MELINOE: FINE-TUNING ENABLES MEMORY-EFFICIENT INFERENCE FOR MIXTURE-OF-EXPERTS MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) model architectures can significantly reduce the number of activated parameters per token, enabling computationally efficient training and inference. However, their large overall parameter counts have precluded their widespread usage in resource-constrained settings as all parameters must still be loaded into GPU memory. Prior works aim to address this memory bottleneck by offloading certain experts into CPU memory and porting them to GPU memory only when they are activated. In practice, these methods suffer from the significant I/O latency incurred by expert transfer. We present MELINOE, a method that fine-tunes an MoE model to more strongly prefer activating a smaller number of experts per sequence. Caching these preferred experts in GPU memory reduces expert churn and CPU-GPU transfer overhead. MELINOE increases throughput by 1.2–$3\times$ over efficient baselines and up to $14.7\times$ over transfer-heavy baselines while retaining or improving the performance of the model on a downstream task, making it a consistently reliable method for improving MoE inference efficiency.

1 INTRODUCTION

Mixture-of-Experts (MoE) models are a Large Language Model (LLM) architecture that aim to reduce the per-token computational cost of training and inference. MoEs achieve this reduction in compute by fragmenting the traditionally dense Feed Forward Network (FFN) layer in the model architecture into a sparsely gated set of multiple FFNs, referred to as "experts" Shazeer et al. (2017); Lepikhin et al. (2021); Fedus et al. (2022). This reparameterization of the FFN used in transformer-based architectures makes MoE models an efficient alternative to their traditional dense counterparts. MoE models have become the flagship models for open-source model developers Jiang et al. (2024) or have been offered as an alternative in a model suite Muennighoff et al. (2025); Abdin & et al. (2024); Dai et al. (2024). However, these gains do not come for free; while MoEs execute only a fraction of their parameters per token, their total parameter footprint remains large. During decoding, any expert may be routed to at any step, so the corresponding expert's weights must be resident in GPU memory.

The necessary residency of experts in GPU memory prior to activation results in one of two scenarios. First, all experts are simultaneously loaded into GPU memory, which can be prohibitive in resource-constrained settings with limited memory. This regime is also wasteful since only a small subset of experts is used per token, yet many unused expert parameters still occupy GPU memory. An alternative paradigm is where only a small subset of the total experts is kept in GPU memory while the remaining experts are *offloaded* into CPU memory Eliseev & Mazur (2023); Xue et al. (2025); Zhou et al. (2025b). The GPU resident experts constitute an expert *cache*, and experts are fetched from CPU memory on demand when selected by the router. While this reduces peak GPU memory usage, it introduces a new

Table 1: Decoding throughput (tokens/s) on an NVIDIA H100 vs. cache size (fraction of experts resident in GPU VRAM). Throughput drops significantly when fewer experts are resident. Mixtral-8x7B does not fit on a single H100 GPU (80 GB VRAM) in FP16.

Model	Cache 25% Experts	Cache 50% Experts	Cache All Experts
OLMoE	15.68	23.84	37.84
Phi-3.5-MoE	4.43	8.04	19.91
Mixtral-8x7B	1.46	2.32	–

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

bottleneck. Frequent expert swaps incur substantial Peripheral Component Interconnect Express (PCIe) transfer latency and can stall generation when the required expert is not already cached. The resulting churn can dominate end-to-end inference latency, especially when expert routing is diverse across tokens. When GPU memory is especially constrained, these problems are exacerbated as only a limited number of experts can be retained in the GPU-resident expert cache (Table 1).

Prior methods propose a variety of alternative approaches to balancing memory efficiency and I/O latency. Mixtral-Offloading Eliseev & Mazur (2023) offloads most experts to CPU DRAM and keeps a small per-layer GPU-resident expert cache. However, I/O slowdowns still hinder improvements in end-to-end inference latency. Their aggressive mixed-precision quantization also trades model quality for memory efficiency. MoE-Infinity Xue et al. (2025) and FLoE Zhou et al. (2025b) seek to reduce I/O transfers by using sophisticated prefetching techniques to predict which experts will be activated prior to the router selecting them. Yet, their gains depend on prediction accuracy and routing locality, where a subset of experts are activated with high probability for consecutive tokens, so diverse routing could still trigger a significant number of cache misses and evictions. Fiddler Kamahori et al. (2025) reduces PCIe traffic by executing some expert computation on the CPU, but its gains are contingent on CPU capability and diminish as per-expert token counts grow, where CPU execution becomes slow and weight transfers to GPU become preferable. Meaningfully, several prior systems do not directly target reducing the *number of CPU–GPU transfers*, which can be a dominant cost in resource-constrained MoE inference. In other words, prior work largely treats routing as fixed and optimizes around it via prefetching, offloading, or quantization. An alternative perspective is to treat routing as malleable and directly shape expert activation patterns.

In this vein, we present **M**ixture-of-**E**xperts with **L**ightweight **I**nference and **N**etwork **O**ffloading **E**fficiency, MELINOE, a framework that targets the number of CPU-GPU transfers in MoE offloading applications to reduce the amount of I/O overhead. With MELINOE, the model is fine-tuned with an auxiliary loss function that penalizes the model for inducing excessive cache transfers. This fine-tuning procedure occurs prior to deploying the MoE model in a memory-constrained system. Since individual sequences already have slight expert preferences, fine-tuning tilts the model towards using a small subset of experts more heavily, increasing activation concentration, while retaining or improving the performance of the model on a downstream task. MELINOE then employs an MLP-based predictor to forecast these preferred experts prior to decoding, making prefetching more reliable since activated experts are less diverse per sequence. MELINOE can improve throughput measured by tokens/s by 1.2–$3\times$ over efficient baselines and up to $14.7\times$ over transfer-heavy baselines across model architectures and hardware configurations, demonstrating its reliability and robustness. Notably, since the fine-tuning procedure is orthogonal to prior offloading techniques, it composes naturally with other baselines and improves their effectiveness. MELINOE provides a practical path to MoE deployment by fine-tuning the model to reduce I/O latency without sacrificing model quality.

2 Problem Setup and Motivation

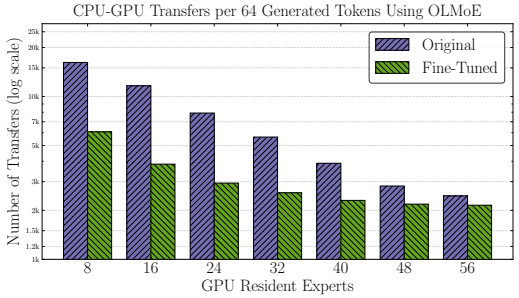
MoE models replace the dense FFN in each transformer block with a sparsely gated set of FFN "experts". For each token in the sequence, a router selects a subset of experts to execute, producing token-level activations, while the remaining experts are inactive. We denote by $\mathbf{E}^{(\ell)}$ the set of experts in the MoE layer $\ell \in [1 \cdots L]$. Let $E := |\mathbf{E}^{(\ell)}| \; \forall \ell \in [1 \cdots L]$. Individual experts in this layer are indexed as $\mathbf{E}_i^{(\ell)}$, where $i \in \{1, \cdots E\}$. The output of the ℓ-th MoE layer for an input $\mathbf{x}_{\text{in}} \in \mathbb{R}^d$ is

$$\mathbf{p}^{(\ell)} \;=\; \text{softmax}\Big(\mathbf{W}_r^{(\ell)}\mathbf{x}_{\text{in}}\Big)\,,\; \mathbf{r}^{(\ell)} \;=\; \text{Top-}K\Big(\mathbf{p}^{(\ell)}\Big)\,,$$

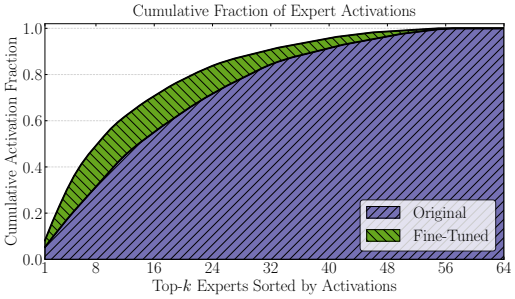
$$\mathbf{y} \;=\; \sum_{\mathbf{r}_i^{(\ell)}=1} \mathbf{p}_i^{(\ell)}\, \mathbf{E}_i^{(\ell)}(\mathbf{x}_{\text{in}}). \tag{1}$$

Here, $\mathbf{W}_r^{(\ell)} \in \mathbb{R}^{E\times d}$ denotes the router weight matrix at layer ℓ. The Top-K operator selects the K experts with the highest routing probabilities, setting the corresponding entries of $\mathbf{r}^{(\ell)}$ to 1 and masking all remaining experts. Thus, $\|\mathbf{r}^{(\ell)}\|_1 = K$. The layer output is then computed by activating only the selected experts, as shown in Equation 1.

The hyperparameter K controls the number of experts activated per token and is model-specific, for example $K = 2$ in Mixtral-8x7B. For hidden dimension d and intermediate dimension d_{ff}, each



(a) Number of Host-to-Device (H2D) and Device-to-Host (D2H) transfers (log scale) when generating 64 tokens using the OLMoE model (original and fine-tuned with auxiliary loss). Fine-tuning can reduce weight transfers by $3.03\times$.



(b) Experts in OLMoE exhibits only *weak* concentration within a layer. Activations are distributed across many experts and the most used experts account for a small fraction of total activations. Fine-tuning increases the concentration of activations.

Figure 1: OLMoE transfer behavior and routing concentration before vs. after fine-tuning.

expert is implemented as an MLP with three linear projections: a gate $\left(\mathbf{W}_{g,i}^{(\ell)} \in \mathbb{R}^{d_{\text{ff}} \times d}\right)$, an up $\left(\mathbf{W}_{u,i}^{(\ell)} \in \mathbb{R}^{d_{\text{ff}} \times d}\right)$, and a down $\left(\mathbf{W}_{d,i}^{(\ell)} \in \mathbb{R}^{d \times d_{\text{ff}}}\right)$ projection. The forward computation is then

$$\mathbf{E}_i^{(\ell)}(\mathbf{x}_{\text{in}}) \;=\; \mathbf{W}_{d,i}^{(\ell)}\Big(\phi\Big(\mathbf{W}_{g,i}^{(\ell)}(\mathbf{x}_{\text{in}})\Big) \odot \mathbf{W}_{u,i}^{(\ell)}(\mathbf{x}_{\text{in}})\Big). \tag{2}$$

where $\odot$ is an element-wise product and ϕ is the gate nonlinearity. Although only a few experts are evaluated per token, the expert parameters dominate the model's memory footprint. In OLMoE, the experts constitute 93% of the weights in the model, and in Mixtral-8x7B, the experts make up 96%. This discordance between expert activations and their memory footprint motivates offloading.

Expert Offloading Systems. Prior works mitigate MoE model's memory requirements by storing most expert weights in CPU DRAM and keeping only a small subset resident in GPU VRAM as an expert cache. During decoding, when the router selects an expert that is not currently resident, the system must fetch that expert's weights from DRAM and transfer them over the CPU–GPU interconnect, typically PCIe, into VRAM before the expert can be executed on the GPU. End-to-end decoding time therefore includes both GPU computation and transfer-induced stalls

$$\text{Time}_{\text{decode}} \;\approx\; \text{Time}_{\text{compute}} \;+\; N_{\text{miss}} \;\cdot\; \text{Time}_{\text{transfer}} \tag{3}$$

where N_{miss} refers to the number of cache misses where the desired expert is not currently stored in GPU VRAM. Since expert weights dominate the model's parameter count, even modest cache budgets can lead to substantial transfer overhead. In many cases, N_{miss} can reach tens of thousands even over short generations (Figure 1a). During pretraining, load balancing objectives encourage broad expert utilization, a beneficial goal in batched, multi-GPU settings to ensure experts on different devices take roughly the same amount of time to execute. However, such objectives increase the number of distinct experts touched during generation, escalating the number of cache transfers.

Expert Specialization. While standard load balancing objectives spread the probability mass of expert activations across many experts, empirical evidence suggests that MoE routing is not arbitrary. Within a given sequence, the router typically exhibits a consistent but weak preference for a subset of experts, indicating a degree of specialization Liang et al. (2025); Jaiswal et al. (2025); Wang et al. (2025). This suggests that some experts contribute disproportionately to a sequence's computation for a given prompt, even if the router still routes a nontrivial fraction of tokens elsewhere. However, in pretrained models this preference is too marginal to yield large gains from naïve caching. Even if we keep the "most preferred" experts in GPU VRAM, they may account for only a modest fraction of the total activations within a sequence, so cache hit rates remain limited. For example, with OLMoE we observe that, on average, the top 8 experts by activation within a sequence account for only $\sim 31\%$ of expert activations (Figure 1b). Even though this is still greater than the expected activation rate of 12.5% from all experts being activated equally, it still leaves substantial routing mass on experts that have been offloaded into CPU DRAM. Thus, the practical value of these preferred experts is paltry.

