FlexMoE: Learning One-for-All Multi-Budget Expert Pruning for Flexible MoE Deployment

Fan Mo[1,*] fan.mo@u.nus.edu
Yuxuan Han[1,*] han_yuxuan@u.nus.edu
Geng Zhang[1,*] zhangg@comp.nus.edu.sg
Wangbo Zhao[2,†] wangbo.zhao96@gmail.com
Yang You[1,†] yangyou@nus.edu.sg

[1] **National University of Singapore**
[2] **The Hong Kong University of Science and Technology**

[*] **Equal contribution.** [†] **Corresponding authors.**

Abstract

Mixture-of-Experts (MoE) language models scale model capacity through sparse expert activation, yet deployment still requires storing and serving all experts. Moreover, deployment budgets can vary substantially across devices, users, and workloads. Existing MoE compression methods typically optimize a separate compressed model for each target budget. We introduce FlexMoE, a post-training one-for-all framework that converts a large pretrained MoE model into a multi-budget family of smaller deployable subnetworks. FlexMoE first ranks expert FFN channels by importance to construct a sliceable weight space shared across budgets, and then learns expert-wise discrete retention actions under gradually increasing cost pressure, allowing a single action-training run to produce reliable pruning masks across budgets. Moreover, only a single mid-budget fine-tune can further recover degraded performance and yields a shared recovered model that transfers to other unseen budgets, avoiding repeated per-budget recovery and separated checkpoint storage. Across tested MoE LLMs with diverse architectures and scales, FlexMoE consistently outperforms strong MoE compression baselines. In particular, on Qwen2-57B-A14B, it retains 99.8% of base performance while pruning 50% of routed-expert parameters even without recovery fine-tuning. For deployment, pruned subnetworks deliver real memory reduction and end-to-end throughput gains, while our kernel co-design enables runtime budget switching without checkpoint reloading and delivers substantial inference acceleration.

1 Introduction

Mixture-of-Experts (MoE) models scale parameter capacity while activating only a small subset of experts per token, achieving strong quality with substantially lower active computation [Lepikhin et al., 2021, Fedus et al., 2022]. This sparse-expert design has therefore become an increasingly common recipe for scaling modern foundation models.

However, sparse activation does not eliminate the deployment cost of storing and serving all experts. In practice, the available deployment budget—including memory, latency/throughput requirements, context demands, and hardware capacity—can vary substantially across users, devices, and workloads. On deployment side, one service may need long-context or agent-level reasoning under a generous budget, while another may serve lightweight chatbot requests under much stricter latency and cost

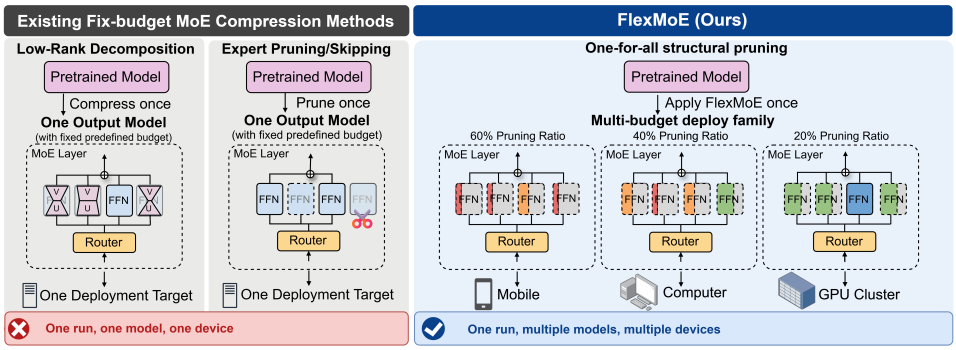


Figure 1: Existing MoE compression methods typically optimize a compressed endpoint for a target budget, whereas FlexMoE learns a shared multi-budget pruning family from one pretrained model, avoiding separate compression and checkpoints across deployment targets.

constraints [Fu et al., 2024, Gao et al., 2026, Ma et al., 2026]. MoE deployment and compression is therefore not only a model-reduction problem, but also a deployment-adaptation problem.

Existing MoE compression methods are nevertheless largely fixed-budget. Expert pruning, skipping, or merging and expert-internal compression typically optimize one compressed endpoint for a chosen target budget [Lu et al., 2024, Bai et al., 2025, Yang et al., 2024b, Gu et al., 2025, Li et al., 2025, Xu et al., 2026]. When the operating budget changes, one may need to rerun compression, reload a different model, or maintain multiple budget-specific checkpoints. This limitation becomes particularly relevant under non-stationary serving workloads, where runtime reconfiguration can be valuable [Gao et al., 2026, Qian et al., 2026].

To address this, we propose **FlexMoE**, a post-training one-for-all intra-expert pruning framework that converts one pretrained MoE model into a family of materializable subnetworks, which are nested within the full model with original size and share its weights across multiple budgets. As illustrated in Figure 2, FlexMoE first ranks expert FFN channels by importance to construct a shared sliceable weight space, and then learns expert-wise discrete retention actions under gradually increasing pruning pressure. A single action-training run thereby exports pruning masks across multiple budgets without saving actual model checkpoints. An optional single mid-budget recovery fine-tune further yields shared recovered weights reusable across the budget family. We additionally co-design the inference kernel to realize efficient online budget switching.

We evaluate FlexMoE on Mixtral-8x7B, Phi-3.5-MoE, and Qwen2-57B-A14B, where it consistently outperforms strong MoE compression baselines, and the single mid-budget recovery transfers well to unseen higher and lower budgets. In particular, on Qwen2-57B-A14B, the pruned subnet retains about 99.8% of the base performance at 50% pruning budget and still preserves about 92.9% at 80% budget. On deployment side, the resulting subnet family provides real end-to-end memory reduction, throughput gains, and enables runtime budget switching.

2 Related Work

2.1 MoE Compression

MoE compression broadly targets either expert-level redundancy or expert-internal weights. At the expert level, NAEE exploits unequal expert importance for expert pruning and skipping [Lu et al., 2024]; DiEP learns expert retention through differentiable pruning [Bai et al., 2025]; HC-SMoE clusters and merges functionally similar experts without retraining [Chen et al., 2025]; and MoNE replaces redundant expert outputs with lightweight novices[Zhang et al., 2026]. Another line compresses expert weights while largely preserving the original MoE structure: MoE-SVD and D^2-MoE perform expert-internal compression, TD-MoE further introduces cross-expert joint tensor decomposition within each layer, and MoE-I^2 combines inter-expert pruning with intra-expert decomposition [Li et al., 2025, Gu et al., 2025, Xu et al., 2026, Yang et al., 2024b]. FlexMoE is closest to this intra-expert family, but instead learns fine-grained prefix-slicing actions over a

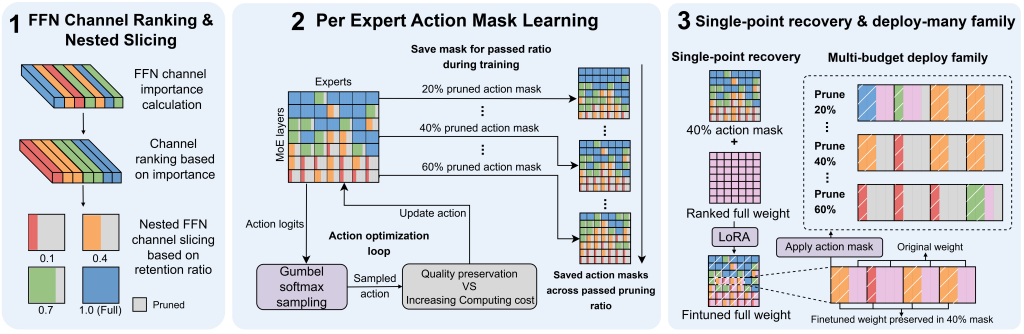


Figure 2: Overview of the FlexMoE pipeline. FlexMoE first importance-ranks expert FFN channels to construct a shared sliceable weight space across multiple retention ratios, then learns expert-wise retention (pruning) actions across pruning budgets. An optional single-point recovery fine-tune produces full recovered weights reusable across the resulting multi-budget subnet family.

shared importance-ordered expert weight space, enabling multiple deployment budgets from the same pretrained MoE while preserving its router allocation and expert topology.

2.2 Nested Subnetworks, Mask Learning, and Ranking

The broader train-once, deploy-many (one-for-all) paradigm is related to elastic and nested subnetwork training. US-Nets introduced universally slimmable networks, while MatFormer, Flextron, and AmoebaLLM extended nested or many-in-one parameter sharing to transformers and large language models [Yu and Huang, 2019, Devvrit et al., 2024, Cai et al., 2024, Fu et al., 2024]. More recent MoE methods introduce elasticity through coarse-to-fine expert ranking or slimmable expert widths [Wang et al., 2026, Tastan et al., 2026]. In contrast, FlexMoE starts from an already pretrained MoE LLM and derives its multi-budget subnet family at the post-training stage.

FlexMoE's shared expert slicable weight space also builds on differentiable model structure learning and gradient-based importance ranking. MaskLLM and Gumbel-Softmax provide tools for gradient-based discrete structural decisions [Fang et al., 2024, Jang et al., 2017], while RECAP and earlier Taylor criteria use grouped first-order saliency for structured pruning [Ilhan et al., 2024, Molchanov et al., 2019]. We combine these ingredients to learn per-expert prefix-slicing actions over importance-ordered FFN channels.

3 Method

We consider a pretrained MoE LLM with L MoE layers and E experts per layer. For a hidden state h routed to expert e in layer l, its FFN is

$$\mathrm{FFN}_{l,e}(h) = W_{l,e}^{\mathrm{down}}\Big(\phi(W_{l,e}^{\mathrm{gate}}h) \odot W_{l,e}^{\mathrm{up}}h\Big), \tag{1}$$

where d_{ff} denotes the intermediate FFN width.

3.1 Intra-Expert Channel Importance Ranking

Directly applying channel prefix slicing to pretrained experts is brittle. We therefore perform a one-time reordering step that converts each expert FFN into an importance-ordered layout, so that prefixes preserve more important hidden-channels under prefix-slice pruning.

For expert (l, e), let $\Theta_{l,e,j}$ denote all parameters associated with hidden channel j: the corresponding rows in expert weights $W_{l,e}^{\mathrm{gate}}$ and $W_{l,e}^{\mathrm{up}}$ and the matching column of $W_{l,e}^{\mathrm{down}}$. Since FFN hidden channels are permutation-invariant when the same permutation is applied consistently to these projections [Navon et al., 2023], they can be reordered without changing the calculation correctness and therefore the full expert function.

