FluxMoE: Decoupling Expert Residency for High-Performance MoE Serving

Qingxiu Liu[1], Cyril Y. He[2*], Hanser Jiang[2], Zion Wang[2],
Alan Zhao[2], Patrick P. C. Lee[1]

[1]The Chinese University of Hong Kong, [2]SCITIX

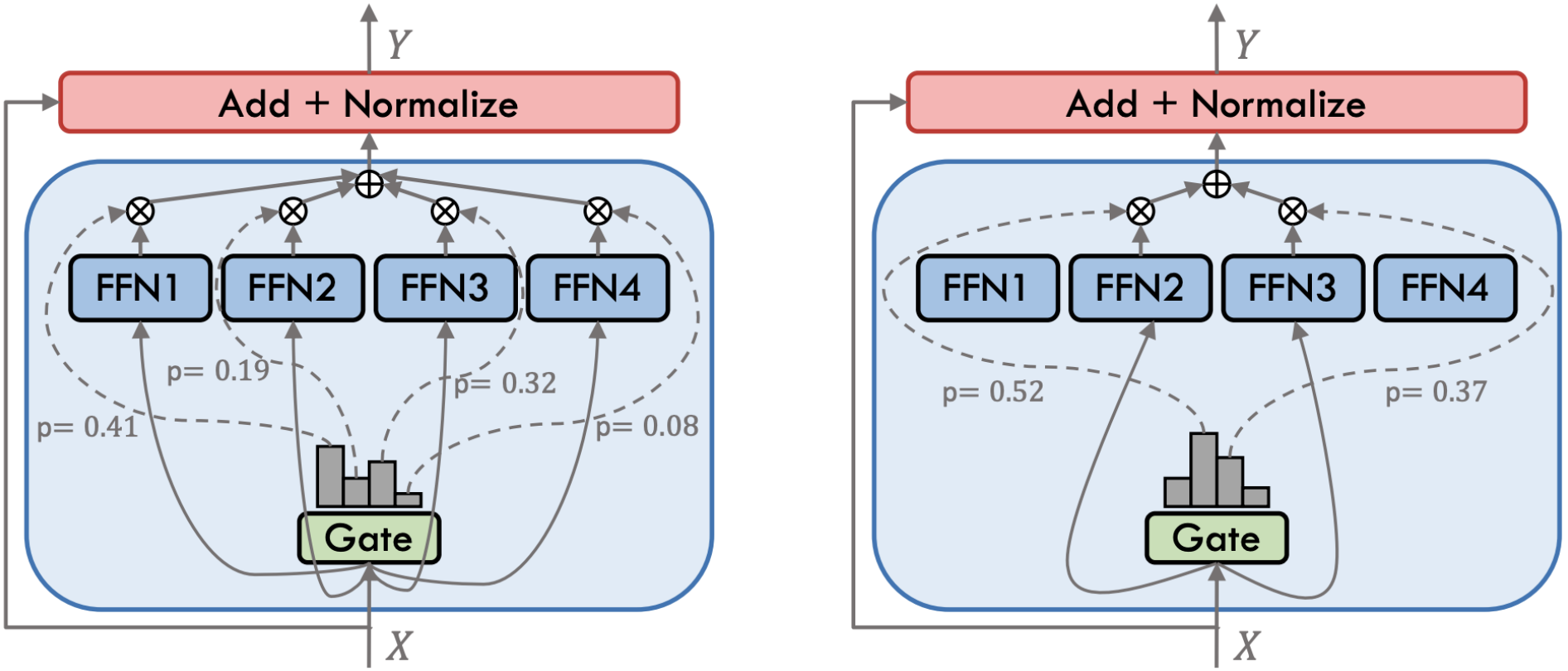
Presenter: Long Zhao

Outline

- ❑Background
- ❑Observation & Challenge
- ❑Design
- ❑Implementation & Evaluation
- ❑Summary

Background

- **MoE models have a sparse expert-based architecture:**
  - **activates only a subset of FFN modules (experts)**

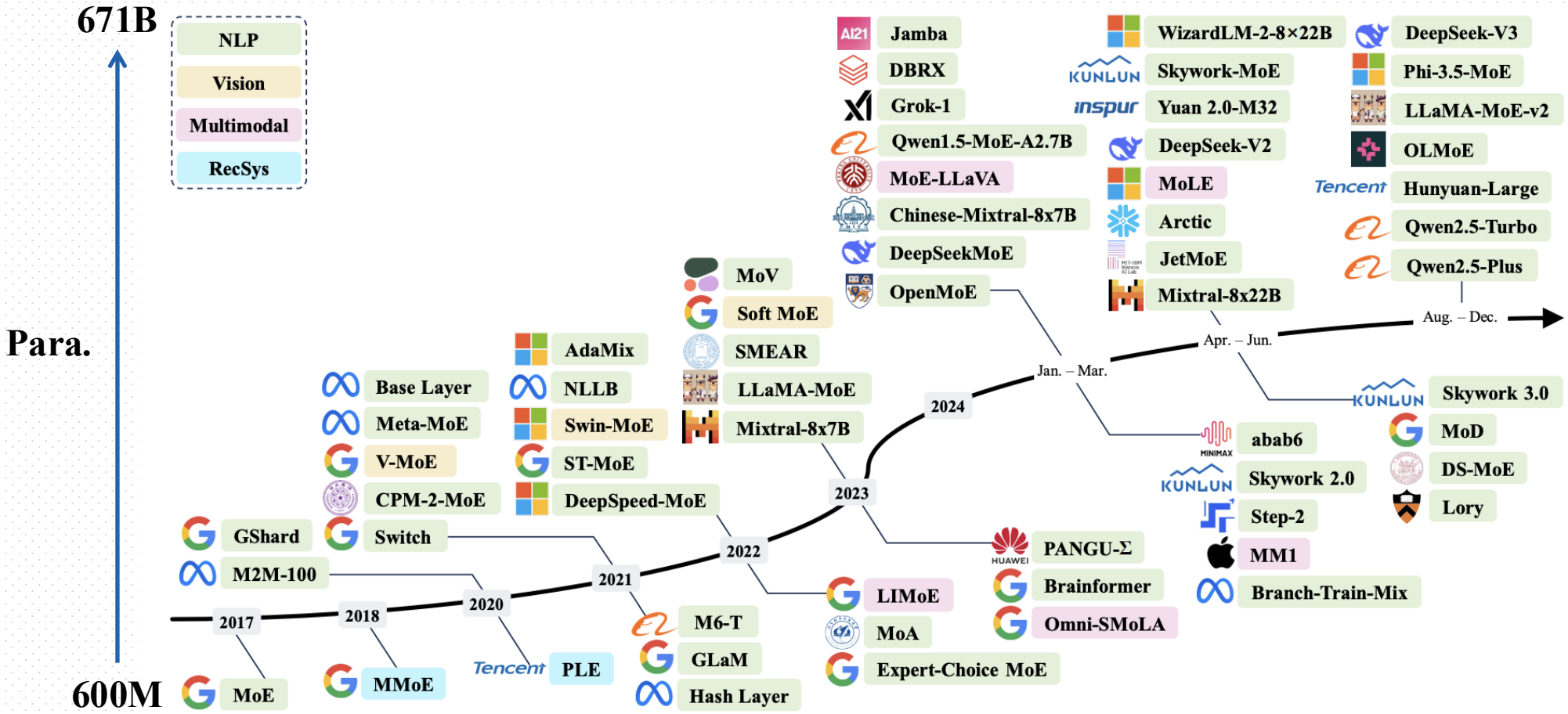


Dense model

MoE model

Background

❑MoE has rapidly evolved and been widely adopted across domains



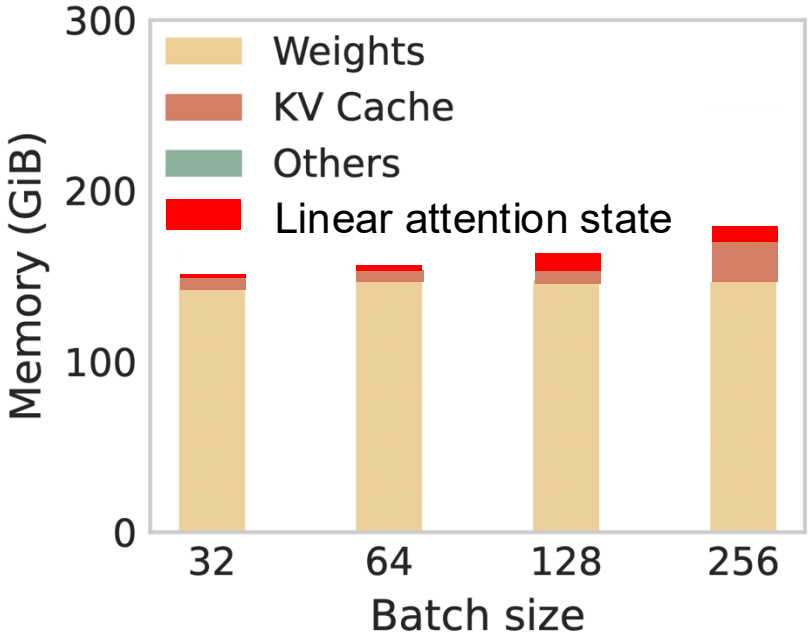
Model capacity has increased by over 1,000×

Background

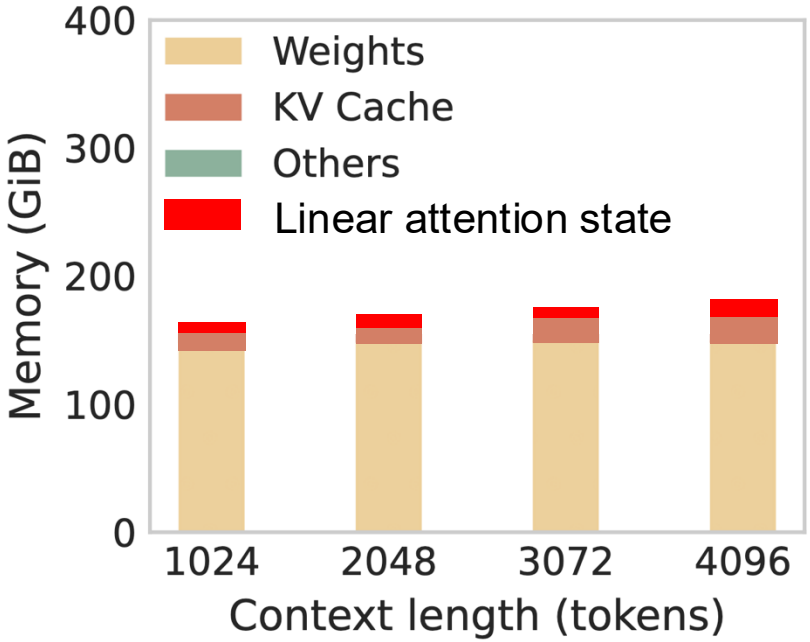
❑MoE inference memory characterization

❖Model weights occupy the majority of GPU memory

head dim=256
attention interval=4
layer=48
kv head=2
256*4096*12*2*2*256*2 ≈ 24G



(a) Context length = 4,096



(b) Batch size = 256

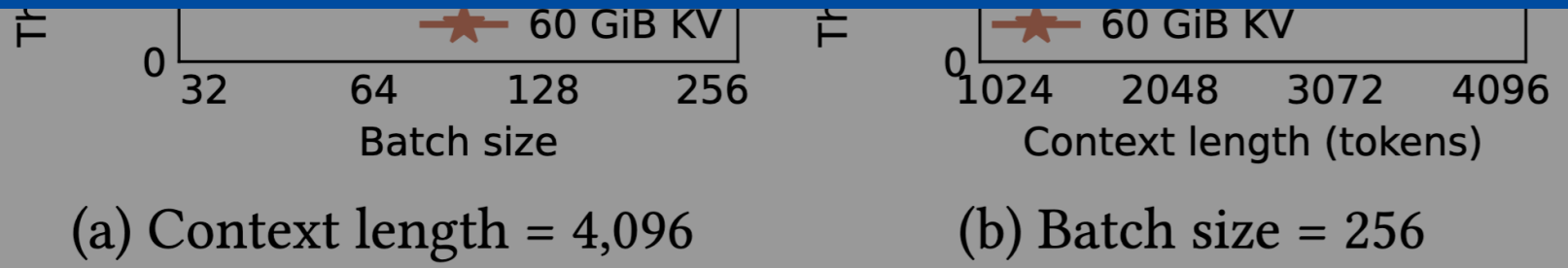
Qwen3-Next-80B-A3B-Instruct

Background

合肥综合性国家科学中心人工智能研究院
Institute of Artificial Intelligence, Hefei Comprehensive National Science Center