Fine-Tuning to Achieve Cache-Friendliness. Nevertheless, the observation of preferred experts motivates an important question: to what extent can we increase the concentration of expert usage

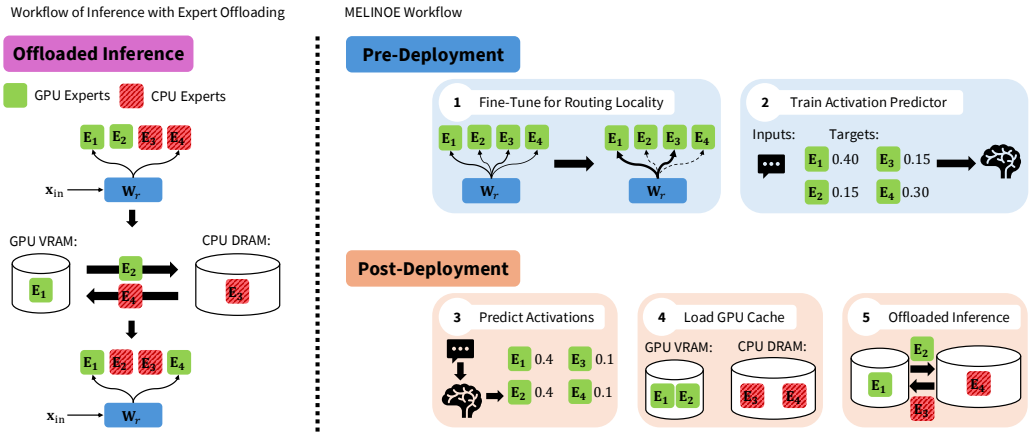


Figure 2: **Overview of MELINOE.** Pre-deployment: fine-tune the model for per-sequence routing locality. Post-deployment: preload a GPU-resident cache and run offloaded inference.

within a sequence to improve expert-cache hit rates? Specifically, since the model already has a set of preferred experts per sequence, tilting the model's routing decisions towards these experts could amplify its natural per-sequence preferences. By tilting routing towards a more consistent set of experts within each sequence, we increase routing locality and make VRAM caching substantially more effective by reducing cache churn. A few considerations are critical for making this approach work in practice. First, we must design an auxiliary objective that encourages the router to reuse a smaller set of experts within a sequence, increasing routing locality and reducing the number of transfers. At the same time, fine-tuning must preserve the base model's quality. A key failure mode that would impair the model's expressiveness is router collapse where the same subset of experts become the preferred experts globally. Here, the remaining experts would not contribute to the model's predictions, degrading the fine-tuned model's integrity. The desired behavior is sequence-specific skew, where expert preferences should be persistent over the course of decoding for a given input but should vary across inputs. Finally, once such per-sequence structure is induced, we need a reliable way to predict the preferred experts before decoding begins. Accurate prediction enables proactive caching of these experts in GPU memory, making the cache substantially more effective because the same experts are likely to remain useful and active throughout generation.

3 METHOD

We present MELINOE, a practical expert-offloading recipe that reduces end-to-end inference latency. We structure the method into two stages, namely pre- and post-deployment, because the resources available to a model provider differ fundamentally before and after releasing a model to a memory-constrained device. In the pre-deployment stage, the provider can access the full-precision MoE weights as well as fine-tuning data. We leverage this setting to *(i)* fine-tune the MoE with an auxiliary objective that increases per-sequence routing locality (Section 3.1.1) and *(ii)* train an activation predictor that anticipates which experts will be most useful for a given input (Section 3.1.2). In the post-deployment stage, the model is executed under tight VRAM budgets. Here, we use the trained predictor to proactively pre-load a GPU-resident expert cache before decoding begins for every sequence. Once these experts have been prefetched, we perform standard offloaded inference with substantially fewer transfers during generation (Section 3.2).

3.1 PRE-DEPLOYMENT STAGE

3.1.1 MOE FINE-TUNING PROCEDURE

MELINOE hinges on the following auxiliary fine-tuning objectives that we use to reshape routing behavior. The following proposed objectives concentrate routing mass on a small set of experts while preserving the router's original specialization patterns to avoid degrading model quality.

Cache Simulation Loss $\mathcal{L}_{cs}$. The first auxiliary objective is a cache simulation loss that directly penalizes routing patterns likely to induce expert transfers under a memory budget. We denote the

router distribution for token $t \in [1, \ldots, T]$ at layer $\ell \in [1, \ldots, L]$ as $\mathbf{p}^{(\ell,t)} \in \mathbb{R}^E$. Since MoE inference routes each token to exactly K experts, we define a binary request vector $\mathbf{r}^{(\ell,t)} \in [0,1]^E$ by selecting the Top-K entries of $\mathbf{p}^{(\ell,t)}$. We additionally define a soft cache state for token t at layer ℓ as $\mathbf{c}^{(\ell,t)} \in \mathbb{R}^E_{\geq 0}$ and initialize each $\mathbf{c}^{(\ell,0)}$ as $\mathbf{0}$. We update this cache state using an exponentially-decayed history of past requests to approximate a recency-weighted cache. Let $\gamma \in [0,1]$ be a decay factor and C be the cache capacity. We first compute an un-normalized update according to the rule

$$\mathbf{c}^{(\ell,t+1)} \;=\; \gamma \mathbf{c}^{(\ell,t)} + \mathbf{r}^{(\ell,t)}$$

During an initial "cache fill" phase, we allow $\|\mathbf{c}^{(\ell,t)}\|_1$ to grow until it reaches C. After this point, we normalize $\mathbf{c}^{(\ell,t)}$ for subsequent tokens so that $\|\mathbf{c}^{(\ell,t)}\|_1 = C$ is preserved at each step

$$\mathbf{c}^{(\ell,t+1)} \;=\; \frac{\gamma Z^{(t)} \mathbf{c}^{(\ell,t)} + \mathbf{r}^{(\ell,t)}}{Z^{(t+1)}},\; Z^{(t+1)} \;=\; \gamma Z^{(t)} + \frac{K}{C}$$

where $Z^{(t)}$ is a scalar normalizer that maintains $\|\mathbf{c}^{(\ell,t)}\|_1 = C$ and $Z^{(1)}$ is initialized to 1. Alternatively, one can also initialize using a uniform vector with $\|\mathbf{c}^{(\ell,1)}\|_1 = C$ to avoid the initial cache fill phase. We set $\gamma = 0.9$, but provide ablation studies for the impact of γ and C on throughput in Appendix D.6 and D.7. As γ controls how long previous routing decisions persist in the cache, smaller values such as $\gamma = 0$ make the cache more reactive and closer to a Least Recently Used (LRU) cache. In contrast, larger values of γ such as $\gamma = 1$ allow previous routing decisions to remain relevant over long horizons similar to a Least Frequently Used (LFU) cache. The cache simulation loss is then

$$\mathcal{L}_{cs} \;=\; \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} \underbrace{\sum_{i=1}^{E} \mathbf{r}_i^{(\ell,t)} \left(1 - \mathbf{c}_i^{(\ell,t)}\right)}_{\text{cache miss proxy at token } t, \text{ layer } \ell} \tag{4}$$

since a value of 1 for $\mathbf{r}_i^{(\ell,t)}$ means the router has selected expert $\mathbf{E}_i^{(\ell)}$ while a higher value of $1 - \mathbf{c}_i^{(\ell,t)}$ means that it is less likely that $\mathbf{E}_i^{(\ell)}$ is already in the GPU-resident expert cache.

Rank Matching Loss $\mathcal{L}_{rm}$. The second auxiliary objective is a rank matching loss that aligns the relative routing preferences of the fine-tuned model with the preferences of the original base model. We change the notation slightly from the previous paragraph and refer to the base model's router distribution for token t at layer ℓ as $\mathbf{p}_b^{(\ell,t)}$ and the fine-tuned model's router distribution as $\mathbf{p}_f^{(\ell,t)}$. We count "mistakes" in the fine-tuned model's ordering relative to the base model's preferences as

$$m^{(\ell,t)} = \sum_{i,j \in [E]} \mathbb{I}\Big\{\mathbf{p}_{b,i}^{(\ell,t)} > \mathbf{p}_{b,j}^{(\ell,t)}\Big\} \Big[\rho - \Big(\mathbf{p}_{f,i}^{(\ell,t)} - \mathbf{p}_{f,j}^{(\ell,t)}\Big)\Big]_+$$

where the operation $[a]_+ := \max(0, a)$. In simple terms, if the base model is more likely to route to an expert $\mathbf{E}_i^{(\ell)}$ over $\mathbf{E}_j^{(\ell)}$, then the fine-tuned model should also prefer $\mathbf{E}_i^{(\ell)}$ over $\mathbf{E}_j^{(\ell)}$ up to some margin ρ. This way, the amount that the fine-tuned model prefers $\mathbf{E}_i^{(\ell)}$ over $\mathbf{E}_j^{(\ell)}$ is nominally factored into the loss, but the underlying ordering still matters. The rank matching loss is calculated as

$$\mathcal{L}_{rm} \;=\; \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} m^{(\ell,t)} \tag{5}$$

From this, we have the full loss calculation for fine-tuning

$$\mathcal{L} \;=\; \mathcal{L}_{nll} \;+\; \lambda_{cs} \mathcal{L}_{cs} \;+\; \lambda_{rm} \mathcal{L}_{rm} \tag{6}$$

where $\mathcal{L}_{nll}$ is the Negative Log-Likelihood loss used in standard language modeling applications. These loss functions jointly reshape routing in a way that is aligned with memory-constrained inference. $\mathcal{L}_{cs}$ promotes within-sequence routing locality by directly penalizing excessive cache transfers, while $\mathcal{L}_{rm}$ prevents router collapse by encouraging the fine-tuned router to preserve the base model's relative expert preferences. To highlight the utility of these specific loss functions, we provide theoretical justifications for both loss functions in Appendix C. As we aim for fine-tuning to be memory-efficient, we update only the the router weights and gate projection and apply low-rank adaptation (LoRA) Hu et al. (2022) to the MLP up and down projections.

3.1.2 EXPERT ACTIVATION PREDICTOR

In prior offloading systems, expert activation predictors often provide only marginal benefit because routing is noisy, so prefetched experts are evicted before they can be reused. In MELINOE, fine-tuning induces stronger prompt-conditioned structure in routing, as activations are more consistent per prompt. Motivated by contextual sparsity Liu et al. (2023); Hou et al. (2025), we learn a prompt-conditioned predictor of expert preferences, Ψ. The activation predictor is trained on a dataset generated as follows. For each prompt $\mathbf{q}$, we compute a fixed-dimensional representation using an embedding model Ψ_{EMB} such that $\Psi_{\text{EMB}}(\mathbf{q}) \in \mathbb{R}^{d_{\text{EMB}}}$. To construct targets for the activation predictor, we generate a response from the MoE and record router probabilities, $\mathbf{p}^{(\ell,t)} \in \mathbb{R}^E$, for each token and layer. The supervised target for the predictor is the per-layer average router probability vector

$$y^{(\ell)}(\mathbf{q}) \;=\; \frac{1}{T}\sum_{t=1}^{T}\mathbf{p}^{(\ell,t)},\; Y(\mathbf{q}) = \Big[y^{(1)}(\mathbf{q})\cdots y^{(L)}(\mathbf{q})\Big]$$

where $Y(\mathbf{q}) \in \mathbb{R}^{L\times E}$. This yields a dataset of pairs $\{\Psi_{\text{EMB}}(\mathbf{q}_n), Y(\mathbf{q}_n)\}_{n=1}^{N}$. We train a lightweight two-layer MLP $\Psi_{\text{MLP}} : \mathbb{R}^{d_{\text{EMB}}} \to \mathbb{R}^{L\times E}$ to predict $\widehat{Y}(\mathbf{q})$ whose rows $[\widehat{Y}(\mathbf{q})]_\ell$ estimate the layerwise expert preference scores. We normalize the ground truth targets and train Ψ_{MLP} by minimizing the KL divergence between the normalized target distribution $[Y(\mathbf{q})]_\ell$ and the predicted distribution by applying a row-wise softmax to $[\widehat{Y}(\mathbf{q})]_\ell$. Because MELINOE reduces routing noise by amplifying prompt-specific expert preferences, even a relatively small Ψ_{EMB} suffices to produce a lightweight predictor of the model's preferred experts at each layer. Thus, we enable accurate prefetching, with minimal prefetching latency, and reduce CPU-GPU transfers in the process.

3.2 POST-DEPLOYMENT STAGE

In the post-deployment stage, the fine-tuned MoE and the activation predictor are executed on a memory-constrained device. Given a prompt $\mathbf{q}$, we predict per-layer expert activations by calculating

$$\Psi_{\text{MLP}}(\Psi_{\text{EMB}}(\mathbf{q})) = \widehat{Y}(\mathbf{q}) \tag{7}$$

For each layer $\ell \in [1,\ldots,L]$, we form a prefetch set $c^{(\ell,1)} = \text{Top-}C([\widehat{Y}(\mathbf{q})]_\ell)$ where C is the cache capacity. We proactively load the corresponding experts into the GPU-resident cache before generation begins. To increase effective cache capacity, all expert weights are maintained in `HQQ INT4`, allowing more experts to remain resident in limited GPU memory. Experts that are not resident in GPU VRAM remain in CPU DRAM and are stored in pinned memory to accelerate transfers from CPU to GPU. Transfers are additionally non-blocking, which ensures asynchronous transfers of experts and minimizes PCIe overhead. These offloaded experts are fetched on demand when selected by the router, evicting cached experts as dictated by the cache policy. Overall, MELINOE localizes routing through fine-tuning, predicts the resulting per-sequence locality, and exploits it via proactive prefetching and quantized cache residency to reduce transfer-induced stalls during generation. In the following section, we demonstrate the effectiveness of these various design elements.

4 RESULTS

4.1 EXPERIMENTAL SETUP

Models and Datasets. We evaluate MELINOE on three MoE backbones that span size and granularity: **OLMoE** (7B params.), **Phi-3.5-MoE** (42B params.), and **Mixtral-8x7B** (47B params.). We train the activation predictor using fixed-dimensional representations from **BGE-Base-EN-v1.5** (109M params./dim.= 768) Xiao et al. (2024). We fine-tune and evaluate across four complementary workloads. We fine-tune on **Dolly15K** Conover et al. (2023), **GSM8K** Cobbe et al. (2021), **WikiText-103** Merity et al. (2016), and **Sciq** Welbl et al. (2017) and benchmark on held-out evaluation splits. Some results on WikiText-103 and SciQ are deferred to Appendix D.1.

