity [4]	MoE	Iteration	Request-level for Cache Iteration-level for prefetch(LFU).

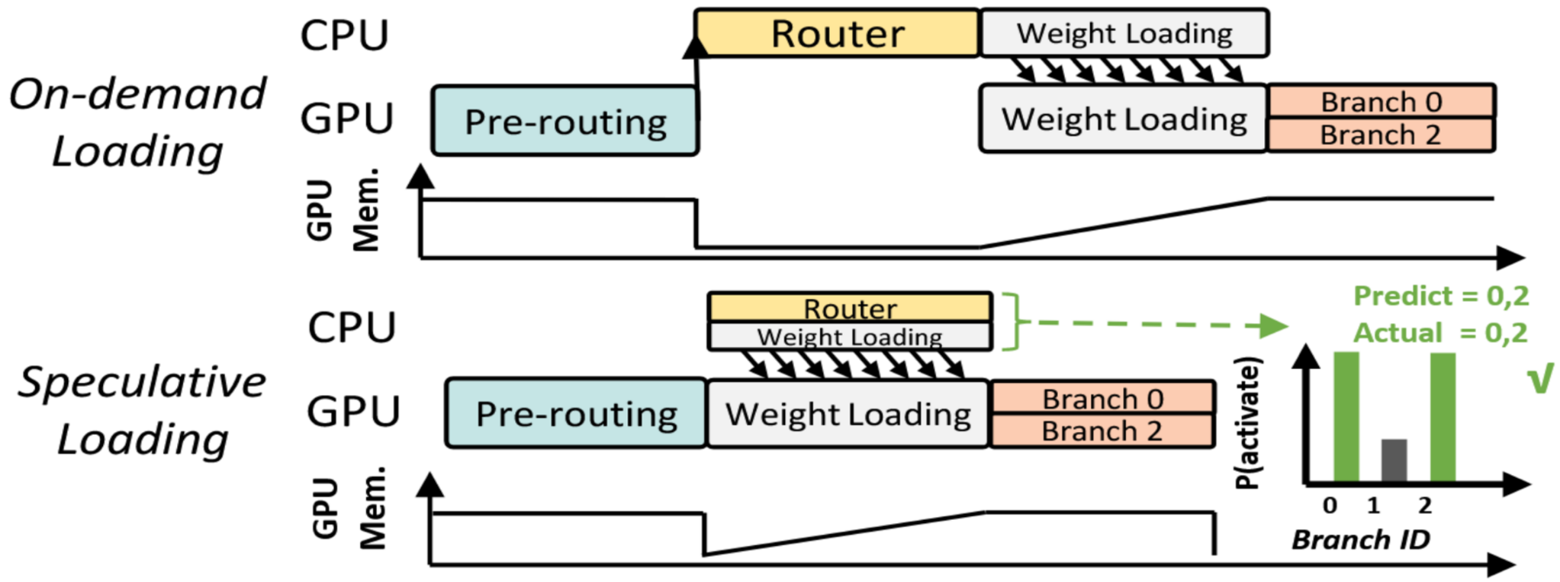
[1] FlexGen: high-throughput generative inference of large language models with a single GPU. ICML23
[2] Brinstorm:Optimizing dynamic neural networks with brainstorm. OSDI 23
[3] MoE-Infinity: Offloading-Efficient MoE Model Serving. arxiv2024
[4] ProMoE: Fast MoE-based LLM Serving using Proactive Caching. arxiv2024

Related Work

DeepSpeed Inference [Naive]

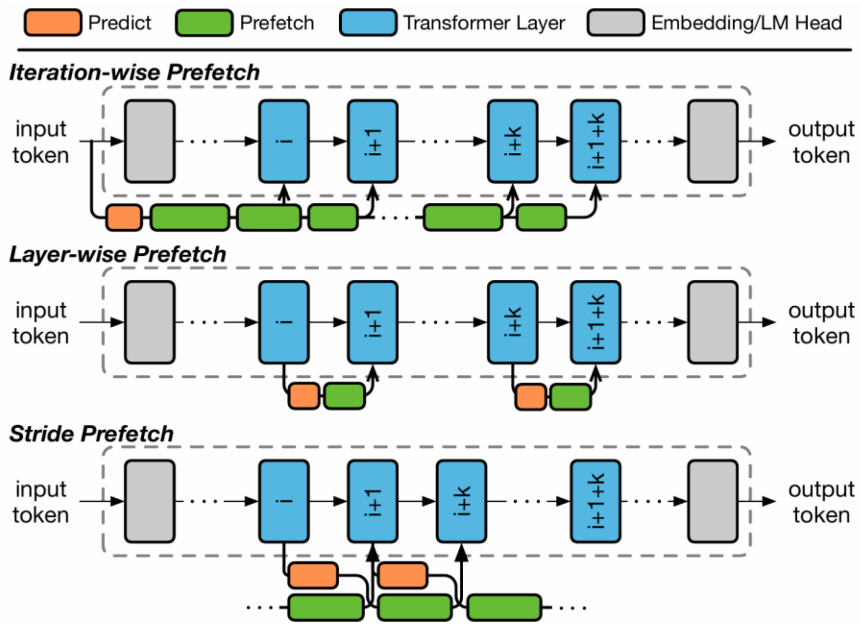
- Cache LRU used Expert
- Prefetch Expert activated at last iteration[dependency]