- MoE inference memory characterization
  - KV Cache capacity determines throughput

Static allocation leaves limited GPU memory for the KV cache



(a) Context length = 4,096 (b) Batch size = 256

Outline

- Background
- **Observation & Challenge**
- Design
- Implementation & Evaluation
- Summary

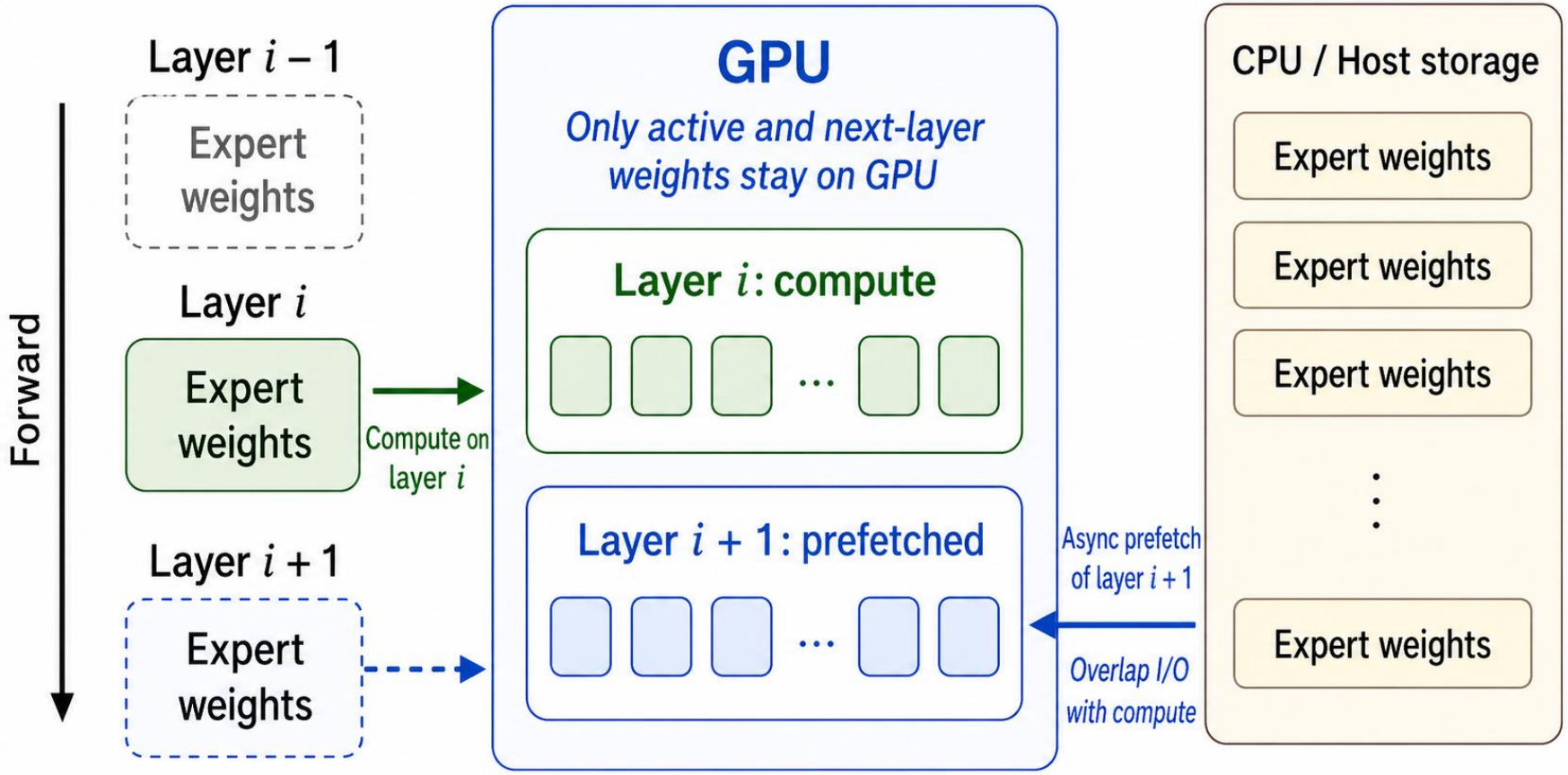
Observation

- Expert weights are only needed during their layer execution
- Expert dominate the model parameters, far exceeding attention[1]

[1] Huang W, Liao Y, Liu J, et al. Mixture compressor for mixture-of-experts llms gains more[J].

Insight

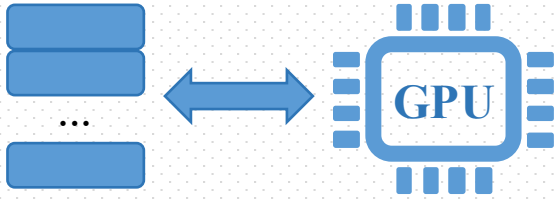
❑Instead of keeping all expert weights in GPU, they can be **loaded on demand** during inference.



Challenge

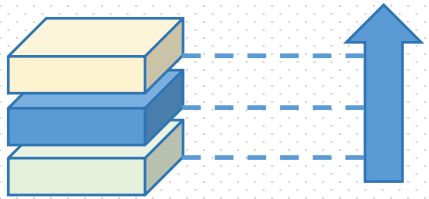
❑Achieve weight offloading and loading comes three challenges:

Logical-Physical Decoupling



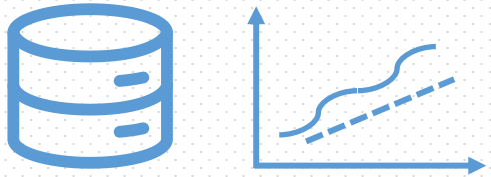
How to maintain virtual address consistency

Hierarchical Parameter transfer



How to transfer expert weight transparently

Workload-Aware Offloading



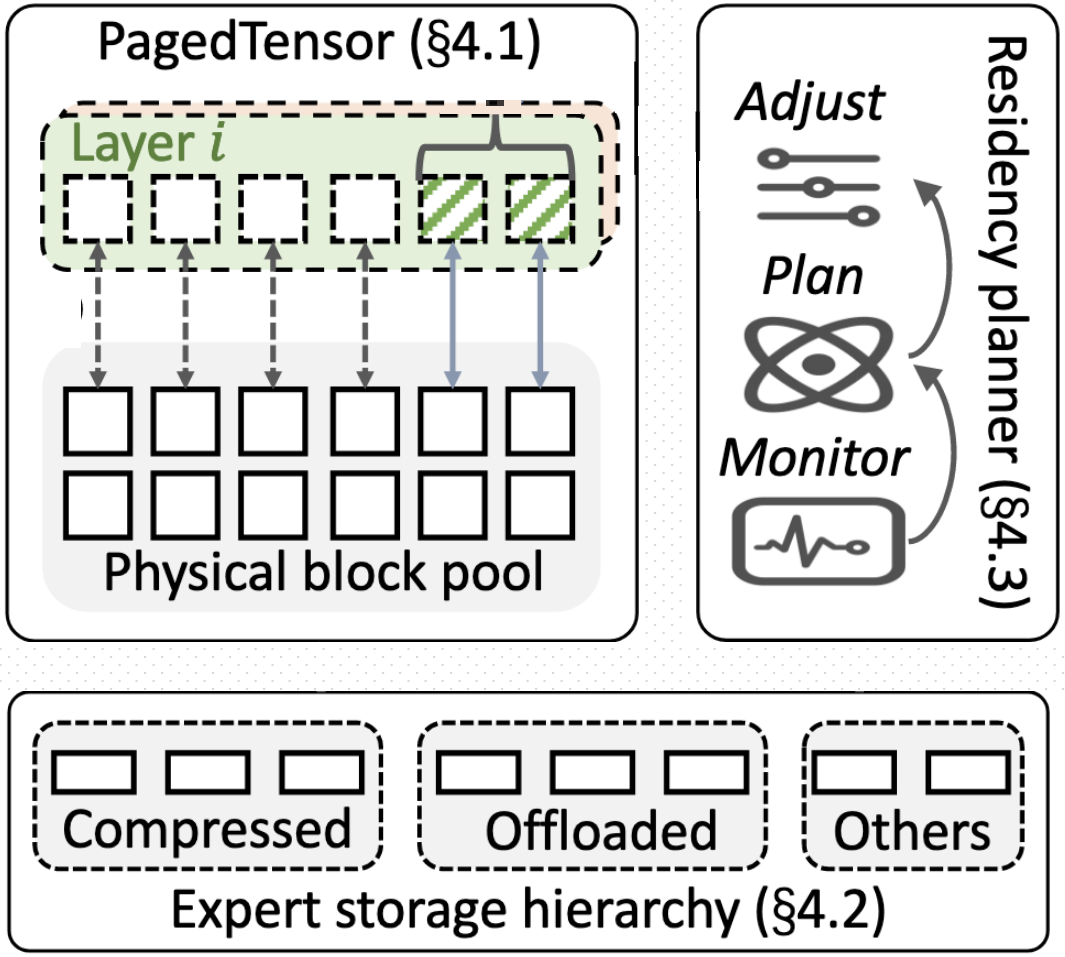
How to handle changing workloads

Outline

- Background
- Observation & Challenge
- **Design**
- Implementation & Evaluation
- Summary

FluxMoE Design

❑To address these challenges, FluxMoE introduces three modules



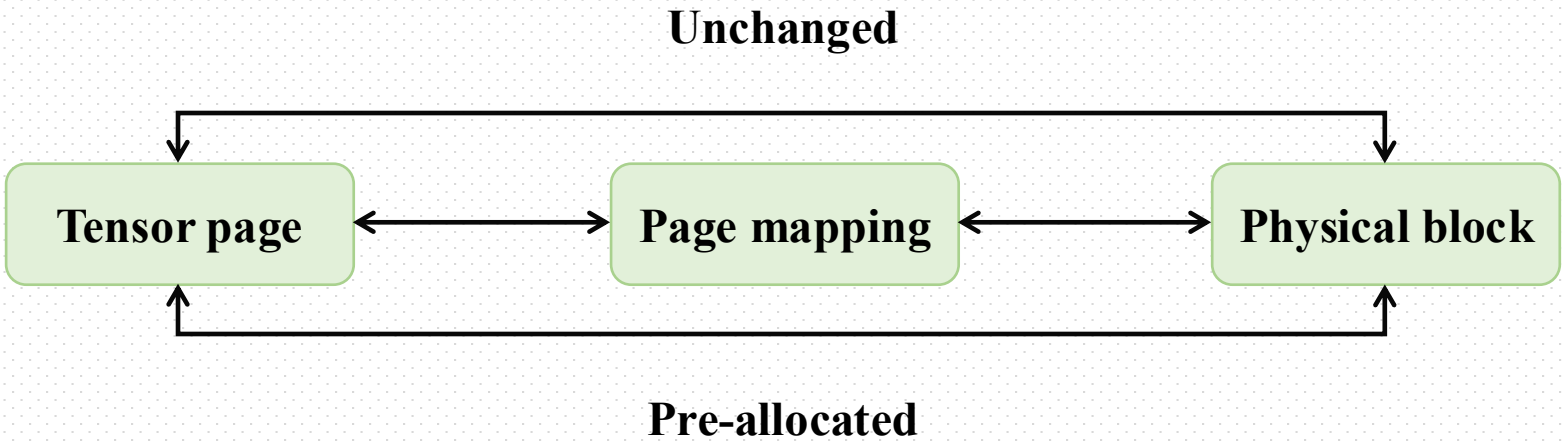
❑Virtualize expert weight addresses for on-demand loading