Fine-Tuning Hyperparameters. Across all models, MELINOE updates the router weights and gate projection layers and applies LoRA rank $r = 32$ to the MLP up and down projections. Remaining weights are fixed at their pretrained initialization. For the cache-simulation loss $\mathcal{L}_{cs}$, we adopt a deliberately restrictive cache capacity to emphasize the intended deployment regime, setting the

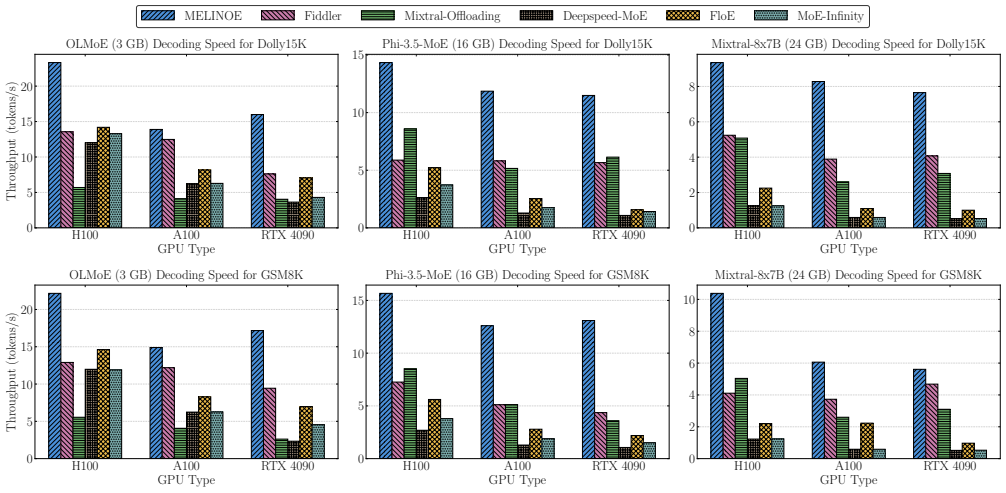


Figure 3: Throughput comparison of MELINOE against baselines across model/dataset/GPU settings.

Table 2: Downstream output quality across baselines. ROUGE-L is reported on Dolly15K. Accuracy is reported on GSM8K.

	Dataset: Dolly15K (ROUGE-L)			**Dataset: GSM8K (Accuracy %)**		
Method	**OLMoE**	**Phi-3.5-MoE**	**Mixtral-8x7B**	**OLMoE**	**Phi-3.5-MoE**	**Mixtral-8x7B**
Base Model	0.1851	0.2067	0.2159	79.21	55.45	77.23
MELINOE	**0.2486**	**0.2270**	**0.2361**	**80.20**	**63.37**	**79.21**
Fiddler	0.1851	0.2067	0.2159	79.21	55.45	77.23
Mixtral-Offloading	0.1734	0.2025	0.2086	72.28	51.49	61.39
DeepSpeed-MoE	0.1851	0.2067	0.2159	79.21	55.45	77.23
FLoE	0.1775	0.1884	0.2212	63.34	53.47	60.40
MoE-Infinity	0.1851	0.2067	0.2159	79.21	55.45	77.23

simulated cache budget to $C = \frac{E}{4}$ (e.g. $C = 16$ for OLMoE). We fix the cache decay parameter to $\gamma = 0.9$ and the rank matching margin to $\rho = 0.1$ throughout. A holistic overview of optimizer settings and fine-tuning hyperparameters are provided in Appendix B.2 for the sake of brevity.

Hardware and Inference Configuration. We validate performance gains across multiple GPU types: **H100 (80GB VRAM)**, **A100 (40GB VRAM)**, and **RTX 4090 (24GB VRAM)**. To ensure that comparisons reflect realistic memory-constrained deployments, even on larger-memory accelerators, we artificially cap per-process GPU memory using PyTorch limits. We allocate 3GB for OLMoE, 16GB for Phi-3.5-MoE, and 24GB for Mixtral-8x7B. The expert cache uses an LFU eviction policy. Sensitivity to cache size and alternative budgets is deferred to Appendix D.5.

4.2 Main Results

We validate the performance of MELINOE against five prior baselines: Fiddler Kamahori et al. (2025), Mixtral-Offloading Eliseev & Mazur (2023), Deepspeed-MoE Rajbhandari et al. (2022), FLoE Zhou et al. (2025b), and MoE-Infinity Xue et al. (2025). Across every configuration, MELINOE significantly improves throughput relative to the baseline methods (Figure 3). Using OLMoE, MELINOE achieves 22.16-23.32 tokens/s on the H100 setup and 15.99-17.17 tokens/s on the RTX 4090 setup. In comparison, the best competing baseline achieves only 14.19-14.62 tokens/s on the H100 setup (FLoE) and 7.61-9.44 tokens/s on the RTX 4090 setup (Fiddler). These improvements persist with the larger architectures. Phi-3.5-MoE reaches 14.34-15.67 tokens/s on the A100 setup whereas the next closest baselines (Fiddler and Mixtral-Offloading) only attain 5.11-5.81 tokens/s. Ultimately, MELINOE improves over the best competing baseline by 1.2-3$\times$ and yields substantially larger gains against baselines that incur frequent weight transfers, for instance a 14.7$\times$ throughput gain over Deepspeed-MoE when using Mixtral-8x7B on the RTX 4090 setup.

Table 3: Impact of fine-tuning and prefetching (64 output tokens). Each entry reports throughput (tokens/s) with average transfers per layer in parentheses (Tx/L).

Setting	Dataset: Dolly15K		Dataset: GSM8K	
	OLMoE ($C = 16$)	**Mixtral-8x7B** ($C = 5$)	**OLMoE** ($C = 16$)	**Mixtral-8x7B** ($C = 5$)
Base Model	15.15 (727)	3.37 (105)	15.10 (771)	3.59 (100)
Fine-Tuned Model	23.35 (240)	8.36 (48)	24.88 (255)	7.82 (52)
Fine-Tuned Model + Prefetch	**24.10 (229)**	**8.98 (46)**	**25.00 (250)**	**8.60 (48)**

Crucially, MELINOE's throughput gains do not come at the expense of task performance (Table 2). Note that the Fiddler, DeepSpeed-MoE, and MoE-Infinity baselines do not change the base model weights and therefore have the same performance. On Dolly15K, MELINOE achieves the best ROUGE-L with OLMoE (0.2486), Phi-3.5-MoE (0.2270), and Mixtral-8x7B (0.2361). On GSM8K, MELINOE again attains the highest accuracy, with the most pronounced improvement occurring with Mixtral-8x7B where MELINOE outperforms the closest baseline by 4.35%. Tilting routing towards more persistent per-sequence experts can maintain or even improve quality on the downstream task.

4.3 Ablation Studies

Relative Impact of Fine-Tuning vs. Prefetching. We aim to disambiguate the influence of fine-tuning and prefetching on MELINOE. Table 3 quantifies the individual contributions of these disparate components for OLMoE and Mixtral-8x7B on Dolly15K and GSM8K. Across both models and datasets, we conclude that the primary factor in throughput improvements is fine-tuning, which can substantially reduce the number of CPU-GPU transfers. In OLMoE, fine-tuning the model can result in $3\times$ fewer transfers relative to the base model. For Mixtral-8x7B, this reduction in transfer counts is especially valuable as transfers are slow in coarse-grained MoE architectures where experts themselves are larger. Prefetching provides supplementary benefits, but the improvements are marginal relative to the gains achieved through fine-tuning. Nonetheless, prefetching itself takes roughly 0.05 seconds but can still reduce end-to-end latency by up to 0.8 seconds.

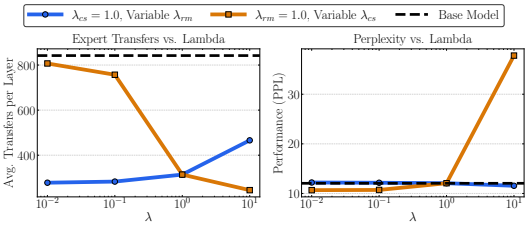


Figure 4: Impact of varying λ_{cs} and λ_{rm} on number of expert transfers and model performance (OLMoE, 64 output tokens).

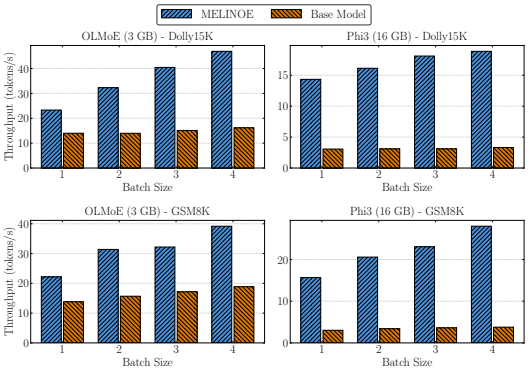


Figure 5: Throughput of MELINOE at various batch sizes with limited GPU VRAM.

Contribution of Loss Functions on Performance. Our fine-tuning objective uses two weighting coefficients, λ_{cs} and λ_{rm}, to control the relative influence of the auxiliary losses. Figure 4 exhibits the significance of these terms on the resulting fine-tuned model's behavior. When holding $\lambda_{cs} = 1.0$ and varying λ_{rm}, perplexity is largely stable for moderate values $\lambda_{rm} \in \{0.01, 0.1, 1.0\}$ while transfers rise slightly from 278 to 314 transfers per layer. When holding $\lambda_{rm} = 1.0$ and varying λ_{cs}, increasing λ_{cs} predictably reduces transfers, confirming that the transfer penalty stabilizes expert residency. However, pushing λ_{cs} too high harms the fine-tuned model's quality, indicating that aggressively minimizing transfers can over-constrain routing. Figure 4 highlights that moderate λ_{cs} and λ_{rm} is an effective balancing point between the objectives.

Effect of Batch Size. We study the impact of batched decoding in Figure 5. We do not include prior offloading baselines as they do not specifically test regimes with batch size > 1. For MELINOE, the activation predictor pools the most likely experts across all sequences in the batch, while all other aspects of the caching policy remain identical to the batch size $= 1$ setting. MELINOE yields

significant throughput gains over the base model and maintains increasing throughput as batch size grows. While speedups modestly diminish at larger batch sizes, fine-tuning to amplify prompt-specific expert preferences remains valuable for memory-constrained MoE inference in batched settings.

Effect of Output Generation Length. A key observation is that fine-tuning does not over-constrain the model and lead to larger performance drops over longer generations. In Table 4, we present the performance of the fine-tuned model at different generation lengths for all three model architectures. Across various output token lengths, the fine-tuned model remains performant. This suggests that MELINOE preserves output quality as generation length increases, rather than trade long-horizon stability for short-context gains. This supports the view that the cache simulation loss remains viable for variable-length generations.

Table 4: Fine-tuned model perplexity across generation lengths. Quality does not degrade with long output horizons.

Output Length	OLMoE	Phi-3.5-MoE	Mixtral-8x7B
64 Tokens	13.11	5.56	6.12
128 Tokens	10.42	4.81	4.80
256 Tokens	9.50	4.18	4.14
512 Tokens	10.56	3.95	3.92
1024 Tokens	9.86	4.38	4.29

Coupling Fine-Tuning with Previous Baselines. Since the fine-tuning procedure in MELINOE is not contingent on specific prefetching, quantization, or sparsity schemes, the fine-tuned model checkpoint can be used as a stand-in for the base model when using prior baselines. Table 5 shows that the fine-tuned model checkpoint can improve the throughput of cache-based methods like FLoE and Mixtral-Offloading. Using the same VRAM restrictions as Section 4.2, swapping the base MoE model for the fine-tuned version can yield improvements of up to 5.49 tokens/s. Therefore, the fine-tuning procedure proposed in MELINOE can be used to augment offloading baselines.

Table 5: Impact of fine-tuning on prior baselines. Each entry reports throughput (tokens/s).

	Dataset: Dolly15K		Dataset: GSM8K	
Method	**OLMoE**	**Phi-3.5-MoE**	**OLMoE**	**Phi-3.5-MoE**
FLoE	14.19	5.23	14.62	5.61
+ Fine-Tuning	20.11	6.28	20.24	9.56
Mixtral-Offloading	5.70	8.58	5.55	8.52
+ Fine-Tuning	9.22	8.96	9.07	8.91

Table 6: WikiText-103 perplexity (PPL) for the base and fine-tuned models. Fine-tuning on task-specific datasets does not meaningfully degrade model performance.

Model	OLMoE	Phi-3.5-MoE	Mixtral-8x7B
Base Model	9.02	4.61	3.87
Fine-Tuned/Dolly15K	8.93	4.25	3.92
Fine-Tuned/GSM8K	9.50	4.64	3.96

Maintaining General-Purpose Language Modeling Capabilities. We show that the fine-tuned models retain their general-purpose language modeling capabilities. Table 6 reports WikiText-103 perplexity for the base and fine-tuned models, demonstrating that fine-tuning on task-specific datasets does not meaningfully degrade general-purpose language modeling. Furthermore, models fine-tuned directly on WikiText-103 preserve the base model's perplexity while still achieving comparable throughput improvements to those observed in our prior experiments (see Appendix D.1).