BrainStorm

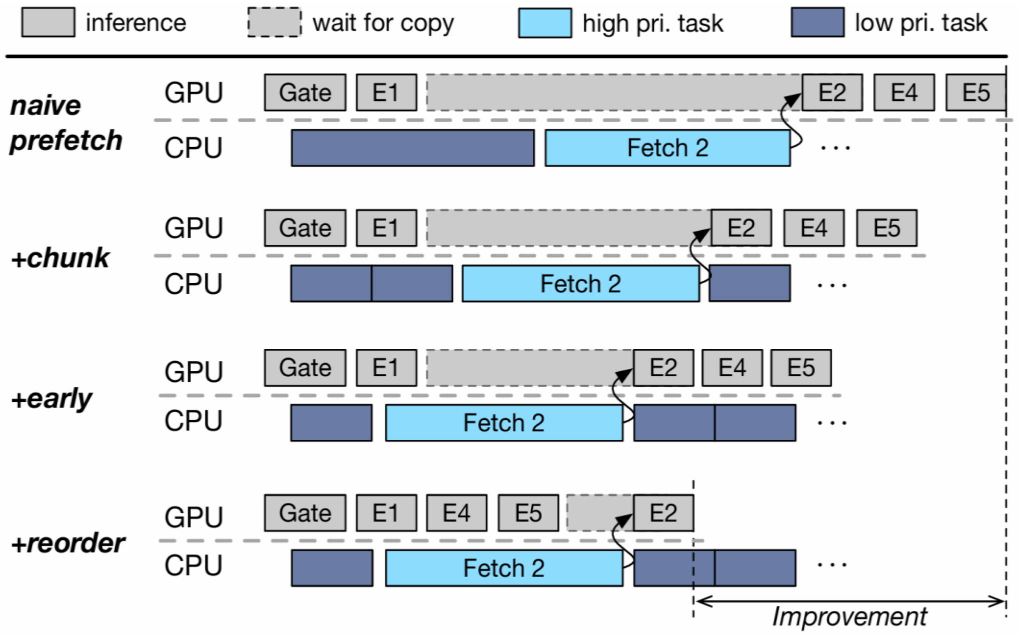


Related Work

ProMoE



Stride Prefetching with Neuron Predictor



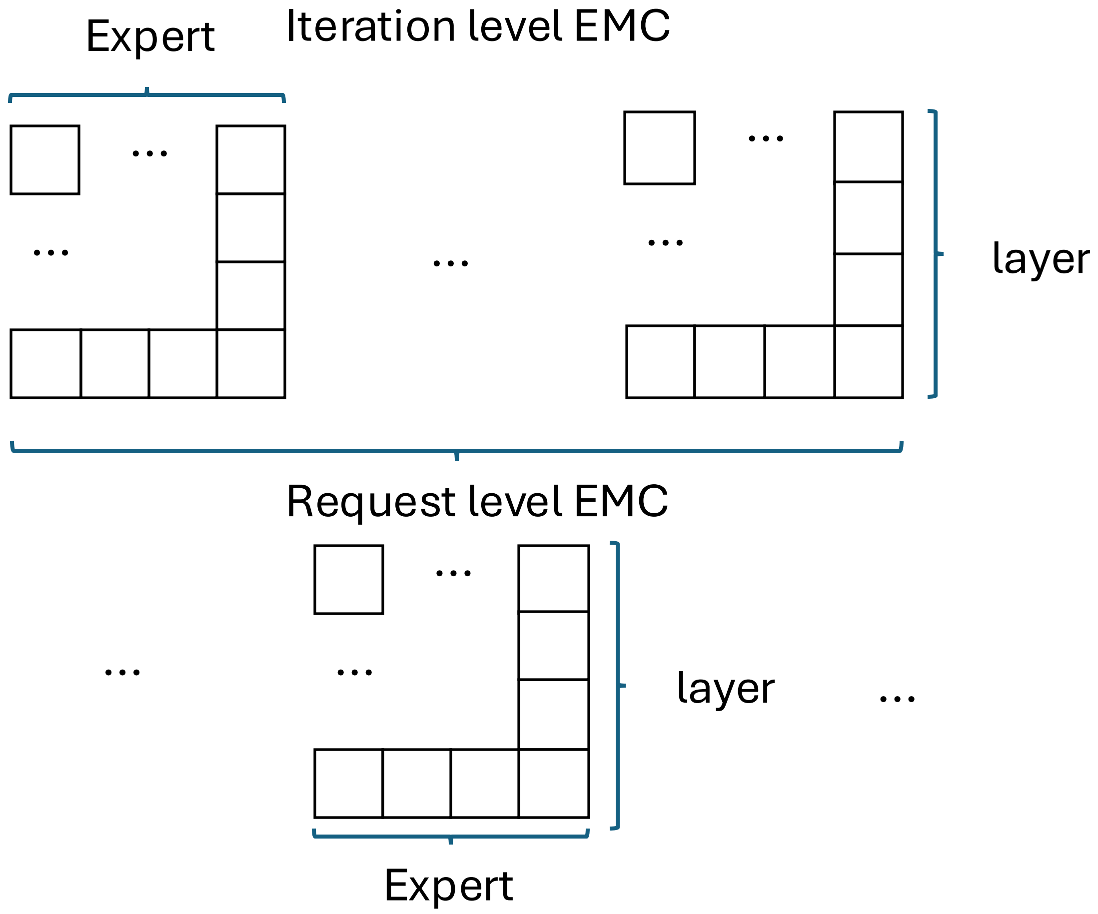
Chunked and Reorder prefetch shedule to overlap prefetch