❑hierarchical storage with balanced bandwidth

❑Dynamically adjust expert residency, prioritizing KV cache

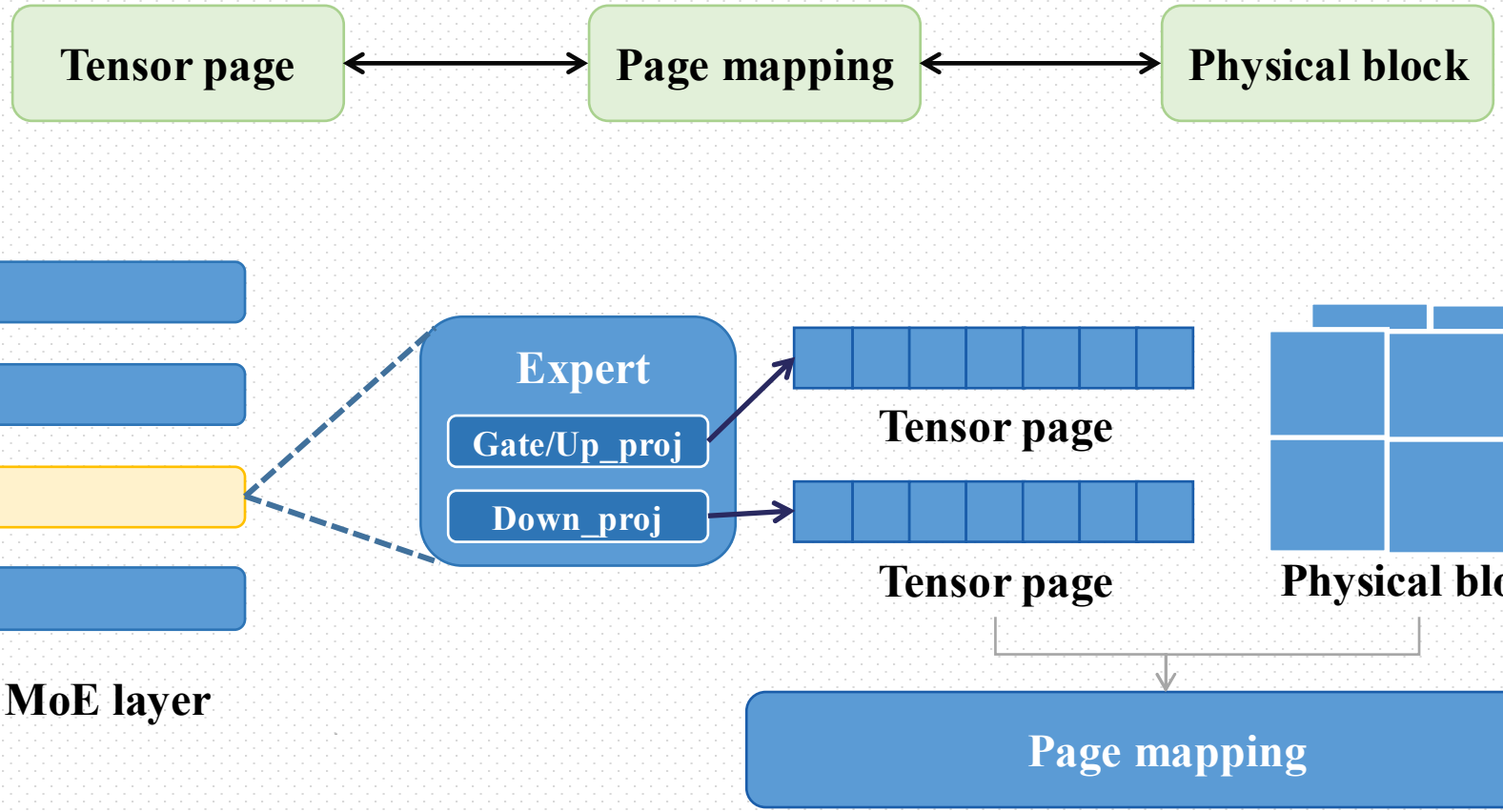
PagedTensor

- Maintain invariant virtual addresses for weight tensors
  - Each expert weight tensor corresponds to one Tensor page
  - Physical blocks are pre-allocated and persistently reserved, with a total capacity equal to the expert weights of two layers



PagedTensor

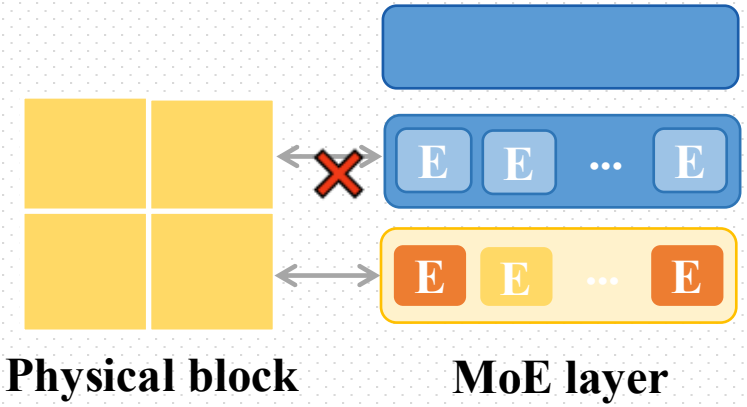
❑Maintain invariant virtual addresses for weight tensors



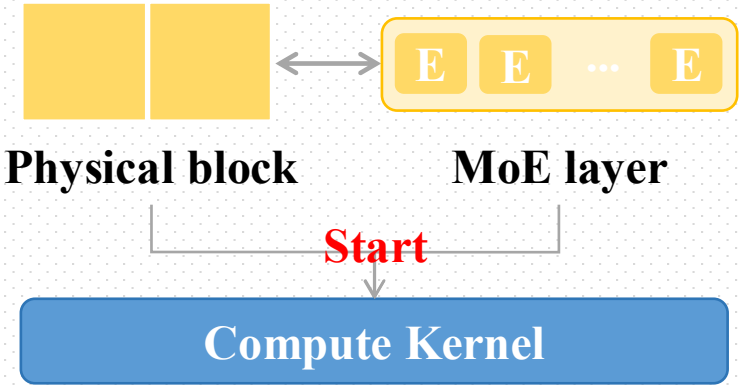
PagedTensor

❑**Asynchronously load expert weights on demand at the layer granularity, which governed by two ordering constraints:**