5 Conclusion

MELINOE fine-tunes an MoE model to prefer a small subset of experts on a per-sequence basis. With these stronger and more consistent expert preferences, caching the preferred experts in GPU memory yields substantially fewer expert transfers during decoding, improving throughput without sacrificing output quality on downstream tasks. We identify three areas for future work. First, while we evaluate the method's generalization in Section 4.3 and Appendix D.1, future work should explore whether this approach is valuable for deploying MoE models across diverse tasks. Second, layer-wise cache budgets may provide increased flexibility. Finally, for very long generations, dynamically adapting cache sizes over time may be beneficial. Overall, we view MELINOE as a step towards deployment-ready MoE models that retain strong quality while operating under real hardware constraints.

AI USE STATEMENT

We used generative AI tools to polish and compress manuscript text to meet the page limit. We also used AI agents to assist with writing code and launching experiments on GPU/compute clusters. Generative AI was not used for the ideation or formulation of the research problem. The authors reviewed the AI-assisted text, references, code, and experimental outputs and take responsibility for the final content and reported results.

ETHICS STATEMENT

This work studies mechanisms to improve resource-constrained and memory-efficient inference for MoE models. We do not identify a new application-specific ethical risk from the fine-tuning method beyond what is already posed by existing models/methods. The authors have read the ICLR code of ethics and believe that the paper complies with it.

REPRODUCIBILITY STATEMENT

We provide the training hyperparameters in the Appendix and the anonymized code in a ZIP file submitted as supplementary material. We plan to release all assets including the code upon publication.

REFERENCES

Marah Abdin and et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023.

Damai Dai, Chengqi Deng, Chenggang Zhao, R.x. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y.k. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70/.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao,

Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Zachary Doucet, Rishi Sharma, Martijn de Vos, Rafael Pires, Anne-Marie Kermarrec, and Oana Balmau. Harmoeny: Efficient multi-gpu inference of moe models, 2025. URL https://arxiv.org/abs/2506.12417.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/du22c.html.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading, 2023. URL https://arxiv.org/abs/2312.17238.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), January 2022. ISSN 1532-4435.

Bairu Hou, Qibin Chen, Jianyu Wang, Guoli Yin, Chong Wang, Nan Du, Ruoming Pang, Shiyu Chang, and Tao Lei. Instruction-following pruning for large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=juARG7yu4P.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Ajay Jaiswal, Jianyu Wang, Yixiao Li, Pingzhi Li, Tianlong Chen, Zhangyang Wang, Chong Wang, Ruoming Pang, and Xianzhi Du. Finding fantastic experts in moes: A unified study for expert dropping strategies and observations, 2025. URL https://arxiv.org/abs/2504.05586.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=N5fVv6PZGz.

Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/lewis21a.html.

Jingcong Liang, Siyuan Wang, Miren Tian, Yitong Li, Duyu Tang, and Zhongyu Wei. Not all models suit expert offloading: On local routing consistency of mixture-of-expert models, 2025. URL https://arxiv.org/abs/2505.16056.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. Deja vu: contextual sparsity for efficient llms at inference time. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoe: Open mixture-of-experts language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xXTkbTBmqq.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, {Reza Yazdani} Aminabadi, {Ammar Ahmad} Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. *Proceedings of Machine Learning Research*, 162:18332–18346, 2022. ISSN 2640-3498. Publisher Copyright: Copyright © 2022 by the author(s); 39th International Conference on Machine Learning, ICML 2022 ; Conference date: 17-07-2022 Through 23-07-2022.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Chenyang Song, Weilin Zhao, Xu Han, Chaojun Xiao, Yingfa Chen, Yuxuan Li, Zhiyuan Liu, and Maosong Sun. Blockffn: Towards end-side acceleration-friendly mixture-of-experts with chunk-level activation sparsity, 2025. URL https://arxiv.org/abs/2507.08771.

The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm, March 2024. URL https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm. Databricks Blog (Mosaic Research).

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts, 2024. URL https://arxiv.org/abs/2408.15664.

Yun Wang, Lingyun Yang, Senhao Yu, Yixiao Wang, Ruixing Li, Zhixiang Wei, James Yen, and Zhengwei Qi. Buddymoe: Exploiting expert redundancy to accelerate memory-constrained mixture-of-experts inference, 2025. URL https://arxiv.org/abs/2511.10054.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 641–649, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache, 2025. URL https://arxiv.org/abs/2401.14361.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a.

Haoqi Yang, Luohe Shi, Qiwei Li, Zuchao Li, Ping Wang, Bo Du, Mengjia Shen, and Hai Zhao. Faster moe llm inference for extremely large models, 2025b. URL https://arxiv.org/abs/2505.03531.

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24(8):7059–7073, 2025. doi: 10.1109/TMC.2025.3546466.

Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. Taming latency-memory trade-off in moe-based llm serving via fine-grained expert offloading, 2025. URL https://arxiv.org/abs/2502.05370.

Jixian Zhou, Fang Dong, Ruijun Huang, Hengjie Cao, Mengyi Chen, Yifeng Yang, Anrui Chen, Mingzhi Dong, Yujiang Wang, Dongsheng Li, David A. Clifton, Qin Lv, Rui Zhu, Chun Zhang, Fan Yang, Tun Lu, Ning Gu, and Li Shang. Oracle-moe: Locality-preserving routing in the oracle space for memory-constrained large language model inference. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=wn6WHREK9k.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y. Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Yuxin Zhou, zheng li, Jun Zhang, Jue WANG, Yiping Wang, Zhongle Xie, Ke Chen, and Lidan Shou. Floe: On-the-fly moe inference on memory-constrained GPU. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=i5aHAkkhJH.

APPENDIX CONTENTS

A Related Works **15**

B Additional Experimental Details **16**
B.1 Model Details 16
B.2 Optimizer and Loss Function Parameters 16
B.3 Hardware Configurations 17
B.4 Evaluation Settings 17
B.5 Training Time and Overhead 17

C Loss Function Design Justification **18**
C.1 Choice of $\mathcal{L}_{cs}$ 18
C.2 Choice of $\mathcal{L}_{rm}$ 20
C.3 Practical Implementation Details 21

D Additional Experiments **22**
D.1 Out-of-Distribution Generalization Performance of MELINOE 22
D.2 Effect of Output Generation Length on Throughput 23
D.3 Ablation on Quantized Experts 23
D.4 Impact of Fine-Tuning on Expert Routing 24
D.5 Ablation on GPU VRAM Budget 28
D.6 Ablation on Soft Cache Capacity in Loss 28
D.7 Ablation on Loss Function Decay Factor γ 29
D.8 Ablation on Cache Eviction Policy 29
D.9 Fine-Tuning without Auxiliary Objectives 29
D.10 Time to First Token (TTFT) Analysis 29
D.11 Alternative Training-Free Methods for Increasing Routing Locality 30

A Related Works

Mixture-of-Experts Model Architectures. Mixture-of-Experts (MoE) model architectures were first proposed as an alternative to traditional transformer-based models Shazeer et al. (2017). In MoE models, the Feed Forward Network (FFN) present in each transformer block is separated into multiple FFNs which are sparsely activated on a per-token basis. These sparse activations allow MoE models to increase model capacity without a proportional increase in training compute, since only a small subset of experts is executed per token Lepikhin et al. (2021); Fedus et al. (2022); Du et al. (2022); Riquelme et al. (2021). A few design choices are critical for making MoE models effective in practice. First, expert granularity controls the size and number of experts in the MoE model. Coarse-grained MoE models such as Mixtral-8x7B Jiang et al. (2024), Phi-3.5-MoE Abdin & et al. (2024), and DBRX The Mosaic Research Team (2024) all use a relatively small number of large experts. In contrast, fine-grained MoE architectures such as OLMoE Muennighoff et al. (2025), Qwen2MoE Yang et al. (2024), Qwen3MoE Yang et al. (2025a), DeepSeek-V2 DeepSeek-AI et al. (2024), and DeepSeek-V3 DeepSeek-AI et al. (2025) use a higher number of small experts to increase expert specialization and reduce training costs. Another axis by which MoE models differ is their token routing strategy. Initial MoE model architectures such as GShard Lepikhin et al. (2021) and GLaM Du et al. (2022) employ router load balancing losses to ensure all experts are utilized in training to prevent expert collapse and token dropping from overloaded experts. BASE Layers Lewis et al. (2021) instead formulates routing as a linear assignment problem, ensuring experts receive an equal number of tokens during generation. Expert Choice Routing Zhou et al. (2022) allows each expert to instead pick its Top-K tokens. More recent works such as Loss-Free Balancing Wang et al. (2024) add expert-wise biases to the router's decisions and update these biases throughout generation to achieve approximate load balancing without an auxiliary loss. Granularity and routing are critical design choices in downstream deployments as they directly shape the tradeoff between capacity, efficiency, and serving latency.

On-Device Mixture-of-Experts. The proliferation of open-source MoE models has made on-device deployments of MoEs a critical research direction. MoE inference is often memory-bound since all experts must be accessible at inference time despite. In this regime, expert offloading techniques have emerged as a promising way of managing the memory constraints imposed by MoE models. Edge-MoE Yi et al. (2025) assumes a highly constrained setting that requires offloading quantized experts to disk memory. However, the following works allow for a larger amount of GPU VRAM and enough CPU DRAM to host all offloaded model weights. Mixtral-Offloading Eliseev & Mazur (2023), which operates in this more lenient regime, uniformly quantizes experts to 3 bits and all remaining weights to 4 bits, allowing for a larger number of experts to remain resident in GPU DRAM. This variable bit-width quantization incurs additional compute overhead and a non-negligible decrease in model performance. FLoE Zhou et al. (2025b) similarly uses quantization and activation sparsity to increase the size of the GPU-resident expert set but applies them to specific layers where they find compression techniques have the least effect on performance. Other techniques avoid lossy compression schemes and exploit other avenues for reducing I/O latency. MoE-Infinity Xue et al. (2025) prefetches experts by profiling previous expert activations and uses K-means to identify experts that are likely to be activated for a specific request. FineMoE Yu et al. (2025) aims to improve on this prefetching scheme by creating expert maps that track token-level activation patterns to prefetch experts in every iteration. An alternative approach proposed by Fiddler Kamahori et al. (2025) is to directly move intermediate activations to CPU memory and perform expert computation on the CPU for CPU-resident experts. As activations are smaller than the expert weights themselves, this method trades compute efficiency for a reduction in I/O latency. Methods such as HarMoEny Doucet et al. (2025) and Faster MoE Yang et al. (2025b) which target the multi-GPU inference setting aim to optimize communication but are not directly applicable to this resource-constrained setting.

Routing and Semantic Locality. While expert offloading is a promising direction for reducing the memory demands of MoE models, recent works suggest that naïvely applying expert offloading may lead to substantial hidden I/O latency. Concretely, MoE models may exhibit low local routing consistency Liang et al. (2025), where nearby tokens may activate different experts. This diversity in activations increases the number of cache transfers, reducing the efficacy of a GPU-resident expert set. Recent literature approaches the issue of routing locality in various ways. Oracle-MoE Zhou et al. (2025a) pretrains MoE architectures to route tokens based on attention score-based semantic similarity

in order to reduce expert transfers. However, its evaluation is limited to GPT-2-scale models, leaving open how well the approach extends to modern MoE models, where larger architectures and tighter GPU memory budgets can make offloading and transfer bottlenecks more pronounced. BlockFFN Song et al. (2025) similarly performs MoE pretraining but utilizes an auxiliary loss function that encourages neighboring tokens to activate similar experts. BlockFFN does not specifically target expert offloading and only mentions speculative decoding as a device-side acceleration technique that composes with their method.

B Additional Experimental Details

B.1 Model Details

In Table 7, we include details of OLMoE, Phi-3.5-MoE, and Mixtral-8x7B, the models that we use in all experiments.

Table 7: Model details for the MoE backbones used in all experiments. Total parameters count all experts; active parameters correspond to the parameters used per token under Top-K routing. FP16 size is the approximate weight footprint assuming 2 bytes/parameter.

Model	Layers	Experts/ Layer	Active Experts	Total Params. (B)	Active Params. (B)	Context Len.	FP16 Size (GB)
OLMoE	16	64	8	6.9	1.3	4K	13.8
Phi-3.5-MoE	32	16	2	42.0	6.6	128K	84.0
Mixtral-8x7B	32	8	2	46.7	12.9	32K	93.4

B.2 Optimizer and Loss Function Parameters

In Tables 8 and 9, we highlight training hyperparameters used for fine-tuning each MoE model and training the MLP activation predictor, respectively.

Table 8: Fine-tuning hyperparameters for each dataset and MoE model backbone.

	Dataset: Dolly15K			Dataset: GSM8K		
Hyperparameter	**OLMoE**	**Phi-3.5-MoE**	**Mixtral-8x7B**	**OLMoE**	**Phi-3.5-MoE**	**Mixtral-8x7B**
Peak Learning Rate	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW	AdamW
Schedule	Linear	Linear	Linear	Linear	Linear	Linear
Warmup Ratio	0.03	0.03	0.03	0.03	0.03	0.03
Max Tokens	512	512	512	512	512	512
LoRA Rank (r)	32	32	32	32	32	32
LoRA α	16	16	16	16	16	16
Epochs	3	3	3	5	5	5
λ_{cs}	0.5	0.5	0.5	0.05	0.05	0.05
λ_{rm}	0.1	0.1	0.1	0.01	0.01	0.01
γ (Cache Decay)	0.9	0.9	0.9	0.9	0.9	0.9
ρ (Rank Margin)	0.1	0.1	0.1	0.1	0.1	0.1
Cache Capacity C	16	4	2	16	4	2

Table 9: Activation predictor MLP training hyperparameters (same values across datasets).

