SP-MoE: Post-Training Neuron Sharing for Memory-Constrained Inference

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) language models activate a few experts per token, but all experts occupy storage, making deployment memory a bottleneck. Existing work addresses this at two levels. At the system level, offloading keeps a subset of experts resident and fetches the rest from host memory; under a tight budget these full-expert transfers can dominate decoding: a Mixtral-8x7B baseline spends 86.8% of decode time waiting on transfers at a 16GiB weight budget, even with prefetching. At the model level, pruning removes experts and merging consolidates them, reducing storage but potentially losing expert-specific capacity; each retained expert remains a whole transfer unit. Both levels leave the transfer unit at expert granularity. We introduce **SP-MoE**, a post-training neuron-sharing method that reduces the size of this unit while retaining every routed expert. It aligns intermediate neurons, selects low-sensitivity ones by reconstruction-gradient magnitude, and aggregates them into one shared component per layer, leaving a private residual per expert. The shared component stays resident, so a miss transfers only that residual. Because a SwiGLU neuron is a gate column, an up column, and a down row whose contributions add in the down projection, decomposed execution computes the shared branch once per token and combines it with routing-weighted private outputs, reproducing the transformed model exactly. Across OLMoE, Qwen1.5-MoE, and Mixtral, SP-MoE achieves the highest average zero-shot accuracy among tested compression baselines at 50% expert-parameter sparsity. On one RTX 4090 at batch size one, under per-model weight budgets including resident shared weights, it improves throughput at every tested sparsity, by up to 1.71× over HC-SMoE and 2.09× over Full MoE. On Mixtral at 50% sparsity, each miss transfers 144 rather than 336 MiB.

1 Introduction

Mixture-of-Experts (MoE) language models increase capacity by replacing a dense feed-forward layer with multiple experts and activating a small subset for each token (Jiang et al., 2024). Sparse activation limits per-token computation, but it does not remove the storage cost of inactive experts. This distinction matters when GPU memory is limited: a model can be affordable to compute yet too large to keep resident. Non-expert weights, the KV cache, and runtime buffers further limit the memory available for experts on consumer-grade GPUs. Deploying such a model therefore requires reducing its GPU memory footprint while keeping both generation latency and model quality acceptable.

One response manages where the model's parameters live. Offloading runtimes keep a subset of experts resident and fetch missing experts from host DRAM over PCIe, using prefetching and scheduling to overlap transfers with computation (Xue et al., 2025). Even with these techniques, when few experts are resident, each decoding step can require substantial parameter movement, and full-expert transfers can exceed the computation window available to hide them. For example, serving Mixtral-8x7B on one RTX 4090 with a 16 GiB weight budget, the full-expert MoE-Infinity baseline spends 86.8% of decode time waiting on expert transfers even with prefetching enabled. Scheduling alone does not reduce the amount of parameter data that each miss must transfer.

Compression addresses the storage cost directly. Expert pruning removes routed choices (Lu et al., 2024); expert merging consolidates several parameter sets into fewer experts (Chen et al., 2025a).

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

Both reduce storage, but they operate at expert granularity, and reducing expert-specific capacity can degrade quality (Liu et al., 2024b). Fewer experts may fit more easily in GPU memory, but when offloading is still needed, each missing expert remains a whole block to transfer. Reducing the number of experts does not by itself reduce the size of that block.

Together, these limitations motivate a finer compression granularity: can a post-training transformation retain every routed expert while reducing the amount of parameter data that must be fetched to execute it? A SwiGLU expert is composed of intermediate-neuron units, each coupling a gate column, an up column, and a down row. These units contribute additively to the expert's output. Sharing selected neurons across experts would reduce stored parameters while leaving a private component for every expert; keeping the shared part resident would then make that smaller private component the transfer unit.

To address these limitations, we propose **SP-MoE**, a post-training neuron-sharing method for routed MoEs. Its *shared-private* representation replaces selected neurons with one shared component per layer and retains private neurons for every expert. The router's parameters and expert roster are retained. This differs from native shared-expert architectures, which learn a shared branch jointly with routed experts during pretraining (Dai et al., 2024). SP-MoE constructs its shared branch from an existing checkpoint, so selecting neurons that tolerate aggregation is essential.

The construction must account for the approximation introduced by sharing: averaging selected parameters changes the functions of the original experts. An offline alignment pass first matches intermediate neurons across experts. We then rank aligned dimensions by reconstruction-gradient magnitude, using low local sensitivity as a heuristic for selecting dimensions to share; higher-scoring dimensions remain private. Calibration routing mass weights each expert's contribution to the shared parameters. Retaining every expert preserves the available routed choices, while the selection criterion aims to limit the quality loss from replacing expert-specific values.

As sharing shortens expert transfers, execution can become the limiting cost of decoding. A straightforward implementation would reconstruct each active expert from the shared and private weights, incurring allocation and copying, and would repeat the same shared matrix products across active experts. This work may be hidden while transfers dominate; as transfers become shorter, it can limit further reductions in decoding latency. SP-MoE exploits the additive structure of SwiGLU to avoid these costs: it computes the shared branch once per token in each layer and combines its output with those of the selected private branches according to the routing weights. This is mathematically equivalent to executing the reconstructed, transformed experts, so the execution scheme introduces no additional approximation.

At Mixtral's 50% matched expert-parameter sparsity point, corresponding to $\alpha = 4/7 = 57.1\%$ shared dimensions, a full-expert miss transfers 336 MiB, whereas an SP-MoE miss transfers a 144 MiB private residual. The shared component itself consumes GPU memory, so the serving benefit must be measured after charging its resident cost. Our evaluation tests quality at matched retained parameter counts and serving throughput under the same per-model weight budget, including dense weights, shared weights, and the expert cache.

Our contributions are:

- We introduce SP-MoE, a post-training neuron-sharing method that preserves the original router and expert roster. Reconstruction-gradient selection and routing-weighted aggregation construct a shared component from aligned pretrained experts. Private components become the units of offloading, and exact decomposed execution reuses the shared computation.
- Across 3 model families, SP-MoE achieves the highest average zero-shot accuracy among the tested compression baselines at 50% expert-parameter sparsity. On one RTX 4090 at batch size one, it improves generation throughput by up to 1.71× over the expert-reduction baseline HC-SMoE at matched parameters and 2.09× over Full MoE under the same per-model weight budget.

2 Related Work

Routed and native shared experts. Modern MoEs combine sparse activation with large expert parameter sets, including Mixtral, OLMoE, DBRX, Grok, and Qwen (Jiang et al., 2024; Muennighoff et al., 2025; Databricks, 2024; xAI, 2024; Yang et al., 2025). DeepSeek-style architectures

explicitly separate shared and routed experts during training (Dai et al., 2024; Liu et al., 2024a; DeepSeek-AI et al., 2025; 2026), so their shared branch co-adapts with the routed ones. SP-MoE instead constructs a shared branch after training by aggregating selected neurons from pretrained routed experts, without retraining. Coordinate correspondence and selection identify neurons that tolerate aggregation. This transformation also applies to models with native shared experts, as our Qwen1.5 experiments show (Section 4.3).

Post-training expert compression. SEER-MoE uses routing information to prune experts (Muzio et al., 2024); NAEE searches expert subsets using reconstruction error (Lu et al., 2024); and EASY-EP uses token-aware contribution estimates (Dong et al., 2025). MC-SMoE combines routing-informed merging, permutation alignment, and compression, while HC-SMoE groups experts by activation similarity (Li et al., 2024; Chen et al., 2025a). These methods remove or merge whole experts. MoNE instead retains routing to lightweight *novices*, constant vectors approximating replaced experts' average outputs (Zhang et al., 2026). REAP favors pruning over merging because merging removes the router's independent modulation of the merged experts; its generative evaluations expose losses that likelihood-scored multiple choice can hide (Lasby et al., 2026). MoBE compresses the up and gate projections using shared basis matrices and expert-specific factors, leaving the down projections unchanged (Chen et al., 2025c). SP-MoE shares complete SwiGLU intermediate neurons while preserving the router and private capacity for every expert. This structure enables exact decomposed execution of the transformed experts and makes smaller private residuals the units of offloading.

Expert offloading. Offloading makes MoEs accessible under limited GPU memory by fetching experts from host memory (Eliseev & Mazur, 2023); MoE-Infinity adds an expert cache and sparsity-aware prefetching (Xue et al., 2025), and other systems explore mixed precision, fine-grained placement, CPU–GPU orchestration, and cross-layer or speculative prefetching (Tang et al., 2024; Yu et al., 2025; Kamahori et al., 2025; Zhou et al., 2025; Chen et al., 2025b).[1] SP-MoE changes the parameter representation presented to an offloading system: the cache-managed object becomes a private residual, so existing policies can operate on smaller transfers.

Relation to quantization. Quantization also shrinks the transferred object, and HOBBIT fetches low-precision experts on demand (Tang et al., 2024). It is a different axis rather than a competing one: quantization reduces bits per parameter, SP-MoE reduces how many parameters are fetched, and the two act on different factors of the same product. Several recent frontier MoEs already ship with expert weights in FP8 or lower (DeepSeek-AI et al., 2026), including two checkpoints in Appendix C.3. Our quality experiments apply SP-MoE to the FP8 DeepSeek-R1 and Qwen3-Coder checkpoints, showing that SP-MoE can be applied directly to low-precision expert weights. When lower precision is acceptable, quantization can further reduce the number of bytes transferred.

3 SP-MoE

Preliminaries. Let $x \in \mathbb{R}^d$ denote the representation of one token entering the layer, and let $i \in \{1, \dots, N\}$ index an expert. An MoE layer contains N parallel expert FFNs and a router. For a SwiGLU expert with hidden width d and intermediate width d_{ffn},

$$E_i(x) = \big(\sigma(xW^{(i)}_{\text{gate}}) \odot xW^{(i)}_{\text{up}}\big)W^{(i)}_{\text{down}}, \tag{1}$$

where $W^{(i)}_{\text{gate}}, W^{(i)}_{\text{up}} \in \mathbb{R}^{d\times d_{\text{ffn}}}$, $W^{(i)}_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}}\times d}$, and σ is the SiLU activation. Early MoE models used simpler FFNs with only up- and down-projection matrices (Fedus et al., 2022); we develop SP-MoE on the now-standard SwiGLU structure, and it applies to the two-matrix design with minor adaptation. Index j selects the j-th gate and up columns together with the j-th down row; the three slices sharing index j constitute one *intermediate neuron*, the unit SP-MoE operates on. We use *dimension* for the index j and *coordinate* when the concern is that index j need not denote the same feature in two experts. The router selects the active expert set $\mathcal{K} = \text{TopK}(R(x))$ and assigns a

[1] Concurrent work of Chen et al. (2025b) uses the same SP-MoE abbreviation for an unrelated speculative-decoding and prefetching system. The two are independent; here SP-MoE denotes our post-training neuron-sharing method.