- ❖**Write-After-Read (WAR)**
- ❖**Read-After-Write (RAW)**



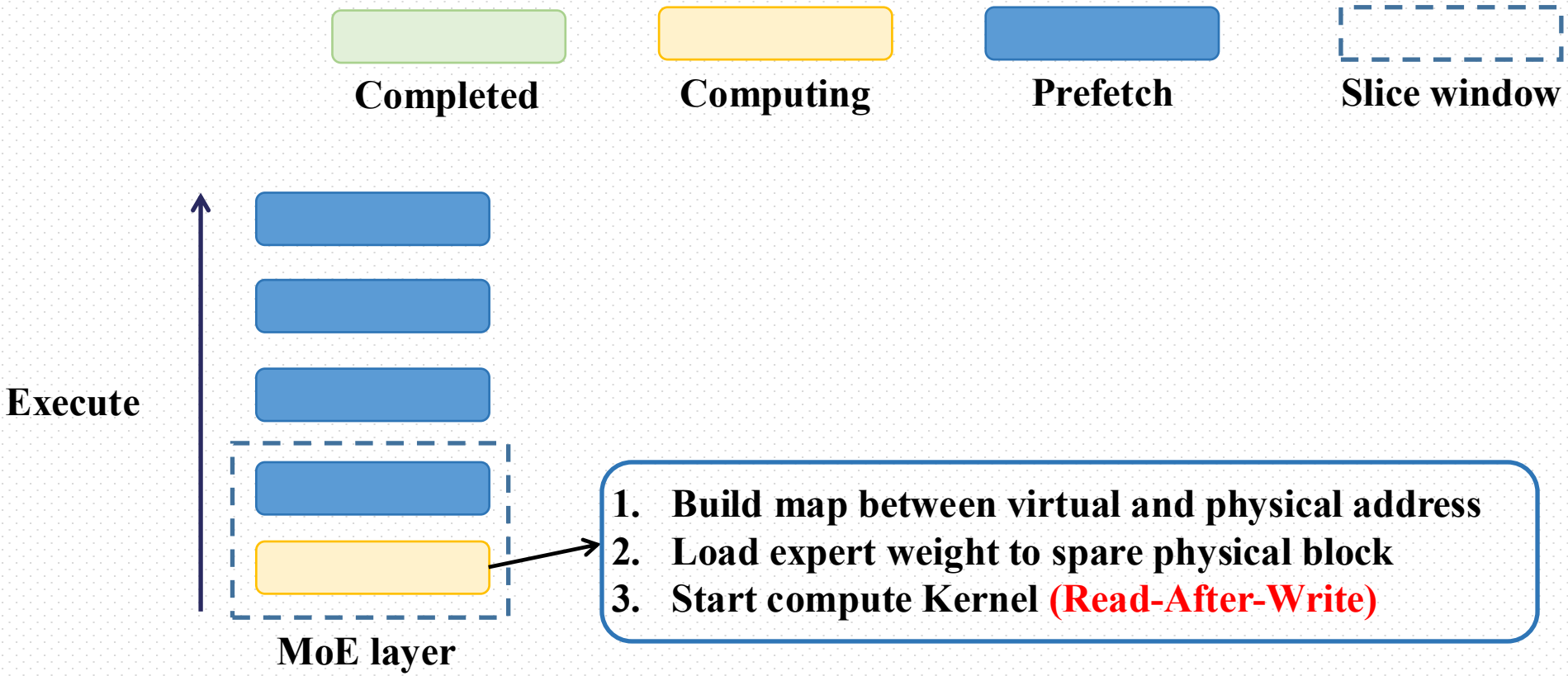
(1) Write-After-Read



(2) Read-After-Write

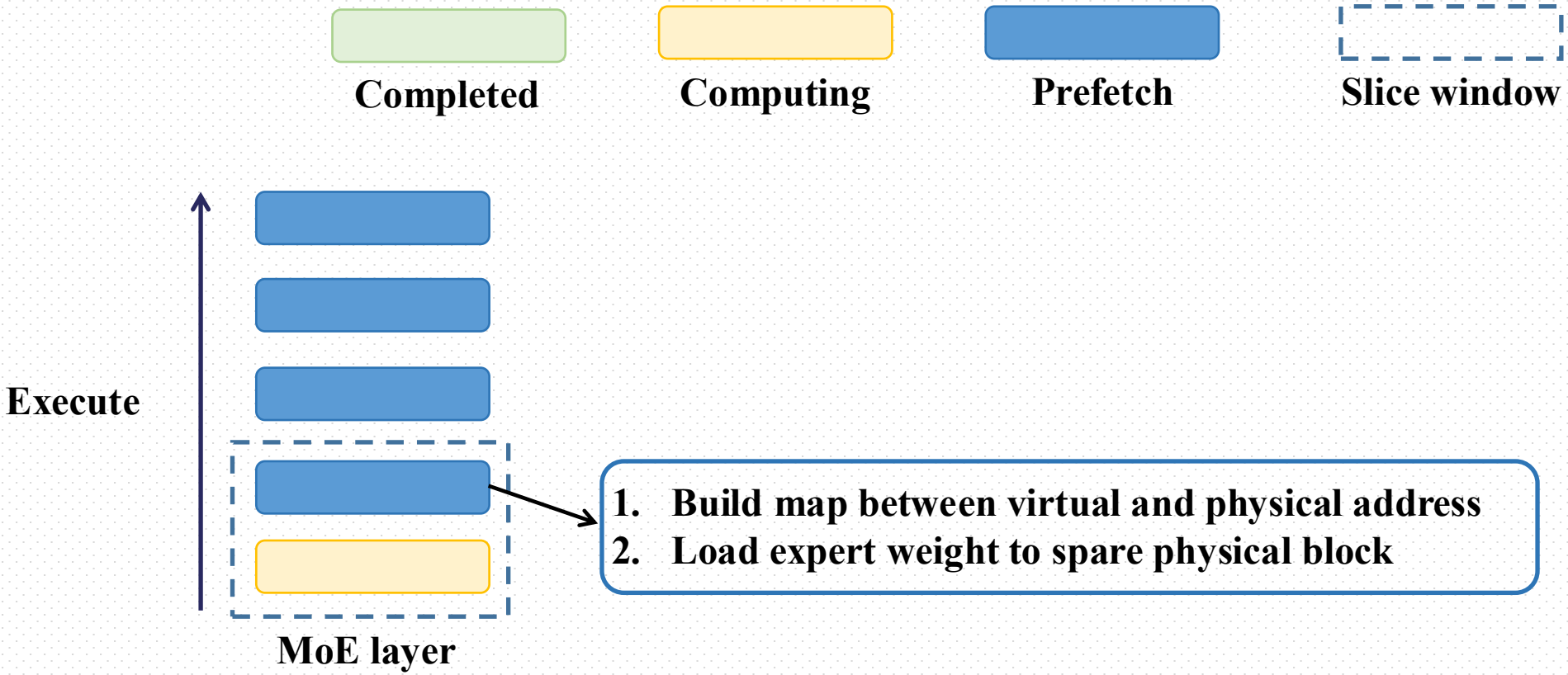
PagedTensor

❑Example for load expert weights on demand



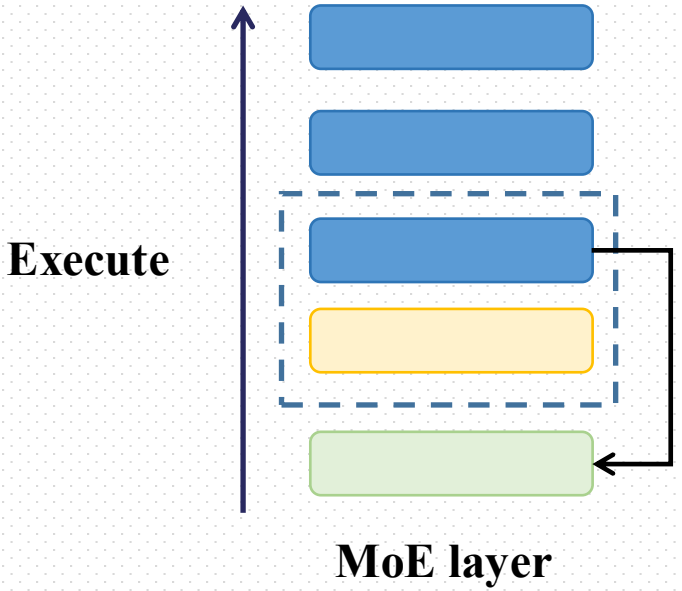
PagedTensor

❑Example for load expert weights on demand



PagedTensor

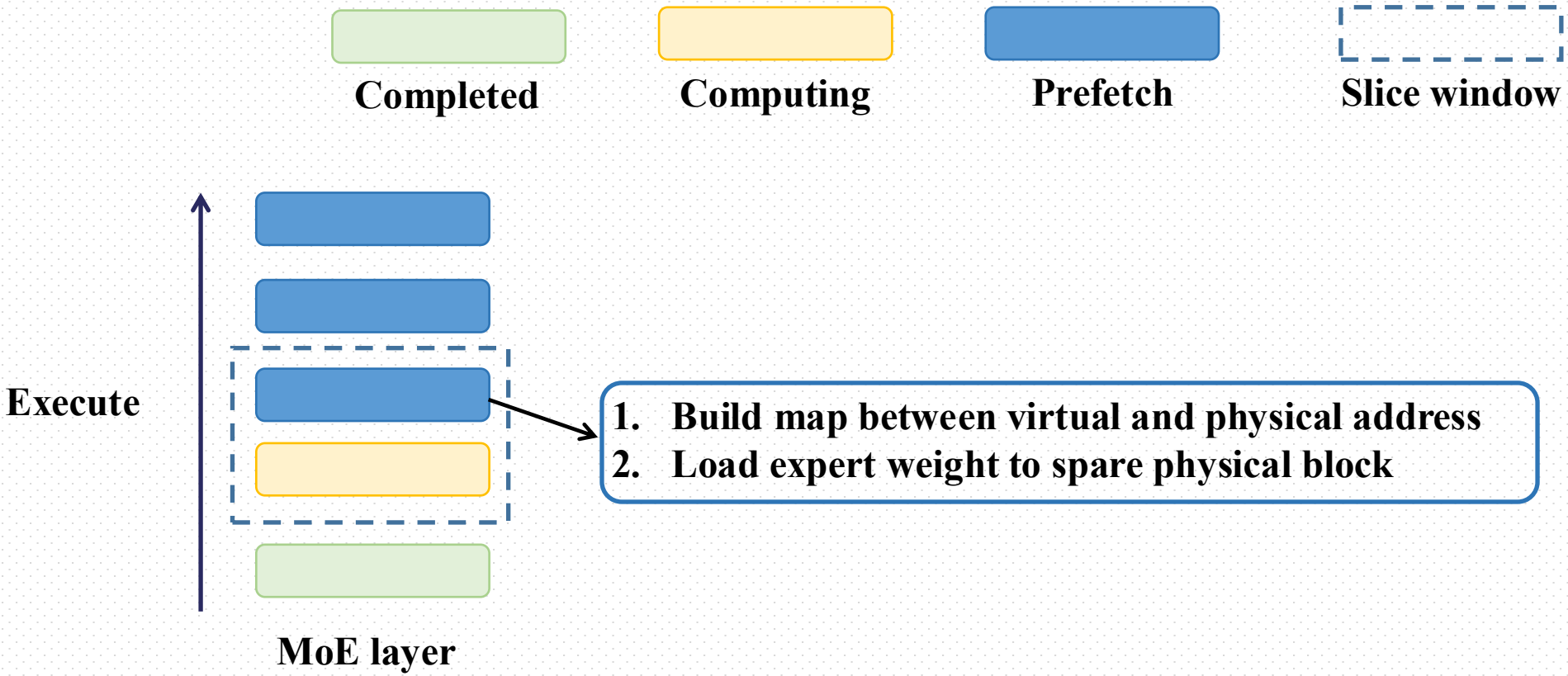
❑Example for load expert weights on demand



Check whether the computation has completed (Write-After-Read)

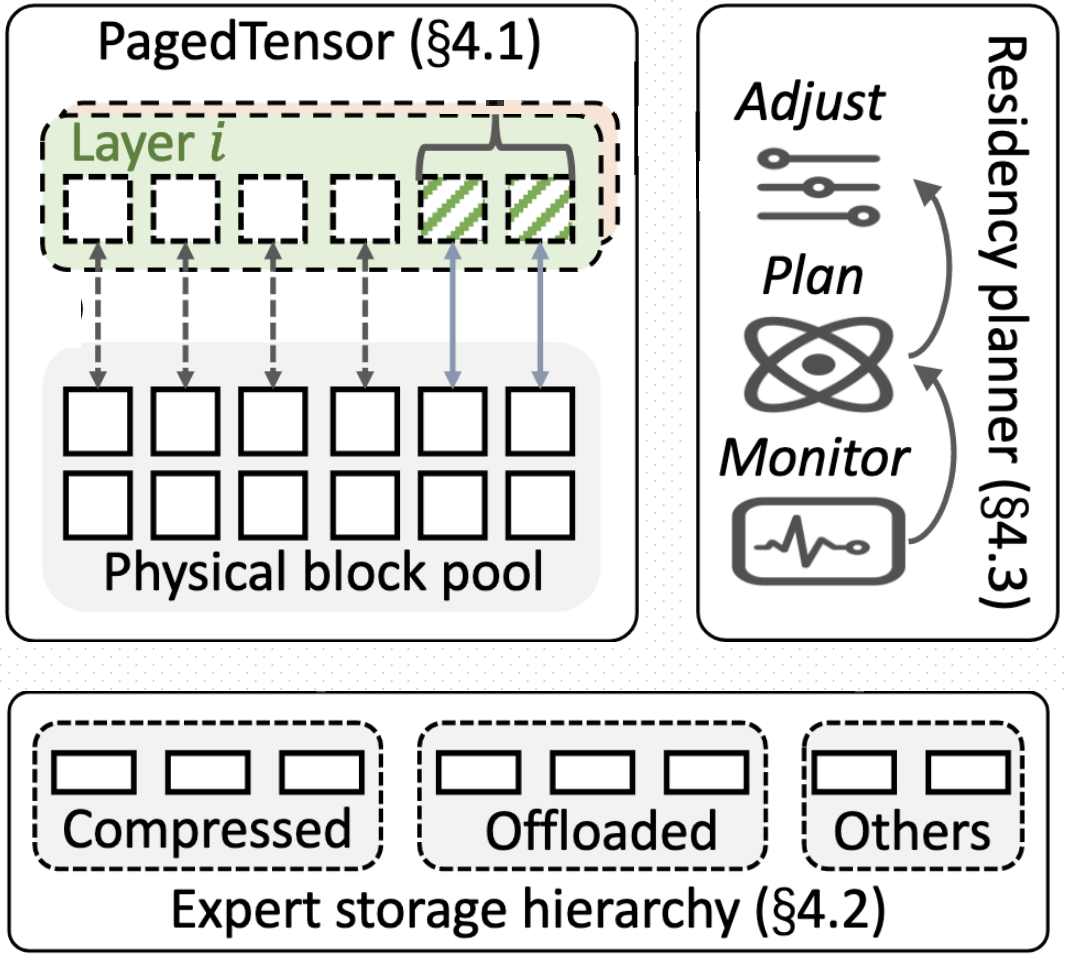
PagedTensor

❑Example for load expert weights on demand



FluxMoE Design

❑To address these challenges, FluxMoE introduces three modules



❑Virtualize expert weight addresses for on-demand loading

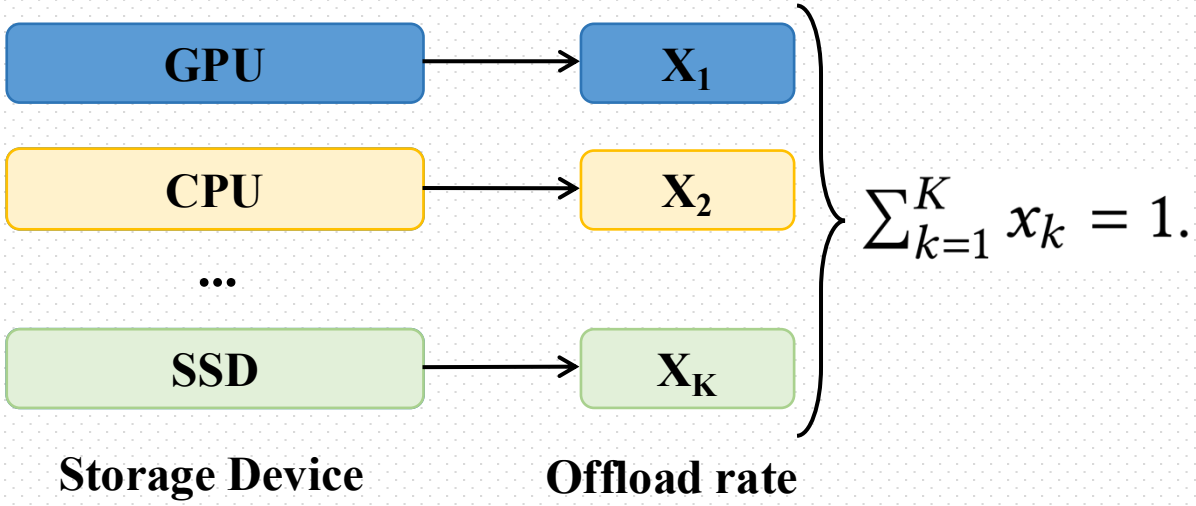
❑hierarchical storage with balanced bandwidth