Hyperparameter	Value
Number of Layers	2
Input Size	768
Hidden Dimension	1024
Loss	KL Divergence
Learning Rate	2×10^{-4}
Epochs	10
Optimizer	SGD
Momentum	0.9
Batch Size	16

B.3 Hardware Configurations

In Table 10, we characterize the various hardware setups and configurations used in our experiments.

Table 10: Hardware configurations used in offloaded MoE inference experiments.

Spec	H100 Setup	A100 Setup	RTX 4090 Setup
GPU	H100	A100	RTX 4090
GPU VRAM (GB)	80	40	24
Host CPU	Intel Xeon Platinum 8470	AMD EPYC 7J13	AMD EPYC 7B13
Host DRAM (GB)	2048	216	270
PCIe Gen xLanes	Gen 5 x16	Gen 4 x16	Gen 4 x16
PCIe Bandwidth (GB/s)	64.0	32.0	32.0

B.4 Evaluation Settings

Finally, in Table 11, we present model settings and decoding metrics used during the evaluation of our method.

Table 11: Evaluation configuration used for throughput measurements.

	OLMoE	Phi-3.5-MoE	Mixtral-8x7B
Resident Experts / Layer	16	8	5
Quantized Modules	`gate_proj, up_proj, down_proj`	`w1, w2, w3`	`w1, w2, w3`
Decoding Strategy	Greedy	Greedy	Greedy
Throughput Metric	Output tokens/s	Output tokens/s	Output tokens/s
Max Output Tokens	256	256	256

B.5 Training Time and Overhead

For OLMoE, all training runs were done on 2xH100 GPUs for a maximum of 8 hours, and for Phi-3.5-MoE and Mixtral-8x7B, all training runs were done on 4xH100 GPUs for the same time. All training runs were done on publicly available resources.

C Loss Function Design Justification

In this section, we provide theoretical justification for the choice of our loss functions. Throughout the section, we will use b and f in the subscript to associate symbols with the base model and the fine-tuned model respectively, $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ refers to the sequence of tokens, and Top-$C(\cdot)$ refers to an operation that returns a vector with L_1 norm C and entries 1 for C positions with the highest magnitude and 0 otherwise. $[N]$ for any natural number $N \in \mathbb{Z}_+$ denotes the set $[1, \ldots, N]$.

C.1 Choice of $\mathcal{L}_{cs}$

The cache simulation loss $\mathcal{L}_{cs}(\mathbf{x})$ is designed to penalize expert activation patterns with excessive switching. Previous work indicates that certain MoE models activate similar experts across adjacent tokens, while others consistently activate a small subset of experts across the decoding trajectory for a given prompt. In the former case, an LRU cache eviction policy is preferable, whereas in the latter an LFU-based eviction policy is more appropriate under conventional caching frameworks.

By fine-tuning the router layers, we can control the underlying expert activation distribution, allowing us to tailor the distribution towards a specific cache eviction policy. The $\mathcal{L}_{cs}$ loss encourages tokens to have similar activation distributions. The decay parameter γ weights older requests relative to newer ones when computing this similarity. In the following proposition, we will first explain how γ can be used to interpolate between LFU and LRU updates with hard cache states.

Definition C.1 (γ cache eviction)**.** For any given sequence of tokens $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, let $\mathbf{r}_f^{(\ell,t)}(\mathbf{x}) \in \{0,1\}^E$ denote the binary request vector at layer ℓ and token position t with $\|\mathbf{r}_f^{(\ell,t)}(\mathbf{x})\|_1 = K$. Then, we define the γ-discounted count and the γ-discounted cache state under model f ($\mathbf{C}_{f,\gamma}$) as follows

$$\text{Count}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x}) = \gamma \text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) + \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \tag{8}$$

$$\mathbf{C}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x}) = \text{Top-}C\left(\text{Count}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x})\right), \tag{9}$$

where $\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ is a vector representing the discounted expert request numbers up to $t-1$ for layer ℓ under sequence $\mathbf{x}$ and $\text{Count}_{f,\gamma}^{(\ell,1)}(\mathbf{x})$ is initialized as a vector with L_1 norm C. Note that $\text{Count}_{f,\gamma}^{(\ell,1)}$ can be set to anything based on the prefetch initialization used. Alternatively, one could also use a uniform initialization.

Remark C.2. From Definition C.1, we have

- Using $\gamma = 1$ corresponds to LFU cache updates, and when $\gamma \to 0^+$ we get LRU cache updates.
- For any γ, the cache can be updated lazily. Specifically, any expert that is present in the cache for layer ℓ at time $t+1$, that is any index with a nonzero entry in $\mathbf{C}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x})$, must either have been present in $\mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ or appear in $\mathbf{r}_f^{(\ell,t)}(\mathbf{x})$. Consequently, the cache can be updated during offloaded inference without incurring additional overhead.

The total number of cache misses equals the number of PCIe transfers. Under a γ-cache eviction policy, the number of cache misses at layer ℓ over T tokens is given by

$$\text{Cache misses at layer } \ell := \sum_{t=1}^{T} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{1} - \mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) \right\rangle,$$

and the hard loss $\mathcal{L}_{cs}^{\text{hard}}(\mathbf{x}, f)$ as

$$\mathcal{L}_{cs}^{\text{hard}}(\mathbf{x}, f) := \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{1} - \mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) \right\rangle. \tag{10}$$

However, the loss above is hard to differentiate with respect to the parameters of f due to the Top-C operation used to define $\mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$. Hence, we use a "soft" cache proxy $\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ instead of $\mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ where

$$\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) := \frac{\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})}{\left\|\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\|_1} \cdot C, \tag{11}$$

which is the $\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ scaled to make its L_1 norm equal to C.

Proposition C.3. *Using the definitions in Equations equation 8 and equation 11, we have*

$$\left\|\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\|_1 = \gamma^{t-1}C + K\sum_{i=1}^{t-1}\gamma^{t-i-1},$$

define $Z^{(t)} = \gamma^{t-1} + \frac{K}{C}\sum_{i=1}^{t-1}\gamma^{t-i-1}$. *Then, by Equation equation 11, one can recursively update the soft cache state as*

$$\mathbf{c}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x}) := \frac{\gamma Z^{(t)}\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) + \mathbf{r}_f^{(\ell,t)}(\mathbf{x})}{Z^{(t+1)}} \qquad \text{and} \qquad Z^{(t+1)} = \gamma Z^{(t)} + \frac{K}{C}$$

Proof. Using the definition in Equation equation 8, we have

$$\left\|\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\|_1 = \left\|\gamma^{t-1}\text{Count}_{f,\gamma}^{(\ell,1)}(\mathbf{x}) + \sum_{i=1}^{t-1}\gamma^{t-i-1}\mathbf{r}_f^{(\ell,t)}(\mathbf{x})\right\|_1 = \gamma^{t-1}C + K\sum_{i=1}^{t-1}\gamma^{t-i-1}$$

Next, using this, one can show $\forall t \in [T]$

$$\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) = \frac{\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})}{\left\|\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\|_1}\cdot C = \frac{\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x})}{Z^{(t)}}.$$

and

$$\begin{aligned}\mathbf{c}_{f,\gamma}^{(\ell,t+1)}(\mathbf{x}) &= \left(Z^{(t+1)}\right)^{-1}\left(\gamma\text{Count}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) + \mathbf{r}_f^{(\ell,t)}(\mathbf{x})\right)\\ &= \left(Z^{(t+1)}\right)^{-1}\left(\gamma Z^{(t)}\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) + \mathbf{r}_f^{(\ell,t)}(\mathbf{x})\right).\end{aligned}$$

The recursion for $Z^{(t)}$ follows directly from the definition. □

Subsequently, we define $\mathcal{L}_{cs}(\mathbf{x}, f)$ by replacing the cache state $\mathbf{C}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ in Equation equation 10 with the "soft" cache state $\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ in order to facilitate fine-tuning the model f via gradient-based methods

$$\mathcal{L}_{cs}(\mathbf{x}, f) := \frac{1}{LT}\sum_{\ell=1}^{L}\sum_{t=1}^{T}\left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{1} - \mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\rangle.$$

The next lemma provides an interpretation of the cache simulation loss $\mathcal{L}_{cs}$.

Lemma C.4. *Let* $\mathbf{x} \sim \mathcal{D}_{train}$ *denote the training dataset of SFT traces. Define* $\mathcal{L}_{cs}(f) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{train}}[\mathcal{L}_{cs}(\mathbf{x}, f)]$, *then we have*

$$\mathcal{L}_{cs}(f) = K - \frac{1}{LT}\sum_{\ell=1}^{L}\sum_{t=1}^{T}\left(Z^{(t)}\right)^{-1}\left(\sum_{i=1}^{t-1}\gamma^{t-1-i}\phi_f^{(\ell)}(t,i) + \gamma^{t-1}\phi_f^{(\ell)}(t,1)\right)$$

where $\phi_f^{(\ell)}(t,i) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{r}_f^{(\ell,i)}(\mathbf{x})\right\rangle\right]$, $\phi_f^{(\ell)}(t,1) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{c}_{f,\gamma}^{(\ell,1)}(\mathbf{x})\right\rangle\right]$, and its derivative with respect to the parameter γ: $\frac{d\mathcal{L}_{cs}(f)}{d\gamma} \le 0$

Proof. Recall, $\mathcal{L}_{cs}(\mathbf{x}, f)$ is given by

$$\mathcal{L}_{cs}(\mathbf{x}, f) = \frac{1}{LT}\sum_{\ell=1}^{L}\sum_{t=1}^{T}\left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{1} - \mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\rangle$$

Let us denote $\mathcal{L}_{cs}^{(\ell,t)}(\mathbf{x}) := \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{1} - \mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})\right\rangle$. Then, for some initialization $\mathbf{c}_{f,\gamma}^{(\ell,1)}(\mathbf{x})$, using Proposition C.3, one can unroll $\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x})$ as follows

$$\mathbf{c}_{f,\gamma}^{(\ell,t)}(\mathbf{x}) = \left(Z^{(t)}\right)^{-1}\left(\sum_{i=1}^{t-1}\gamma^{t-1-i}\mathbf{r}_f^{(\ell,i)}(\mathbf{x}) + \gamma^{t-1}\mathbf{c}_{f,\gamma}^{(\ell,1)}(\mathbf{x})\right)$$

Using this expression and the fact $\|\mathbf{r}_f^{(\ell,t)}(\mathbf{x})\|_1 = K$, one can obtain

$$\mathcal{L}_{cs}^{(\ell,t)}(\mathbf{x}, f) = K - \left(Z^{(t)}\right)^{-1} \left(\sum_{i=1}^{t-1} \gamma^{t-1-i} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{r}_f^{(\ell,i)}(\mathbf{x}) \right\rangle + \gamma^{t-1} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{c}_{f,\gamma}^{(\ell,1)}(\mathbf{x}) \right\rangle\right)$$

and its derivative with respect to γ is given by

$$\begin{aligned}
&\frac{d\mathcal{L}_{cs}^{(\ell,t)}(\mathbf{x}, f)}{d\gamma} \\
&= -\left(Z^{(t)}\right)^{-1} \left(\sum_{i=1}^{t-2} (t-1-i)\gamma^{t-2-i} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{r}_f^{(\ell,i)}(\mathbf{x}) \right\rangle + (t-1)\gamma^{t-2} \left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{c}_{f,\gamma}^{(\ell,1)}(\mathbf{x}) \right\rangle\right) \\
&\leq 0
\end{aligned}$$

Taking the expectation with respect to $\mathbf{x} \sim \mathcal{D}_{\text{train}}$ completes the result. □

Remark C.5. From Lemma C.4, one can observe that minimizing the loss $\mathcal{L}_{cs}(f)$ entails maximizing the terms $\phi_f^{(\ell)}(t,i) := \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{train}}}\left[\left\langle \mathbf{r}_f^{(\ell,t)}(\mathbf{x}), \mathbf{r}_f^{(\ell,i)}(\mathbf{x}) \right\rangle\right]$ which encourages the expert request distributions at layer ℓ for token positions t and i to be similar. Note that $\gamma < 1$ implies that $\phi_f^{(\ell)}(t,i)$ receives a higher weight when t and i are closer, and this incentivizes reduced switching.

Remark C.6. We choose $\gamma = 0.9$ in our experiments since $\frac{d\mathcal{L}_{cs}(f)}{d\gamma} \leq 0$ while also enabling the resulting model to work with both LRU and LFU cache eviction policies.

C.2 Choice of $\mathcal{L}_{rm}$

Given a sequence of tokens $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, the purpose of the term $\mathcal{L}_{rm}(\mathbf{x})$ is to discourage router collapse onto a small subset of experts. A broadly adopted strategy for preventing distributional collapse while retaining desirable properties of the base model is KL-regularization. Incorporating a KL penalty is ubiquitous across domains including safety fine-tuning, RLHF, knowledge distillation, and reasoning model training where it serves as a general mechanism for constraining policy updates relative to the base model. In our setting, let $\mathbf{p}_f^{(\ell,t)}(\mathbf{x})$ and $\mathbf{p}_b^{(\ell,t)}(\mathbf{x})$ denote the router distributions at layer ℓ and token position t induced by the fine-tuned and base routers, respectively. One may then define the KL loss as follows

$$\mathcal{L}_{rm}^{\text{KL}}(\mathbf{x}, f) = \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} \text{KL}\left(\mathbf{p}_f^{(\ell,t)}(\mathbf{x}) \| \mathbf{p}_b^{(\ell,t)}(\mathbf{x})\right).$$

Although this may be a natural choice in tasks such as knowledge distillation where the objective is to match the teacher's distribution, our setting is different. We are less concerned with divergence between the two distributions in the KL or entropy sense and are ultimately interested in the induced rankings, since the model selects the Top-K experts from the router distribution. We, therefore, choose a loss that is more directly aligned with mismatches in the induced rankings. Concretely, we model this using the Kendall rank correlation coefficient Kendall (1938).