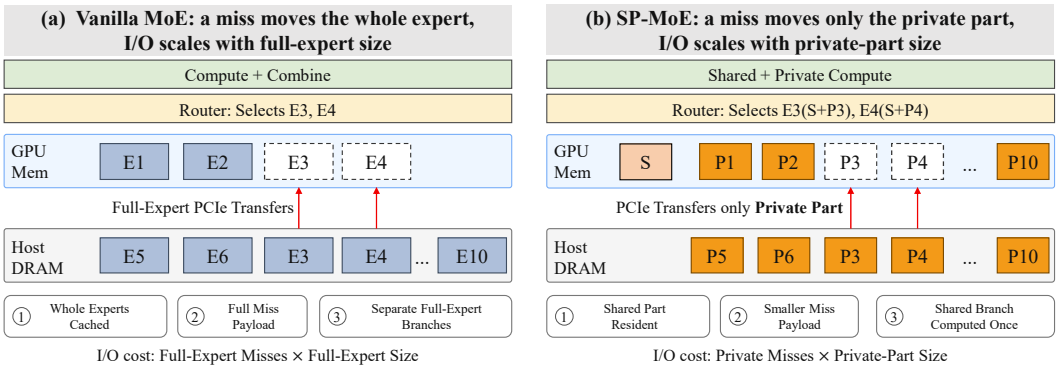


Figure 1: Neuron sharing changes the offloading unit. Full-expert offloading fetches all weights for a missing expert. SP-MoE keeps a shared component S resident and fetches only private components P_i. For a fixed layer input, the unchanged router selects the same expert IDs. Box sizes are schematic and do not represent a measured cache capacity.

post-top-k routing weight $P_i(x)$ to each active expert:

$$y = \sum_{i \in \mathcal{K}} P_i(x) E_i(x), \tag{2}$$

where y is the routed-expert output for x and different tokens can select different subsets. Throughout this section, any native shared expert remains unchanged and its output is excluded from the mixture.

SP-MoE shares intermediate neurons across experts after training. It partitions each expert's intermediate neurons into a subset shared across the layer and a subset that stays private to the expert: the shared subset is stored once per layer while each expert retains its private component. The router's parameters and the expert roster are unchanged, so every routed choice that existed before the transformation still exists after it. Figure 1 contrasts the resulting offloading unit with full-expert offloading.

3.1 Shared-private parameterization

For a layer with N experts, let $\mathcal{S}$ be the selected set of intermediate dimensions, with $|\mathcal{S}| = \lfloor \alpha d_{\text{ffn}} \rfloor$, and let $\mathcal{P}$ be its complement. After the coordinate alignment described below, all experts use the same selected index set $\mathcal{S}$, whose indices need not be contiguous. Parameters at each selected index are aggregated across experts to form one shared neuron, giving the shared component

$$\boldsymbol{\theta}_{\mathcal{S}} = \text{Aggregate}(\boldsymbol{\theta}_{\mathcal{S}}^{(1)}, \ldots, \boldsymbol{\theta}_{\mathcal{S}}^{(N)}), \tag{3}$$

while expert i retains $\boldsymbol{\theta}_{\mathcal{P}}^{(i)}$. Sharing a fraction α of intermediate dimensions reduces the expert-parameter count by $(N-1)\alpha/N$, because the selected dimensions are stored once instead of N times. Under the pruning convention, sparsity s denotes the fraction of original expert parameters removed. We therefore choose $\alpha = sN/(N-1)$ to match target sparsity. For a fixed layer input, routing weights are unchanged, and the private residual remains available for every routed expert.

3.2 Dimension selection

Aggregating index j across experts presumes that the j-th neuron means the same thing in each of them, which trained experts do not guarantee: a SwiGLU expert's intermediate coordinates are permutation-symmetric. We therefore first establish correspondence, choosing a medoid expert per layer and matching the other experts' neurons to it by cosine similarity over the three slices together. This is one-time offline preprocessing that preserves each expert's function, adds no serving-time work, and follows the alignment principle used by MC-SMoE (Li et al., 2024); Appendix B gives the procedure. Selection then operates on aligned coordinates.

We calibrate on routed tokens. Let $\mathcal{D}$ be the calibration set and $\mathcal{D}_i$ its tokens whose original top-k decision selects expert i. The target $y_{\mathrm{MoE}}(x)$ is the original layer's mixture output in Equation 2, held fixed when computing gradients. For a candidate dimension, we measure its sensitivity through the reconstruction objective

$$\mathcal{L}_{\mathrm{recon}}^{(i)} = \frac{1}{|\mathcal{D}_i|} \sum_{x \in \mathcal{D}_i} \|E_i(x) - y_{\mathrm{MoE}}(x)\|_2^2, \tag{4}$$

The mixture output supplies a common reconstruction target that captures the layer's collective behavior on routed tokens. We score a candidate dimension by the local sensitivity of this objective to its three parameter slices:

$$\mathrm{score}_{i,j} = \|\nabla_{W_{\mathrm{gate}}^{(i)}[:,j]} \mathcal{L}_{\mathrm{recon}}^{(i)}\|_2 + \|\nabla_{W_{\mathrm{up}}^{(i)}[:,j]} \mathcal{L}_{\mathrm{recon}}^{(i)}\|_2 + \|\nabla_{W_{\mathrm{down}}^{(i)}[j,:]} \mathcal{L}_{\mathrm{recon}}^{(i)}\|_2. \tag{5}$$

Sharing replaces each expert-specific slice with an aggregate value. For a comparable replacement perturbation, a small gradient predicts a small first-order change in reconstruction loss, so low-gradient dimensions are candidates for sharing while high-gradient dimensions remain private. This is a local-sensitivity heuristic, not a bound on finite aggregation error. We average scores across experts,

$$\mathrm{AggregatedScore}(j) = \frac{1}{N} \sum_i \mathrm{score}_{i,j}, \tag{6}$$

and choose the $|\mathcal{S}|$ lowest-scoring dimensions for $\mathcal{S}$. The gradient ranking is computed once from the calibration set; it does not update the router's parameters.

3.3 Routing-weighted aggregation

For the selected dimensions, the shared value is a routing-mass-weighted average. With $\widetilde{P}_i(x)$ denoting the post-top-k routing weight and zero for an unselected expert,

$$w_i = \frac{\sum_{x \in \mathcal{D}} \widetilde{P}_i(x)}{\sum_{r=1}^{N} \sum_{x \in \mathcal{D}} \widetilde{P}_r(x)}, \qquad \boldsymbol{\theta}_{\mathcal{S}} = \sum_i w_i \boldsymbol{\theta}_{\mathcal{S}}^{(i)}. \tag{7}$$

Experts with greater calibration routing mass contribute more to the shared value, while the parameters on $\mathcal{P}$ remain unchanged. Sharing is the only approximation in this transformation; its quality is measured in Section 4.2.

3.4 Exact decomposed execution

The shared and private neurons of each transformed expert can be packed into compact matrices by reordering their gate/up columns and down rows together. Because intermediate neurons contribute independently to the down projection, the transformed expert's output can be computed without reconstructing a full matrix. For active experts $\mathcal{K}$,

$$\widetilde{y} = \sum_{i \in \mathcal{K}} P_i(x)[E(x; \boldsymbol{\theta}_{\mathcal{S}}) + E(x; \boldsymbol{\theta}_{\mathcal{P}}^{(i)})] = \left(\sum_{i \in \mathcal{K}} P_i(x) \right) E(x; \boldsymbol{\theta}_{\mathcal{S}}) + \sum_{i \in \mathcal{K}} P_i(x) E(x; \boldsymbol{\theta}_{\mathcal{P}}^{(i)}). \tag{8}$$

Here $\widetilde{y}$ is the transformed layer's output. The equality is between full execution of the transformed experts and decomposed execution; sharing can change the original y in Equation 2, while decomposition introduces no additional approximation. Appendix A proves the equality between the two execution forms in Equation 8. The shared branch is computed once per token in each layer, and the private branches follow the original routing weights. Compared with executing k full transformed experts, projection FLOPs decrease by $(k-1)\alpha/k$. Thus decomposed execution is exact for the transformed parameters and avoids reconstructing full weight matrices and repeating the shared matrix products. The sum of routing weights is kept explicit; it is one only when the router normalizes weights over the active experts.

Deployment interpretation. The shared-private representation makes each expert's private component the cache-managed unit. The shared component is pinned at load time, and the runtime's existing prefetch, eviction, and scheduling policies operate on private components, so a miss transfers a private residual instead of a full expert. Expert-granularity pruning and merging leave whole expert blocks as the offloading unit.

Memory trade-off. The GPU weight budget includes one resident shared component and the cached private components. Consider one layer with equal-size experts of B bytes each, and let M be the memory available to this layer's shared weights and private cache after other resident state has been charged. At shared fraction α, caching C private components must satisfy $\alpha B + C(1-\alpha)B \leq M$. The first term is paid once regardless of the active expert set, while each cache miss transfers a private component of $(1-\alpha)B$ bytes instead of B. For Mixtral with $N = 8$, a 50% matched expert-parameter sparsity gives $\alpha = 4/7 \approx 57.1\%$: the shared component costs $4B/7$ and each private component costs $3B/7$. Section 4.3 therefore holds every comparison to a single weight budget that charges the resident shared component, making the measurements iso-memory by construction.

4 EXPERIMENTS

We compare quality at matched model size, measure offloaded serving, and ablate neuron sharing, coordinate alignment, dimension selection, and execution.