We rank channels using grouped first-order Taylor saliency [Molchanov et al., 2019, Ilhan et al., 2024]. On calibration sample batches $\mathcal{B}_{\text{rank}}$, the importance score of channel j is

$$s_{l,e,j} = \frac{1}{|\mathcal{B}_{\text{rank}}|} \sum_{b \in \mathcal{B}_{\text{rank}}} \sum_{\theta \in \Theta_{l,e,j}} \left(\theta \frac{\partial \mathcal{L}^{(b)}}{\partial \theta} \right)^2 . \tag{2}$$

Intuitively, $s_{l,e,j}$ estimates the extent of param-group $\Theta_{l,e,j}$ (channel j) affects the loss $\mathcal{L}$. Sorting $s_{l,e,j}$ in descending order gives permutation matrix $P_{l,e}$, which is applied consistently across the FFN hidden dimension:

$$\widetilde{W}^{\text{gate}}_{l,e} = P_{l,e} W^{\text{gate}}_{l,e}, \quad \widetilde{W}^{\text{up}}_{l,e} = P_{l,e} W^{\text{up}}_{l,e}, \quad \widetilde{W}^{\text{down}}_{l,e} = W^{\text{down}}_{l,e} P^{\top}_{l,e}. \tag{3}$$

After reordering, a retention ratio $r \in (0, 1]$ corresponds to keeping only the top $k = \lceil r \, d_{\text{ff}} \rceil$ channels in all ranked channels. Let $m(r)$ denote the corresponding prefix mask. The sliced expert FFN is

$$\text{FFN}^{(r)}_{l,e}(h) = \widetilde{W}^{\text{down}}_{l,e} \Big(m(r) \odot \phi(\widetilde{W}^{\text{gate}}_{l,e} h) \odot \widetilde{W}^{\text{up}}_{l,e} h \Big). \tag{4}$$

During slice action learning, masks are only used to simulate channel removal and mark action training results (which channels are retained), while all ranked model weights remain unchanged; at deployment, masked channels can instead be truly removed or excluded from computation. The resulting importance-ordered layout forms a shared prefix-sliceable weight space from which different expert widths can be selected across pruning budgets.

3.2 Action Mask Learning

Given the ranked experts, we learn one discrete retention action per expert from predefined ratio set $\mathcal{A} = \{r_1, \ldots, r_K\}$, where $r_k \in (0, 1]$. For every expert (l, e), we maintain trainable action logits $\alpha_{l,e} \in \mathbb{R}^K$. To sample subnetworks under one-hot operation while preserving gradients for action logits training, we use Straight-Through Gumbel-Softmax [Jang et al., 2017, Fang et al., 2024]:

$$z^{\text{soft}}_{l,e} = \text{softmax}\left(\frac{\alpha_{l,e} + g_{l,e}}{\tau} \right), \qquad z^{\text{hard}}_{l,e} = \text{one_hot}\left(\arg\max_k z^{\text{soft}}_{l,e,k} \right). \tag{5}$$

where $g_{l,e,k} \sim \text{Gumbel}(0, 1)$ are added to the original action distribution as action noise sampling, and introduce a linearly annealed τ to control the degree of impact of Gumbel noise to promote early stage action exploration. The sampled expert subnet forward pass uses the straight-through estimator

$$\tilde{z}_{l,e} = \text{sg}\big[z^{\text{hard}}_{l,e} - z^{\text{soft}}_{l,e} \big] + z^{\text{soft}}_{l,e}, \tag{6}$$

where $\text{sg}(.)$ means stop gradient. $\tilde{z}_{l,e}$ ensures actions are discrete in the forward pass but differentiable in the backward pass. All straight-through estimator $\tilde{z}$ jointly defines a sampled action mask during training stage which can be applied to each expert and produce a sampled subnet nested in full ranked MoE model.

Importantly, we freeze all pretrained model weights during action learning and optimize only the action logits. This is essential to our one-for-all design: if model weights were updated together with the pruning actions, masks exported at different points of the training trajectory would correspond to different underlying weight states, breaking the shared-weight property across budgets. By keeping the ranked weights fixed, all exported action masks are optimized and will operate on the same parameter set, allowing a single action-training run to produce a consistent multi-budget subnet family nested in full ranked MoE model.

We optimize the action logits using

$$\mathcal{L}_{\text{action}} = \mathcal{L}_{\text{qual}} + \lambda_{\text{cost}}(t) \sum_{l=1}^{L} \sum_{e=1}^{E} q_{l,e} \sum_{k=1}^{K} p_{l,e,k} r_k - \beta(t) \mathcal{H}(p), \tag{7}$$

$\mathcal{L}_{\text{qual}}$ combines the student language-modeling cross-entropy with teacher–student KL distillation from the frozen full model. $p_{l,e} = \text{softmax}(\alpha_{l,e})$ denotes the currently learned clean action distribution without Gumbel sampling noise and τ scaling. $q_{l,e}$ is the frequency of assigning tokens to expert

(l, e) in this MoE layer. This cost term is both parameter and load weighted: assigning thicker actions to highly routed experts preserves task accuracy, but also brings more computation and incurs larger cost penalties, so actions must learn an expert-wise accuracy–efficiency trade-off. We further include an entropy regularizer $\mathcal{H}(p)$ computed from all $p_{l,e}$. It's the mean action categorical entropy over all experts averaged across layers and experts. This term encourages exploration, prevents premature collapse to a single action, and helps the model discover more reliable MoE subnets. Entropy weight $\beta(t)$ is also annealed linearly.

Action logits are initialized as the fully retained action. We gradually increase $\lambda_{\text{cost}}(t)$ so that the learned policy moves from thicker to thinner experts and traverses progressively stronger pruning budgets. At any checkpoint, the training have:

$$\hat{r}_{l,e} = r_{\arg\max_k \alpha_{l,e,k}}, \qquad \hat{\mathcal{M}} = \{\hat{r}_{l,e}\}_{l,e}, \qquad \hat{\rho} = 1 - \frac{1}{LE}\sum_{l=1}^{L}\sum_{e=1}^{E}\hat{r}_{l,e}. \tag{8}$$

Here, $\hat{r}_{l,e}$ is the currently trained deterministic expert action, $\hat{\mathcal{M}}$ specifies one retained prefix mask for each expert and $\hat{\rho}$ is the global fraction of routed-expert parameters pruned. By saving masks $\hat{\mathcal{M}}$ along this optimization trajectory, FlexMoE obtains a sequence of pruned subnetworks across budgets in one training run, and marked by trained masks without actually store separated model checkpoints.

FlexMoE's Kernel Co-Design. Shared-weight online budget switching adds two runtime costs absent from statically pruned subnetworks: the expert-specific retained widths make the expert FFNs irregular, fragmented GEMMs, and reconstructing each clipped weight by slicing and concatenating the gate–up tensor adds per-call tensor manipulation and host-side dispatch. During decoding these are dominated by kernel-launch and scheduling overhead that does not shrink with the reduced expert width, so the theoretical FLOP savings of pruning stay hidden. We therefore co-design a clipped-FFN kernel that (i) stores the gate–up weights interleaved and the down projection transposed, so a retained prefix weight is selected by loop bounds and masking rather than any weight slicing, copy or concatenation, and (ii) turns the routing into fixed-shape, device-resident block metadata and reads each expert's retained width from a device buffer inside the kernel, pruning on-device—skipping pruned channel blocks and clamping the contraction loop—while the launch shape stays constant. Eliminating every host synchronization and data-dependent branch makes the whole expert forward capturable in a single CUDA graph; decoding then becomes compute-bound, so the smaller active width turns into real speedup, and switching the budget only overwrites the tiny device width buffer, requiring no weight reload and no graph re-capture. See Appendix B for implementation details.

3.3 Recovery Fine-Tuning

Parameter pruning can still degrade model quality, while independently recovering every budget would undermine the one-for-all objective. We therefore choose one mid-budget mask $\mathcal{M}^{\text{mid}}$ and perform a single-point recovery fine-tune. The importance-ranked base weights W_0 are frozen, and LoRA adapters [Hu et al., 2022] with parameters φ are trained on the retained expert FFN channels chosen by $\mathcal{M}^{\text{mid}}$ at this budget.

Let $o^S_{\text{mid}} = f_{\text{stu}}(x; \mathcal{M}^{\text{mid}}, W_0, \varphi)$ denote the outputs from LoRA-augmented subnet materialized by mask $\mathcal{M}^{\text{mid}}$ and $o^F = f_{\text{tea}}(x; W_0)$ denote the full-model teacher outputs, respectively. We optimize

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{rec}}}\left[\lambda_{\text{task}}\text{CE}(o^S_{\text{mid}}, y) + \lambda_{\text{kl}}\text{KL}(o^F \| o^S_{\text{mid}})\right]. \tag{9}$$

to let the pruned sub-network mimic the full teacher model. After training, the adapters are merged into the ranked base model to obtain one recovered model $\overline{W}$. Because all exported masks select prefixes from the same importance-ordered expert weight space while preserving the router and expert topology, the same $\overline{W}$ can be reused with every budget mask $\mathcal{M}^{(b)}$ without additional recovery fine-tuning or budget-specific checkpoints. Thus, the one-for-all property extends from action learning to recovery.

4 Experiments

4.1 Experimental Setup

Implementation and Evaluation Details. We use three pretrained MoE LLMs: Mixtral-8x7B [Jiang et al., 2024], Phi-3.5-MoE [Abdin et al., 2024] and Qwen2-57B-A14B [Yang et al.,

Prune	Method	Mixtral-8×7B								Phi-3.5-MoE	Qwen2-57B-A14B
		ARC-c	ARC-e	HellaS	OBQA	PIQA	WinoG	MathQA	Avg. (SD)	Avg. (SD)	Avg. (SD)
0%	Base model	57	84	65	36	82	76	43	63.29	62.00	58.71
20%	NAEE2024	47	76	58	32	79	72	40	57.71	N/A	55.86
	MoE-I^{2}2024b	48	79	55	32	78	74	37	57.57	N/A	N/A
	MoE-SVD*2025	50	80	60	32	80	71	35	58.29	60.57	58.43
	TD-MoE*2026	47	77	56	28	79	74	36	56.71	58.43	58.00
	FlexMoE	52.00 ±2.00	80.00 ±2.00	63.67 ±0.58	35.67 ±0.58	80.00 ±1.73	77.00 ±1.00	40.00 ±1.00	61.19 ±0.92	60.24 ±0.36	58.81 ±0.16
	FlexMoE(SP-XFER)	53.67 ±1.53	81.33 ±1.15	64.33 ±0.58	35.67 ±0.58	80.33 ±1.15	77.00 ±1.00	40.00 ±1.00	**61.76** ±0.86	**63.29** ±0.00	**60.24** ±0.7
40%	NAEE2024	36	63	46	25	72	64	35	48.71	57.57	53.14
	MoE-I^{2}2024b	38	71	43	26	69	66	31	49.14	45.29	N/A
	MoE-SVD*2025	38	72	49	27	75	65	27	50.43	55.57	54.86
	TD-MoE*2026	30	57	39	20	66	62	24	42.57	55.14	55.43
	FlexMoE	45.00 ±1.00	75.67 ±0.58	55.33 ±1.53	31.67 ±1.53	78.33 ±0.58	71.33 ±0.58	31.00 ±1.00	55.48 ±0.87	55.43 ±0.43	58.86 ±0.14
	FlexMoE(SP-FT)	48.33 ±1.15	77.00 ±1.00	60.00 ±1.00	33.67 ±1.15	79.67 ±0.58	73.33 ±0.58	33.67 ±0.58	**57.95** ±0.5	**59.62** ±0.33	**60.00** ±0.42
60%	NAEE2024	23	42	33	17	62	55	26	36.86	N/A	44.00
	MoE-I^{2}2024b	22	44	32	18	58	55	23	36.00	N/A	N/A
	MoE-SVD*2025	28	60	40	22	66	62	25	43.29	50.00	51.71
	TD-MoE*2026	22	37	29	16	58	54	22	34.00	46.14	51.43
	FlexMoE	31.33 ±1.53	61.33 ±2.31	46.33 ±0.58	23.00 ±1.00	73.33 ±0.58	65.00 ±2.65	26.00 ±1.00	46.62 ±0.81	50.43 ±0.25	58.62 ±0.08
	FlexMoE(SP-XFER)	34.33 ±2.08	64.33 ±3.05	50.33 ±1.15	23.33 ±0.58	73.67 ±1.53	70.33 ±0.58	26.33 ±1.53	**48.95** ±0.84	**52.81** ±0.22	**59.76** ±0.58

Table 1: Zero-shot accuracy (%) across seven downstream tasks. FlexMoE results report mean ± sample standard deviation over three independent full-pipeline runs. * denotes locally reproduced method results. We apply single-point recovery fine-tuning (SP-FT) at the 40% budget; the 20% and 60% results reuse the same recovered weights (SP-XFER) without additional fine-tuning.