Related Work (MoE Infinity)

1、Maintain multi-level data structure EMC

2、Cache router results

3、Prefetch and Cache Expert Based EMC



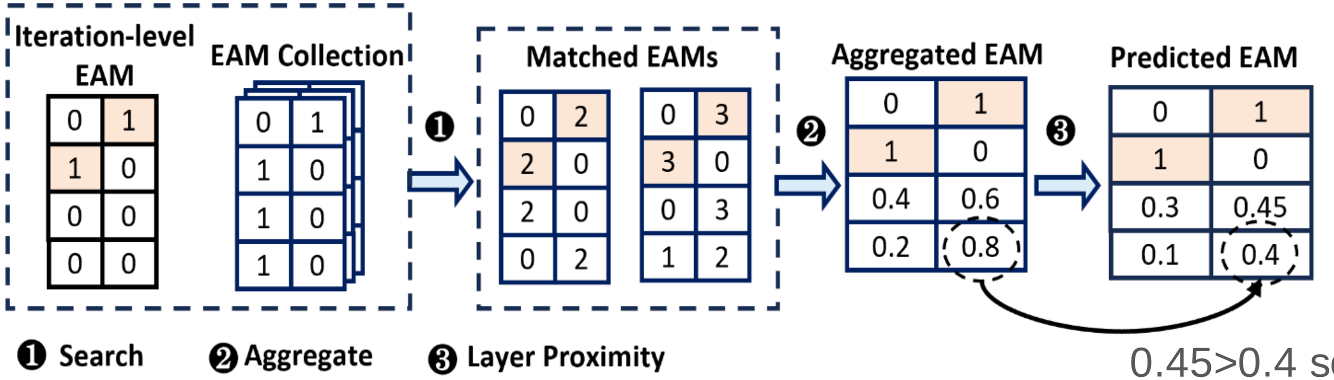
Related Work (MoE Infinity)

Prefetch

① Find prior matched EAMs From EAM Collection

② Compute activation probability for each expert

③ Adjusts the value in each cell through the formula (1 − (i − l)/L) [1]



0.45>0.4 so E[3,2] prefeteched first than E[4,2]

[1] where i is the future layer ID, l is the current layer ID and L is the layer number.

Related Work (MoE Infinity)

Catching schedule

Reserve a portion of the slots for prefetching.

\+ Prefetching Schedule +

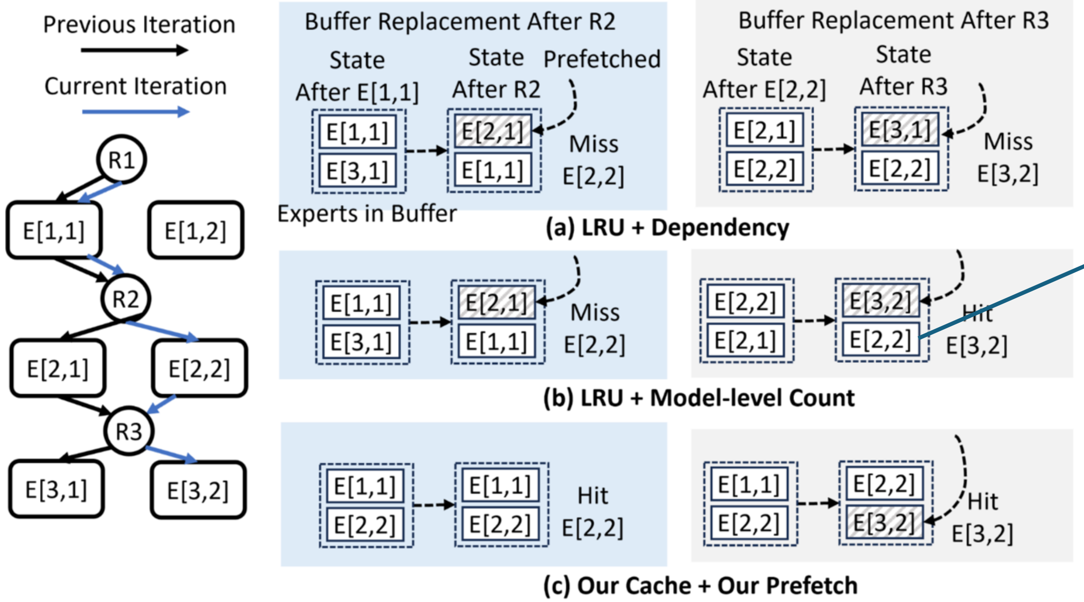
Cache the highest priority expert initially by the request-level EAM