4.1 SETUP

Models and baselines. We evaluate the base versions of OLMoE-1B-7B (Muennighoff et al., 2025), Qwen1.5-MoE-A2.7B (Qwen Team, 2024), and Mixtral-8x7B (Jiang et al., 2024), with routed-expert counts $N = 64$, 60, and 8, top-k routing of $k = 8$, 4, and 2, and intermediate widths $d_{\text{ffn}} = 1{,}024$, $1{,}408$, and $14{,}336$. Qwen1.5 pairs its routed experts with a native shared expert that compression leaves untouched, so N counts only the routed experts SP-MoE targets. Quality comparisons include SEER-MoE, NAEE, HC-SMoE, MC-SMoE, and EASY-EP (Muzio et al., 2024; Lu et al., 2024; Chen et al., 2025a; Li et al., 2024; Dong et al., 2025). NAEE's subset search is capped at 1000 combinations for models with many experts. All compressed variants use a uniform sparsity across MoE layers and a matched retained expert-parameter budget. Non-expert weights are fixed; the uncompressed model is the reference. Reported sparsity follows the pruning convention, with SP-MoE's shared width adjusted by $\alpha = sN/(N-1)$ to include the shared component in the parameter count.

Quality evaluation. We report zero-shot accuracy using LM-Eval Harness (Gao et al., 2024) on nine tasks: ARC-Challenge, ARC-Easy, HellaSwag, PIQA, WinoGrande, OpenBookQA, BoolQ, RTE, and MMLU (Clark et al., 2018; Zellers et al., 2019; Bisk et al., 2019; Sakaguchi et al., 2019; Mihaylov et al., 2018; Clark et al., 2019; Wang et al., 2019; Hendrycks et al., 2021). We use LM-Eval Harness 0.4.9.2, with `acc_norm` for ARC-Challenge, ARC-Easy, HellaSwag, PIQA, and OpenBookQA and `acc` for WinoGrande, BoolQ, RTE, and MMLU; prompt construction, chat-template handling, and quality-evaluation batching follow the harness protocol. The reported average gives equal weight to each task. Methods requiring calibration use C4 text concatenated into 16 sequences of 2048 tokens (Raffel et al., 2023).

Serving evaluation. Quality comparisons match retained expert parameters; serving comparisons hold a model-weight budget M fixed. The *GPU expert cache* holds full experts for the baselines and private residuals for SP-MoE; an activated unit absent from the cache must be fetched before its computation can complete. Comparisons are *iso-memory* for model weights: the runtime enforces and reports a single budget for ordinary dense weights, resident shared weights, and the expert cache,

$$\text{dense} + \text{shared} + \text{cache} \leq M. \tag{9}$$

Resident shared weights therefore reduce the cache capacity by their exact byte size. We set M to 3 GiB for OLMoE, 6 GiB for Qwen1.5, and 16 GiB for Mixtral. Unless otherwise stated, these per-model budgets apply to all offloaded serving results; comparisons between methods always use the same budget. M excludes the KV cache, activations, CUDA context, allocator overhead, and temporary buffers. Workloads are identical across methods; total GPU memory use is not the quantity held fixed.

All serving variants use BF16 on one RTX 4090 in a workstation with 256 GB host DRAM and PCIe 4.0 x16, on MoE-Infinity with unchanged prefetch, eviction, and scheduling policies (Xue et al., 2025). We compare Full MoE, the uncompressed model of Section 4.2 run under the same budget,

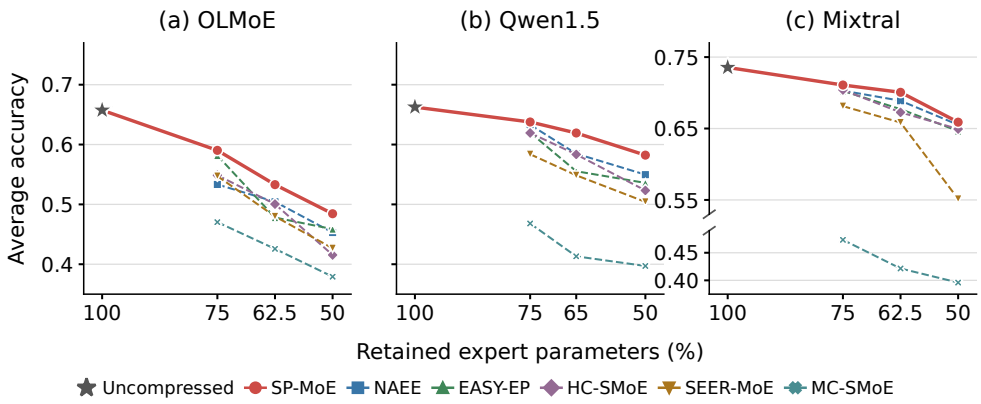


Figure 2: Quality at matched expert-parameter budgets. Each point is the mean zero-shot accuracy across nine tasks; the star at 100% is the uncompressed model. SP-MoE is joined to it to show the degradation from the original checkpoint. Full task-level values appear in Appendix C.1.

against Pruned MoE and SP-MoE at batch size one on BIG-bench's abstract narrative-understanding task (Srivastava et al., 2023), which asks a model to match a narrative to the proverb it illustrates, with 128 input and 128 output tokens, discarding five warmup batches and averaging the rest: 100 batches for OLMoE and Qwen1.5, 60 for Mixtral, whose per-batch decode is several times longer. *Pruned MoE* is any baseline that reduces the routed expert count at expert granularity, so a miss still transfers a whole expert; we use HC-SMoE (Chen et al., 2025a). Generation throughput is output tokens divided by end-to-end generation time; at batch size one, its reciprocal is the end-to-end time per output token (TPOT). Each SP-MoE variant is compared against that baseline at its matched retained-parameter budget by Welch t over completed batches, with $|t| > 2$ for every sparsity of every model at batch size one. Dispersion is within-run batch-level variation, not across runs or seeds.

4.2 Quality at matched model size

Figure 2 compares average accuracy over the tested expert-parameter budgets, and Table 1 lists the 50% sparsity averages. SP-MoE has the highest average accuracy among the compressed methods shown at each reported sparsity for all three models. The largest margin is on Qwen1.5, where it leads the strongest baseline by 3.5 percentage points at 35% sparsity and 3.3 at 50%, both computed from the full-precision scores in Table 6. On Mixtral, SP-MoE and NAEE have comparable measured accuracy at 50% sparsity, with SP-MoE ahead by 0.4 percentage points in single runs without seed replication. Accuracy nevertheless falls as more neurons are shared, especially on OLMoE. Preserving the router's parameters and expert roster retains the available routed choices, but earlier-layer changes can alter hidden states and hence later routing decisions. Appendix C.1 reports every task score, including tasks on which a baseline scores higher. A supplementary Mixtral comparison finds competitive accuracy with REAP across 25–50% sparsity (Appendix C.2).

Beyond multiple choice. The nine tasks above score answer-choice likelihoods without generating tokens. Lasby et al. (2026) report that such discriminative scores can overstate methods that average expert parameters, which is exactly SP-MoE's approximation, so we test the ordering where tokens are generated. Appendix C.3 evaluates DeepSeek-R1, Qwen3-Coder-480B, and Qwen3-Next-80B on AIME 2024, MATH-500, HumanEval, and BIG-Bench Hard. At the evaluated 20–25% sparsities, SP-MoE's mean clipped per-task loss is 2.1–3.0 percentage points against 22.1–33.3 for HC-SMoE and EASY-EP. On Qwen3-Next HumanEval, SP-MoE retains 91.5% pass@1 against 95.7% uncompressed, while the two baselines fall to 2.4% and 9.8%. All compressed variants use the same C4 calibration set. More aggressive sharing has a larger quality cost: on DeepSeek-R1, mean clipped loss rises from 2.1 percentage points at 25% sparsity to 18.3 at 50%.

4.3 Serving measurements

At batch size one, SP-MoE outperforms Pruned MoE in generation throughput at every tested sparsity on all three models under the same per-model weight budget (Figure 3). Mixtral at 50% sparsity

Table 1: Average zero-shot accuracy at 50% expert-parameter sparsity, with the uncompressed model as reference. Bold indicates the best compressed model in each row.

	Uncompressed	SP-MoE	NAEE	HC-SMoE	EASY-EP	SEER-MoE	MC-SMoE
OLMoE	0.657	**0.484**	0.453	0.416	0.459	0.427	0.379
Qwen1.5	0.663	**0.582**	0.550	0.523	0.536	0.504	0.397
Mixtral	0.736	**0.659**	0.655	0.649	0.647	0.551	0.396

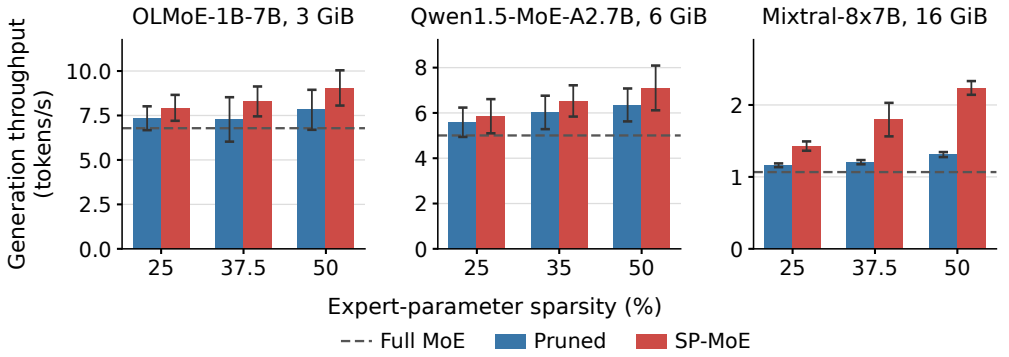


Figure 3: Generation throughput under the same weight budget of Equation 9, at the budget in each panel title. Bars are batch means, whiskers batch-level standard deviation; each SP-MoE bar sits beside the Pruned baseline at the same sparsity. The dashed line is Full MoE at the same budget.

combines higher quality with faster inference: SP-MoE achieves 65.89% average accuracy versus HC-SMoE's 64.93% (Table 7), while throughput rises from 1.31 to 2.24 tokens/s under the same 16 GiB weight budget (Table 2). This is the largest gain, at 1.71× over Pruned MoE and 2.09× over Full MoE; Qwen1.5 shows the smallest gains, at 1.05–1.12× over Pruned MoE.

The gain comes from per-miss transfer size, not coverage. Charging the shared component does raise the number of cached objects, because each object is smaller: on Mixtral the cache holds 39 full experts for the baselines and 42, 45, and 49 private residuals at 25%, 37.5%, and 50% sparsity. But the count does not explain the speedup. SP-MoE is faster than its paired Pruned MoE baseline in all nine comparisons, despite a lower cache hit rate in six. At 50% sparsity on Mixtral, its hit rate is 11.9 percentage points lower (25.9% versus 37.8%), yet it is 1.71× faster. Each miss transfers 144 rather than 336 MiB, a 2.33× reduction in payload; traffic per token of context falls by 1.93× after accounting for the additional misses.