❑Dynamically adjust expert residency, prioritizing KV cache

Expert Storage Hierarchy

- **Build storage hierarchy model and make weight transfer transparent**
  - **Assign an offload rate to each storage device**

$$\mathcal{S} = \{S_1, \ldots, S_K\}$$



$$\sum_{k=1}^{K} x_k = 1.$$

- **Define load time**

$$\tau_k = \frac{x_k \times P_{\text{layer}}}{B_k}.$$

Data

Bandwidth

- **Determine offload rate**

$$\frac{x_k \times P_{\text{layer}}}{B_k} = \frac{x_\ell \times P_{\text{layer}}}{B_\ell}, \qquad \frac{x_k}{B_k} = \frac{x_t}{B_t}$$

$$x_k = \frac{B_k}{\sum_{\ell=1}^{K} B_\ell}, \quad \forall k \in 1, \ldots, K.$$

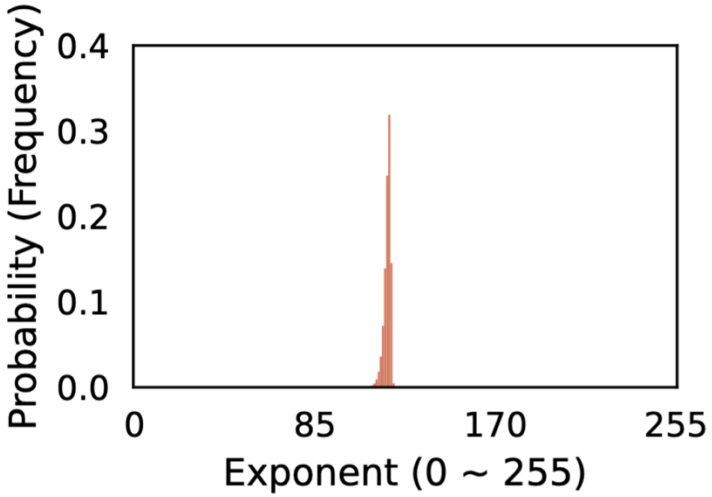
Expert Storage Hierarchy

- GPU bandwidth is high, more weights should be kept on the GPU

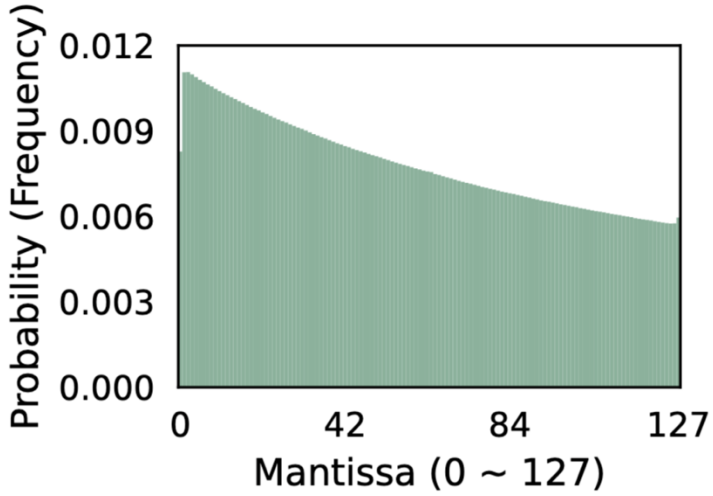
$$x_k = \frac{B_k}{\sum_{\ell=1}^{K} B_\ell}, \quad \forall k \in 1, \ldots, K.$$

- Fortunately, expert weights can be compressed losslessly

BF16 = Sign+Exponent+Mantissa



Low-entropy ✅

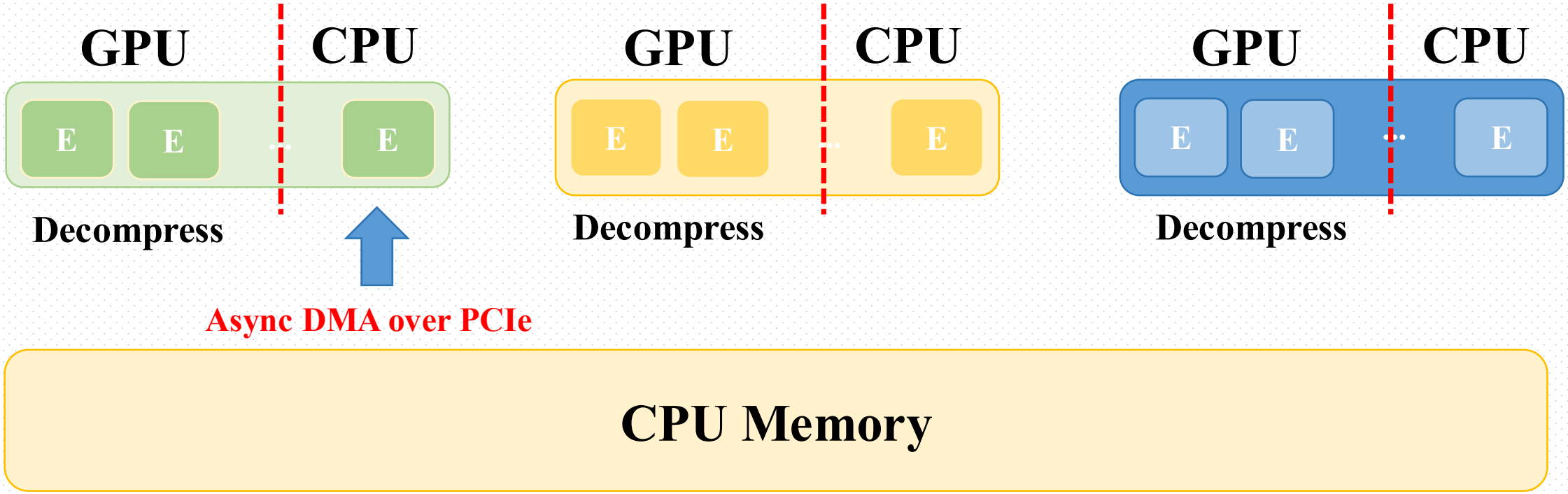


High-entropy ❌

Expert Storage Hierarchy

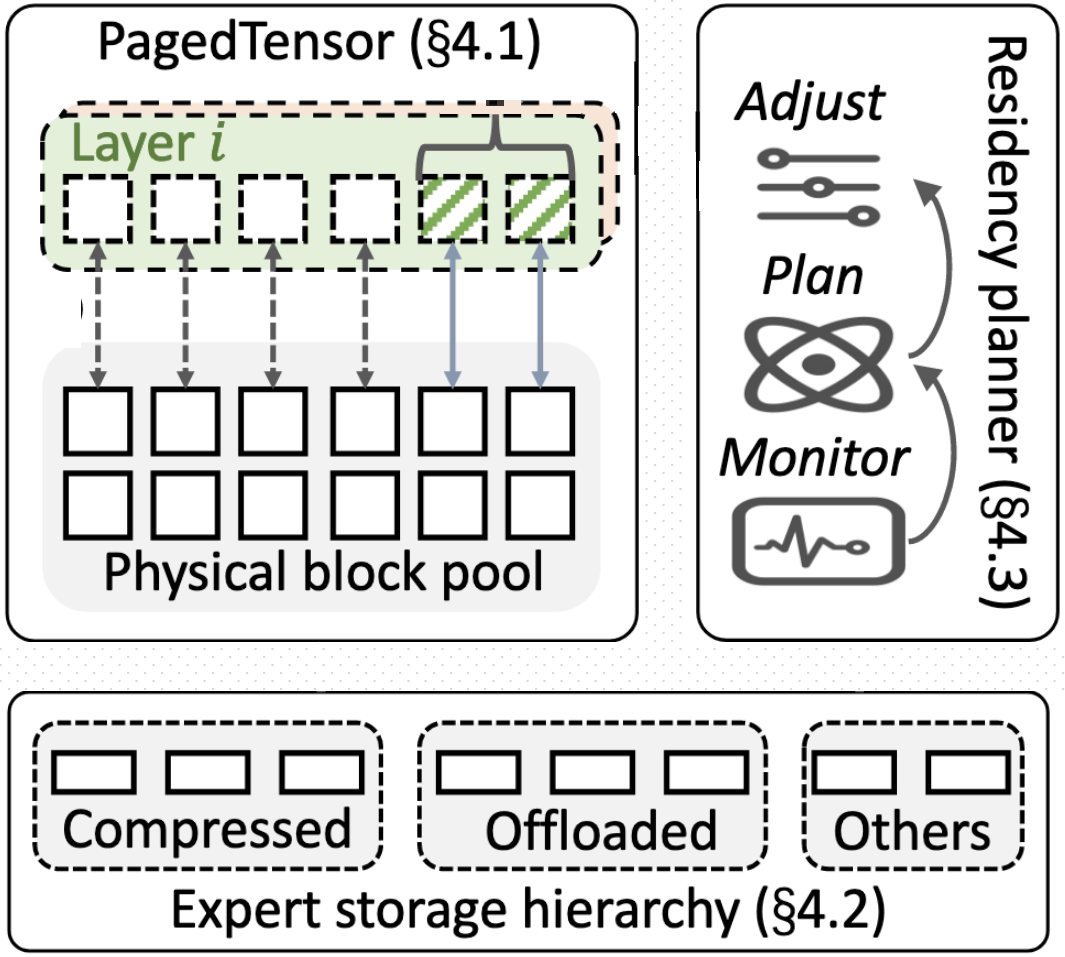
❑Example for Hierarchy Storage Loading

❖The experts in a layer are distributed across all storage devices



FluxMoE Design

❑To address these challenges, FluxMoE introduces three modules



❑Virtualize expert weight addresses for on-demand loading

❑Hierarchical storage with balanced bandwidth

❑Dynamically adjust expert residency, prioritizing KV cache

Residency Planner

❑Determine how many weights are kept on the GPU

❖Theoretical analysis

- ➢ the per-layer latency is set by the slowest backend

One Backend ⟵ $\tau_k = \frac{x_k \times P_{\text{layer}}}{B_k}.$ $\tau^{(i)}_{\text{load\_layer}} = \max_k \tau_k, \quad \forall k \in 1, \ldots, K.$ ⟶ One Layer

- ➢ The total time is the sum of the time across all layers.

$$\tau_{\text{load}} = \sum_{i=1}^{N} \tau^{(i)}_{\text{load\_layer}}.$$

- ➢ Define the compute-to-load ratio as ρ to describe the current bottleneck

$$\rho = \frac{\tau_{\text{comp(theory)}}}{\tau_{\text{load}}}.$$

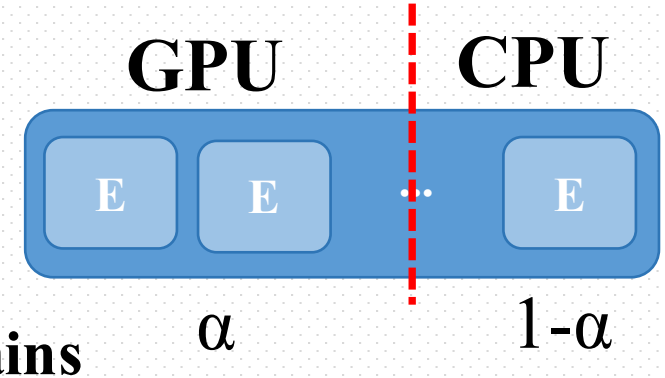
The ideal value of ρ is 1, where computation and loading can overlap

Residency Planner

❑Determine how many weights are kept on the GPU

❖Residency control model

➢ Define α as the fraction kept in GPU memory.