Definition C.7. Given two router output distributions $\mathbf{p}, \mathbf{q} \in \Delta^E$, the Kendall rank correlation coefficient $\tau(\mathbf{p}, \mathbf{q}) \in [-1, 1]$ is defined as

$$\tau(\mathbf{p}, \mathbf{q}) = 1 - \frac{2\text{Inv}(\mathbf{p}, \mathbf{q})}{\binom{n}{2}},$$

where $\text{Inv}(\mathbf{p}, \mathbf{q})$ is the number of pairwise inversions between distributions $\mathbf{p}$ and $\mathbf{q}$.

Maximizing the rank correlation coefficient is equivalent to minimizing the inversion number. Ideally, one could use the following loss function

$$\mathcal{L}_{rm}^{\text{inv}}(\mathbf{x}) = \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} \text{Inv}\left(\mathbf{p}_f^{(\ell,t)}(\mathbf{x}), \mathbf{p}_b^{(\ell,t)}(\mathbf{x})\right).$$

Note that when this loss is 0, the experts activated by the fine-tuned router and the base router will be exactly the same. However, the term $\text{Inv}(\cdot,\cdot)$ is difficult to parameterize and differentiate with respect to the parameters of the fine-tuned model f. One can express the inversion number as follows

$$\text{Inv}\left(\mathbf{p}_f^{(\ell,t)}(\mathbf{x}), \mathbf{p}_b^{(\ell,t)}(\mathbf{x})\right) = \sum_{i,j\in[E]} g_b^{(\ell,t)}(\mathbf{x},i,j)\; g_{f,\text{Inv}}^{(\ell,t)}(\mathbf{x},i,j),$$

where

$$g_b^{(\ell,t)}(\mathbf{x},i,j) := \mathbb{I}\left\{\mathbf{p}_{b,i}^{(\ell,t)}(\mathbf{x}) > \mathbf{p}_{b,j}^{(\ell,t)}(\mathbf{x})\right\} \quad \text{and} \quad g_{f,\text{Inv}}^{(\ell,t)}(\mathbf{x},i,j) := \mathbb{I}\left\{\mathbf{p}_{f,i}^{(\ell,t)}(\mathbf{x}) < \mathbf{p}_{f,j}^{(\ell,t)}(\mathbf{x})\right\}.$$

To facilitate learning using gradient-based methods, we use $m^{(\ell,t)}$ as a proxy for the inversion number $\text{Inv}(\cdot,\cdot)$. Specifically, we replace $g_{f,\text{Inv}}$ by g_f, i.e.,

$$m^{(\ell,t)}(\mathbf{x}) = \sum_{i,j\in[E]} g_b^{(\ell,t)}(\mathbf{x},i,j)\; g_f^{(\ell,t)}(\mathbf{x},i,j) \tag{12a}$$

with

$$g_f^{(\ell,t)}(\mathbf{x},i,j) := \left[\rho - \left(\mathbf{p}_{f,i}^{(\ell,t)}(\mathbf{x}) - \mathbf{p}_{f,j}^{(\ell,t)}(\mathbf{x})\right)\right]_+. \tag{12b}$$

and define $\mathcal{L}_{rm}$ as

$$\mathcal{L}_{rm}(\mathbf{x},f) = \frac{1}{LT}\sum_{\ell=1}^{L}\sum_{t=1}^{T} m^{(\ell,t)}(\mathbf{x}).$$

Lemma C.8. *Define* $m^{(\ell,t)}(\mathbf{x}) = \sum_{i,j\in[E]} \mathbb{I}\left\{\mathbf{p}_{b,i}^{(\ell,t)}(\mathbf{x}) > \mathbf{p}_{b,j}^{(\ell,t)}(\mathbf{x})\right\}\left[\rho - \left(\mathbf{p}_{f,i}^{(\ell,t)}(\mathbf{x}) - \mathbf{p}_{f,j}^{(\ell,t)}(\mathbf{x})\right)\right]_+$ *as in Equation equation 12. Then, we have*

$$m^{(\ell,t)}(\mathbf{x}) \geq \rho\, \text{Inv}\left(\mathbf{p}_f^{(\ell,t)}(\mathbf{x}), \mathbf{p}_b^{(\ell,t)}(\mathbf{x})\right).$$

Proof. Without loss of generality, assume that no two entries for either vector $\mathbf{p}_f^{(\ell,t)}(\mathbf{x})$ or $\mathbf{p}_b^{(\ell,t)}(\mathbf{x})$ are equal. Then, we have

$$\begin{aligned}
\rho\, \text{Inv}\left(\mathbf{p}_f^{(\ell,t)}(\mathbf{x}), \mathbf{p}_b^{(\ell,t)}(\mathbf{x})\right) &= \sum_{i,j\in[E]} \rho\mathbb{I}\left\{\mathbf{p}_{f,i}^{(\ell,t)}(\mathbf{x}) < \mathbf{p}_{f,j}^{(\ell,t)}(\mathbf{x})\right\}\mathbb{I}\left\{\mathbf{p}_{b,i}^{(\ell,t)}(\mathbf{x}) > \mathbf{p}_{b,j}^{(\ell,t)}(\mathbf{x})\right\} \\
&\leq \sum_{i,j\in[E]} \left[\rho - \left(\mathbf{p}_{f,i}^{(\ell,t)}(\mathbf{x}) - \mathbf{p}_{f,j}^{(\ell,t)}(\mathbf{x})\right)\right]_+ \mathbb{I}\left\{\mathbf{p}_{b,i}^{(\ell,t)}(\mathbf{x}) > \mathbf{p}_{b,j}^{(\ell,t)}(\mathbf{x})\right\} \\
&= m^{(\ell,t)}(\mathbf{x}),
\end{aligned}$$

where the last line follows from $\rho\,\mathbb{I}(a<b) \leq [\rho-(a-b)]_+$ □

Remark C.9. As a consequence of Lemma C.8, minimizing loss $\mathcal{L}_{rm}$ is equivalent to maximizing a lower bound on the rank correlation coefficient between the fine-tuned and base router expert request distributions averaged across layers $\ell \in [L]$ and token positions $t \in [T]$.

C.3 Practical Implementation Details

We briefly describe how the model is optimized with the cache simulation loss $\mathcal{L}_{cs}$ in practice. During fine-tuning, the forward pass uses a hard top-K binary mask to select experts and compute the cache state, consistent with standard MoE routing. To enable gradient flow through this non-differentiable operation, we apply a straight-through estimator where the backward pass substitutes gradients through the scaled soft router probabilities. The soft cache state update itself, defined in Section 3.1.1, is fully differentiable. The combined effect is that the router receives gradients encouraging it to assign higher probability to experts with high cache residency and lower probability to experts likely to trigger cache misses. Implementation details for this gradient flow can be found in our supplementary materials.

D ADDITIONAL EXPERIMENTS

D.1 OUT-OF-DISTRIBUTION GENERALIZATION PERFORMANCE OF MELINOE

As mentioned in Section 5, computational constraints prevented us from performing large-scale fine-tuning on a comprehensive, general-purpose dataset. Instead, we evaluate MELINOE across two complementary axes of generalization: (1) cross-task transfer, where the fine-tuning data and downstream task differ, and (2) broader benchmark coverage on WikiText-103 (general web text) and SciQ (scientific question answering).

Cross-Task Generalization. In Table 12, we evaluate MELINOE under the same settings and resource constraints as Sections 4.1 and 4.2, but cross the fine-tuning dataset with the evaluation dataset. We find that throughput improvements relative to prior baselines persist even when fine-tuning and evaluation distributions differ, though they are partially dampened. For example, Phi-3.5-MoE fine-tuned on GSM8K and evaluated on Dolly15K still achieves 10.44 tokens/s, exceeding the next-best baseline (Mixtral-Offloading) by 1.86 tokens/s. Mixtral-8x7B is particularly robust: fine-tuning on GSM8K and evaluating on Dolly15K yields 8.21 tokens/s, only 1.14 tokens/s below the in-distribution result. This suggests that MELINOE's gains largely persist even when the deployment device's local data distribution differs from the fine-tuning corpus.

Table 12: Decoding throughput (tokens/s) under cross-task evaluation. For MELINOE, we report throughput after fine-tuning on either Dolly15K or GSM8K; baselines are shown for reference.

	Eval: Dolly15K		**Eval: GSM8K**	
Method	**Phi-3.5-MoE**	**Mixtral-8x7B**	**Phi-3.5-MoE**	**Mixtral-8x7B**
MELINOE (FT: Dolly15K)	14.34	9.35	13.52	8.25
MELINOE (FT: GSM8K)	10.44	8.21	15.67	10.38
Fiddler	5.88	5.24	7.26	4.11
Mixtral-Offloading	8.58	5.08	8.52	5.04
DeepSpeed-MoE	2.63	1.25	2.69	1.23
FLoE	5.23	2.25	5.61	2.20
MoE-Infinity	3.73	1.25	3.79	1.24

Broader Benchmark Coverage. To further demonstrate robustness beyond Dolly15K and GSM8K, we fine-tune OLMoE on two additional benchmarks: WikiText-103 (general web text) and SciQ (scientific question answering). For WikiText-103, we limit fine-tuning to 30k examples due to time constraints. Table 13 shows that on WikiText-103, MELINOE incurs a modest perplexity increase of 0.46 while delivering $1.5\times$ the throughput of the next-fastest baseline. On SciQ, MELINOE delivers both the highest throughput and accuracy of all offloading methods. To verify that these results extend to larger MoE architectures, we additionally fine-tune Phi-3.5-MoE on WikiText-103 in Table 14, where MELINOE again achieves the best throughput while preserving base-model perplexity.

Table 13: OLMoE throughput (t/s), perplexity (PPL), and accuracy (Acc) on WikiText-103 and SciQ across competing baselines. MELINOE delivers the strongest throughput on both benchmarks while incurring only minor quality degradation.

	WikiText-103		**SciQ**	
Method	**Throughput (t/s)**	**PPL**	**Throughput (t/s)**	**Acc**
MELINOE	**22.84**	9.48	**27.08**	**60.68**
Fiddler	15.11	9.02	11.41	57.26
Mixtral-Offloading	7.63	10.47	8.96	58.11
DeepSpeed-MoE	12.56	9.02	12.29	57.26
FLoE	14.59	15.02	16.15	51.28
MoE-Infinity	11.67	9.02	12.51	57.26

Table 14: Phi-3.5-MoE throughput (t/s) and perplexity (PPL) on WikiText-103. Results extend to the larger Phi-3.5-MoE architecture, with MELINOE delivering the highest throughput while matching base-model perplexity.

Method	Throughput (t/s)	PPL
MELINOE	**11.84**	4.26
Fiddler	6.90	4.61
Mixtral-Offloading	7.67	4.69
DeepSpeed-MoE	2.49	4.61
FLoE	4.97	4.94
MoE-Infinity	3.46	4.61

Across four benchmarks spanning instruction following, mathematical reasoning, general web text, and scientific question answering, MELINOE consistently delivers throughput improvements while retaining or improving task performance. Combined with the cross-task results above, this breadth of evaluation provides strong evidence that MELINOE's benefits are not benchmark-specific.

D.2 Effect of Output Generation Length on Throughput

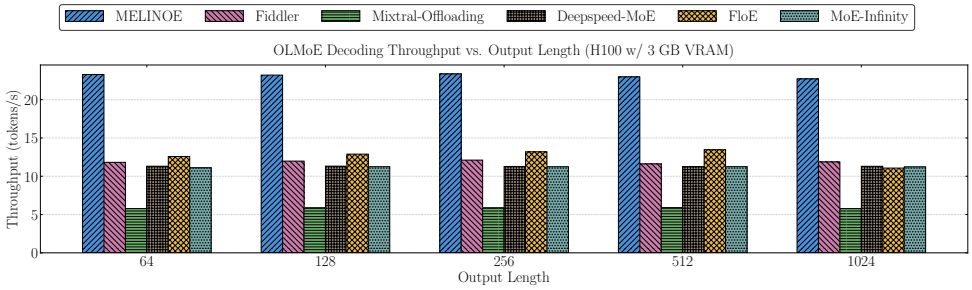


Figure 6: Throughput of baselines with various output lengths using OLMoE on the H100 setup with 3GB of VRAM.

As output length increases, MELINOE maintains stable throughput (Figure 6), which suggests that the fine-tuned model's expert preferences endure even in longer generations. In this regime, memory-efficiency becomes increasingly critical as the KV cache grows with the number of generated tokens.

D.3 Ablation on Quantized Experts

Quantizing experts to `HQQ INT4` allows more experts to remain resident in GPU VRAM. In Table 15, we analyze the effects of quantization on throughput. Each setting has roughly equal GPU VRAM usage. Quantization improves throughput, but its benefit is not proportional to the increase in resident experts due to compute overhead. The fine-tuned model with just 8 resident experts has greater throughput than the quantized base model with 24 resident experts.

Table 15: Impact of quantized experts for OLMoE. We report the number of GPU experts per layer and decoding throughput (tokens/s).