Table 2 separates the two quantities. Within each model, traffic falls monotonically as the cache-managed object shrinks across the SP-MoE settings, while hit rate does not follow the same order: on Qwen1.5 SP-MoE sits 11.0 points below Pruned MoE on hit rate yet moves 1.62× less data and runs 1.12× faster. Pruning leaves the object at full expert size, so its traffic reduction comes only from missing less often, not from a smaller object.

Qwen1.5 also shows that a native shared expert does not make post-training neuron sharing redundant. Its native branch is 5,632 wide, uncompressed, and resident, charged against the dense term of Equation 9 rather than the cache, and sharing neurons across routed experts still improves throughput at all three sparsities.

Reducing transfer volume yields larger speedups in more I/O-bound deployments. At 50% sparsity, OLMoE cuts traffic per token more than Mixtral (2.12× versus 1.93×), yet gains less throughput (1.16× versus 1.71×). The Full baseline spends 86.8% of decode waiting for expert transfers on Mixtral, versus 48.1% on OLMoE and 47.6% on Qwen1.5; comparable traffic reductions therefore yield larger gains when transfers occupy more of decoding. At batch sizes one, four, and eight, SP-MoE moves less data per token and stalls less than either baseline, with throughput gains over Full MoE of 1.33×, 1.44×, and 1.43×, while its margin over the pruned baseline narrows with batch size and reverses at batch eight, where SP-MoE is 4.0% slower: batching amortizes each transfer, so the stall saving that favors SP-MoE shrinks, while the cost of executing all routed experts exactly does not. Appendix D reports every sparsity and varies batch size, weight budget, and workload.

Table 2: Serving counters at 50% sparsity under the same weight budget. Tok/s is generation throughput with batch-level standard deviation; Hit is cache hit rate (%); MB/tok is PCIe traffic per token of context. Rows issue equal lookups per token, so a miss fetches a full expert for Full and Pruned and a private residual for SP-MoE. Every sparsity appears in Appendix D.

	OLMoE (3 GiB)			Qwen1.5 (6 GiB)			Mixtral (16 GiB)		
Method	Tok/s	Hit (%)	MB/tok	Tok/s	Hit (%)	MB/tok	Tok/s	Hit (%)	MB/tok
Full	6.78 (0.71)	35.0	552	5.01 (0.63)	17.4	760	1.07 (0.04)	20.8	9142
Pruned	7.82 (1.12)	56.3	360	6.35 (0.73)	42.2	504	1.31 (0.03)	37.8	7071
SP-MoE	**9.05** (0.99)	59.2	**170**	**7.10** (0.99)	31.2	**311**	**2.24** (0.10)	25.9	**3666**

Table 3: Dimension Selection on OLMoE: mean accuracy over nine tasks.

Selection	37.5%	50%
Gradient magnitude	**0.5330**	**0.4844**
Min norm	0.5163	0.4698
Wanda	0.5142	0.4718
Activation sparsity	0.5131	0.4788
Max similarity	0.5012	0.4265

Table 4: Execution Ablation on OLMoE: median tokens/s, weights resident; B is batch size.

B	Execution	25%	37.5%	50%
1	Reconstruct	12.37	12.09	12.02
	Decompose	**13.18**	**13.06**	**12.60**
4	Reconstruct	30.59	29.87	31.22
	Decompose	**35.79**	**33.57**	**34.17**

4.4 Ablations

Neuron sharing and coordinate alignment. At 37.5% sparsity on OLMoE, we compare SP-MoE with matched neuron removal and with the offline alignment pass disabled. All three use one harness configuration, separate from the main quality tables. Removal keeps 640 private dimensions per expert against SP-MoE's 390 shared plus 634 private, matching retained expert parameters to within 0.02%. Removing the shared branch costs 1.2 percentage points of nine-task average accuracy; skipping alignment at the same shared width costs 0.7. Both components contribute modestly at this operating point. Removal still wins four of nine tasks, and the average advantage rests mainly on MMLU. Appendix E.1 gives the protocol and per-task scores.

Dimension selection. The alternatives rank the same neurons by different signals: *min norm* by smallest weight norm across the three slices, *Wanda* by weight magnitude scaled by input activation norm (Sun et al., 2024), *activation sparsity* by how often the activation is near zero, and *max similarity* by agreement with other experts' corresponding neurons. The first three criteria were designed for parameter removal; SP-MoE instead ranks dimensions for replacement by shared values using reconstruction-gradient magnitude. This criterion gives the highest average accuracy at both OLMoE sparsities (Table 3). Its local sensitivity score provides a heuristic for sharing; Appendix E.2 compares all five criteria with measured single-dimension distortion.

Execution path. Reconstruction concatenates the shared and private weights before each expert computation; decomposed execution computes the two branches directly and reuses the shared output. With weights resident, decomposition improves throughput by 1.05–1.08× at $B = 1$ and 1.09–1.17× at $B = 4$ (Table 4). Both paths compute the same function (Appendix A), so the gain adds no approximation. All offloaded serving results above use decomposed execution.

5 Conclusion

SP-MoE addresses the storage and transfer costs of memory-constrained MoE inference. Offloading moves whole experts, while pruning and merging reduce expert count without shrinking the transfer unit. SP-MoE shares complete SwiGLU intermediate neurons while retaining the router and private components for every routed expert. Unlike native shared experts learned during pretraining, it constructs the shared component from a checkpoint through reconstruction-gradient selection over aligned neurons and routing-weighted aggregation. Keeping shared weights resident makes private components the transfer units. The neurons' additive structure enables exact decomposed execution of the transformed model, computing the shared branch once per token in each layer. One transformation delivers all three: fewer stored parameters, smaller transfers, and shared computation reused across active experts.

AI USE STATEMENT

The research ideas were developed by the authors without LLM assistance. Large language models assisted with manuscript refinement, LaTeX formatting, implementation of code for the ablation studies, experiment execution, and collection, organization, and internal-consistency analysis of experimental results. Reported measurements come from model runs on the stated hardware. The authors reviewed all AI-assisted work, verified the code and experimental results, and take responsibility for the final text, code, experiments, and claims.

ETHICS STATEMENT

This work studies inference efficiency and model compression using public pretrained models and benchmark datasets. It does not collect data from human participants. Compressed models can inherit the biases and failure modes of their source models; the benchmarks used here do not establish deployment safety.

REPRODUCIBILITY STATEMENT

Section 3 describes checkpoint construction and decomposed execution. Section 4.1 specifies the models, calibration data, comparison budgets, hardware, and metrics. Appendix C.1 provides task-level scores, and Appendix D gives serving counters and sensitivity measurements. Appendix A establishes the algebraic equivalence of decomposed execution, and Appendix B specifies the alignment pass, including its similarity criterion, assignment solver, and one-off cost. Appendix E.1 specifies the protocol for the construction ablations, and Appendix C.3 describes the frontier-model quality evaluation.

We will provide code for checkpoint construction and the runtime, construction and evaluation commands, evaluation configurations, and available result records to reviewers and area chairs on request through an anonymous repository during the discussion period. The runtime code will be pinned to the revision used in the experiments, including the accounting of Equation 9. We will release this material publicly upon acceptance.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language, 2019. URL https://arxiv.org/abs/1911.11641.

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 *International Conference on Machine Learning*, 2025a. URL https://proceedings.mlr.press/v267/chen25aq.html.

Liangkun Chen, Zijian Wen, Tian Wu, Xiaoxi Zhang, and Chuan Wu. SP-MoE: Speculative decoding and prefetching for accelerating MoE-based model inference, 2025b. URL https://arxiv.org/abs/2510.10302.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Xiaodong Chen, Mingming Ha, Zhenzhong Lan, Jing Zhang, and Jianguo Li. MoBE: Mixture-of-basis-experts for compressing MoE-based LLMs, 2025c. URL https://arxiv.org/abs/2508.05257.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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), pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

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, 2018. URL https://arxiv.org/abs/1803.05457.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

Databricks. Introducing DBRX: A new state-of-the-art open LLM. https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024. Accessed: 2025-09-15.

DeepSeek-AI. DeepSeek-R1-0528. https://huggingface.co/deepseek-ai/DeepSeek-R1-0528, 2025. Model card. Accessed: 2026-06-08.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. DeepSeek-V3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Zican Dong, Han Peng, Peiyu Liu, Wayne Xin Zhao, Dong Wu, Feng Xiao, and Zhifeng Wang. Domain-specific pruning of large mixture-of-experts models with few-shot demonstrations, 2025. URL https://arxiv.org/abs/2504.06792.

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

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://arxiv.org/abs/2101.03961.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, et al. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2021. URL https://arxiv.org/abs/2012.14913.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

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, 2024. URL https://arxiv.org/abs/2401.04088.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models, 2025. URL https://arxiv.org/abs/2402.07033.

Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. REAP the experts: Why pruning prevails for one-shot MoE compression. In *International Conference on Learning Representations*, 2026.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. Merge, then compress: Demystify efficient SMoE with hints from its routing policy. In *International Conference on Learning Representations*, pp. 14234–14256, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3d09a88c3372cdb79401fde592ca4db8-Paper-Conference.pdf.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, et al. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024a. URL https://arxiv.org/abs/2405.04434.

Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Matthew B. Blaschko, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs, 2024b. URL https://arxiv.org/abs/2407.00945.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient LLMs at inference time, 2023. URL https://arxiv.org/abs/2310.17157.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6159–6172, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.334. URL https://aclanthology.org/2024.acl-long.334/.

Mathematical Association of America. American invitational mathematics examination – AIME 2024. https://huggingface.co/datasets/AI-MO/aimo-validation-aime, 2024. Accessed: 2026-06-10.

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 Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open mixture-of-experts language models, 2025. URL https://arxiv.org/abs/2409.02060.

Alexandre Muzio, Alex Sun, and Churan He. SEER-MoE: Sparse expert efficiency through regularization for mixture-of-experts, 2024. URL https://arxiv.org/abs/2404.05089.

Qwen Team. Qwen1.5-MoE: Matching 7b model performance with 1/3 activated parameters, March 2024. URL https://qwenlm.github.io/blog/qwen-moe/.