2024a] as cross-model validation with different MoE architectures and sizes. We defined discrete action set as $\mathcal{A} = \{0.1, 0.4, 0.7, 1\}$, and throughout this section, the pruning ratio (budget) is defined by the global prune budget $\hat{\rho}$ in Eq. (8) of an action mask applied to the ranked full MoE model. During action learning, we export one action mask whenever expert parameters globally pruned by roughly 1% ($\hat{\rho}$ increase 1%). For recovery fine-tuning, we choose the 40% budget action mask as the recovery point, and let recovered ability transfer to other budgets (20% and 60% here). Channel importance ranking, action learning, and recovery fine-tuning stages use Zyda-2 [Tokpanov et al., 2024] as the calibration dataset. As quality evaluation, we report zero-shot accuracy on seven widely used reasoning benchmarks implemented with `lm-eval-harness`: ARC-Challenge, ARC-Easy, HellaSwag, OpenBookQA, PIQA, WinoGrande, and MathQA [Gao et al., 2021, Clark et al., 2018, Zellers et al., 2019, Mihaylov et al., 2018, Bisk et al., 2020, Sakaguchi et al., 2020, Amini et al., 2019]. For each model, we independently rerun FlexMoE pipeline three times using different random seeds to account for randomness in calibration-sample selection and parameter initialization, and report the mean and sample standard deviation. More hyper-parameters and reproduction details please see Appendix C.2.

Baselines. We fix the baseline family throughout the main comparison. Our primary baseline methods are MoE-SVD and TD-MoE, since both are latest strong pretrained MoE compression methods [Li et al., 2025, Xu et al., 2026]. We additionally report NAEE and MoE-I^2 as broader references, representing expert-level pruning/skipping and mixed inter-/intra-expert compression, respectively [Lu et al., 2024, Yang et al., 2024b]. For fair comparison, we locally reproduced the two strongest baseline methods, MoE-SVD and TD-MoE on all 3 backbones by using same Zyda-2 calibration dataset, same local `lm-eval-harness` protocal over 7 tasks. To avoid selective reporting, we use a shared comparison grid centered on 20%, 40%, and 60% prune budgets, and include each baseline whether the corresponding model–budget pair is publicly available or we locally reproduced. When a baseline is still unavailable for a given model–budget pair, we marked as "N/A".

4.2 Main Results and Analysis

Task Accuracy Results Analysis. Across models, the main practical pattern is consistent: FlexMoE achieves competitive or superior pruning quality without recovery across most evaluated settings. With a single shared recovery fine-tuning run, it consistently surpasses all strong baselines, and the repeated runs support that FlexMoE preserves quality with varied initialization and sample

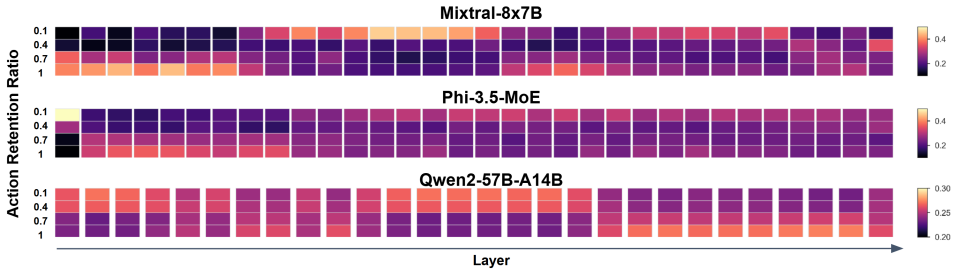


Figure 3: Layer-wise action distributions learned by FlexMoE at 40% prune budget. Lighter cell color indicate the higher fraction of experts in that layer assigned to the corresponding action.

batches. For recovery strategy, the shared weight fine-tuned at the 40% point transferred recovery to other (20% and 60%) budget points and achieves higher task accuracy. Observing the excellent performance of Qwen2, in addition to comparing it with the baseline, we also tested the performance on Qwen2 at compression ratios of 50% and 80% (results see appendix Table 7). Surprisingly, without fine-tuning recovery, average scores show that it remains nearly lossless at a 50% parameter prune budget ($\sim 99.8\%$ of base performance), even at an 80% prune budget, it still retains about 92.9% of the base average score. These results demonstrate the effectiveness of FlexMoE in pruning quality preservation. These results confirming that our proposed expert-wise action learning captures meaningful architecture-dependent structure in MoE layers rather than random slicing allocation and budget compression.

Effectiveness of Action Learning. Figure 3 and Figure 4 jointly show that the learned action distributions are clearly non-uniform across experts, layers and models, indicating that FlexMoE does not merely prune experts, but learns structured budget allocation. Table 3 shows results for further ablation experiments of slicing action assignment. The learned action assignment achieves the highest mean accuracy across 4 model-budget settings, outperforming all uniform, Gaussian-random, globally shuffled, or in-layer-shuffled action assignment control. Since global shuffling preserves the overall prune ratio and in-layer shuffling additionally preserves layer-wise capacity, their degradation indicating that destroying learned action structure consistently hurts performance, and expert-specific assignments matter. These results strongly confirm that our proposed expert-wise action learning captures meaningful quality-dependent structure rather than random budget convergence and allocation, which is the key to FlexMoE maintaining strong quality across pruning budgets.

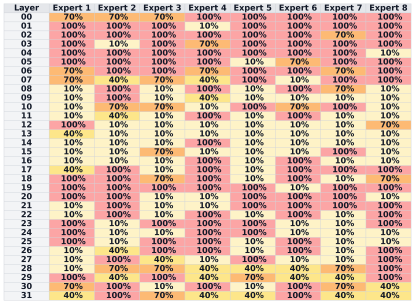
Effectiveness of Single-Point Fine-Tuning. Table 2 shows that the quality gains obtained by SP-FT transfer effectively to both tighter and looser pruning budgets. This transfer follows from the shared importance-ordered prefix structure: tighter budget action masks retain smaller subsets of the channels updated in the recovery fine-tuning, whereas looser masks preserve these recovered prefixes while additionally exposing more tail channels not attended in recovery.

We also present an ablation of SP-FT at different budget points. Among three recovery points, SP-FT at 40% achieves the highest cross-budget average score compared with recovery at 20% and 60%. More broadly, the shared model recovered at 40% slightly exceeds the average performance of independently fine-tuning each budget, while requiring only one recovery run and one recovered weight set. Thus, a single point recovery fine-tune produces one shared weight that remains strong quality across the budget trajectory, avoiding repeated per-budget recover and the storage of separate budget-specific weights. Together with one-run slice action learning, this directly supports FlexMoE's train-once, deploy-many objective.

Table 2: Cross-budget transfer ability of SP-FT recovered weights on Mixtral-8×7B. Values are seven-task average accuracies (%). The last column independently fine-tunes at each test budget ratio.

Test Ratio	SP-FT at 20%	SP-FT at 40%	SP-FT at 60%	Independent per-budget FT
20%	60.86	**61.86**	60.71	60.86
40%	54.57	**58.00**	57.14	**58.00**
60%	42.86	49.57	**50.29**	**50.29**
Average	52.76	**56.48**	56.05	56.38

Figure 4: Learned action mask for Mixtral-8×7B at 40% pruning. Each cell denotes one expert's selected retention ratio.



Layer	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5	Expert 6	Expert 7	Expert 8
00	70%	70%	70%	100%	100%	100%	100%	100%
01	100%	100%	100%	10%	100%	100%	100%	100%
02	100%	100%	100%	100%	100%	100%	70%	100%
03	100%	10%	100%	70%	100%	100%	100%	100%
04	100%	100%	100%	100%	100%	100%	100%	10%
05	100%	100%	100%	100%	10%	70%	100%	100%
06	70%	100%	100%	70%	100%	100%	70%	100%
07	70%	40%	70%	40%	100%	10%	100%	100%
08	10%	100%	10%	100%	10%	100%	70%	10%
09	10%	100%	10%	40%	10%	10%	10%	10%
10	10%	70%	70%	10%	100%	70%	100%	10%
11	10%	40%	10%	100%	10%	100%	10%	10%
12	100%	10%	10%	10%	10%	10%	10%	70%
13	40%	10%	10%	10%	10%	10%	10%	10%
14	10%	10%	10%	100%	10%	10%	10%	10%
15	10%	10%	70%	10%	10%	10%	100%	10%
16	10%	10%	10%	100%	10%	100%	10%	10%
17	40%	100%	10%	100%	10%	100%	100%	100%
18	100%	100%	70%	100%	10%	100%	10%	70%
19	100%	100%	100%	100%	100%	10%	100%	100%
20	100%	100%	10%	10%	100%	100%	100%	10%
21	10%	100%	10%	10%	100%	100%	100%	100%
22	10%	100%	10%	100%	10%	100%	10%	100%
23	100%	10%	100%	100%	100%	10%	10%	100%
24	100%	10%	10%	100%	100%	10%	10%	10%
25	100%	10%	100%	100%	10%	100%	10%	10%
26	10%	40%	100%	100%	10%	100%	10%	100%
27	10%	100%	40%	10%	100%	10%	10%	100%
28	10%	70%	70%	40%	40%	40%	70%	100%
29	100%	40%	100%	40%	70%	40%	40%	100%
30	70%	100%	10%	100%	10%	100%	70%	40%
31	40%	100%	70%	40%	40%	100%	40%	40%

Table 3: Expert-wise action-allocation ablation. Values are seven-task average zero-shot accuracies (%). ± denotes sample SD;

Allocation	**Mixtral-8×7B**		**Qwen2-57B-A14B**	
	20%	**60%**	**20%**	**60%**
Gaussian random	60.00 ±0.68	44.14 ±2.08	57.86 ±0.94	57.00 ±1.21
Uniform assign	60.14	45.14	58.14	57.43
Learned shuffled	59.14 ±0.55	44.14 ±1.84	55.71 ±0.46	56.14 ±0.57
In-layer shuffled	60.43 ±0.44	46.00 ±0.21	58.29 ±0.13	58.43 ±0.0
Learned	61.19 ±0.92	46.62 ±0.81	58.81 ±0.16	58.62 ±0.08

Table 4: End-to-end serving throughput and memory allocation of statically materialized FlexMoE subnetworks under a fixed GPU-memory budget.