	Dolly15K		GSM8K	
	GPU Experts	**Throughput**	**GPU Experts**	**Throughput**
Base Model	8	13.98	8	13.66
Base Model + Quantized Experts	24	15.80	24	17.95
Fine-Tuned Model	8	22.77	8	19.89
Fine-Tuned Model + Quantized Experts	24	25.99	24	28.59

D.4 Impact of Fine-Tuning on Expert Routing

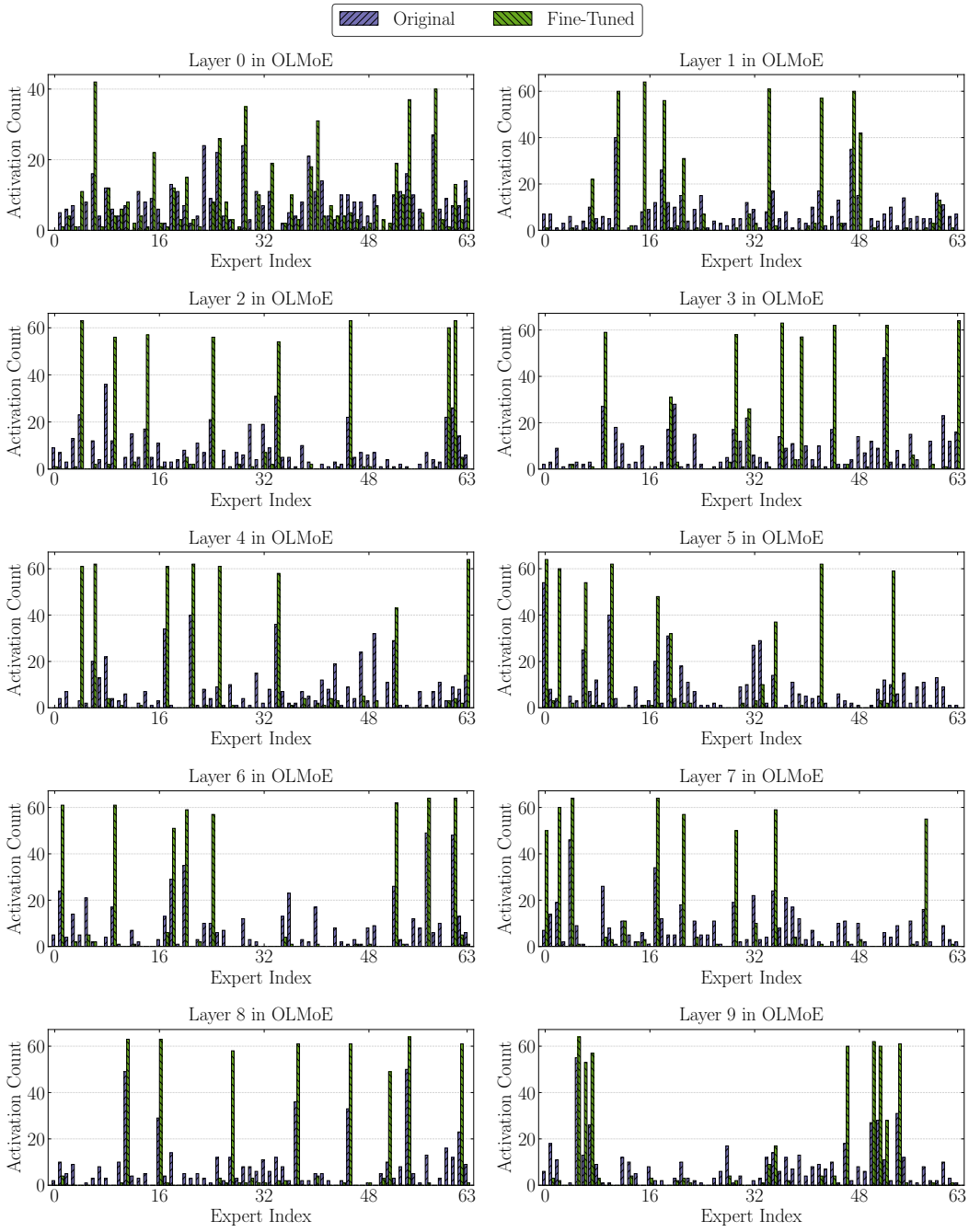


Figure 7: Activations of experts for a single sequence in the first 10 layers of OLMoE.

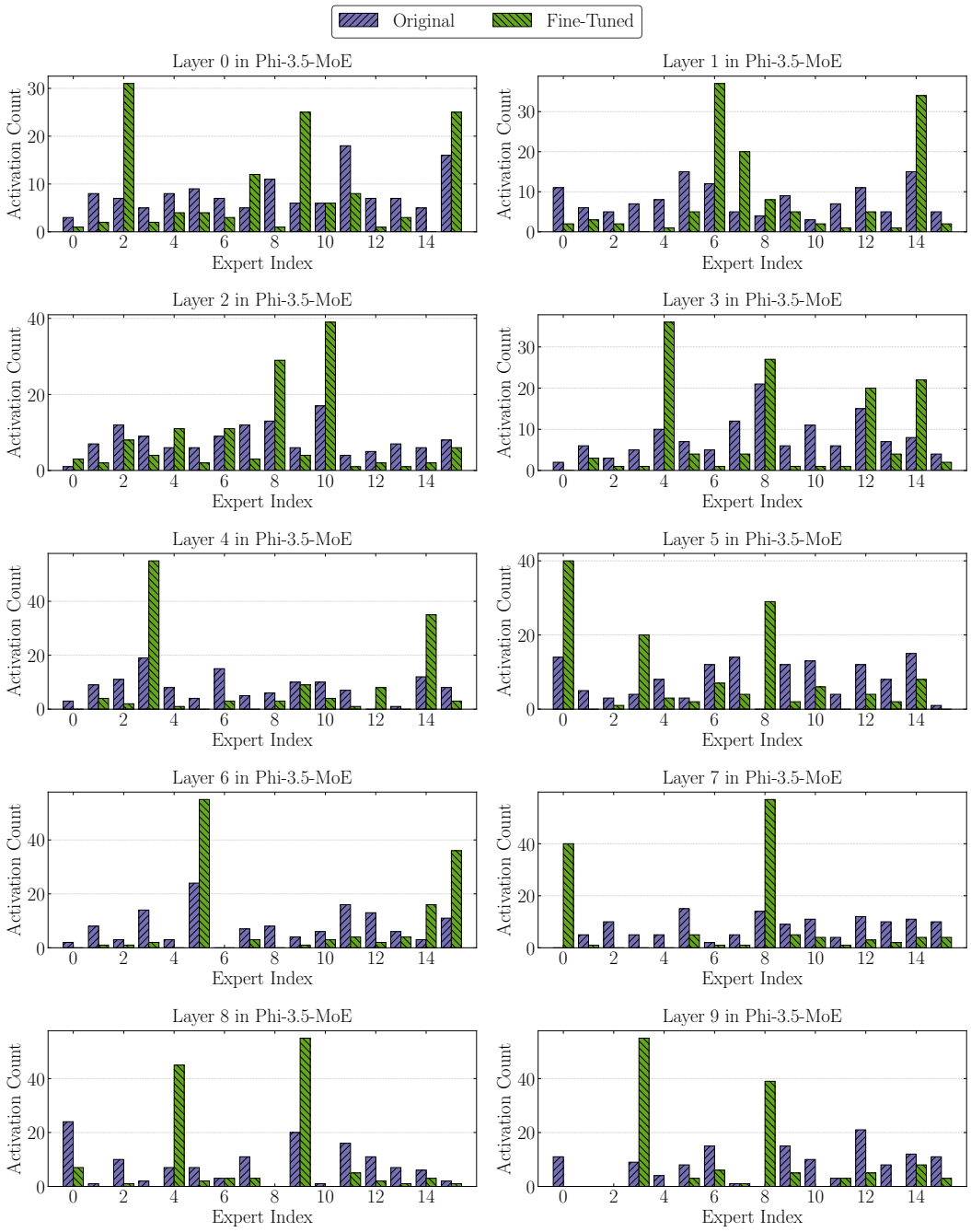


Figure 8: Activations of experts for a single sequence in the first 10 layers of Phi-3.5-MoE.

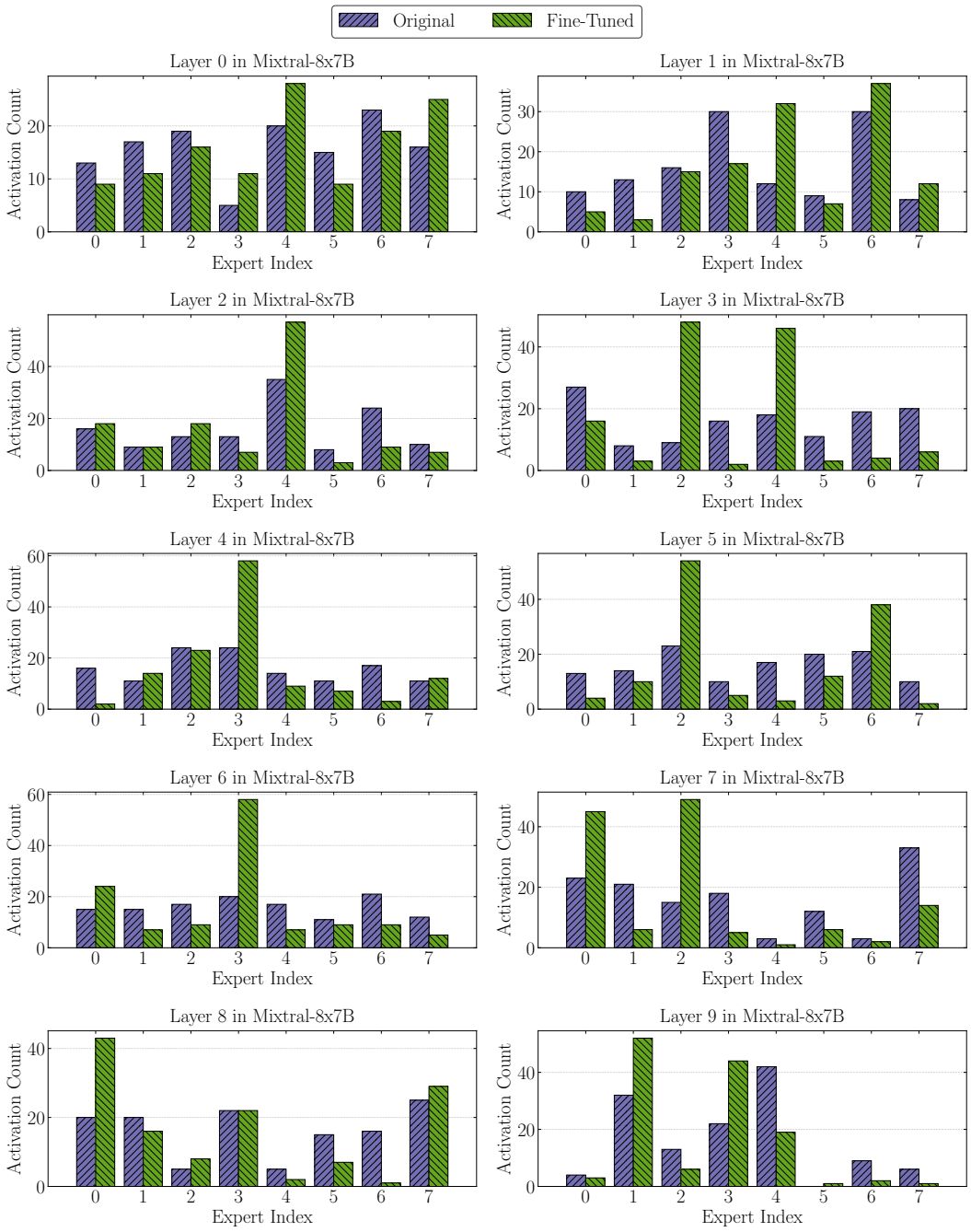


Figure 9: Activations of experts for a single sequence in the first 10 layers of Mixtral-8x7B.