Qwen Team. Qwen3-Coder-480B-A35B-Instruct-FP8. https://huggingface.co/Qwen/Qwen3-Coder-480B-A35B-Instruct-FP8, 2025a. Model card. Accessed: 2026-06-08.

Qwen Team. Qwen3-Next-80B-A3B-Instruct. https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct, 2025b. Model card. Accessed: 2026-06-08.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text Transformer, 2023. URL https://arxiv.org/abs/1910.10683.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale, 2019. URL https://arxiv.org/abs/1907.10641.

Noam Shazeer. GLU variants improve Transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023. URL https://arxiv.org/abs/2206.04615.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.11695.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, et al. Challenging BIG-Bench tasks and whether chain-of-thought can solve them, 2022. URL https://arxiv.org/abs/2210.09261.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. HOBBIT: A mixed precision expert offloading system for fast MoE inference, 2024. URL https://arxiv.org/abs/2411.01433.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL https://arxiv.org/abs/1804.07461.

xAI. Grok open release. https://github.com/xai-org/grok-1, 2024. Accessed: 2025-09-15.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. fMoE: Fine-grained expert offloading for large mixture-of-experts serving, 2025. URL https://arxiv.org/abs/2502.05370.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

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*, 2026.

Yuxin Zhou, Zheng Li, Jun Zhang, Jue Wang, Yiping Wang, Zhongle Xie, Ke Chen, and Lidan Shou. FloE: On-the-fly MoE inference on memory-constrained GPU, 2025. URL https://arxiv.org/abs/2505.05950.

A Exact Decomposed Execution for SwiGLU

This appendix establishes the equivalence between full-matrix execution and decomposed execution of the transformed SwiGLU experts.

The property it rests on is that a SwiGLU feed-forward layer (Shazeer, 2020) is a sum of independent per-neuron terms, so its intermediate dimensions carry no cross-coordinate interaction. That additive view is the same one under which Geva et al. (2021) read a feed-forward layer as a set of key–value memories whose contributions superpose, and under which Liu et al. (2023) skip individual intermediate neurons at inference time without altering the remaining ones. Both exploit the decomposability of the sum; we use it to split the sum once, along a fixed partition, and compute the shared part a single time per token.

Proposition. Let $\mathcal{S}$ and $\mathcal{P}$ be disjoint sets partitioning the d_{ffn} intermediate dimensions of a SwiGLU expert. For any fixed parameters θ,

$$E(x;\theta) = E(x;\theta_{\mathcal{S}}) + E(x;\theta_{\mathcal{P}}). \tag{10}$$

Proof. Write g_j and u_j for gate and up columns and v_j for a down row. SwiGLU applies its activation and product independently to each intermediate dimension, so

$$E(x;\theta) = \sum_{j=1}^{d_{\text{ffn}}} \underbrace{\sigma(xg_j)(xu_j)}_{h_j(x)} v_j. \tag{11}$$

Partitioning this sum gives

$$E(x;\theta) = \sum_{j\in\mathcal{S}} h_j(x)v_j + \sum_{j\in\mathcal{P}} h_j(x)v_j = E(x;\theta_{\mathcal{S}}) + E(x;\theta_{\mathcal{P}}). \tag{12}$$

Applying the identity to the transformed experts makes the first term identical across all active experts. Factoring it out of their routing-weighted sum yields Equation 8. Thus decomposed execution and full-matrix execution of the transformed model are equal in exact arithmetic; floating-point implementations may differ through accumulation order. □

Cost. Let $m = |\mathcal{S}|$. Storing one shared component and N private components retains $m+N(d_{\text{ffn}}-m)$ intermediate dimensions, compared with Nd_{ffn} originally. The realized expert-parameter sparsity is $s_{\text{realized}} = (N-1)m/(Nd_{\text{ffn}})$. Computing the shared branch once and k private branches likewise reduces expert projection work by $(k-1)m/(kd_{\text{ffn}})$. Substituting $m = \lfloor \alpha d_{\text{ffn}} \rfloor$ accounts for integer rounding. Ignoring rounding gives $s = (N-1)\alpha/N$, hence $\alpha = sN/(N-1)$ for a target sparsity s, as in Section 3.

B Coordinate Correspondence

Coordinate-wise aggregation across experts is only well defined once corresponding intermediate neurons have been identified. The construction below supplies that correspondence. It is offline preprocessing, not a contribution of this work, and follows the alignment principle used by MC-SMoE (Li et al., 2024).

Why a permutation is needed. The intermediate coordinates of a SwiGLU expert are permutation-symmetric: reordering the columns of $W^{(i)}_{\text{gate}}$ and $W^{(i)}_{\text{up}}$ together with the rows of $W^{(i)}_{\text{down}}$ leaves E_i unchanged. Two independently trained experts therefore have no reason to place the same feature at the same index, and averaging index j across experts would combine unrelated features.

Procedure. For each layer we select a reference expert, then match every other expert's neurons to it and permute all three of that neuron's slices together; permuting fewer than three would change the expert's function.

The reference is the layer's medoid. We flatten each expert's gate and up matrices into one vector, normalize it to unit length, and take the expert whose summed cosine similarity to all experts in the layer is largest.

Matching is a square linear assignment problem. For experts r and i we build a $d_{\text{ffn}} \times d_{\text{ffn}}$ score matrix whose (a, b) entry adds three cosine similarities, between gate column a of r and gate column b of i, and likewise for the up columns and the down rows, each vector ℓ_2-normalized before the products. We maximize the total score over permutations with the Jonker–Volgenant algorithm, as implemented by `scipy.optimize.linear_sum_assignment` on the negated matrix, which returns an exact optimum rather than a greedy match.

Cost. The pass runs once per checkpoint and adds no serving-time work. Similarities are computed in FP32 on the device holding the weights, and the assignment solver runs on CPU. In a CPU measurement on OLMoE, with 16 layers of 64 experts at $d_{\text{ffn}} = 1{,}024$, medoid selection takes 0.7 s per layer and the 63 assignments take 7.2 s, so the matching is about 2 minutes for the whole checkpoint; loading and re-saving the weights dominates end-to-end time. Cost grows with the number of experts and with d_{ffn}^3 in the assignment step, so it is a one-off preparation cost, not a per-deployment one.

Effect of alignment. At 37.5% expert-parameter sparsity on OLMoE, disabling alignment at the same α and shared width lowers nine-task average accuracy from 0.5313 to 0.5244 under the matched evaluation protocol of Appendix E.1.

C Quality Evaluation

This appendix expands Section 4.2 with task-level results on the three main models, a comparison with REAP, and generative evaluations on frontier models. Each comparison states its evaluation protocol.

C.1 Detailed Task-Level Results

Tables 5–7 give the zero-shot scores underlying Figure 2. Each model is separated into sparsity groups; bold marks the highest value in each task column and the average column among compressed variants in a group, including ties. Hella, WinoG, and OBQA abbreviate HellaSwag, WinoGrande, and OpenBookQA. Scores are fractions; the average gives equal weight to the nine tasks.

Table 5: OLMoE-1B-7B: zero-shot accuracy by task and expert-parameter sparsity.

Method	ARC-C	ARC-E	Hella	PIQA	WinoG	OBQA	BoolQ	RTE	MMLU	Avg.
Uncompressed reference										
Uncompressed	0.5000	0.7710	0.7809	0.7851	0.6882	0.4420	0.6997	0.7148	0.5351	0.6574
25% sparsity										
SP-MoE	0.3976	0.6902	0.6807	0.7481	0.6401	0.3760	**0.6985**	**0.6606**	**0.4204**	**0.5903**
NAEE	0.3567	0.6044	0.6832	0.7394	0.6346	0.3660	0.6251	0.5054	0.2849	0.5333
EASY-EP	0.4019	**0.6978**	**0.7341**	**0.7650**	**0.6472**	**0.4340**	0.6920	0.5632	0.3001	0.5817
HC-SMoE	**0.4036**	0.6317	0.6613	0.6953	0.6433	0.3600	0.5306	0.6245	0.3883	0.5487
SEER-MoE	0.3703	0.6410	0.7040	0.7644	0.6188	0.4160	0.6043	0.5162	0.2911	0.5473
MC-SMoE	0.2713	0.4672	0.5521	0.6627	0.6062	0.3160	0.6012	0.5235	0.2328	0.4703
37.5% sparsity										
SP-MoE	0.3404	**0.6338**	0.6002	0.7095	**0.6140**	0.3240	**0.6884**	**0.5740**	**0.3125**	**0.5330**
NAEE	0.3294	0.6246	**0.6080**	**0.7111**	0.5825	**0.3740**	0.5153	0.5307	0.2641	0.5044
EASY-EP	0.3072	0.5189	0.5786	0.6474	0.5683	0.3240	0.5789	0.5126	0.2663	0.4780
HC-SMoE	**0.3703**	0.5981	0.5652	0.6692	0.6062	0.3200	0.5217	0.5632	0.2910	0.5005
SEER-MoE	0.2944	0.4487	0.5645	0.6877	0.5651	0.3440	0.6156	0.5343	0.2652	0.4799
MC-SMoE	0.2406	0.3801	0.4442	0.6121	0.5470	0.2620	0.5939	0.5235	0.2283	0.4257
50% sparsity										
SP-MoE	**0.2995**	**0.5564**	0.5008	**0.6649**	**0.5801**	0.3140	**0.6398**	0.5199	**0.2848**	**0.4844**
NAEE	0.2756	0.4348	0.4758	0.6453	0.5391	0.3180	0.5969	**0.5523**	0.2396	0.4530
EASY-EP	0.2790	0.4794	**0.5138**	0.6284	0.5643	**0.3220**	0.5673	0.5343	0.2398	0.4587
HC-SMoE	0.2730	0.4503	0.4369	0.5990	0.5249	0.2760	0.4284	0.4982	0.2527	0.4155
SEER-MoE	0.2551	0.3779	0.4236	0.6328	0.5146	0.3100	0.5661	0.5271	0.2329	0.4267
MC-SMoE	0.2560	0.3350	0.3249	0.5658	0.4988	0.2480	0.4367	0.5162	0.2317	0.3792

Table 6: Qwen1.5-MoE-A2.7B: zero-shot accuracy by task and expert-parameter sparsity. Qwen1.5 uses a middle setting of 35% sparsity, corresponding to 39 of 60 retained experts in the expert-reduction baselines; OLMoE and Mixtral use 37.5%.