	Mixtral-8×7B			**Phi-3.5-MoE**			**Qwen2-57B-A14B**		
Pruning Ratio	tok/s	ckpt size (GB)	KV (GB)	tok/s	ckpt size	KV	tok/s	ckpt size	KV
0%	8276 ±67	87.03	24.26	8226 ±112	78.05	33.24	1302 ±9	107.02	4.27
20%	10478 ±115	70.13	41.15	9761 ±128	63.08	48.21	4455 ±18	88.62	22.67
40%	11972 ±107	53.56	57.73	11341 ±134	48.05	63.24	5214 ±32	70.21	41.08
60%	12296 ±135	37.09	74.20	11802 ±148	36.61	74.68	5185 ±44	51.95	59.34

4.3 Deployment Performance and Analysis

We finally evaluated deployment performance of FlexMoE's pruned models under 2 scenarios: Fixed hardware budget and under MoE kernel co-design. We evaluate these two settings separately to distinguish their sources of throughput improvement. We use the SGLang engine [Zheng et al., 2024] on a single H200 GPU under a synthetic workload with 4096 prompt requests, input length = 64, output length = 256. Our primary metric is model throughput (tok/s), defined by summarizing the model prefill input and decoding output throughput. Results are averaged across multiple runs ± sample standard deviation, and speedups are normalized to the unpruned model within each execution path.

Fixed hardware budget. For deployments with a fixed GPU-memory budget, we materialize each target subnetwork by physically removing the pruned expert channels at that budget. This provides three coupled benefits: it reduces the resident weight footprint, decreases the FLOPs and weight traffic of active experts, and allows the serving engine to allocate the released memory to additional KV-cache slots. The resulting throughput therefore represents the practical end-to-end benefit available under the same deployment (GPU-memory) budget. As shown in Table 4, static pruned model consistently improves throughput across all three architectures.

This fixed-memory-budget experiment reflects realistic deployment: parameter memory released through compression is naturally available to the serving engine for KV-cache allocation. Thus, the KV-cache benefit is not an unrelated artifact, but an inherent deployment advantage of reducing the checkpoint footprint.

Online budget switching with the co-designed kernel. We next isolate the computational benefit of activating thinner experts: the full importance-ordered weights stay resident and the per-expert retained widths are selected at runtime, so the resident model size, KV-cache capacity, and memory headroom are identical across budgets. A direct Python implementation cannot realize the FLOP savings—during decoding the step is dominated by kernel launch and host-side scheduling that does not shrink with the retained width (Table 5, *naive*). Our co-designed kernel removes every host synchronization and data-dependent host branch, so the entire expert forward is captured in a single

Table 5: Inference throughput with FlexMoE co-designed kernel enabled, compared with naive Python inference implementation on Mixtral.

Pruning	Mixtral-8×7B			Phi-3.5-MoE		Qwen2-57B-A14B	
Ratio	Naive Python tok/s	Kernel tok/s	speedup×	Kernel	speedup	Kernel	speedup
0%	7780	5818 ±31	1.00×	10639 ±100	1.00×	7367 ±30	1.00×
20%	7345	6860 ±52	1.18×	12395 ±110	1.17×	9093 ±28	1.23×
40%	7087	8222 ±43	1.41×	15286 ±135	1.44×	10744 ±36	1.46×
60%	6649	11895 ±73	**2.04**×	18319 ±166	**1.72**×	13018 ±45	**1.77**×

Table 6: Cost of switching the serving budget (Qwen2-57B-A14B, H200, bf16). Our in-place width-buffer swap replaces checkpoint reload and graph re-capture at 10^4–$10^6\times$ lower latency, while keeping the shared weights resident and outputs bit-identical. Reload is a lower bound (weight transfer and reload only under our cluster at a cross-node sequential checkpoint read bandwidth of ~ 10.4 GB/s and a host-to-GPU transfer bandwidth of ~ 53.2 GB/s.

budget switch mechanism	Switch latency	Reloads weights	Re-captures graph
Reload materialized per-budget checkpoint	6.4–13.2 s	yes	yes
Re-capture CUDA graph per budget	108 ms	no	yes
In-place width-buffer swap (ours)	**4.2** μs	no	no

CUDA graph and replayed; decoding then becomes compute-bound and the reduced active width turns into real speedup. As shown in Table 5, the kernel yields consistent throughput gains on all three backbones as expert parameter reduce.

Runtime budget switching is instant and almost free. Because the kernel reads each expert's retained width from a device buffer and its launch shape is width-independent, a single captured graph serves every budget: switching the budget only overwrites that $O(E \cdot L)$ width buffer (~ 7 KB). This costs $4.2\ \mu s$—four orders of magnitude below re-capturing the graph (108 ms) and six below reloading a materialized per-budget checkpoint (6.4–13.2 s; Table 6). The switched graph is bit-identical to a graph captured natively at that budget (max. abs. difference 0; and $\leq 8 \times 10^{-6}$ against an eager reference) and stays within 0.7% of its latency, so this flexibility costs neither accuracy nor speed, while the MoE decode step itself shrinks by up to $2.37\times$ as the budget drops. As shown in Figure 5, under the kernel, decoding step latency tracks the scheduled serving budget instantly and reversibly at fixed resident memory, with no extra graph re-capture and weight reload cost (Qwen2-57B-A14B, H200).

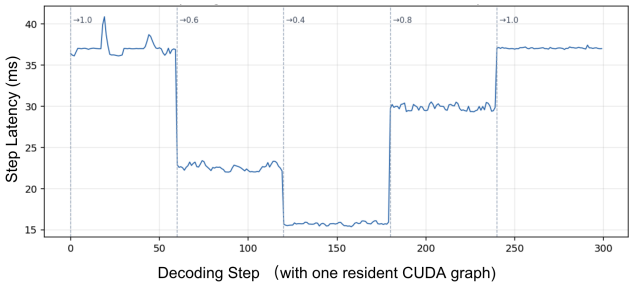


Figure 5: MoE decode-step latency under co-designed kernel with the serving budget scheduled as $1.0 \to 0.6 \to 0.4 \to 0.8 \to 1.0$, 60 steps each budget.

5 Conclusion

We presented FlexMoE, a post-training compression framework that converts a pretrained MoE LLM into a family of deployable subnetworks nested in the full model. By ranking expert FFN channels by estimated importance and learning per-expert one discrete retention action across budgets, our method obtains a series of reliable pruned subnetworks nested in large pretrained MoE from a single action-training run. We further showed that a one-step recovery fine-tune at a single mid-budget point is already sufficient to produce shared recovered model that transfers well to other unseen budgets. Experiments on various MoE models show that the proposed framework surpasses recent strong MoE compression baselines. Finally, offline pruned subnetworks deliver real throughput gains at

their fixed budget, and a kernelized co-design makes runtime budget switching feasible. We hope this work provides a new perspective of MoE model structure search and a practical foundation for budget-adaptive MoE model deployment and inference.

Limitations and Future Works. While our main focus is MoE structure search and static expert-parameter pruning, the multi-budget shared-weight family produced by FlexMoE opens a promising direction for future work on stronger system-level co-design and input-aware dynamic budget-adaptation strategies for deployment and online serving.

Acknowledgments and Disclosure of Funding

We would like to acknowledge that computational work involved in this research work is supported by NUS IT's Research Computing group using grant numbers CFP03-CF-088. Yang You's research group is being sponsored by ByteDance grant, Alibaba grant, Meta Research Award, AliCloud grant, Apple grant, Google Research Award, Microsoft Research grant, and Adobe gift.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, 2019. doi: 10.18653/v1/N19-1245.

Sikai Bai, Haoxi Li, Jie Zhang, Zicong Hong, and Song Guo. DiEP: Adaptive mixture-of-experts compression through differentiable expert pruning. In *Advances in Neural Information Processing Systems*, 2025.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5298–5311, 2024.

I-Chun Chen, Hsu-Shen Liu, Wei-Fang Sun, Chen-Hao Chao, Yen-Chang Hsu, and Chun-Yi Lee. Retraining-free merging of sparse MoE via hierarchical clustering. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 8594–8620. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25aq.html.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. Matformer: Nested transformer for elastic inference. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.5555/3737916.3742377.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. In *Advances in Neural Information Processing Systems*, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Yonggan Fu, Zhongzhi Yu, Junwei Li, Jiayi Qian, Yongan Zhang, Xiangchi Yuan, Dachuan Shi, Roman Yakunin, and Yingyan Celine Lin. Amoeballm: Constructing any-shape large language models for efficient and instant deployment. In *Advances in Neural Information Processing Systems*, 2024.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. Zenodo, September 2021.

Shouwei Gao, Junqi Yin, Feiyi Wang, and Wenqian Dong. FLYING SERVING: On-the-fly parallelism switching for large language model serving. In *Proceedings of the 40th ACM International Conference on Supercomputing*, ICS '26, pages 17–29. Association for Computing Machinery, 2026. doi: 10.1145/3797905.3800525. URL https://doi.org/10.1145/3797905.3800525.

Hao Gu, Wei Li, Lujun Li, Qiyuan Zhu, Mark G. Lee, Shengjie Sun, Wei Xue, and Yike Guo. Delta decompression for MoE-based LLMs compression. In *International Conference on Machine Learning*, 2025.

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.

Fatih Ilhan, Gong Su, Selim Furkan Tekin, Tiansheng Huang, Sihao Hu, and Ling Liu. Resource-efficient transformer pruning for finetuning of large models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16206–16215, 2024.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

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.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Wei Li, Lujun Li, Hao Gu, You-Liang Huang, Mark G. Lee, Shengjie Sun, Wei Xue, and Yike Guo. MoE-SVD: Structured mixture-of-experts LLMs compression via singular value decomposition. In *International Conference on Machine Learning*, 2025.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6159–6172, 2024. doi: 10.18653/v1/2024.acl-long.334.

Xinyue Ma, Heelim Hong, Taegeon Um, Jongseop Lee, Seoyeong Choy, Woo-Yeon Lee, and Myeongjae Jeon. Orbitflow: Slo-aware long-context llm serving with fine-grained kv cache reconfiguration. *arXiv preprint arXiv:2601.10729*, 2026.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, 2018. doi: 10.18653/v1/D18-1260.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning (ICML)*, 2023.

Shangshu Qian, Kipling Liu, P. C. Sruthi, Lin Tan, and Yongle Zhang. Towards resiliency in large language model serving with kevlarflow. *arXiv preprint arXiv:2601.22438*, 2026.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020.

Nurbek Tastan, Stefanos Laskaridis, Karthik Nandakumar, and Samuel Horváth. MoSE: Mixture of slimmable experts for efficient and adaptive language models. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 119816–119836. PMLR, 2026. URL https://proceedings.mlr.press/v306/tastan26a.html.

Yury Tokpanov, Paolo Glorioso, Quentin Anthony, and Beren Millidge. Zyda-2: a 5 trillion token high-quality dataset. *arXiv preprint arXiv:2411.06068*, 2024.

Yaoxiang Wang, Qingguo Hu, Yucheng Ding, Ruizhe Wang, Yeyun Gong, Jian Jiao, Yelong Shen, Peng Cheng, and Jinsong Su. Training matryoshka mixture-of-experts for elastic inference-time expert utilization. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 33093–33111, Budapest, Hungary, 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.emnlp-main.1449/.

Yuebin Xu, Yanhong Wang, Xuemei Peng, Hui Zang, Minghao Chen, Pengfei Xia, and Zeyi Wen. TD-MoE: Tensor decomposition for MoE models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=D9cnZNZfxX.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

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. *arXiv preprint arXiv:2505.09388*, 2025. doi: 10.48550/arXiv.2505.09388. URL https://arxiv.org/abs/2505.09388.

Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. MoE-I^2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10456–10466, 2024b. doi: 10.18653/v1/2024.findings-emnlp.612.

Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, 2019. doi: 10.18653/v1/P19-1472.