Replace the lowest priority slots by the request-level EAM

1	2
0	3
0	3

request level EAM

=



(a) LRU + Dependency
(b) LRU + Model-level Count
(c) Our Cache + Our Prefetch

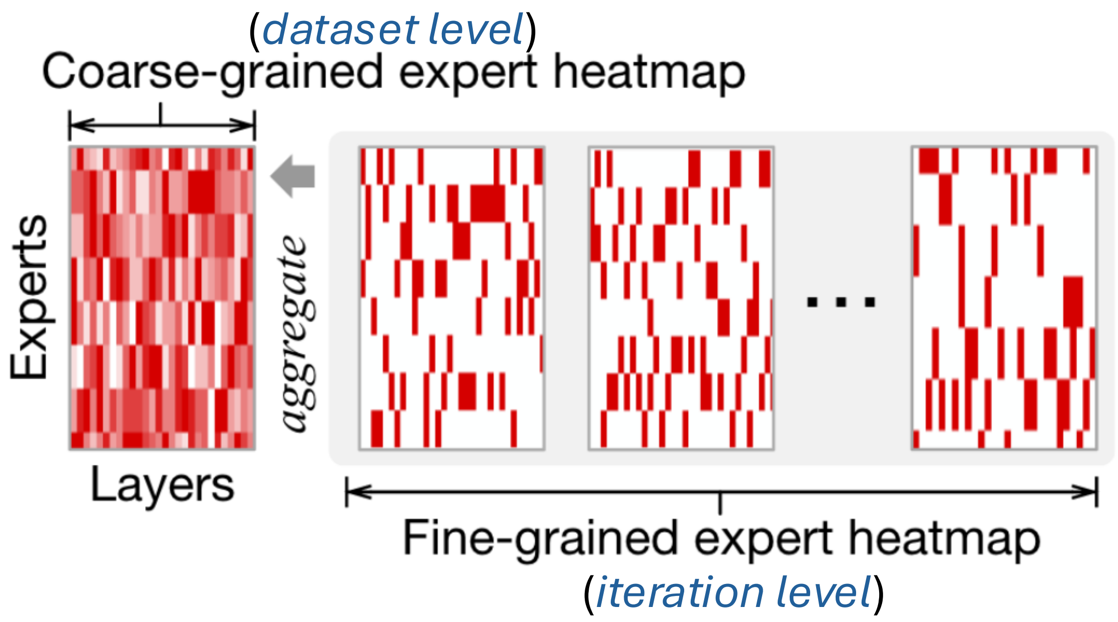
Risk!

Global Hot ≠ Local Hot

Contents

Problems in Current Expert Offloading

- ❑ Existing expert offloading solutions prefetch experts using inefficient guide data
  - ❖ Naive/Model level offloading solutions rely on coarse-grained expert patterns(e.g., dataset level), **but "global hot" doesn't equal to "local hot"!**



Problems in Current Expert Offloading

- Existing expert offloading solutions prefetch experts using inefficient guide data
  - Naive/Model level offloading solutions rely on coarse-grained expert patterns(e.g., dataset level), **but “global hot” doesn’t equal to “local hot”!**
  - Iteration level offloading solutions like MoE-Infinity only record expert hit counts, diminish the **contained possibility information** assigned to each expert by gate

0.1	0.1	0.1	0.2	0.1	0.1	0.2	0.1

Router Possibility

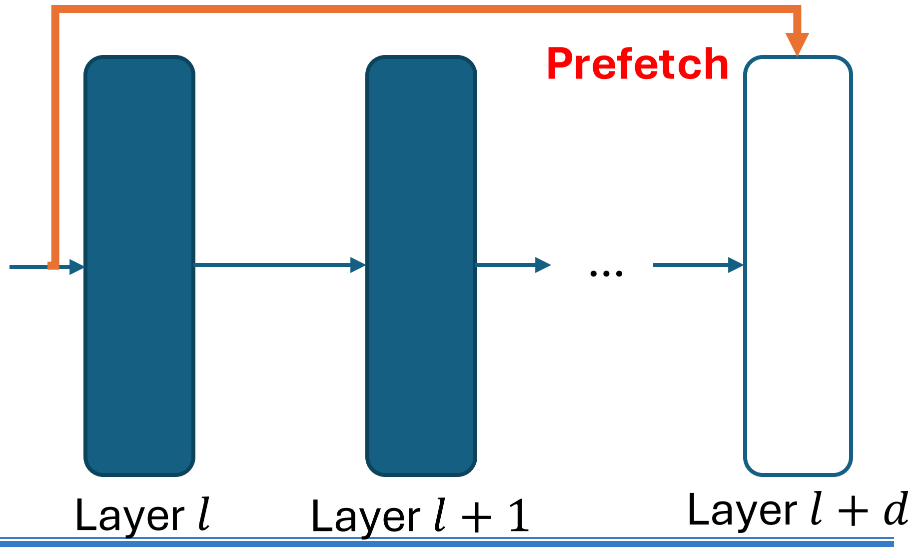
$\neq$

0	0	0	1	0	0	1	0

Hit Count

Problems in Current Expert Offloading

- ❑ Existing expert offloading solutions prefetch experts using inefficient guide data
- ❑ Ignorance of input prompts' heterogeneity
  - ❖ To overlap model inference with expert transmission(CPU → GPU), existing offloading methods will **prefetch $l + d$ layer's expert parameter at layer l,** d is the **prefetch distance**