Method	ARC-C	ARC-E	Hella	PIQA	WinoG	OBQA	BoolQ	RTE	MMLU	Avg.
Uncompressed reference										
Uncompressed	0.4386	0.7319	0.7734	0.7976	0.6867	0.4480	0.7969	0.6823	0.6084	0.6626
25% sparsity										
SP-MoE	0.4130	**0.7269**	0.7252	0.7802	**0.6914**	0.4060	**0.7847**	0.6534	**0.5588**	**0.6377**
NAEE	**0.4181**	0.7243	0.7268	0.7655	0.6772	0.4120	0.7606	0.6715	0.5414	0.6330
EASY-EP	0.4096	0.6898	**0.7525**	**0.7884**	0.6875	**0.4260**	0.6615	0.6282	0.5354	0.6199
HC-SMoE	0.4044	0.6768	0.7250	0.7661	0.6803	0.3680	0.7486	**0.6823**	0.5229	0.6194
SEER-MoE	0.4044	0.6717	0.7038	0.7726	0.6180	0.3700	0.6602	0.5812	0.4721	0.5838
MC-SMoE	0.2969	0.5000	0.5647	0.6730	0.6338	0.3320	0.4474	0.5199	0.2451	0.4681
35% sparsity										
SP-MoE	**0.3976**	**0.7033**	0.6982	0.7568	**0.6732**	0.3780	**0.7703**	**0.6570**	**0.5390**	**0.6193**
NAEE	0.3916	0.6848	**0.7199**	**0.7650**	0.6385	**0.4040**	0.6719	0.5632	0.4237	0.5847
EASY-EP	0.3797	0.6397	0.6774	0.7214	0.6440	0.3580	0.6688	0.5307	0.3789	0.5554
HC-SMoE	0.3763	0.6372	0.6820	0.7345	0.6448	0.3620	0.7064	0.6173	0.4913	0.5835
SEER-MoE	0.3746	0.6124	0.6582	0.7535	0.5943	0.3720	0.6575	0.5451	0.3699	0.5486
MC-SMoE	0.2628	0.3792	0.4242	0.6050	0.5651	0.2940	0.4324	0.5054	0.2510	0.4132
50% sparsity										
SP-MoE	**0.3660**	**0.6524**	0.6329	0.7312	**0.6709**	0.3560	**0.7352**	0.6029	**0.4929**	**0.5823**
NAEE	0.3464	0.6271	0.6462	**0.7410**	0.6117	0.3480	0.6401	0.5884	0.3980	0.5497
EASY-EP	0.3575	0.6111	**0.6713**	0.7165	0.6448	**0.3800**	0.6373	0.5235	0.2814	0.5359
HC-SMoE	0.3302	0.5316	0.5959	0.6899	0.6101	0.3060	0.6394	**0.6462**	0.3601	0.5233
SEER-MoE	0.3285	0.5446	0.5561	0.7176	0.5612	0.3260	0.6459	0.5162	0.3413	0.5042
MC-SMoE	0.2321	0.3439	0.3452	0.5843	0.5288	0.2540	0.5443	0.5090	0.2326	0.3971

Table 7: Mixtral-8x7B: zero-shot accuracy by task and expert-parameter sparsity.

Method	ARC-C	ARC-E	Hella	PIQA	WinoG	OBQA	BoolQ	RTE	MMLU	Avg.
Uncompressed reference										
Uncompressed	0.5981	0.8380	0.8425	0.8226	0.7727	0.4940	0.8587	0.7112	0.6814	0.7355
25% sparsity										
SP-MoE	0.5606	**0.8060**	**0.8243**	**0.8177**	0.7514	**0.4780**	0.8315	0.6895	**0.6387**	**0.7109**
NAEE	**0.5683**	0.8051	0.8211	0.8058	**0.7735**	0.4560	0.8272	0.6823	0.5880	0.7030
EASY-EP	0.5461	0.8018	0.8204	0.8063	0.7609	**0.4780**	0.8138	0.6931	0.6035	0.7026
HC-SMoE	0.5512	0.7967	0.8096	0.7992	0.7545	0.4480	**0.8459**	**0.7148**	0.6163	0.7040
SEER-MoE	0.5606	0.8026	0.8185	0.8101	0.7672	0.4500	0.7804	0.5884	0.5539	0.6813
MC-SMoE	0.2722	0.4356	0.5961	0.6170	0.5714	0.3180	0.6401	0.5343	0.2755	0.4734
37.5% sparsity										
SP-MoE	**0.5418**	**0.7912**	**0.8121**	**0.8047**	0.7585	**0.4620**	0.8489	**0.6859**	**0.6012**	**0.7007**
NAEE	0.5341	0.7828	0.8007	0.7976	**0.7609**	0.4560	0.8367	0.6787	0.5529	0.6889
EASY-EP	0.5145	0.7879	0.8023	0.8030	0.7451	0.4500	0.7908	0.6606	0.5395	0.6771
HC-SMoE	0.5034	0.7601	0.7842	0.7758	0.7340	0.4360	**0.8544**	0.6462	0.5594	0.6726
SEER-MoE	0.5171	0.7778	0.7987	0.7927	0.7569	0.4460	0.7385	0.5957	0.5023	0.6584
MC-SMoE	0.2329	0.3666	0.4306	0.5522	0.5446	0.2880	0.6150	0.5271	0.2364	0.4215
50% sparsity										
SP-MoE	**0.5119**	0.7660	**0.7758**	**0.7922**	**0.7474**	**0.4460**	0.8021	0.5523	**0.5360**	**0.6589**
NAEE	0.4906	0.7534	0.7702	0.7845	0.7332	0.4420	0.8003	**0.6606**	0.4616	0.6552
EASY-EP	0.4846	**0.7668**	0.7658	0.7671	0.7372	0.4320	0.7795	0.6029	0.4826	0.6465
HC-SMoE	0.4633	0.7517	0.7575	0.7650	0.7222	0.4100	**0.8333**	0.6318	0.5089	0.6493
SEER-MoE	0.3857	0.6545	0.6240	0.7476	0.5888	0.3000	0.6856	0.5776	0.3993	0.5514
MC-SMoE	0.2167	0.2908	0.3325	0.5539	0.5351	0.2620	0.5765	0.5307	0.2661	0.3960

C.2 Comparison with REAP

We compare SP-MoE with REAP (Lasby et al., 2026) and HC-SMoE on Mixtral-8x7B at 25%, 37.5%, and 50% expert-parameter sparsity. All three methods are re-evaluated in the rebuilt LM-Eval Harness 0.4.9.2 environment with Transformers 4.57.3 described in Appendix E.1. We use the nine zero-shot tasks of Section 4.1, with `acc_norm` for ARC-C, ARC-E, HellaSwag, PIQA, and OpenBookQA and `acc` for the others, and average the task scores equally. Table 8 reports the results. Absolute scores are not directly comparable to Table 7, which uses the main evaluation configuration.

Table 8: Mixtral-8x7B comparison with REAP under the rebuilt evaluation environment, at matched retained expert-parameter budgets. Bold marks the highest task score and average within each sparsity.

Method	ARC-C	ARC-E	Hella	PIQA	WinoG	OBQA	BoolQ	RTE	MMLU	Avg.
25% sparsity										
SP-MoE	0.5427	0.7778	**0.8276**	**0.8259**	**0.7616**	**0.4580**	0.8465	0.6859	**0.6190**	0.7050
REAP	**0.5648**	0.7879	0.8155	0.8205	0.7577	0.4560	**0.8538**	**0.7148**	0.6028	**0.7082**
HC-SMoE	0.5580	**0.8064**	0.8071	0.8145	0.7388	0.4380	0.8498	0.7004	0.6134	0.7029
37.5% sparsity										
SP-MoE	0.5179	0.7483	**0.8106**	**0.8161**	**0.7466**	**0.4440**	0.8453	0.6173	**0.5875**	**0.6815**
REAP	**0.5299**	0.7597	0.7900	0.7960	0.7435	0.4320	0.8398	**0.6859**	0.5380	0.6794
HC-SMoE	0.5128	**0.7689**	0.7801	0.7911	0.7459	0.4200	**0.8587**	0.6426	0.5553	0.6750
50% sparsity										
SP-MoE	**0.4761**	0.7142	**0.7779**	**0.7943**	**0.7332**	0.4100	0.8138	0.5307	**0.5006**	**0.6390**
REAP	0.4462	0.7041	0.7440	0.7590	0.7182	**0.4140**	0.8110	**0.6390**	0.4551	0.6323
HC-SMoE	0.4548	**0.7159**	0.7308	0.7601	0.7151	0.4020	**0.8343**	0.5921	0.4763	0.6313

SP-MoE achieves comparable average accuracy to REAP, leading by 0.21 and 0.67 percentage points at 37.5% and 50% sparsity, respectively; REAP leads by 0.32 points at 25%. SP-MoE also scores above HC-SMoE at all three sparsities. The task-level ordering varies: SP-MoE scores higher than REAP on HellaSwag, PIQA, WinoGrande, and MMLU at each sparsity, while REAP scores higher on RTE.

SP-MoE pairs this competitive accuracy with smaller offloading units. REAP prunes whole experts, leaving each retained expert at full width, so expert offloading still transfers a full expert on a cache miss. SP-MoE keeps the shared component resident, charging it to the weight budget in Equation 9, and transfers only the missing private residual. This representation reduces the bytes transferred per miss while preserving the router and private capacity for every expert.

C.3 FRONTIER-MODEL GENERATIVE EVALUATION

We additionally evaluate DeepSeek-R1-0528, Qwen3-Coder-480B-A35B-Instruct-FP8, and Qwen3-Next-80B-A3B-Instruct (DeepSeek-AI, 2025; Qwen Team, 2025a;b). These experiments use eight NVIDIA H20-3e GPUs and differ from the main evaluation in tasks, precision, and sparsity. They establish quality only, and support no serving-throughput claim.

Unlike the nine multiple-choice tasks of Section 4.2, every task here requires the model to generate a response that is then checked, so this subsection tests the ordering under the generative conditions Lasby et al. (2026) identify as the harder case for parameter aggregation. All variants of a model are built from the same C4 calibration set and evaluated under one harness configuration per model.