Geng Zhang, Yuxuan Han, Yuxuan Lou, Yiqi Zhang, Wangbo Zhao, and Yang You. MoNE: Replacing redundant experts with lightweight novices for structured pruning of MoE. In *International Conference on Learning Representations*, pages 39983–40009, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/431c7c0b59aeafa2182b4d6617884b53-Abstract-Conference.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pages 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

A Extended Experimental Results, Ablations and Discussion

A.1 Full Cross-Model Results, Baseline Comparability, and Multi-Budget Trajectories

Table 7 reports the complete cross-model full results among 7 used down-stream tasks. In addition to the main-text model–budget pairs, 20%, 40%, and 60% points reported in the main text, we also report additional intermediate budgets available from the same FlexMoE pruning trajectory using seed 42, to show FlexMoE's robustness across the entire budget trajectory. And we retain source-reported baseline results from their original papers for reference, paired with our locally reproduced MoE-SVD and TD-MoE results.

Table 7: Full zero-shot accuracy (%) across seven downstream tasks. † denotes publicly available baseline results retained for reference, while * denotes our local reproduction under the unified calibration and evaluation setting. Repeated FlexMoE results report mean ± sample standard deviation over three independent full-pipeline runs. SP-FT denotes recovery at the 40% budget and SP-XFER reuses the same recovered weights at another budget. CP-FT denotes the cross-point fine-tuning ablation, detailed in Appendix A.5.

Ratio	Method	ARC-c	ARC-e	HellaS	OBQA	PIQA	WinoG	MathQA	Avg.
	Mixtral-8×7B (8 experts per layer, top-2 activated per token)								
0%	Base model	57	84	65	36	82	76	43	63.29
20%	NAEE†	47	76	58	32	79	72	40	57.71
	MoE-I²†	48	79	55	32	78	74	37	57.57
	MoE-SVD†	55	80	61	33	81	73	38	60.14
	MoE-SVD*	50	80	60	32	80	71	35	58.29
	TD-MoE†	53	83	64	33	82	77	40	61.71
	TD-MoE*	47	77	56	28	79	74	36	56.71
	FlexMoE	52.00 ±2.00	80.00 ±2.00	63.67 ±0.58	35.67 ±0.58	80.00 ±1.73	77.00 ±1.00	40.00 ±1.00	61.19 ±0.92
	Ours (CP-FT)	53	81	62	33	81	76	37	60.43
	FlexMoE (SP-XFER)	53.67 ±1.53	81.33 ±1.15	64.33 ±0.58	35.67 ±0.58	80.33 ±1.15	77.00 ±1.00	40.00 ±1.00	61.76 ±0.86
25%	FlexMoE (CP-FT)	48	79	61	33	80	75	36	58.86
	FlexMoE (SP-XFER)	53	81	63	34	80	75	38	60.57
40%	NAEE†	36	63	46	25	72	64	35	48.71
	MoE-I²†	38	71	43	26	69	66	31	49.14
	MoE-SVD†	38	72	43	27	71	67	32	50.00
	MoE-SVD*	38	72	49	27	75	65	27	50.43
	TD-MoE†	47	77	57	28	79	76	35	57.00
	TD-MoE*	30	57	39	20	66	62	24	42.57
	FlexMoE	45.00 ±1.00	75.67 ±0.58	55.33 ±1.53	31.67 ±1.53	78.33 ±0.58	71.33 ±0.58	31.00 ±1.00	55.48 ±0.87
	Ours (CP-FT)	44	75	57	31	79	68	31	55.00
	FlexMoE (SP-FT)	48.33 ±1.15	77.00 ±1.00	60.00 ±1.00	33.67 ±1.15	79.67 ±0.58	73.33 ±0.58	33.67 ±0.58	57.95 ±0.5
45%	FlexMoE (CP-FT)	43	74	56	30	78	72	28	54.43
	FlexMoE (SP-XFER)	44	76	58	31	78	72	31	55.71
50%	MoE-SVD†	37	67	50	25	73	64	28	49.14
	FlexMoE (CP-FT)	43	73	56	27	78	67	27	53.00
	FlexMoE (SP-XFER)	39	71	56	30	78	71	29	53.43
55%	FlexMoE (CP-FT)	35	66	48	28	75	63	26	48.71
	FlexMoE (SP-XFER)	36	69	54	28	77	69	28	51.57
60%	NAEE†	23	42	33	17	62	55	26	36.86
	MoE-I²†	22	44	32	18	58	55	23	36.00
	MoE-SVD†	23	45	33	19	62	55	25	37.43
	MoE-SVD*	28	60	40	22	66	62	25	43.29
	TD-MoE†	28	55	38	21	65	62	24	41.86
	TD-MoE*	22	37	29	16	58	54	22	34.00
	FlexMoE	31.33 ±1.53	61.33 ±2.31	46.33 ±0.58	23.00 ±1.00	73.33 ±0.58	65.00 ±2.65	26.00 ±1.00	46.62 ±0.81
	Ours (CP-FT)	33	64	46	28	73	62	27	47.57
	FlexMoE (SP-XFER)	34.33 ±2.08	64.33 ±3.05	50.33 ±1.15	23.33 ±0.58	73.67 ±1.53	70.33 ±0.58	26.33 ±1.53	48.95 ±0.84
	Phi-3.5-MoE (16 experts per layer, top-2 activated per token)								
0%	Base model	56	77	68	40	79	76	38	62.00
20%	MoE-SVD†	54	81	65	39	79	74	36	61.14
	MoE-SVD*	55	78	66	38	80	72	35	60.57
	TD-MoE†	55	77	65	39	79	74	38	61.00

Continued on next page

Table 7 – continued

Ratio	Method	ARC-c	ARC-e	HellaS	OBQA	PIQA	WinoG	MathQA	Avg.
	TD-MoE*	52	76	63	35	78	71	34	58.43
	FlexMoE	52.33 ±1.53	78.33 ±0.58	64.67 ±0.58	39.00 ±1.73	76.33 ±1.53	78.00 ±1.00	33.00 ±1.00	60.24 ±0.36
	FlexMoE (SP-XFER)	56.00 ±1.00	83.00 ±0.00	65.33 ±0.58	39.67 ±1.15	79.00 ±1.00	80.67 ±0.58	39.33 ±0.58	63.29 ±0.00
40%	NAEE†	48	73	61	35	76	73	37	57.57
	MoE-I^2 †	40	59	27	29	70	67	25	45.29
	MoE-SVD†	48	72	58	35	75	72	31	55.86
	MoE-SVD*	48	71	63	33	76	70	28	55.57
	TD-MoE†	50	75	61	35	78	73	33	57.86
	TD-MoE*	47	73	60	30	76	69	31	55.14
	FlexMoE	50.00 ±1.73	74.67 ±2.08	58.00 ±0.00	35.00 ±1.00	76.00 ±0.00	65.33 ±1.53	29.00 ±1.00	55.43 ±0.43
	FlexMoE (SP-FT)	53.00 ±0.00	80.67 ±1.15	59.00 ±1.00	39.00 ±1.00	78.33 ±0.58	71.00 ±1.00	36.33 ±1.15	59.62 ±0.33
60%	MoE-SVD†	40	60	46	30	71	68	25	48.57
	MoE-SVD*	42	63	48	31	74	68	24	50.00
	TD-MoE†	41	70	46	28	73	68	23	49.86
	TD-MoE*	36	65	43	26	68	64	21	46.14
	FlexMoE	43.67 ±1.53	71.00 ±1.00	47.33 ±0.58	29.00 ±1.00	73.33 ±0.58	63.33 ±0.58	25.33 ±0.58	50.43 ±0.25
	FlexMoE (SP-XFER)	46.00 ±1.00	74.33 ±0.58	51.33 ±0.58	31.00 ±1.00	75.00 ±1.00	64.33 ±0.58	27.67 ±0.58	52.81 ±0.22
	Qwen2-57B-A14B (8 shared + 64 routed experts per layer, 8 shared + top-8 routed activated per token)								
0%	Base model	47	75	63	33	80	74	39	58.71
20%	NAEE†	42	72	59	31	79	72	36	55.86
	MoE-SVD†	45	74	61	30	78	73	35	56.57
	MoE-SVD*	46	75	62	31	79	74	42	58.43
	TD-MoE†	49	79	59	31	80	73	37	58.29
	TD-MoE*	47	78	59	33	80	73	36	58.00
	FlexMoE	47.33 ±0.58	75.33 ±0.58	62.67 ±0.58	33.67 ±0.58	80.67 ±0.58	74.33 ±0.58	37.67 ±0.58	58.81 ±0.16
	FlexMoE (SP-XFER)	48.67 ±0.58	78.67 ±1.53	64.00 ±1.00	34.33 ±0.58	80.00 ±1.00	75.33 ±0.58	40.67 ±0.58	60.24 ±0.7
40%	NAEE†	39	72	54	28	76	71	32	53.14
	MoE-SVD†	33	63	45	29	71	65	31	48.14
	MoE-SVD*	39	68	56	32	76	72	41	54.86
	TD-MoE†	45	77	54	29	78	71	35	55.57
	TD-MoE*	48	77	50	29	77	72	35	55.43
	FlexMoE	48.00 ±0.00	76.67 ±0.58	62.33 ±0.58	33.00 ±1.00	80.00 ±0.00	73.67 ±0.58	38.33 ±1.15	58.86 ±0.14
	FlexMoE (SP-FT)	49.67 ±0.58	79.33 ±1.15	62.00 ±1.00	34.00 ±1.00	79.67 ±0.58	74.67 ±0.58	40.67 ±0.58	60.00 ±0.42
50%	FlexMoE	47	76	61	33	80	73	40	58.57
60%	NAEE†	29	58	44	21	69	61	26	44.00
	MoE-SVD†	32	62	44	27	69	64	30	46.86
	MoE-SVD*	42	71	46	29	74	68	32	51.71
	TD-MoE†	41	73	46	28	74	68	31	51.57
	TD-MoE*	42	73	45	28	75	67	30	51.43
	FlexMoE	49.00 ±1.00	79.33 ±0.58	59.33 ±0.58	30.67 ±0.58	80.00 ±0.00	73.67 ±1.15	38.33 ±0.58	58.62 ±0.08
	FlexMoE (SP-XFER)	50.67 ±0.58	80.33 ±1.53	59.67 ±0.58	32.33 ±1.15	79.00 ±1.00	76.67 ±0.58	39.67 ±0.58	59.76 ±0.58
80%	FlexMoE	44	73	54	29	77	73	32	54.57
	Qwen3-30B-A3B (128 routed experts) (128 experts per layer, top-8 activated per token)								
0%	Base model	53	79	60	34	79	71	58	62.00
20%	FlexMoE (SP-XFER)	55	81	60	35	80	73	59	63.29
25%	FlexMoE	51	76	58	36	79	69	53	60.29
	FlexMoE (SP-XFER)	54	81	59	35	80	74	58	63.00
40%	FlexMoE	50	74	54	33	77	65	49	57.43
	FlexMoE (SP-FT)	51	81	57	34	79	70	53	60.71
60%	FlexMoE	36	66	43	29	72	60	35	48.71
	FlexMoE (SP-XFER)	41	76	47	29	76	65	40	53.43

Baseline reproduction and comparability. The source-reported baseline results in Table 7 are retained only as contextual references, since their calibration and recovery settings are not fully controlled. In particular, the published methods are all data-dependent but use different calibration corpora: NAEE and MoE-I^2 use C4-based calibration, whereas MoE-SVD and TD-MoE use substantially smaller WikiText-2 calibration sets. To provide a fair comparison, we locally reproduce the two strongest and latest baselines, MoE-SVD and TD-MoE, using the same Zyda-2 calibration corpus, hardware environment, pruning-budget accounting, and lm-eval-harness protocol as FlexMoE. We

Table 8: LongBench-E results across different pruning ratios. Scores are reported in percentage points and grouped by context length.