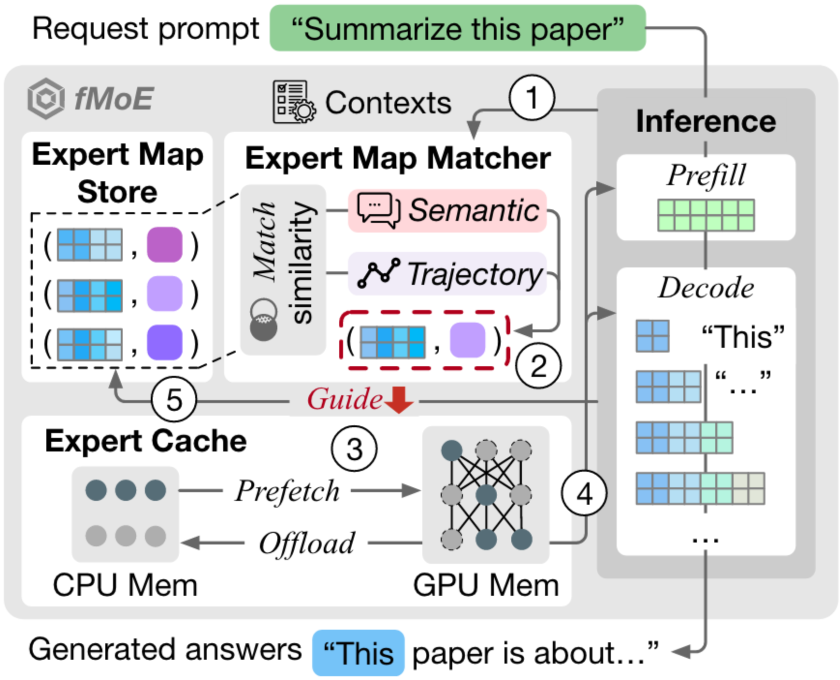
Problems in Current Expert Offloading

- ❑ Existing expert offloading solutions prefetch experts using inefficient guide data
- ❑ Ignorance of input prompts' heterogeneity
  - ❖ To overlap model inference with expert transmission(CPU → GPU), existing offloading methods will **prefetch $l + d$ layer's expert parameter at layer l,** d is the **prefetch distance**
  - ❖ For layers $\in [1, d]$ that don't have predecessor, existing methods (e.g., MoE-Infinity) prefetches the most popular exports from history, **ignoring the unique semantic information of input prompts**

Contents

fMoE Overview

- ❑ Track fine-grained iteration-level expert activation information (***Moti.1***)
  - ❖ Expert Map Store
- ❑ Use semantic/trajectory info of current batch to match best expert map (***Moti.2***)
  - ❖ Semantic/Trajectory Expert Map Matcher
- ❑ Expert prefetching guided by matched expert map
- ❑ Cache management & Expert map deduplication



Expert Map & Expert Map Store

- Expert Map
  - Data structure that tracks expert activation possibility (**router result**) in each model layer l at iteration level i
  - Recording every iteration's expert activation possibility makes it *fine-grained* and *lossless*

$$map_i := \{\mathbf{P}_1^{(i)}, \ldots, \mathbf{P}_l^{(i)}, \ldots, \mathbf{P}_L^{(i)}\}, \quad l \in [L].$$

$$\mathbf{P}_l^{(i)} := \{p_{l,1}^{(i)}, \ldots, p_{l,j}^{(i)}, \ldots, p_{l,J}^{(i)}\}, \quad \sum_{j \in [J]} p_{l,j}^{(i)} = 1, \ \forall p_{l,j}^{(i)} \geq 0.$$

abb.	full
i	iteration
L	layer num
J	Expert num in layer

Expert Map & Expert Map Store

- Expert Map
  - Data structure that tracks expert activation possibility (**router result**) in each model layer l at iteration level i
  - Recording every iteration's expert activation possibility makes it *fine-grained* and *lossless*

$$map_i := \{\mathbf{P}_1^{(i)}, \ldots, \mathbf{P}_l^{(i)}, \ldots, \mathbf{P}_L^{(i)}\}, \quad l \in [L].$$

$$\mathbf{P}_l^{(i)} := \{p_{l,1}^{(i)}, \ldots, p_{l,j}^{(i)}, \ldots, p_{l,J}^{(i)}\}, \quad \sum_{j \in [J]} p_{l,j}^{(i)} = 1, \ \forall p_{l,j}^{(i)} \geq 0.$$

- Expert Map Store
  - Dynamically keeps the most useful and unique expert maps for real-time queries during inference by *deduplication (later)*

Expert Map Matcher

- When a request prompt arrives, Expert map matcher searches the expert map store **for appropriate expert maps** to guide expert prefetching before inference

- Expert map matcher works in two scenarios divided by *prefetch distance* d
  - for layer $[d+1, L]$, how to efficiently choose the expert map?
    - **Solution**: Trajectory Similarity

Expert Map Matcher

- When a request prompt arrives, Expert map matcher searches the expert map store **for appropriate expert maps** to guide expert prefetching before inference

- Expert map matcher works in two scenarios divided by *prefetch distance* d
  - for layer $[d+1, L]$, how to efficiently choose the expert map?
    - **Solution**: Trajectory Similarity
  - for layer $[1, d]$, how to choose the expert map without sufficient distance?
    - **Solution**: Semantic Embedding Similarity