The tasks are AIME 2024, MATH-500, three BIG-Bench Hard tasks, and HumanEval variants (Mathematical Association of America, 2024; Lightman et al., 2023; Lewkowycz et al., 2022; Suzgun et al., 2022; Chen et al., 2021). In Table 9, BBH-C, BBH-F, and BBH-T denote `causal_judgement`, `formal_fallacies`, and the seven-object version of `tracking_shuffled_objects`. Qwen3-Next uses zero-shot BBH and HumanEval-Instruct; the other models use few-shot BBH and HumanEval. Consequently, scores should be compared within a model block.

We report the average task score and mean clipped loss. For uncompressed scores a_t and compressed-model scores $\widetilde{a}_t$, clipped loss is

$$\frac{100}{T}\sum_{t=1}^{T}\max(0, a_t - \widetilde{a}_t), \tag{13}$$

expressed in percentage points and displayed with a percent sign in the table. This metric prevents improvements on some tasks from cancelling losses on others. Table 9 reports two SP-MoE sparsities for DeepSeek-R1, one for each Qwen model, and two compression baselines.

Table 9: Supplementary quality results on frontier models. Clip loss is the mean per-task drop, clipped at zero, in percentage points. Bold marks the highest scores and lowest clip loss among compressed methods at matched sparsity, including ties; DeepSeek-R1 comparisons use 25% sparsity. These experiments provide no serving-throughput evidence.

Variant	Sparsity	AIME24	MATH	BBH-C	BBH-F	BBH-T	HEval	Avg.	Clip loss
DeepSeek-R1									
Uncompressed	–	0.833	0.848	0.668	0.808	0.968	0.829	0.826	–
HC-SMoE	25%	0.033	0.550	**0.690**	0.816	0.780	0.177	0.508	32.31%
EASY-EP	25%	0.033	0.302	0.674	0.796	0.944	0.213	0.494	33.30%
SP-MoE	25%	**0.800**	**0.914**	0.647	**0.856**	**0.996**	**0.756**	**0.828**	**2.13%**
SP-MoE	50%	0.333	0.776	0.652	0.696	0.928	0.470	0.643	18.33%
Qwen3-Coder									
Uncompressed	–	0.267	0.936	0.647	0.456	0.912	0.695	0.652	–
HC-SMoE	20%	0.000	0.488	**0.701**	0.704	0.968	0.085	0.491	22.07%
EASY-EP	20%	0.000	0.456	0.663	0.708	0.960	0.073	0.477	22.81%
SP-MoE	20%	**0.267**	**0.864**	0.658	**0.864**	**0.976**	**0.610**	**0.706**	**2.62%**
Qwen3-Next									
Uncompressed	–	0.533	0.958	0.727	1.000	1.000	0.957	0.863	–
HC-SMoE	25%	0.167	0.538	**0.711**	0.964	0.996	0.024	0.567	29.59%
EASY-EP	25%	0.033	0.452	0.706	0.948	**1.000**	0.098	0.539	32.32%
SP-MoE	25%	**0.500**	**0.910**	0.684	**0.988**	**1.000**	**0.915**	**0.833**	**2.98%**

D SERVING EVALUATION

This appendix expands Section 4.3 with serving counters at every sparsity, the accuracy–throughput trade-off, and sensitivity to batch size, weight budget, and workload.

D.1 PER-SPARSITY SERVING RESULTS

All runs use the shared weight budget of Equation 9, 128 input and 128 output tokens, batch size one, seed 1024, and five discarded warmup batches. Completed batches are 100 for OLMoE and Qwen1.5 and 60 for Mixtral. The 4 GiB OLMoE budget run executes 200 batches; we report its first 100 so that every budget is compared over the same number of batches. Table 10 gives the per-sparsity comparison summarized in Section 4.3.

Table 10: Every serving sparsity under the shared weight budget. SP denotes SP-MoE and HC denotes HC-SMoE. Reported sparsity is expert-parameter sparsity, so SP-MoE's shared-dimension fraction is $\alpha = sN/(N-1)$ and HC-SMoE retains $(1-s)N$ experts. Qwen1.5 uses a middle setting of 35% sparsity, corresponding to 39 of 60 retained experts; OLMoE and Mixtral use 37.5%. Object is the size of one SP private residual or HC full expert in MiB, equal to the data fetched on a cache miss. MB/tok is PCIe traffic in decimal megabytes per token of context, prompt plus generated, as in Table 2. Δhit is SP minus HC cache hit rate in percentage points. Cache counts resident objects: SP counts private residuals and HC counts full experts. HC cache counts also apply to Full under these budgets. t is Welch's statistic over completed batches.

		Tok/s					Object (MiB)		MB/tok		Cache	
Model	Sparsity	SP	HC	SP/HC	t	Δhit	SP	HC	SP	HC	SP	HC
OLMoE	25%	7.93	7.35	1.08	5.9	+1.3	8.95	12	361	488	236	180
(3 GiB)	37.5%	8.29	7.28	1.14	6.7	+0.9	7.43	12	276	443	281	180
	50%	9.05	7.82	1.16	8.2	+2.9	5.91	12	170	360	349	180
Qwen1.5	25%	5.86	5.59	1.05	2.7	−4.1	12.32	16.5	538	664	202	157
(6 GiB)	35%	6.53	6.02	1.08	5.0	−7.0	10.63	16.5	448	607	231	157
	50%	7.10	6.35	1.12	6.1	−11.0	8.12	16.5	311	504	295	157
Mixtral	25%	1.43	1.16	1.23	29.7	−3.4	240	336	6361	8447	42	39
(16 GiB)	37.5%	1.80	1.21	1.49	19.5	−3.0	192	336	4754	7882	45	39
	50%	2.24	1.31	1.71	70.8	−11.9	144	336	3666	7071	49	39

The Full-model baselines are 6.78, 5.01, and 1.07 tokens/s for OLMoE, Qwen1.5, and Mixtral, spending 48.1%, 47.6%, and 86.8% of decode time waiting on transfers. Mixtral's cache holds 39 full experts for the baselines against 42, 45, and 49 private residuals for SP-MoE. These counts match $\lfloor C/B \rfloor$ for a full expert and $\lfloor C/((1-\alpha)B) \rfloor$ for a private residual, where C is the remaining cache capacity after resident charges and $B = 336\,\mathrm{MiB}$.

D.2 Accuracy–Throughput Trade-off across Sparsities

Increasing sparsity trades accuracy for generation throughput on OLMoE at the fixed 3 GiB weight budget (Figure 4). From 25% to 50% sparsity, throughput rises from 7.93 to 9.05 tokens/s, while average accuracy falls from 0.5903 to 0.4844. The intermediate 37.5% setting records 8.29 tokens/s with average accuracy 0.5330. All three settings exceed Full MoE's 6.78 tokens/s at the same budget.

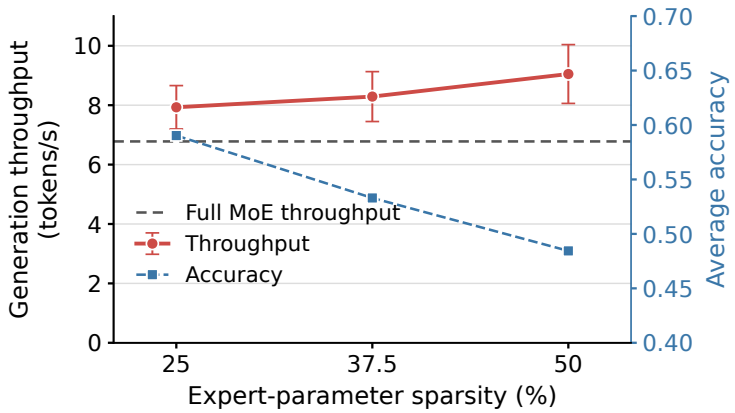


Figure 4: OLMoE throughput at the 3 GiB weight budget against average zero-shot accuracy, across the three reported settings. Sparsity and shared width are related by $\alpha = sN/(N-1)$, not identical. Whiskers are batch-level standard deviation; the dashed line is Full MoE's throughput at the same budget.

D.3 Batch-Size Sensitivity

Table 11 repeats the OLMoE comparison at batch sizes one, four, and eight under the same 3 GiB budget. SP-MoE achieves higher generation throughput than Full MoE at all three batch sizes, by 1.33×, 1.44×, and 1.43×. Against HC-SMoE at matched sparsity, the advantage narrows from 1.16× at batch one to 1.08× at batch four. At batch eight, SP-MoE is 4.0% slower (20.77 against 21.64 tokens/s; batch-level Welch $t = -2.83$).

PCIe traffic per token falls for every method as batching amortizes each transfer over more sequences. SP-MoE moves the least data at every tested batch size: HC-SMoE transfers 2.1×, 1.9×, and 1.7× as much data per token. SP-MoE also has the lowest transfer-stall fraction, at 24.8%, 38.2%, and 41.3% of decode time, compared with 39.9%, 50.1%, and 52.7% for HC-SMoE. These results show that the transfer savings persist with batching even as the throughput advantage over HC-SMoE narrows and reverses. Cache occupancy remains at 349 private residuals for SP-MoE and 180 full experts for each baseline across all three batch sizes, as set by the fixed weight budget. Each cache miss fetches a 5.91 MiB private residual for SP-MoE or a 12 MiB full expert for either baseline. Each method has 100 measured batches at $B = 1, 4$ and 60 at $B = 8$, after five discarded warmup batches.

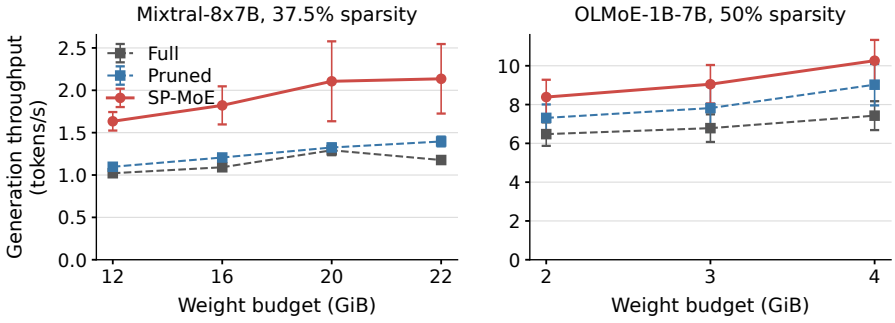
Splitting per-token decode time into its stalled and non-stalled parts accounts for the crossover. SP-MoE's stall saving over HC-SMoE shrinks from 23.6 ms per token at $B = 1$ to 10.3 at $B = 4$ and 4.5 at $B = 8$, because batching amortizes each transfer over more sequences. Its non-stalled time exceeds HC-SMoE's by 6.2, 5.0, and 6.4 ms, a penalty that does not shrink with batch size. HC-SMoE reduces the number of experts and so evaluates fewer of them, whereas decomposed execution is exact over all routed experts, leaving SP-MoE's non-stalled time close to Full MoE's (1.09×, 1.00×, and 1.04×). The reversal therefore occurs where the shrinking transfer saving falls below this roughly batch-invariant execution cost, between $B = 4$ and $B = 8$. The same accounting predicts that the crossover moves to larger batch sizes in more I/O-bound deployments, where the stall term starts higher. The current implementation retains a per-expert dispatch loop; reducing this overhead with matched batched kernels for SP-MoE and HC-SMoE is left to future work.