❖Memory-aware constrains

➢ Prioritize GPU memory for the KV cache.

$$\mathcal{C}_{\text{res}} = \mathcal{C}_{\text{gpu}} - \mathcal{C}_{\text{kv}}$$

GPU memory reserved for expert residency

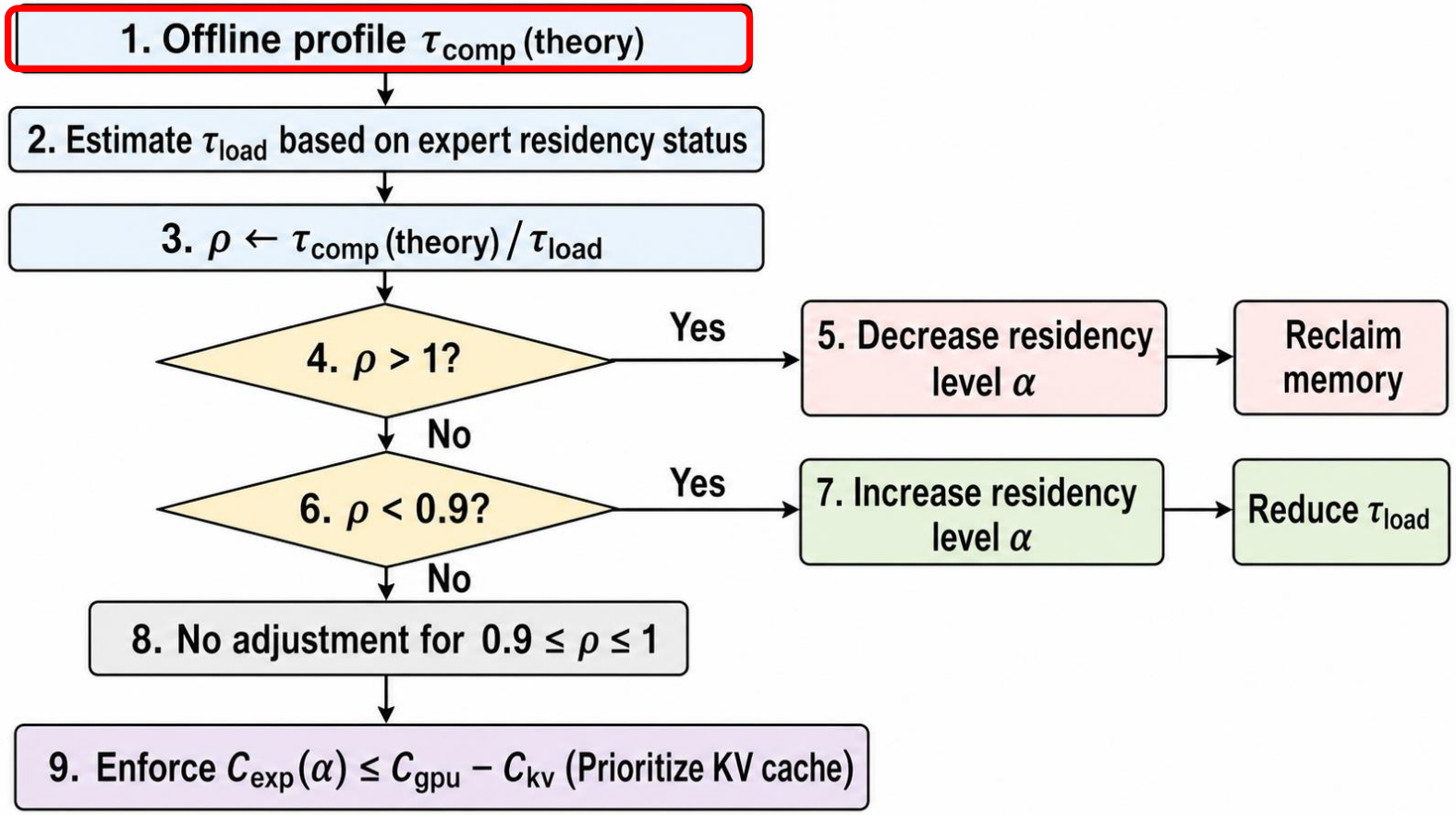
$$\mathcal{C}_{\text{exp}}(\alpha) \leq \mathcal{C}_{\text{res}}$$

GPU memory constrain

Residency Planner

❑ **Adaptively adjust the offloading strategy at runtime**

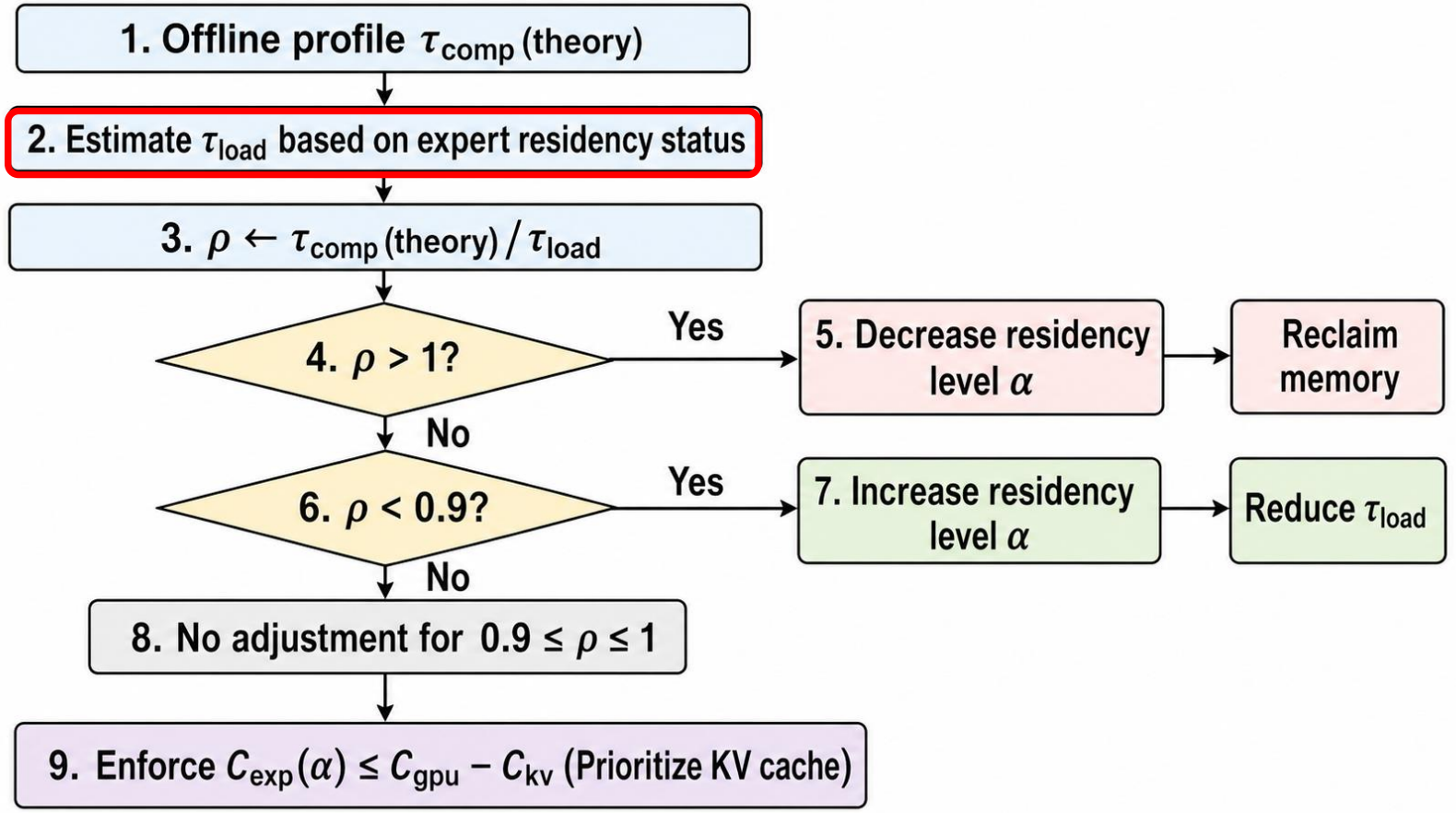
$$\rho = \frac{\tau_{\text{comp(theory)}}}{\tau_{\text{load}}}.$$



Residency Planner

❑ **Adaptively adjust the offloading strategy at runtime**

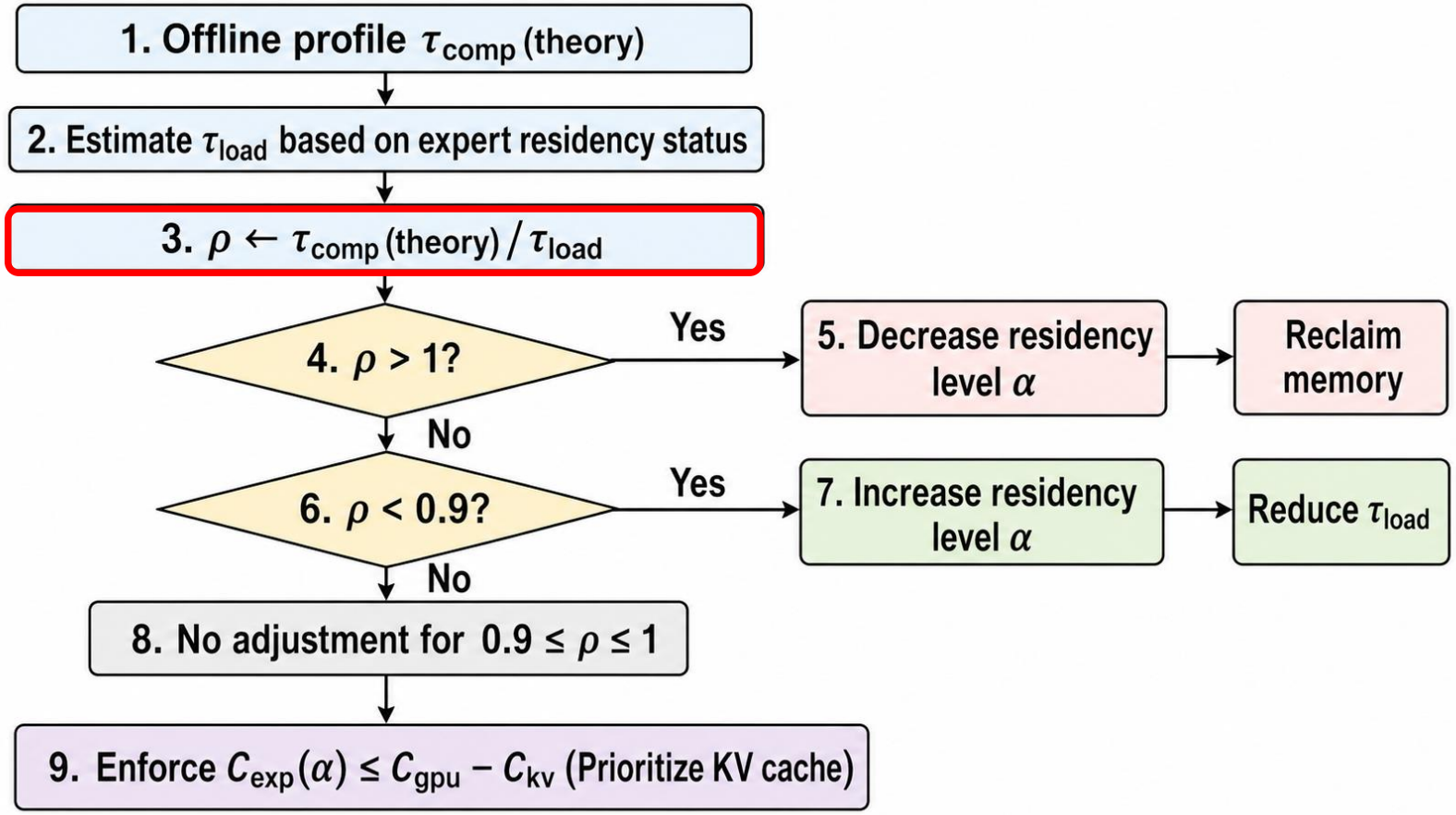
$$\rho = \frac{\tau_{\text{comp(theory)}}}{\tau_{\text{load}}}.$$