Matcher — Trajectory Similarity

- Trajectory Similarity
  - Record $(l-1)$ layers expert activation probabilities (router result) **as trajectories** to search expert map for $l+d$ layer
  - Given trajectories $map^{new} \in \mathbb{R}^{B \times (l-1)J}$ and historical expert maps $map^{old} \in \mathbb{R}^{C \times (l-1)J}$, compute pairwise cosine similarity:

$$score^{map}_{x,y} := \frac{map^{new}_x \cdot map^{old}_y}{\|map^{new}_x\| \cdot \|map^{old}_y\|}, \quad x \in [B],\ y \in [C].$$

abb.	full
B	batch size
C	map store capacity
x	one prompt
y	one history iter

Matcher — Trajectory Similarity

❑ Trajectory Similarity

❖ Record $(l-1)$ layers expert activation probabilities (router result) **as trajectories** to search expert map for $l+d$ layer

❖ Given trajectories $map^{new} \in \mathbb{R}^{B\times(l-1)J}$ and historical expert maps $map^{old} \in \mathbb{R}^{C\times(l-1)J}$, compute pairwise cosine similarity:

$$score_{x,y}^{map} := \frac{map_x^{new} \cdot map_y^{old}}{\|map_x^{new}\| \cdot \|map_y^{old}\|}, \quad x \in [B],\ y \in [C].$$

abb.	full
B	batch size
C	map store capacity
x	one prompt
y	one history iter

❑ Matching Method

❖ Select historical iteration y with **the highest similarity**, use $P_{l+d}^{y} \in map_y^{old}$ to guide the expert prefetching for target layer $l+d$

Matcher — Semantic Embedding Similarity

❑ Semantic Embedding Similarity

- ❖ Record input layer's embedding **as semantic input embedding** to search expert map **for $[1, d]$ layers** both in collecting expert(historical) maps and inference(new)
- ❖ Given embedding $sem^{new} \in \mathbb{R}^{B \times h}$ and historical embedding collection $sem^{old} \in \mathbb{R}^{C \times h}$, compute pairwise cosine similarity:

$$score^{sem}_{x,y} := \frac{sem^{new}_x \cdot sem^{old}_y}{\|sem^{new}_x\| \cdot \|sem^{old}_y\|}, \quad x \in [B],\ y \in [C]$$

abb.	full
B	batch size
C	map store capacity
x	one prompt
y	one history iter

Matcher — Semantic Embedding Similarity

❑ Semantic Embedding Similarity

- ❖ Record input layer's embedding **as semantic input embedding** to search expert map **for $[1, d]$ layers** both in collecting expert(historical) maps and inference(new)
- ❖ Given embedding $sem^{new} \in \mathbb{R}^{B \times h}$ and historical embedding collection $sem^{old} \in \mathbb{R}^{C \times h}$, compute pairwise cosine similarity:

$$score_{x,y}^{sem} := \frac{sem_x^{new} \cdot sem_y^{old}}{\|sem_x^{new}\| \cdot \|sem_y^{old}\|}, \quad x \in [B],\ y \in [C]$$

abb.	full
B	batch size
C	map store capacity
x	one prompt
y	one history iter

❑ Matching Method

- ❖ Select historical iteration y with **the highest similarity**, use $\{P_1^{(y)}, \ldots, P_d^{(y)}\} \in map_y^{old}$ to guide the expert prefetching for target layer $[1, d]$

Expert Prefetching

- Expert Map Guided Prefetching
  - Select **experts that have the highest probability score** from the expert map of one layer
  - If the similarity score from map matcher is high, prefetch less experts(but no less than TopK), if similarity score is low, prefetch more experts. Practical number is controlled by a threshold δ_l.

Cache Management

- Cache Management — Latency
  - For prefetching experts to cache, prioritize experts that have *higher probability* and are *closer to the current layer* :

$$PRI_{l,j}^{prefetch} := \frac{p_{l,j}}{l - l_{now}}, \quad l \in [L],\ j \in [J].$$

Cache Management

- Cache Management — Latency
  - For prefetching experts to cache, prioritize experts that have *higher probability* and are *closer to the current layer* :

$$PRI_{l,j}^{prefetch} := \frac{p_{l,j}}{l - l_{now}}, \quad l \in [L],\ j \in [J]$$

  - For evicting when cache is full, prioritize experts that have *lower probability* and are *less frequently hit*:

$$PRI_{l,j}^{evict} := \frac{1}{p_{l,j} \cdot freq_{l,j}}, \quad l \in [L],\ j \in [J]$$

Expert Map Deduplication

- Expert Map Deduplication — Memory
  - Compute the pairwise redundancy score $RDY_{x,y}$ to determine which old iterations to drop, update the old one with the expert map from new iteration

$$RDY_{x,y} := \frac{d}{L} \cdot score_{x,y}^{sem} + \frac{L-d}{L} \cdot score_{x,y}^{map}, \quad x \in [B],\ y \in [C]$$

Expert Map Deduplication

❑ Expert Map Deduplication — Memory

❖ Compute the pairwise redundancy score $RDY_{x,y}$ to determine which old iterations to drop, update the old one with the expert map from new iteration

$$RDY_{x,y} := \frac{d}{L} \cdot score_{x,y}^{sem} + \frac{L-d}{L} \cdot score_{x,y}^{map}, \quad x \in [B],\ y \in [C]$$