Table 11: OLMoE batch-size sensitivity under the 3 GiB weight budget. Pruned denotes HC-SMoE (HC), SP denotes SP-MoE; both use 50% sparsity. Tok/s is mean end-to-end generation throughput over the batch, with batch-level standard deviation. SP/HC and SP/Full are throughput ratios; t is Welch's statistic for SP minus HC over completed batches. Stall is the share of decode time waiting on expert transfers; other counters follow Table 2.

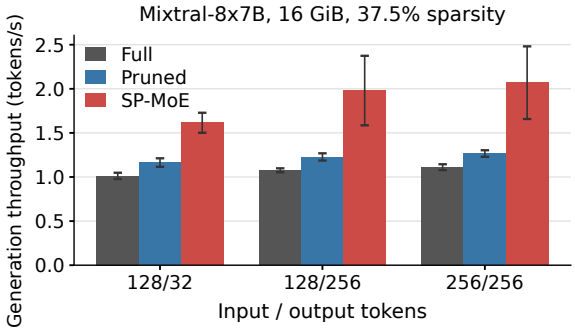
B	Method	Tok/s	Hit (%)	MB/tok	Stall (%)	Cache	SP/HC	SP/Full	t
1	Full	6.78 (0.71)	35.0	552	48.1	180			
	Pruned	7.82 (1.12)	56.3	360	39.9	180	1.16	1.33	8.20
	SP-MoE	**9.05** (0.99)	**59.2**	**170**	**24.8**	349			
4	Full	10.76 (0.80)	30.1	434	57.3	180			
	Pruned	14.36 (1.15)	47.0	263	50.1	180	1.08	1.44	6.62
	SP-MoE	**15.54** (1.34)	**54.1**	**138**	**38.2**	349			
8	Full	14.49 (0.75)	26.0	346	60.5	180			
	Pruned	**21.64** (1.83)	41.4	190	52.7	180	0.96	1.43	−2.83
	SP-MoE	20.77 (1.52)	**49.5**	**113**	**41.3**	349			

D.4 Memory-Budget and Workload Sensitivity

SP-MoE achieves higher generation throughput than HC-SMoE at every tested weight budget and input/output workload (Figure 5). Under Equation 9, throughput ratios on Mixtral at 37.5% sparsity are 1.49, 1.49, 1.59, and 1.53 at 12, 16, 20, and 22 GiB; on OLMoE at 50% sparsity, they are 1.15, 1.16, and 1.14 at 2, 3, and 4 GiB. On Mixtral at 16 GiB and 37.5% sparsity, ratios of 1.39, 1.61, and 1.63 at 128/32, 128/256, and 256/256 input/output tokens show that the advantage persists across short and long generations.



(a) Weight budget, at fixed sparsity



(b) Input/output workload

Figure 5: Sensitivity to the weight budget and to the input/output workload, both under Equation 9. Every point compares SP-MoE against HC-SMoE at the same weight budget. Whiskers are batch-level standard deviation.

E CONSTRUCTION ABLATIONS AND DIAGNOSTICS

This appendix expands Section 4.4 with the neuron-sharing and alignment ablation and dimension-selection diagnostics.

E.1 NEURON SHARING, REMOVAL, AND ALIGNMENT

This subsection gives the per-task values behind Section 4.4's comparison of neuron sharing against removal.

Controlled comparison. Table 12 evaluates three variants from the same OLMoE-1B-7B base checkpoint, using the same C4 calibration (16 sequences of 2,048 tokens) and reconstruction-gradient selection criterion. All three evaluations use LM-Eval Harness 0.4.9.2 with Transformers 4.57.3, zero-shot prompts without a chat template, BF16. We average the nine task scores equally, using `acc_norm` for ARC-C, ARC-E, HellaSwag, PIQA, and OpenBookQA, and `acc` for the others. The reference row re-measures the checkpoint in Table 3 under this rebuilt evaluation environment. Its average is 0.5313 versus 0.5330 in that table; construction variants are compared only within this matched set. The REAP comparison in Appendix C.2 uses the same rebuilt evaluation environment.

Table 12: Neuron sharing, removal, and alignment on OLMoE at 37.5% expert-parameter sparsity. *No shared branch* deletes the lowest-scoring dimensions instead of aggregating them, keeping 640 private dimensions per expert; *no alignment* rebuilds from the un-aligned base at the same α and shared width. Retained intermediate-neuron counts per MoE layer are 40,966, 40,960, and 40,966 against $64 \times 1{,}024 = 65{,}536$ originally. Each neuron contains $3d$ parameters, so the three rows are matched to within 0.02%. All three rows are one matched set under a single harness configuration; differences within the table are meaningful, and absolute values are not comparable to Table 3, which uses the configuration of the main quality evaluation.

Variant	ARC-C	ARC-E	Hella	PIQA	WinoG	OBQA	BoolQ	RTE	MMLU	Avg.
SP-MoE	0.3490	0.6002	**0.6307**	0.6915	0.5927	**0.3620**	0.5743	**0.6065**	**0.3744**	**0.5313**
no shared branch	**0.3609**	**0.6141**	0.5822	**0.7345**	0.5746	0.3280	0.6443	0.5487	0.2904	0.5197
no alignment	0.3490	0.6002	0.5965	0.7089	**0.6140**	0.3340	**0.6651**	0.5812	0.2709	0.5244

Both construction changes lower the average, by 1.2 and 0.7 percentage points. The picture is mixed per task: removing the shared branch improves ARC-C, ARC-E, PIQA and BoolQ, and the average difference comes mainly from MMLU, where it loses 8.4 points. We therefore read these as evidence that both components contribute at this operating point, not as a uniform advantage across tasks. The removal variant retains 640 private dimensions per expert, matching SP-MoE's retained expert-parameter count to within 0.02%.

E.2 Dimension-Selection Diagnostics

Reconstruction-gradient magnitude gives the highest average accuracy at both tested OLMoE sparsities in Table 3. Figure 6 supports this choice through a single-dimension distortion diagnostic on OLMoE layer 0. In the log-scale comparison, gradient magnitude has the strongest correlation with measured distortion ($r = 0.70$), compared with Wanda ($r = 0.16$), Max Similarity ($r = 0.09$), Activation Sparsity ($r = -0.02$), and Min Norm ($r = 0.00$). This stronger association supports using gradient magnitude to identify dimensions whose replacement is more disruptive, retaining them as private and sharing less sensitive dimensions.

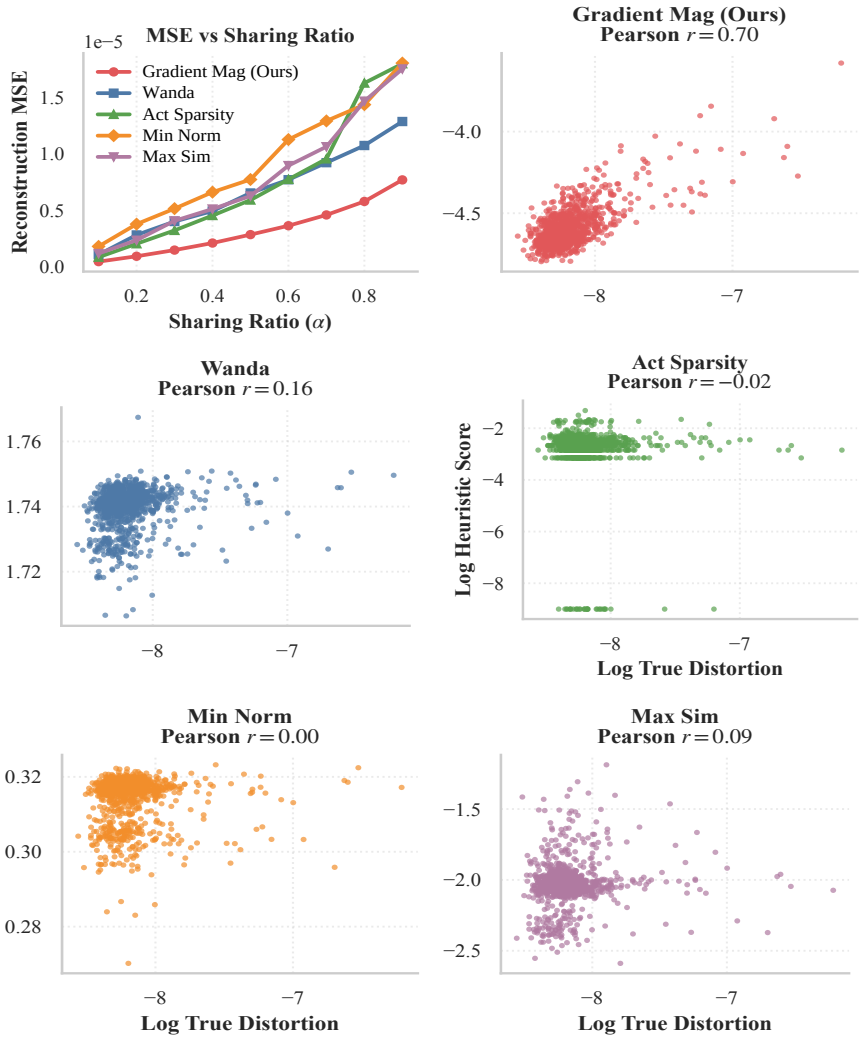


Figure 6: Selection heuristics on OLMoE layer 0. The first panel compares reconstruction distortion across sharing ratios. The remaining panels relate log heuristic scores to log measured single-dimension distortion. Gradient magnitude has the strongest correlation, supporting its use as a local sensitivity signal for dimension selection.