	Qwen2-57B-A14B			Mixtral-8×7B			Phi-3.5-MoE		
Prune ratio	0–4k	4–8k	8k+	0–4k	4–8k	8k+	0–4k	4–8k	8k+
Base	44.43	41.02	40.55	39.85	40.50	36.70	41.73	42.16	42.61
20%	44.43	41.12	39.68	35.43	37.31	34.41	37.32	38.96	39.45
40%	42.92	40.33	40.71	31.15	30.71	29.23	36.70	36.16	36.74
60%	36.38	33.69	32.34	22.39	24.24	23.84	28.32	28.04	28.04

therefore base the quantitative comparisons in the main text on these locally reproduced results (*), while retaining the published values here for completeness.

During baseline reproduction, here are two factors that can make source-reported and locally reproduced results differ. First, decomposition-based methods estimate their compression factors from a relatively small calibration subset and optimize a separate compressed endpoint for each target budget. Their resulting decomposition behavior can therefore vary with the calibration samples and random seed, making cross-corpus values not directly interchangeable with our unified Zyda-2 reproduction. Second, for TD-MoE, we found an additional difference in pruning-ratio accounting: its reported ratio is defined over the selected MoE layers to which decomposition is applied, whereas our main comparison uses the globally removed routed-expert parameters as the pruning ratio. The same nominal TD-MoE ratio can therefore correspond to a smaller actual global parameter reduction. For the other compared baselines, we verified that the parameter-ratio accounting is consistent with our global routed-expert definition. This is another reason why our main claims rely on the locally reproduced MoE-SVD and TD-MoE results. Notably, even against the source-reported TD-MoE numbers under this more favorable compression ratio convention, the shared-recovered FlexMoE model remains competitive or better across the matched budgets.

Additional validation on Qwen3-30B-A3B. We additionally evaluate Qwen3-30B-A3B [Yang et al., 2025], a newer and smaller-checkpoint MoE backbone with 128 routed experts. Compared with our three main backbones, it provides a substantially more expert-rich and strongly sparser routing topology. Results show that FlexMoE still remains effective on this architecture, and recovery performed only at the 40% budget transfers well to both looser and tighter masks. Because Qwen3-30B-A3B is newer and directly comparable results for NAEE, MoE-I^2, MoE-SVD, and TD-MoE are unavailable under our seven-task protocol, we use it only as additional architectural evidence to show backbone robustness rather than include it in the main baseline comparison, and exclude it from rerun and main text results.

Dense coverage of the budget trajectory. The 20%, 40%, and 60% points in the main text are only shared comparison anchors with baselines, rather than the only operating points supported by FlexMoE. Since the global budget is induced by independent expert-wise actions, a four-action expert space still yields a much finer global budget resolution (depends on the sparsity of MoE layers). In practice, FlexMoE's one-for-all action optimization stage allows to obtain subnetworks across the entire budget trajectory from 100% to 20% retained expert parameters, with a fine granularity of approximately 0.5% . So action training traverses a dense sequence of global pruning ratios and masks can be exported throughout this trajectory, while each exported mask stores only one action per expert and can be applied to the same importance-ranked full model, avoiding the need to rerun action learning or store another full checkpoint. The additional intermediate results in Table 7 further show that the learned budget family remains stable between the three main comparison points.

A.2 Long-Context Evaluation on LongBench-E

To test whether intra-expert channel pruning causes a disproportionate degradation as context length increases, we additionally evaluate FlexMoE on LongBench-E [Bai et al., 2024], the length-balanced evaluation setting introduced with LongBench. We evaluated all three main backbones over the 13 LongBench-E tasks. Table 8 reports these preliminary bucket-level averages.

Table 9: Sensitivity to the cost-pressure schedule on Qwen2-57B-A14B. "Step" is the action-learning step at which the corresponding global pruning budget is reached. The gentler schedule requires more optimization but yields consistently stronger masks.

Prune budget	Cost schedule	Reached at step	Seven-task Avg.
20%	$1 \rightarrow 15$	220	57.57
	$0.1 \rightarrow 5$	822	$\mathbf{58.81 \pm 0.16}$
40%	$1 \rightarrow 15$	225	57.86
	$0.1 \rightarrow 5$	1114	$\mathbf{58.86 \pm 0.14}$
60%	$1 \rightarrow 15$	266	54.86
	$0.1 \rightarrow 5$	4500	$\mathbf{58.62 \pm 0.08}$

As expected, stronger pruning lowers absolute model quality. However, across all models, the results do not show abrupt representation collapse in the longer 4–8k and 8k+ bucket. At a fixed pruning budget, performance across 4–8k and 8k+ contexts generally remains comparable to the shorter bucket or follows the same context-length trend already present in the corresponding original full model. These results suggest that FlexMoE primarily reduces overall model capacity as expected to enhance deployability, rather than disproportionately damaging long-context representations as sequence length increases.

A.3 Action-Learning Robustness: Cost Pressure and Calibration Resampling

Cost-pressure schedule. The cost coefficient $\lambda_{\text{cost}}(t)$ controls how quickly action optimization traverses the pruning-budget axis. A stronger schedule pushes experts toward thinner actions earlier, so a target pruning ratio is encountered after fewer optimization steps and fewer observed calibration samples. This accelerates training, but leaves less opportunity for the quality-preservation objective to distinguish which expert-width assignments should absorb the pruning budget. Conversely, a gentler schedule takes more exploration actions, spends more updates and calibration samples at intermediate budgets, produces more fine-grained budget trajectory, effectively increasing quality-guided exploration before reaching an aggressive pruning point.

We verify this trade-off on Qwen2-57B-A14B by comparing two linear cost schedules:

The gentler $0.1 \rightarrow 5$ schedule substantially improves pruning quality, particularly at 60%, where the optimizer receives substantially more quality-guided updates before reaching the target budget. We therefore view λ_{cost} as a trajectory-speed controller: stronger cost pressure trades search quality for a faster budget traversal, while weaker pressure allocates more optimization and calibration samples per unit of pruning. This also explains why the preferred scale of λ_{cost} is backbone-dependent: in a more strongly sparse architecture such as Qwen2, reducing the width of one routed expert perturbs the full-model output less, so the same cost force can move actions toward thin experts much faster.

Calibration resampling and seed robustness. The main experiments use the same selected cost schedule for each model, but rerun the complete FlexMoE pipeline three times with different random seeds. Each run independently resamples Zyda-2 calibration batches and therefore changes the channel-ranking permutation, Gumbel action trajectory, and recovery optimization. Despite these changes, the resulting masks are stable. This indicates that FlexMoE's strong performance are not tied to one specific calibration subset and action trajectory, but learns structured, inner-expert level importance representations under given cost pressure, thus achieving a balance between output quality and parameter pruning.

A.4 Ablation: Importance-Aware Channel Ranking

We ablate the importance-aware reordering stage on Mixtral-8x7B. The goal is to test whether prefix slicing remains effective without first sorting expert FFN channels by importance. We compare three variants: base model, which directly applies our action-learning pipeline on the original unranked model; 100 ranked, where channels are ranked using 100 calibration samples; and 5k ranked, where the same reordering procedure uses 5,000 calibration samples. A subtle but important point is that, at the same target prune ratio, the action masks are not shared across these variants. Instead, each

model variant runs its own action-learning process and uses the exported mask from that run. This is the fairest comparison: different channel orderings change the meaning of prefix slicing itself, so forcing the same mask across different orderings would confound the effect of reordering with an incompatible pruning pattern. In contrast, our current setup keeps the overall training pipeline identical to each model and only changes the ranking stage to obtain model variants. For each model variant, the action masks at different prune ratios are all exported from a single action-training run. There's no recovery fine-tuning applied.

Table 10: Ablation on importance-aware reordering for prefix slicing on Mixtral-8x7B. All numbers are zero-shot accuracy (%). "100 ranked" and "5k ranked" denote that the channel ordering is estimated using 100 and 5,000 calibration samples, respectively.

Ratio	Method	ARC-c	ARC-e	HellaS	OBQA	PIQA	WinoG	MathQA	Avg
20%	base model	50	79	59	24	79	56	34	54.43
20%	100 ranked	51	79	62	34	81	75	36	59.71
20%	5k ranked	53	81	62	33	81	73	37	**60.00**
40%	base model	44	71	51	21	75	52	28	48.86
40%	100 ranked	41	72	55	29	76	71	30	53.43
40%	5k ranked	39	70	56	30	78	72	32	**53.86**

Table 10 shows that importance-aware reordering is crucial for making prefix slicing effective. Without reordering, the unreordered base model reaches only 54.43 average accuracy at the 20% prune ratio and 48.86 at 40%. In contrast, the ranked variants substantially improve performance, reaching 59.71/60.00 at 20% and 53.43/53.86 at 40%. The improvement is especially clear at the tighter 40% budget, where preserving a more informative prefix matters more. This supports our core motivation: prefix channel retention action is only meaningful when the channel layout has first been transformed into an importance-ordered space. Once a modest number of calibration samples (steps) is available, the learned order converges and already becomes stable enough to support effective action learning, and further increasing the ranking set yields only marginal gains. This suggests that the ranking stage is not only effective but also practically sample-efficient: the importance estimates converge quickly, and the resulting channel-ranked layout is robust enough to support top-retained channel prefix slicing actions.

A.5 Ablation: Efficiency of Single Point Fine-Tuning (SP-FT) Recovery

Cross-budget Fine-Tuning (CP-FT) Ablation Details. To obtain one recovered model that can serve multiple action masks at different budgets, we first implemented a cross-point fine-tuning strategy (CP-FT) inspired by the sandwich rule in universally slimmable networks (US-Nets) and the many-subnet distillation idea of AmoebaLLM Yu and Huang [2019], Fu et al. [2024]. In each micro-step, we load a pool of trained action masks across budgets. We attached LoRA to full expert FFN projections and activate it for both the full teacher and subnet forwards. The full subnet is constrained with standard language-model CE loss to prevent full model degradation and affecting the inplace distillation objectives of subnets. While each sampled subnet is trained by soft distillation against the full-subnet teacher. Following the AmoebaLLM-style balancing factor used in our implementation, each subnet distillation loss is reweighted by the ratio between the magnitude of the full-subnet CE loss and that of the subnet distillation loss, so that no low-budget subnets dominates optimization because of higher distillation loss. Concretely, with full-subnet loss $\mathcal{L}_{\text{full}}$ and sampled subnet losses $\{\mathcal{L}_{\text{sub}}^{(i)}\}_{i=1}^{K}$, we optimize

$$\mathcal{L}_{\text{CP-FT}} = \mathcal{L}_{\text{full}} + \sum_{i=1}^{K} \frac{|\mathcal{L}_{\text{full}}|}{\left|\mathcal{L}_{\text{sub}}^{(i)}\right| + \epsilon} \mathcal{L}_{\text{sub}}^{(i)}, \tag{10}$$

where $\mathcal{L}_{\text{full}}$ is the full-subnet CE loss and each $\mathcal{L}_{\text{sub}}^{(i)}$ is the distillation loss from a sampled action mask. For each optimization step, the sampled masks (subnets) follow a US-Nets sandwich pattern consisting of the minimum-ratio mask and several random intermediate masks from the action pool, while the full model is always included as teacher.