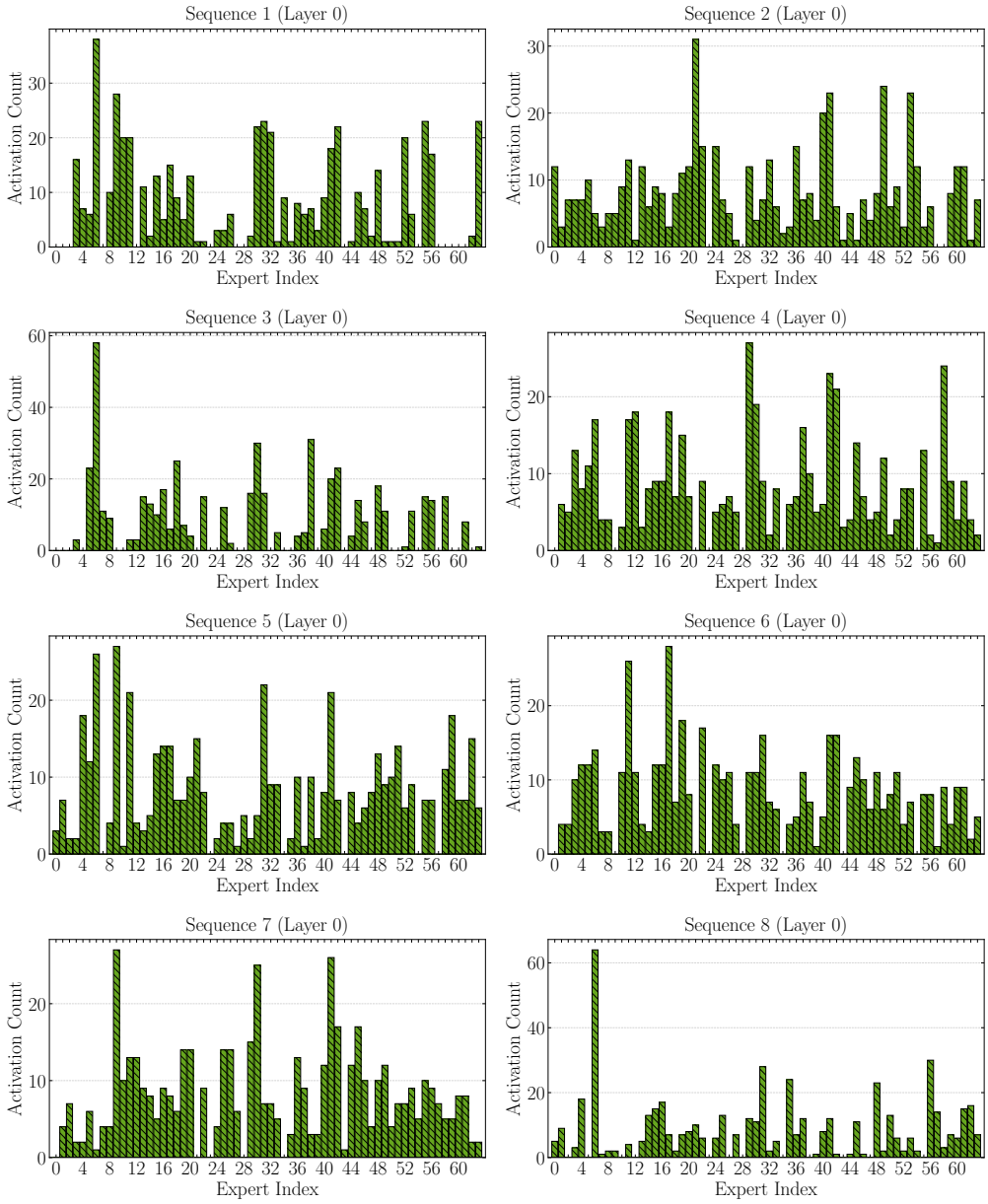


Figure 10: Activations of experts for 8 different sequences in layer 0 of OLMoE.

Figures 7, 8, and 9 demonstrate the effects of fine-tuning on expert activations for OLMoE, Phi-3.5-MoE, and Mixtral-8x7B, respectively. Across all model architectures, fine-tuning skews activations towards a few highly preferred experts, justifying fine-tuning as a meaningful approach to make routing more predictable for MoE models. Additionally, Figure 10 demonstrates that routing still remains diverse across multiple sequences, suggesting that the combined effect of the fine-tuning procedure presented in MELINOE is to create sequence-specific skew but retain global expert usage diversity.

D.5 Ablation on GPU VRAM Budget

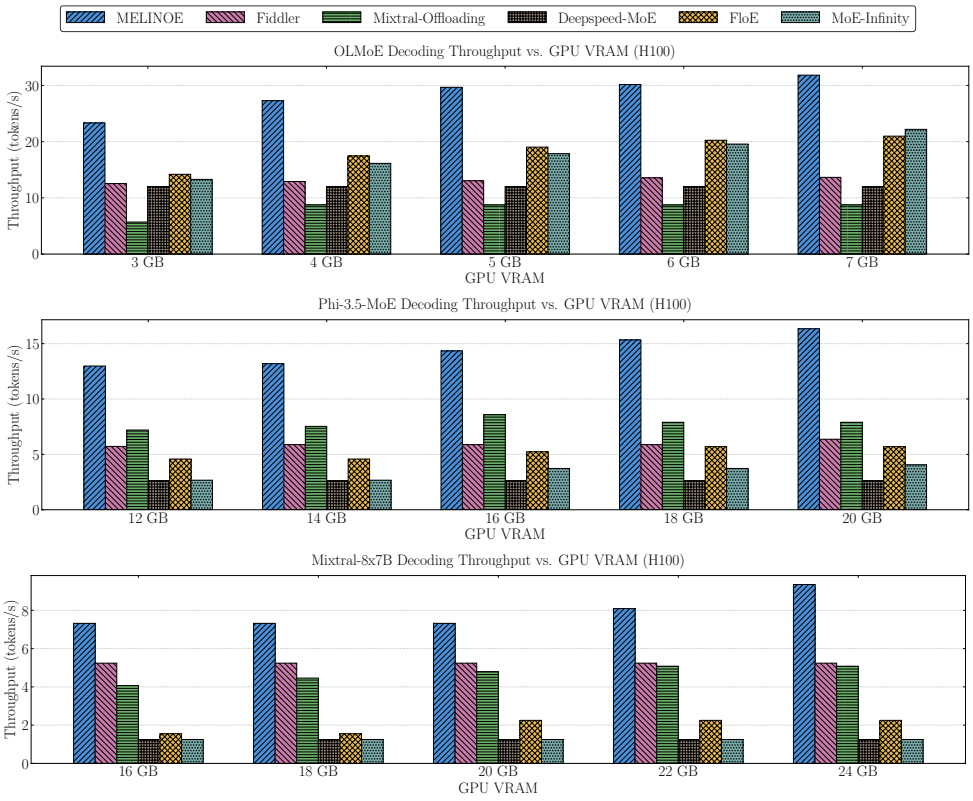


Figure 11: Throughput of baselines under GPU VRAM restrictions on H100 across three MoE models.

Figure 11 displays the impact of GPU VRAM on decoding throughput. Across all models and VRAM budgets, MELINOE outperforms prior offloading baselines, demonstrating its robustness across a diverse array of downstream deployments.

D.6 Ablation on Soft Cache Capacity in Loss

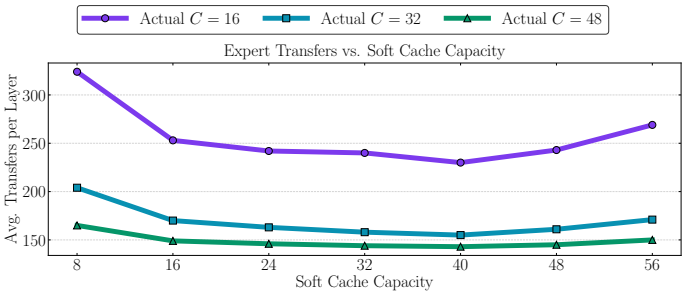


Figure 12: Transfers per layer with fine-tuned model using different soft cache capacities (OLMoE, 64 output tokens).

In Figure 12, we evaluate the effect of the soft cache capacity used in fine-tuning on the average number of transfers per layer in the downstream evaluation. We consider three different cache budgets during evaluation, $C = 16$, $C = 32$, and $C = 48$. Performance is noticeably worse when the soft cache capacity is set too low as transfers are dominated by forced evictions, making subtle routing choices more difficult to learn. There is also a slight degradation in the number of transfers when the

soft cache budget is set too high as few transfers occur in this regime so transfer penalties remain limited.

D.7 Ablation on Loss Function Decay Factor

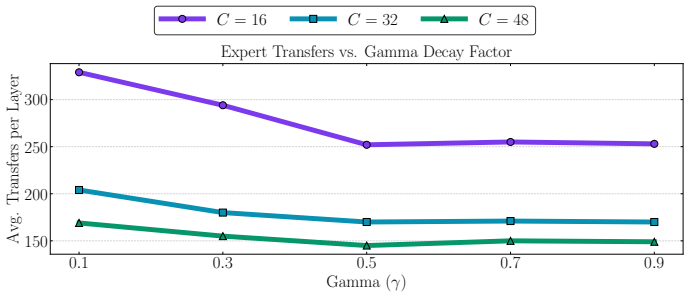


Figure 13: Transfers per layer with fine-tuned model using different γ decay factors (OLMoE, 64 output tokens).

Figure 13 studies the decay factor γ used in the cache simulation loss, which controls how long past expert usage is factored into current eviction decisions. We find that transfers are high when γ is too small but decrease rapidly as γ increases across all cache budgets. Thus, overly aggressive decay makes routing decisions too myopic when using an LFU eviction policy.

D.8 Ablation on Cache Eviction Policy

Finally, in Table 16, we quantify the impact of the cache eviction policy on the number of transfers per layer. For smaller values of γ, an LRU policy results in fewer cache transfers as the model was fine-tuned with a more reactive cache simulation loss. However, using a larger γ with an LFU policy results in the fewest number of transfers overall.

Table 16: Transfers per layer for OLMoE under different decay factors γ and different eviction policies during inference.

	LRU Eviction Policy	**LFU Eviction Policy**
Fine-Tuned w/ $\gamma = 0.1$	314	329
Fine-Tuned w/ $\gamma = 0.3$	287	294
Fine-Tuned w/ $\gamma = 0.5$	262	252
Fine-Tuned w/ $\gamma = 0.7$	264	255
Fine-Tuned w/ $\gamma = 0.9$	262	253

D.9 Fine-Tuning without Auxiliary Objectives

In the following results, we compare models that are fine-tuned on the downstream task with only NLL loss with models that are fine-tuned with the auxiliary objectives we detail in the paper. We establish that the auxiliary objectives are responsible for reshaping routing behavior and improving the models' throughput as the models fine-tuned with the auxiliary objectives achieve greater throughput than their vanilla fine-tuned counterparts. Fine-tuning on its own does not increase expert activation concentration or improve efficiency, making the MELINOE-specific loss functions the core of the efficiency gains that our paper highlights.

D.10 Time to First Token (TTFT) Analysis

MELINOE achieves competitive TTFT across all configurations. Increased routing concentration reduces eviction-and-reload cycles even when the overall expert working set is large while processing the whole prompt simultaneously.

Table 17: Throughput (tokens/s) comparison between models fine-tuned with and without MELINOE's auxiliary objectives ($\mathcal{L}_{cs}$ and $\mathcal{L}_{rm}$). Across both architectures and datasets, the auxiliary objectives are essential to realizing MELINOE's throughput improvements.

	OLMoE		Phi-3.5-MoE	
Method	**Dolly15K (t/s)**	**GSM8K (t/s)**	**Dolly15K (t/s)**	**GSM8K (t/s)**
Fine-Tuned with MELINOE Objectives	**23.32**	**22.16**	**16.33**	**14.89**
Fine-Tuned without MELINOE Objectives	13.74	14.28	8.48	7.97

Table 18: TTFT (ms) across competing baselines. MELINOE achieves competitive prefill latency on OLMoE and the lowest prefill latency on Phi-3.5-MoE, demonstrating that the auxiliary objectives and prefetching strategy do not introduce prefill-time overhead.

	OLMoE		Phi-3.5-MoE	
Method	**Dolly15K (ms)**	**GSM8K (ms)**	**Dolly15K (ms)**	**GSM8K (ms)**
MELINOE	578	559	**1189**	**1226**
Fiddler	**583**	**549**	1212	1091
Mixtral-Offloading	970	998	1312	1353
DeepSpeed-MoE	647	663	3014	3234
FLoE	582	594	2204	2344
MoE-Infinity	622	651	2827	3013

D.11 Alternative Training-Free Methods for Increasing Routing Locality

To demonstrate the importance of MELINOE's auxiliary fine-tuning losses, we test two methods that naïvely amplify expert biases at inference time without modifying the model weights:

- **Temperature Scaling:** Multiplying per-token router logits by a scaling factor α before applying softmax. Higher temperatures sharpen the router distribution, amplifying existing per-token expert preferences.
- **Prompt-Biased Routing:** Computing expert activation frequencies from the prompt tokens and adding α times these frequencies to per-token router logits during decoding. This tilts routing toward experts that were active during prefill, encouraging a more consistent expert keep set.

Tables 19 and 20 report throughput, generation quality (ROUGE-L on Dolly15K), and language modeling quality (PPL on WikiText-103) for both methods across two values of α, alongside MELINOE for reference.

Table 19: OLMoE comparison against naïve routing-bias methods.

	Dolly15K		WikiText-103	
Method	**Throughput (t/s)**	**ROUGE-L**	**Throughput (t/s)**	**PPL**
MELINOE	**23.32**	**0.2486**	**22.84**	9.48
Temperature ($\alpha = 1.0$)	13.47	0.1851	14.12	9.02
Temperature ($\alpha = 2.0$)	13.62	0.0995	13.72	52.96
Prompt Bias ($\alpha = 1.0$)	15.45	0.1397	15.71	11.33
Prompt Bias ($\alpha = 2.0$)	18.43	0.1037	18.02	64.85

Temperature scaling has virtually no effect on throughput because the top-K expert selection is invariant to logit scaling. Prompt-biased routing does improve throughput by concentrating activations, but at a steep quality cost. On WikiText-103, perplexity degrades from 9.02 to 64.85 for OLMoE at $\alpha = 2.0$, rendering the model unusable. We additionally observe that fine-tuning the model with entropy regularization techniques on routing probabilities encounters these same failure modes, with significant performance degradation and increased expert starvation. By contrast, the combination of

Table 20: Phi-3.5-MoE comparison against naïve routing-bias methods.

	Dolly15K		WikiText-103	
Method	**Throughput (t/s)**	**ROUGE-L**	**Throughput (t/s)**	**PPL**
MELINOE	**14.34**	**0.2270**	**11.84**	4.26
Temperature ($\alpha = 1.0$)	3.04	0.2067	2.99	4.61
Temperature ($\alpha = 2.0$)	3.08	0.1880	2.95	4.94
Prompt Bias ($\alpha = 1.0$)	3.89	0.1667	3.18	4.68
Prompt Bias ($\alpha = 2.0$)	5.68	0.0941	3.57	5.74

MELINOE's auxiliary objectives avoids these failure modes by encouraging within-sequence routing bias without impacting performance.