❖ 50K expert maps take 200MB CPU memory which is considerably memory efficient

Contents

Evaluation

Hardware

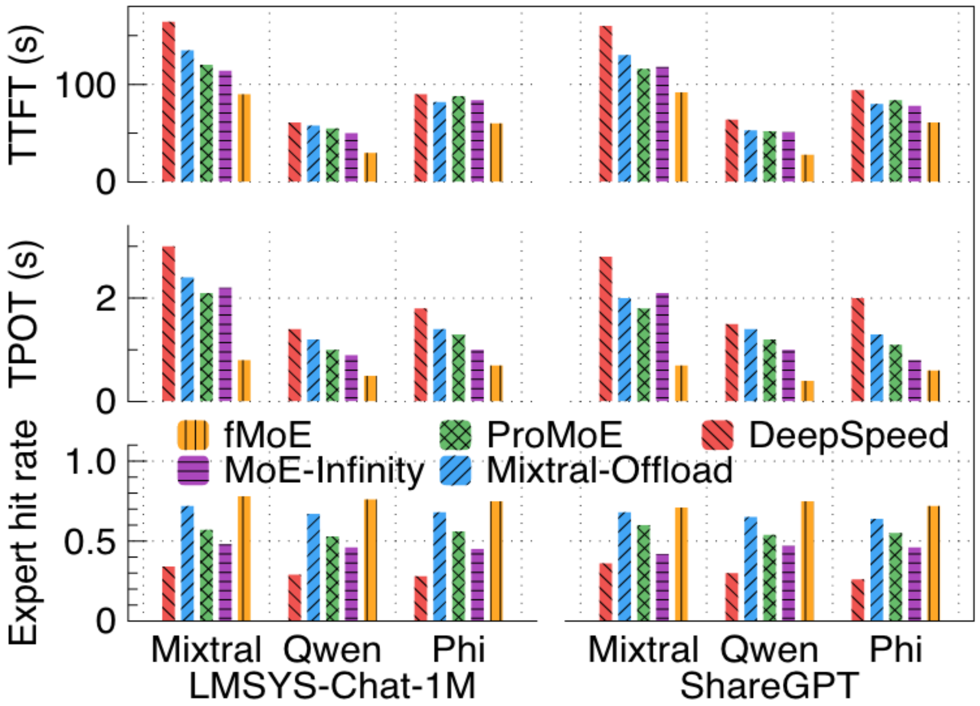
- 6 × NVIDIA RTX 3090 (24GB memory)
- Interconnect: NVlink(NVBridge)　　Host: PCIe 4.0 (32GB/s)
- 32 AMD Ryzen Threadripper PRO 3955 with WX480 GB CPU memory

Model

- Mixtral-8×7B
- Qwen1.5-MoE (8 × 2.7 B)
- Phi-3.5-MoE (16×3.8 B)

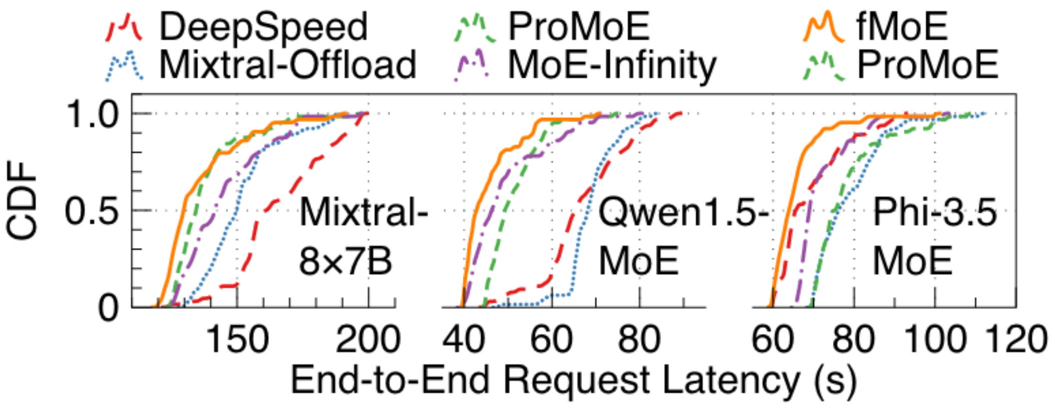
Evaluation

TTFT ⬇ TPOT ⬇ Expert Hit Rate ⬆



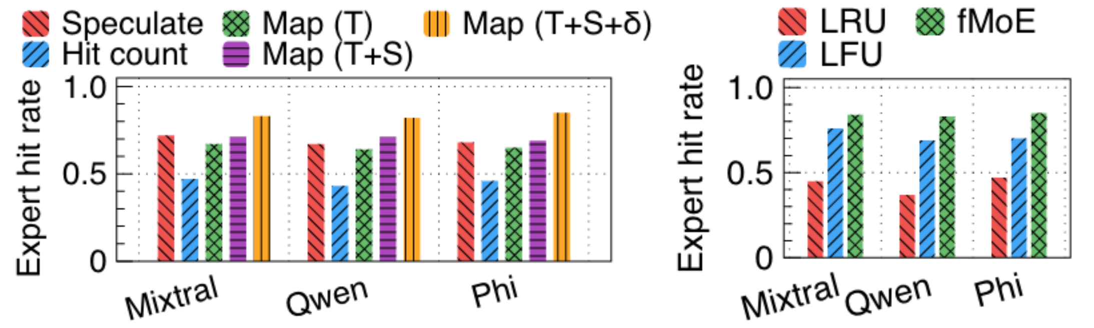
Offline Serving

[7:3]