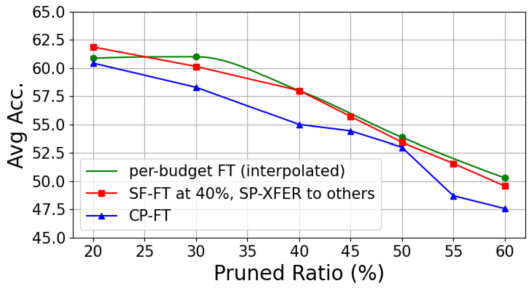


Figure 6: Average downstream accuracy of different recovery strategies on Mixtral-8x7B across prune budgets. Red curve is the recovery strategy used in FlexMoE

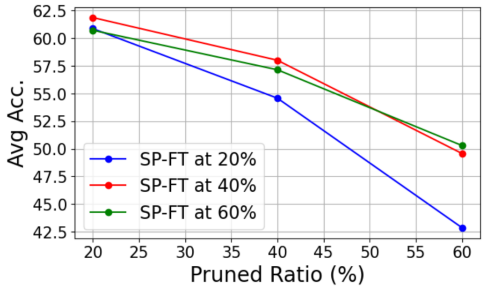


Figure 7: Recovery-point generalization curve, plot by results shown in Table 2. Red curve is the recovery point used in FlexMoE

Why Single Point Fine Tuning (SP-FT) is More Effective in Practice. We plot ablation results in Figure 6, with our SP-FT approach and per-budget fine-tuning results, the accuracy values are averaged from same 7 tasks in main experiments. We also found that this CP-FT strategy underperforms both per-budget fine-tuning and SP-FT over most budgets, whereas SP-FT followed by direct transfer already stays much closer to the per-budget fine-tuning results. We believe the reason is structural. CP-FT jointly optimizes many masks with substantially different active channel prefixes, so the shared LoRA update must satisfy several incompatible subnet configurations at once. In practice this might still weaken specialization and blur the recovery signal at any one budget. By contrast, leveraging the invariant nesting property of frozen route and expert topology, SP-FT concentrates all recovery capacity on one concrete subnet and then reuses the merged weights across nearby masks. Although this sacrifices a small amount of accuracy relative to independently fine-tuning every budget, it removes the need to train, store, and maintain one recovered model per budget point.

Why the Middle Budget Generalizes Best. Table 2 and Figure 7 show a clear asymmetry. Fine-tuning at the high-budget point (20%) generalizes poorly when transferred downward: performance degrades monotonically as pruning becomes stronger, which is consistent with the intuition that high-budget recovery still spends its capacity on tail channels that are later removed by tighter masks. Interestingly, it is also not the best even on the 20% target itself, where the mid-budget recovery point (40%) performs better. A plausible explanation is that moderate masking might act as a useful regularizer: compared with 20%-FT, 40%-FT is forced to recover only the more reusable and important prefix channels, which improves robustness even at slightly looser budgets. On the other side, low-budget recovery (60%) behaves as expected: it is stronger near its own fine-tuned budget, but its upward transfer remains weaker because the recovered update only covers a relatively small active prefix and therefore cannot adequately restore the additional channels exposed at looser masks. Overall, the 40% recovery point provides the best global trade-off between coverage and specialization: it does not fully optimize any endpoint, but it yields the strongest one-step generalization across the whole budget family.

B Kernel Co-Design Implementation Details

Our deployment goal is a single shared, importance-ordered MoE checkpoint that stays fully resident and serves every pruning budget, with the serving budget switched at runtime (Section 4.3). This imposes two requirements on the expert forward. **(R1)** A smaller per-expert retained width k_e must translate into proportionally less GPU work, not merely a masked full-width computation. **(R2)** The entire routed-expert forward must be capturable in a single CUDA graph: In decoding the per-step cost is otherwise dominated by kernel launch and host-side scheduling, which does not shrink with k_e, so without graph capture the pruning savings stay hidden behind launch overhead.

B.1 Naive Python Online Clipped FFN

Our first online clipping approach was directly implemented based on HuggingFace-style original model forward path and applies budget-conditioned weights clipping inside the expert FFN forward

Algorithm 1: Naive Python Online Clipped FFN

Input : hidden states X, routed expert assignments, router weights, per-expert retained widths $\{k_e\}$, dense expert weights $\{W_e^{gu}, W_e^{down}\}$

Output : final routed experts FFN output Y

Initialize $Y \leftarrow 0$

Find all active experts in the current batch

foreach *active expert* e **do**

  Gather routed token states X_e and router weights for expert e

  Read retained width k_e

  if $k_e = I$ **then**

    `// Original full path: use full contiguous gate-up weight`

    $Z_e^{gu} \leftarrow X_e(W_e^{gu})^\top$

    $(G_e, U_e) \leftarrow \mathrm{chunk}(Z_e^{gu}, 2)$

    $H_e \leftarrow \mathrm{SiLU}(G_e) \odot U_e$

    $O_e \leftarrow H_e(W_e^{down})^\top$

  else

    `// Online clipped path: extra slicing and reconstruction`

    $W_e^{gate} \leftarrow W_e^{gu}[0{:}k_e, :]$ `// extra slice`

    $W_e^{up} \leftarrow W_e^{gu}[I{:}I+k_e, :]$ `// extra slice`

    $\widehat{W}_e^{down} \leftarrow W_e^{down}[:, 0{:}k_e]$ `// extra slice`

    $\widehat{W}_e^{gu} \leftarrow \mathrm{cat}(W_e^{gate}, W_e^{up})$ `// hotspot`

    $\widehat{Z}_e^{gu} \leftarrow X_e(\widehat{W}_e^{gu})^\top$

    $(\widehat{G}_e, \widehat{U}_e) \leftarrow \mathrm{chunk}(\widehat{Z}_e^{gu}, 2)$

    $\widehat{H}_e \leftarrow \mathrm{SiLU}(\widehat{G}_e) \odot \widehat{U}_e$

    $O_e \leftarrow \widehat{H}_e(\widehat{W}_e^{down})^\top$

  Weight O_e by router scores and scatter-add into Y

return Y

at runtime. This path first identifies all routed experts in the current batch, and then iterates over them. In the original full-path (without weights pruning) implementation, the gate and up projections are stored contiguously as a gate–up weight tensor, so one expert forward can be executed with a single larger GEMM to enable larger GPU utilization, followed by a chunk into gate and up projection outputs. By contrast, online clipping breaks this fast path: once a runtime mask requests a retained width $k_e < I$, the implementation must first slice gate rows and up rows separately, concatenates into a clipped contiguous gate–up weight tensor, slice the down projection to the same width, and then launch the matrix multiplication and chunk. Since this happens inside a Python-side per-expert loop, the runtime pays both dispatch overhead and extra tensor-manipulation overhead before the actual GEMM.

This implementation is simple but inefficient for two reasons. First, once the learned action mask assigns different retained widths to different experts, the runtime can no longer naturally batch expert FFNs into one regular MoE matrix multiplication; it instead degrades toward many small expert-wise GEMMs plus Python-side scheduling. Second, compared with the dense full path, clipped execution inserts extra slice and cat operations on the weight tensors before every clipped expert forward. Especially for tensor concatenation step, where PyTorch's internal implementation involves additional empty memory allocation and tensor data copying. These overheads are specific to our nested prefix-slicing setting: they arise because the scenario is asked to materialize budget-specific clipped subnetworks on the fly from one shared MoE checkpoint across all budget family.

B.2 Online-Reordered Shared Weight Layout

To reduce the online clipping overhead, we first arrange the cross-budget shared model weights into a layout that is more convenient for runtime prefix slicing actions. In the original expert, the connected gate-up projection weight is stored as

$$W^{gu}_{\text{orig}} = [g_0, \ldots, g_{I-1}, u_0, \ldots, u_{I-1}],$$

so to obtain W^{gu}_{clip} with only top k channels requires two separate row slices followed by one concatenation:

$$W^{gate}_{[:k]},\ W^{up}_{[:k]},\ W^{gu}_{\text{clip}} = \text{cat}(W^{gate}_{[:k]}, W^{up}_{[:k]})$$

Meanwhile, the original down projection is stored as $W^{down}_{\text{orig}} \in \mathbb{R}^{H\times I}$, so clipping also requires a larger overhead column slice.

Our export path rewrites the shared weights into this new weights layout:

$$W^{gu}_{\text{reord}} = [g_0, u_0, g_1, u_1, \ldots, g_{I-1}, u_{I-1}], \qquad W^{down}_{\text{reord}} = (W^{down}_{\text{orig}})^\top \in \mathbb{R}^{I\times H}.$$

Under this layout, clipping to W^{gu}_{clip} with top k channels becomes one prefix slice on the interleaved gate-up tensor and one row slice on the transposed down tensor:

$$W^{gu}_{\text{clip}} = W^{gu}_{\text{reord}}[0:2k,:], \qquad W^{down}_{\text{clip}} = W^{down}_{\text{reord}}[0:k,:].$$

In the original online path, the runtime must rebuild a W^{gu}_{clip} before every clipped expert forward. In the reordered layout, we can obtain requested weights by performing only 1 direct prefix slicing. This reduces the online scheduling cost from repeated tensor assembly.

B.3 Graph-Capturable Fused Clipped FFN Forward

On top of the reordered shared weights, we co-design a fused Triton kernel that removes every host-side, input-dependent operation for decoding, so the whole routed-expert forward is captured once and replayed. It consumes the same reordered layout as Section B.2: the interleaved gate–up tensor $\tilde{W}^{gu} \in \mathbb{R}^{E\times 2I\times H}$ (gate channel j at row $2j$, up channel j at $2j{+}1$) and the transposed down tensor $\tilde{W}^{down} \in \mathbb{R}^{E\times I\times H}$, so retaining the top-$k_e$ mid channels reduced to just one prefix in the channel dimension. Algorithm 2 summarizes the forward; three design choices make it both budget-proportional (R1) and capturable (R2).

Device-resident fixed-shape routing. Routing is turned into fixed-size, device-resident block metadata by a single device kernel (MoEAlignBlockSize from SGLang): a padded `sorted_ids` list, a per-block `expert_ids` array, and a `num_post` count. No `bincount`, no device-to-host sync, and no host bucketing; across decode steps only the *values* of these buffers change, never their shapes. The token gather is folded into the first GEMM (each block reads its hidden rows through `sorted_ids`) and the scatter is folded into the second GEMM's epilogue, so there is no separate indexing step.

On-device pruning under a fixed launch grid. Both GEMMs launch on a grid sized to the *full* width ($\lceil I/B_N\rceil$ and $\lceil H/B_N\rceil$ column blocks), independent of the budget. Each program reads its expert's retained width k_e and realizes pruning purely by data-dependent control flow on the device: in GEMM-1 a column block of mid channels early-returns when its offset $n_0 \geq k_e$ (the pruned channels are never computed), and in GEMM-2 the contraction loop runs only over $k_0 \in [0, k_e)$. The work thus shrinks with k_e while the launch shape, kernel arguments, and grid stay constant — exactly what a CUDA graph requires.