Residency Planner

❑ **Adaptively adjust the offloading strategy at runtime**

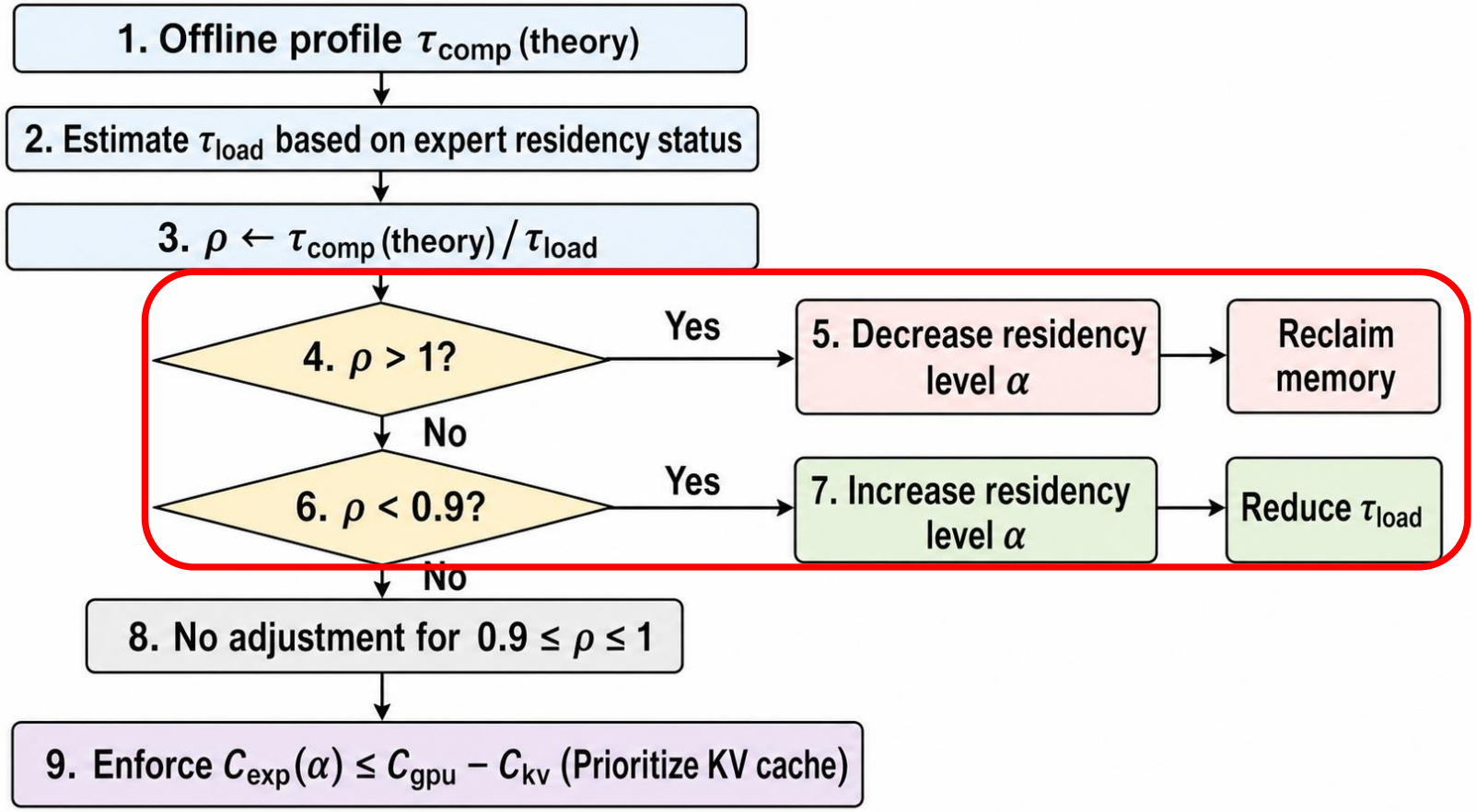
$$\rho = \frac{\tau_{\text{comp(theory)}}}{\tau_{\text{load}}}.$$



Residency Planner

❑ **Adaptively adjust the offloading strategy at runtime**

$$\rho = \frac{\tau_{\text{comp(theory)}}}{\tau_{\text{load}}}.$$



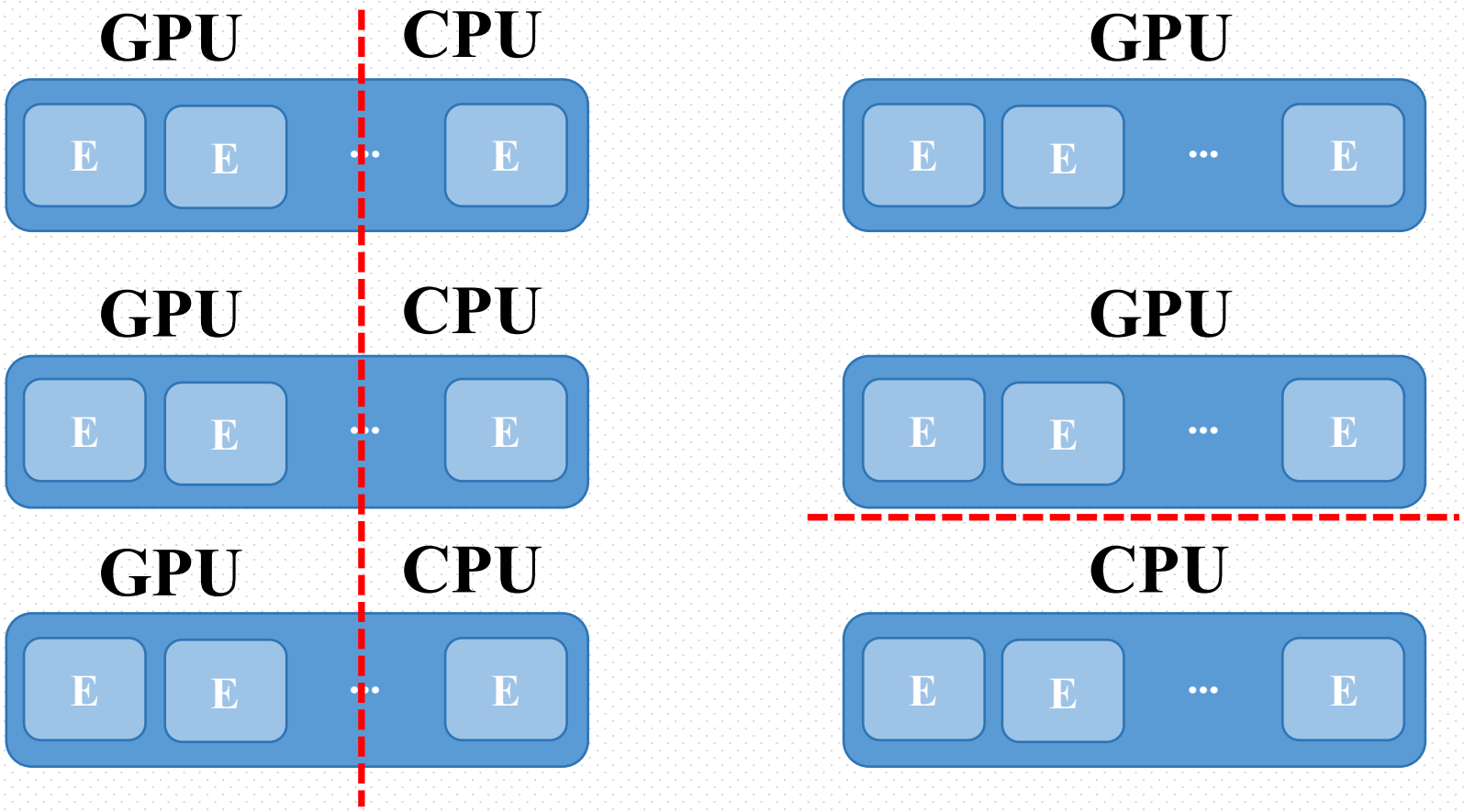
Outline

- Background
- Observation & Challenge
- Design
- **Evaluation**
- Summary

Evaluation

- **Testbed**
  - One node with Four NVIDIA L40 PCIe GPUs, 48 GB of GDDR6 memory
  - 2 TiB of host DRAM, and 3 TB of disk storage
  - Use tensor parallelism to employ MoE model
- **Model**
  - Mixtral-8× 7B-Instruct, 32 layer and 47B
  - Qwen3-Next-80B-A3BInstruct, 48 layer and 80B
- **Baseline**
  - vLLM
  - vLLM-O: offload 12.5% weight to DRAM
  - FluxMoE-H: Layer-wise compress 87.5% of expert weights in HBM, Lack of balanced loading

FluxMoE vs. FluxMoE-H



FluxMoE

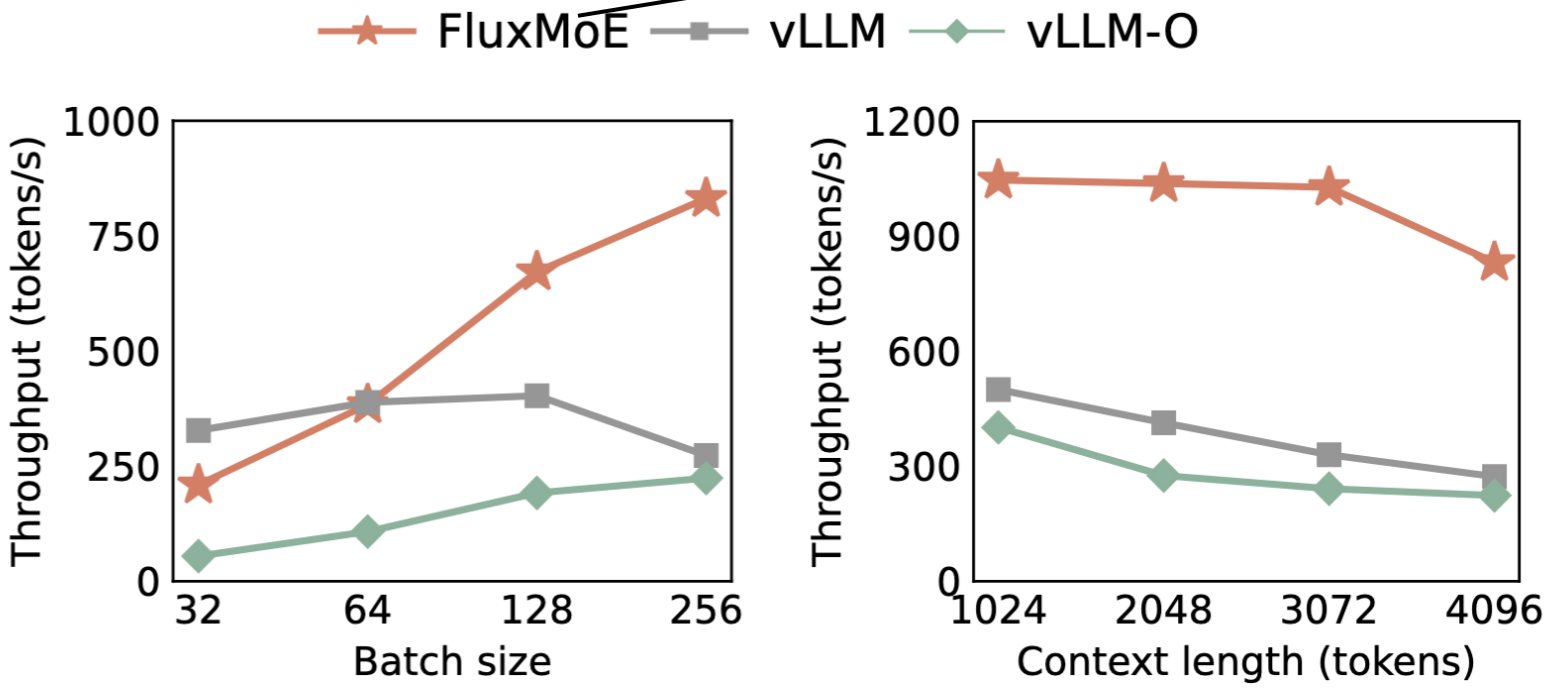
FluxMoE-H

End-to-End Performance

❑Throughput under a performance-bound regime

❖With sufficient memory for the standard serving framework vLLM to run natively without OOM