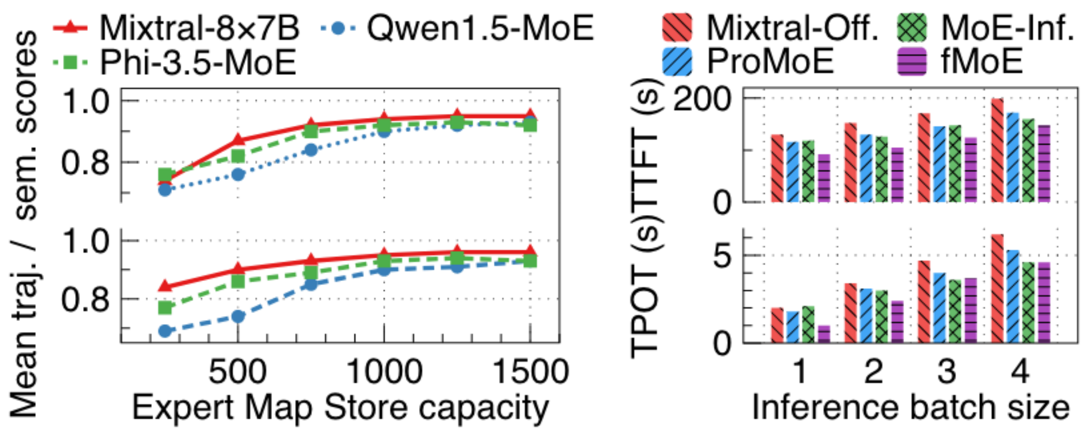
Online Serving

Evaluation



(a) Expert pattern tracking approaches. **(b)** Prefetch and caching.

Mean traj. \ Sem. scores ↑

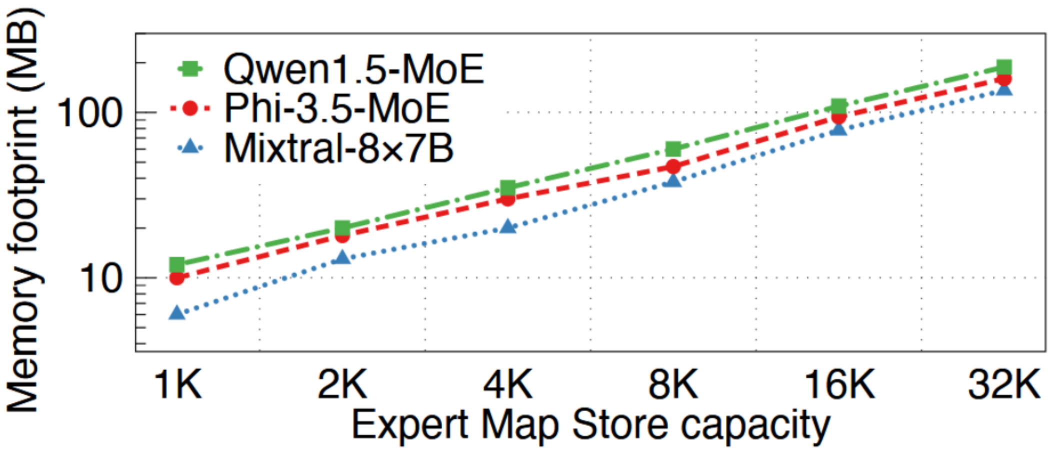


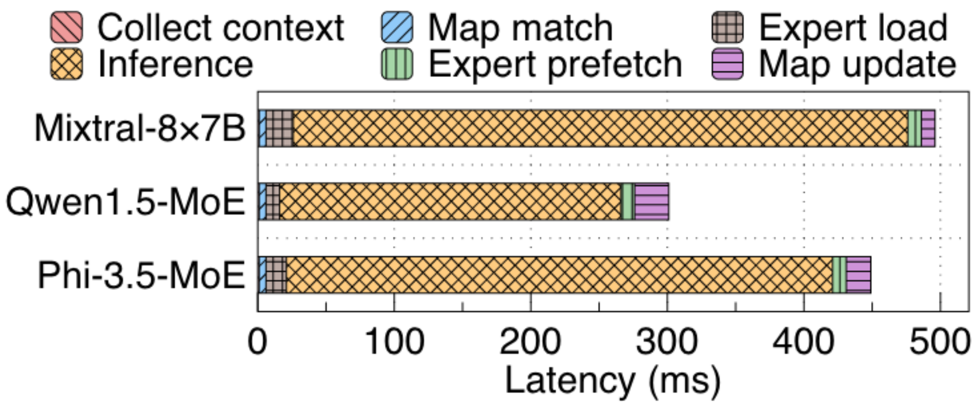
(a) Expert Map Store capacity. **(b)** Inference batch size.

Ablation Study

Sensitive Analysis

Evaluation





Different Model with different Expert number

Thanks!