Width read from a device buffer at replay time. Crucially, k_e is read from a device-resident buffer $\text{keep} \in \mathbb{Z}^E$ (`keep[e]`) *inside* the kernel, not baked into the launch configuration. Because a captured graph replays its recorded launches but re-reads buffer *contents*, a single captured graph serves every budget: switching the serving budget only overwrites keep (an $O(E)$ per layer device write), with no graph re-capture and no weight reload (Section 4.3). To keep capture deterministic the block sizes B_M, B_N, B_K are fixed rather than autotuned.

Once replayed from a graph, decoding becomes GPU-bound, so the expert weight-bandwidth term — which *does* scale with k_e — becomes a real fraction of the step, and the pruning budget turns into the decode throughput reported in Table 5.

Algorithm 2: Graph-Capturable Fused Clipped FFN Forward (deployed decode path)

Input : hidden states $X \in \mathbb{R}^{T \times H}$, routed expert ids eid, router weights α; reordered interleaved $\tilde{W}^{gu} \in \mathbb{R}^{E \times 2I \times H}$ and transposed $\tilde{W}^{down} \in \mathbb{R}^{E \times I \times H}$; *device-resident* retained widths $\text{keep} \in \mathbb{Z}^{E}$

Output : routed-expert FFN output $Y \in \mathbb{R}^{T \times H}$

$(\text{sid}, \text{eblk}, n_{\text{post}}) \leftarrow$ `MoEAlignBlockSize`(eid, B_M, E) `// fixed-shape device metadata, no host sync`

$n_{\text{valid}} \leftarrow T \cdot K$

`// GEMM-1 (gate-up + SiLU): fixed grid` $(\lceil n_{\max}/B_M \rceil,\ \lceil I/B_N \rceil)$

foreach *block* (p_m, p_n) ***in parallel*** **do**

  if $p_m B_M \geq n_{\text{post}}$ ***or*** $\text{eblk}[p_m] < 0$ **then return**

  $e \leftarrow \text{eblk}[p_m];\ k_e \leftarrow \text{keep}[e];\ n_0 \leftarrow p_n B_N$

  if $n_0 \geq k_e$ **then return** `// pruned channel block: no work`

  $\text{rows} \leftarrow \text{sid}[\text{block}];\ t \leftarrow \lfloor \text{rows}/K \rfloor$ `// fused token gather`

  $a_g, a_u \leftarrow 0$

  for $k_0 = 0$ **to** H ***step*** B_K **do**

    $a_g \mathrel{+}= X[t, k_0{:}] \cdot \tilde{W}^{gu}[e,\ 2n,\ k_0{:}];\ a_u \mathrel{+}= X[t, k_0{:}] \cdot \tilde{W}^{gu}[e,\ 2n{+}1,\ k_0{:}]$

  $\text{mid}[\text{block},\ n] \leftarrow \text{SiLU}(a_g) \odot a_u$ `// masked to` $n < k_e$

`// GEMM-2 (down): fixed grid` $(\lceil n_{\max}/B_M \rceil,\ \lceil H/B_N \rceil)$

foreach *block* (p_m, p_n) ***in parallel*** **do**

  if $p_m B_M \geq n_{\text{post}}$ ***or*** $\text{eblk}[p_m] < 0$ **then return**

  $e \leftarrow \text{eblk}[p_m];\ k_e \leftarrow \text{keep}[e];\ o \leftarrow 0$

  for $k_0 = 0$ **to** k_e ***step*** B_K **do**

    $o \mathrel{+}= \text{mid}[\text{block}, k_0{:}] \cdot \tilde{W}^{down}[e, k_0{:}, :]$ `// K-loop clamped to` k_e

  $o \leftarrow o \cdot \alpha[\text{sid}]$ `// fused router weight`

  scatter o into $\text{out}[\text{sid}, :]$ `// flat` $[T{\cdot}K,\ H]$ `buffer`

$Y \leftarrow$ `Sum`$(\text{out.view}(T, K, H),\ \text{axis}{=}1)$ `// fixed-shape top-`k `reduction`

return Y

C Reproducibility Details

C.1 Reproduction Protocol

Models and calibration data, and evaluation. We apply FlexMoE to Mixtral-8×7B, Phi-3.5-MoE, Qwen2-57B-A14B and Qwen3-30B-A3B using their original pretrained checkpoints and routing structures. All three post-training stages—channel ranking, action learning, and recovery fine-tuning—use only the train split of our Zyda-2 `sample-100BT` subset.

Pruning protocol. The pruning budget is the fraction of routed-expert FFN parameters removed; routers and other non-routed-expert modules are unchanged. We use the discrete expert-retention action space $\mathcal{A} = \{0.1, 0.4, 0.7, 1.0\}$. Channel ranking is performed once before action learning. During action learning, all pretrained model weights are frozen and only the expert action logits α are optimized, ensuring that masks obtained at different points of the trajectory refer to the same ranked weight set. Each exported mask therefore stores only one retention action per routed expert and can be directly reapplied to the shared ranked model.

Recovery protocol. For the main recovery experiments, we select the action mask at 40% pruning and freeze the importance-ranked base weights. LoRA parameters are only attached to channels exposed at this budget, and only LoRA parameters attached are optimized at this stage. After optimization, LoRA parameters are merged into the ranked base weights; the resulting recovered weight set is reused directly with the 20% and 60% masks as SP-XFER without further fine-tuning.

C.2 Complete Hyperparameter Configuration

Table 11 summarizes the complete configuration used for the three backbones shown in main text results. Unless explicitly listed otherwise, model-specific settings are identical across backbones.

Table 11: Complete FlexMoE reproduction configuration for the three main backbones. "Same" indicates the same setting as Mixtral-8×7B.

Stage	Hyperparameter	Mixtral-8×7B	Phi-3.5-MoE	Qwen2-57B-A14B
Importance-Aware Channel Ranking				
Ranking	Dataset	Zyda-2 `sample-100BT`, train split	Same	Same
	Steps	700	Same	Same
	Batch	Batch size 8; sequence length 1,024	Same	Same
	Weight update	None; gradients are used only to estimate channel saliency	Same	Same
Expert-Wise Action Learning				
Action	Dataset	Zyda-2 `sample-100BT`, train split	Same	Same
	Trainable parameters	Action logits α only; pretrained weights frozen	Same	Same
	Retention actions	$\{0.1, 0.4, 0.7, 1.0\}$; full-action initialization bias 0.05	Same	$\{0.1, 0.4, 0.7, 1.0\}$; full-action initialization bias 0.3
	Optimizer	AdamW; LR 2×10^{-4}; $\beta = (0.9, 0.99)$; weight decay 0; grad. clip 1.0	Same	Same
	Maximum steps	20,000; one sampled action mask per batch	Same	Same
	Batch	Batch size 4; grad. accumulation 4; sequence length 1,024	Same	Batch size 4; grad. accumulation 8; sequence length 1,024
	Quality loss	CE weight 1; KL weight 1; distillation temperature $T = 2$	Same	Same
	Gumbel temperature τ	Piecewise linear $4.0 \to 0.8 \to 0.5$; milestones 2,000 / 8,000	Same	Same
	Cost weight λ_{cost}	Linear $1 \to 15$ over 10,000 steps	Same	Linear $0.1 \to 5$ over 10,000 steps
	Entropy weight β	Linear $0.2 \to 0$ over 5,000 steps	Same	Same
	LR schedule	200-step warmup; decay to $0.1\times$ initial LR	Same	Same
	Mask export interval	Every 1% decrease in global retention	Every 5%	Every 5%
Single-Point Recovery Fine-Tuning				
Recovery	Dataset	Zyda-2 `sample-100BT`, train split	Same	Same
	Recovery budget	40% pruning	Same	Same
	Trainable parameters	LoRA only; ranked base weights frozen	Same	Same
	Optimizer	AdamW; LR 1×10^{-4}; $\beta = (0.9, 0.99)$; weight decay 0; grad. clip 1.0	Same	Same
	Training steps	Maximum 8,000; selected checkpoint 7,800	Maximum 8,000; selected checkpoint 3,001	Maximum 6,000; selected checkpoint 4,501
	Batch	Batch size 2; grad. accumulation 4; sequence length 2,048	Same	Same
	LR schedule	100-step linear warmup; decay to $0.1\times$ initial LR	Same	Same
	LoRA	Rank 64; $\alpha = 128$; dropout 0; no bias	Same	Same
	LoRA targets	`gate_up_proj` and `down_proj` in all 32 MoE layers	Same	`gate_up_proj` and `down_proj` in all 28 MoE layers
	Recovery loss	CE weight 1; KL weight 1; distillation temperature $T = 2$	Same	Same

C.3 Independent Runs, Statistics, and Compute

Independent runs. For the reported repeated-run experiments, we execute the complete pipeline three times with different seeds, and we report the sample mean and sample standard deviation s. Each run independently changes LoRA parameter initialization (action logits initialization are not affected by the random seed, since they are initialized as pointing to the full expert, and push them to thinner actions during training) and independently samples Zyda-2 batches, therefore each run can produce a different optimization trajectory for channel permutation, action-learning and recovered model. Despite rerun variation stochastic optimization, FlexMoE consistently converges to strong solutions across multi-budget with low performance variance, demonstrating the robustness of the learned compression trajectory.

Compute. Importance ranking is performed on one NVIDIA H200, while action learning and recovery fine-tuning use two H200 GPUs. The observed action-learning wall-clock times are approximately 48, 30, and 7 hours for Mixtral-8$\times$7B, Phi-3.5-MoE, and Qwen2-57B-A14B, respectively, to optimize the action trajectory once and produce fine-grained budget points from 0% to 70% prune ratio. By comparison, our local reproductions of MoE-SVD and TD-MoE require several hours to produce an independent compressed model for each target budget in 20%, 40% and 60%, whereas a single FlexMoE action-learning run amortizes its cost over the entire fine-grained budget trajectory, with additional operating points represented only by lightweight action masks rather than separately trained and stored checkpoints.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction match the paper's actual scope: the paper proposes a MoE pruning and deploy framework, evaluates cross-budget transfer via single-point recovery, and reports both offline deployment gains and an exploratory online kernel co-design. These claims are supported by the method, main tables, and deployment experiments. See paper abstract and Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper include a limitation discussion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theorem/lemma-style theoretical results that require formal assumptions and proofs. The equations in the paper define the method and objectives rather than establishing formal theoretical guarantees.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper discloses most important ingredients to reproduce the result, as well as a hyper-parameter table in Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We'll release code repository as promised.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies all the training and test details necessary to understand the results, as well as a hyper-parameter table in Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper provides sample std for main experiment results, calculated from 3 repeated independent runs with different seeds.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports main compute information, including the use of 2× H200 for action learning and recovery fine-tuning, single H200 for other experiments. Memory cost for base model, produced checkpoints and action-training times are also provided.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: To the best of our knowledge, the research described in the paper is consistent with the NeurIPS Code of Ethics. The work studies post-training compression and deployment of existing MoE language models, and does not involve human subjects or deceptive experimental practices.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: The paper does not include a discussion of broader societal impacts.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [No]

Justification: The paper does not currently describe safeguards for releasing models, masks, or code related to large language models

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [No]

Justification: The paper credits the creators of the main datasets, models, and benchmarks through citations, but it does not explicitly state the licenses or terms of use of the external assets used in the paper.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We consider our contribution as a method and empirical study rather than an openly released benchmark, asset or artifact.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects. The experiments are conducted on pretrained models and public benchmarks only.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The paper does not involve human subjects research, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components, but we admit that LLMs are used in code writing and paper polishing.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.