keep all compressed expert weights in GPU



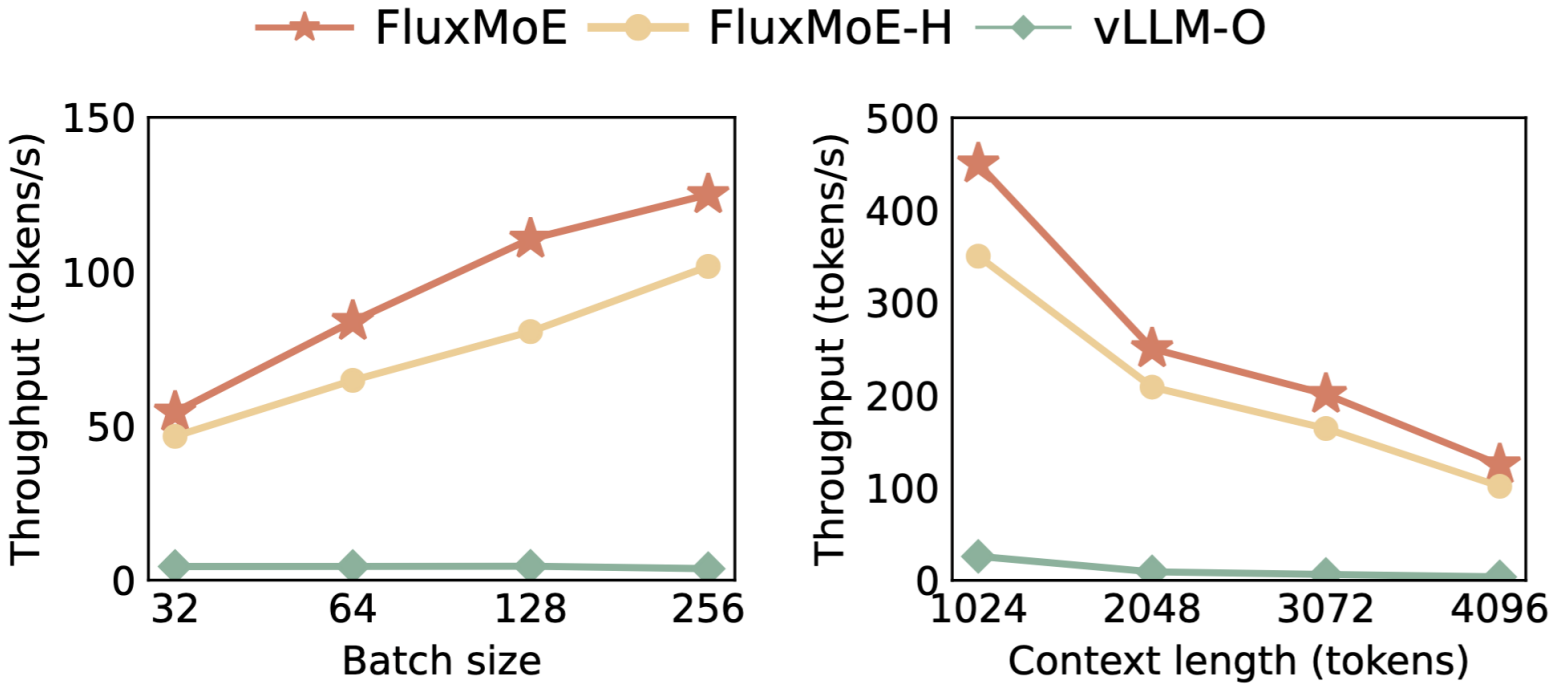
(a) Context length = 4,096 (b) Batch size = 256

Qwen3-Next-80B-A3B-instruct on 4GPU

End-to-End Performance

❑Throughput under a capacity-bound regime

❖A highly resource-constrained setup , forcing the system to rely on expert offloading



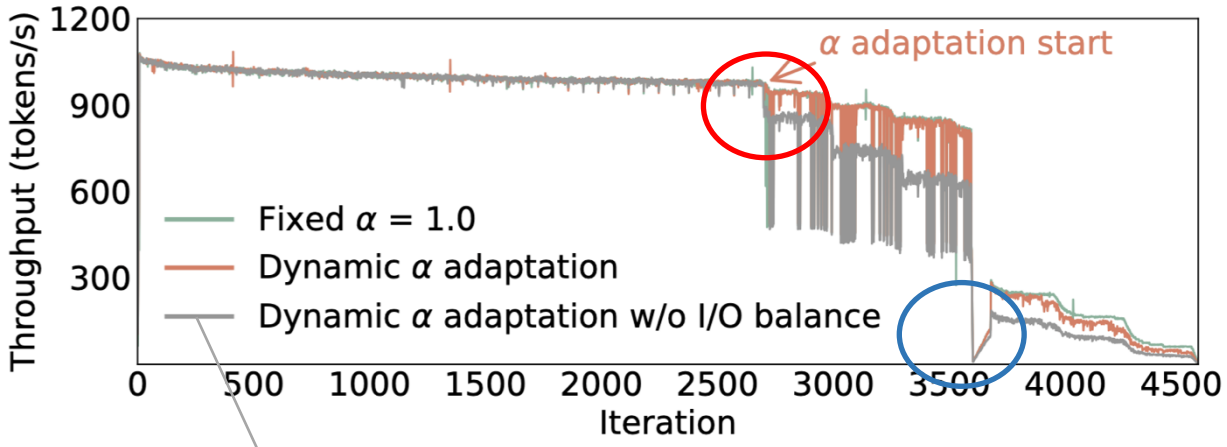
(a) Context length = 4,096 (b) Batch size = 256

Mixtral-8× 7B-instruct on 2GPU

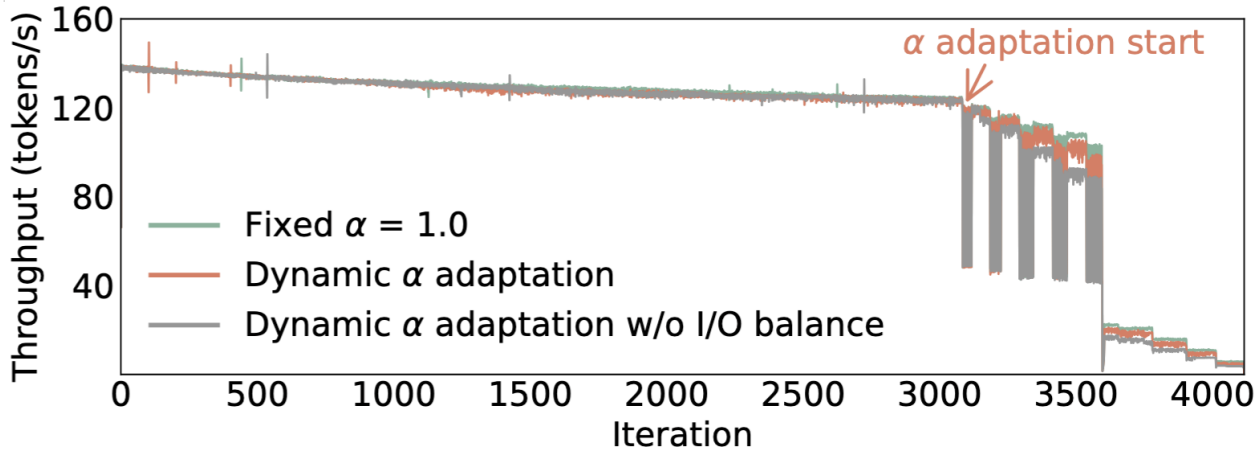
Adaptation of Expert Residency

❑Throughput stability during runtime residency adaptation

❖FluxMoE frees GPU memory without throughput loss



(a) Qwen3-Next-80B-A3B-Instruct [52] (TP=4, BS=256)



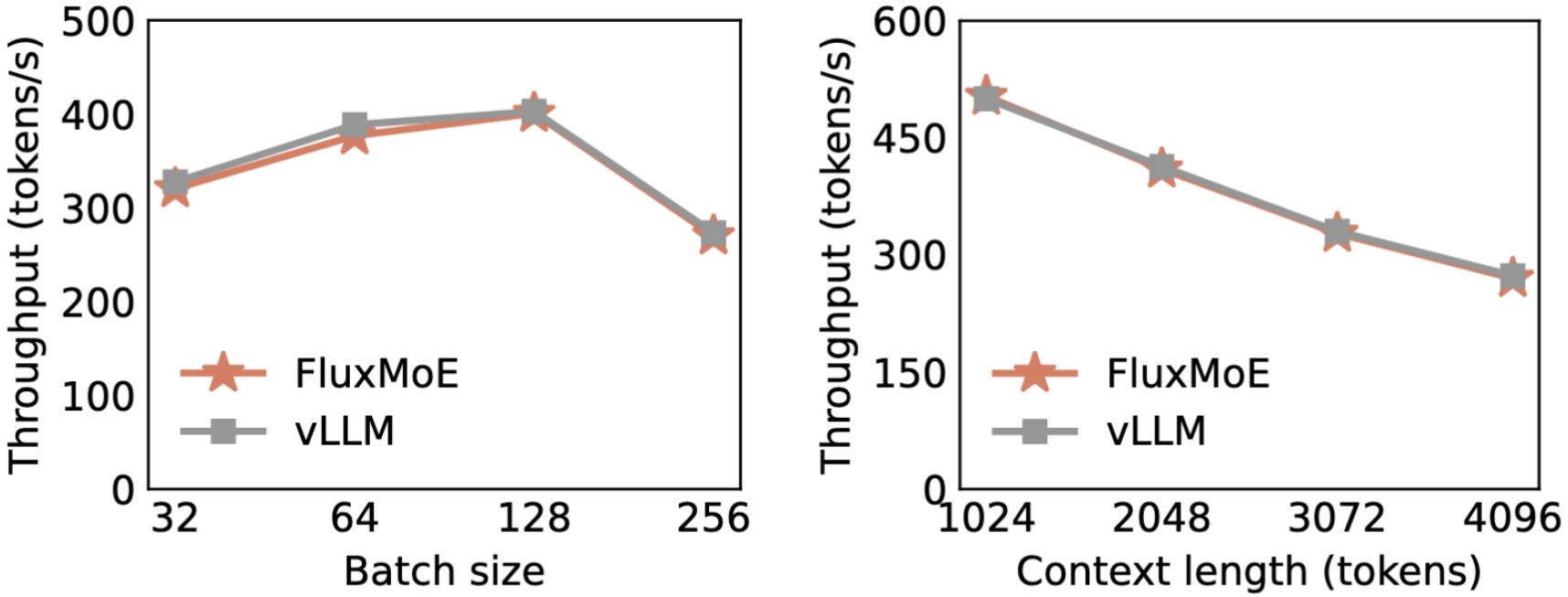
(b) Mixtral-8×7B-Instruct [28] (TP=2, BS=32)

Adjust expert weight offloading within a single layer

Overhead Analysis

❑PagedTensor overhead analysis

❖The result shows only ~3% PagedTensor overhead



(a) Context length = 4,096 (b) Batch size = 256

Qwen3-Next-80B-A3B-instruct on 4GPU

Outline

- Background
- Observation & Challenge
- Design
- Evaluation
- **Conclusion & Disscussion**

Conclusion

- FluxMoE loads experts on demand to free GPU memory for KV cache
- PagedTensor and hierarchy storage enable efficient expert loading
- FluxMoE maintains accuracy while improving throughput in memory-limited settings

Disscussion

- The evaluation is throughput-centric, lacking SLO, TTFT, TPOT
- Offload overhead is unclear, since the analysis only has PagedTensor
- FluxMoE's paging idea may generalize beyond MoE

FluxMoE vs. KTransformers

Core Dimension	KTransformers	FluxMoE
Goal	Local low-concurrency MoE serving	High-throughput MoE serving
Offload Type	expert weight&computation	expert weight
Main Bottleneck	CPU compute and sync	GPU memory and PCIE
Best Fit	Single-user, small batch	Multi-request, long context

FluxMoE: Decoupling Expert Residency for High-Performance MoE Serving

Thanks for listening !

Presenter:Long Zhao