MoRE: Scaling mixture of experts with hardware-aware low-rank routing

Anonymous authors
Paper under double-blind review

Abstract

Mixture-of-Experts (MoE) layers are central to frontier language models, and recent architectures push toward more and smaller experts. In this regime, the standard linear router becomes a bottleneck: with M experts and hidden dimension h, its per-token cost $\Theta(Mh)$ dominates the MoE layer once M is large. We introduce MoRE (**M**ixture **o**f **R**ank-reduced-routed **E**xperts), which factorizes the router weight matrix at rank r and reduces the routing cost to $O((h+M)r)$. We prove that rank logarithmic in M suffices for routing expressivity when the number of active experts is fixed, and is necessary up to precision factors. We also prove that logarithmic rank preserves load balance in a Gaussian memorization model, and training on a synthetic phonebook task shows that low rank does not hurt memorization. At matched active FLOPs, the factorization allows a factor of $\Theta(h/r)$ more experts. To realize this gain in wall-clock time, we design a fused Triton kernel at inference that avoids expensive memory operations on HBM. Empirically, MoRE improves memorization on the phonebook task and performance on knowledge-intensive Q&A benchmarks after pretraining, while matching reasoning ability.

1 Introduction

Mixture-of-experts (MoE) is now the dominant architectural paradigm for frontier language models (DeepSeek-AI, 2024a;b; Jiang et al., 2024; xAI, 2024; Yan et al., 2026). In these architectures, the standard dense MLP is replaced by a pool of independent experts and a router layer that activates only a small subset of experts per token. Recent advances in model scaling rely heavily on increasing the number of experts in MoE layers (Abnar et al., 2025). This approach expands the total parameter count, allowing models to benefit from neural scaling laws without proportionally increasing active compute (Kaplan et al., 2020; Abnar et al., 2025). In particular, at a fixed active compute budget, MoE layers achieve significantly better *memorization and factual recall* than dense MLPs (Jelassi et al., 2025). Recent frontier MoE models are also increasingly operating in the *fine-grained regime* (Appendix A), in which the layer is divided into a larger number of smaller experts, each with intermediate width below the hidden dimension (Krajewski et al., 2024).

However, further scaling the number of experts in MoE layers hits a bottleneck: routing costs begin to dominate the expert cost as the expert pool grows. This is driven by two distinct factors: the increased active compute required to calculate the routing function, and the heavy memory-bandwidth cost of writing and reading the routing scores. These combined costs can offset the efficiency gains of sparsity in the many-expert regime (see Figure 1). This motivates our main question:

Can we scale the number of experts in MoEs efficiently?

1.1 Our contributions

In this work, we propose **M**ixture **o**f **R**ank-reduced-routed **E**xperts (MoRE), an architectural design that allows for scaling to more experts with an asymptotically lower computational cost. **MoRE replaces the full-rank linear router with a low-rank router**. This factorization reduces the number of parameters in the routing function and the FLOPs required for a forward pass, allowing scaling to more experts. Our router design is inspired by the low-rank parameterization of key-query (KQ)

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.

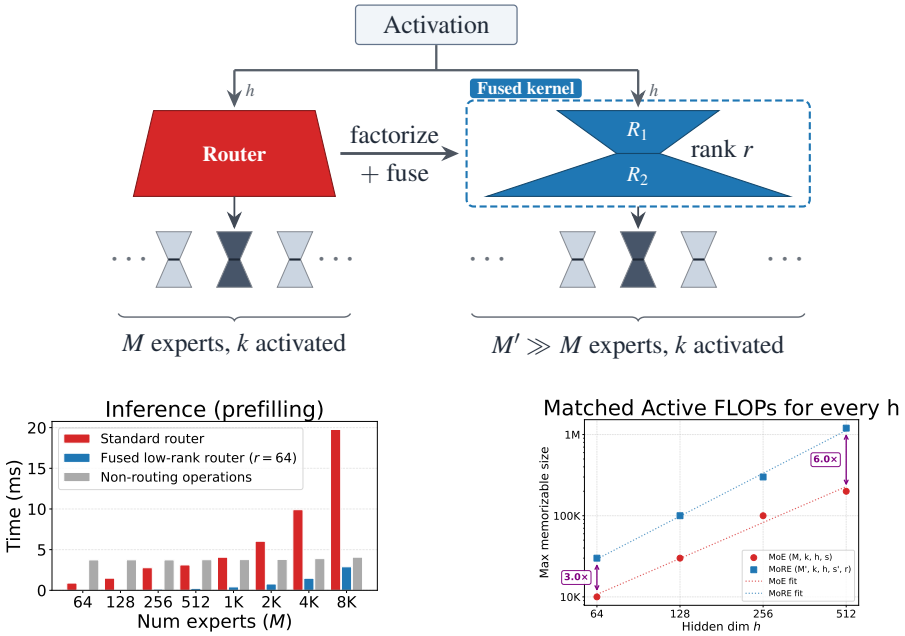


Figure 1: **MoRE unlocks more scaling with the same active FLOPs. (Top)** Replacing the full-rank $h \times M$ router with a low-rank factorization R_1, R_2 inside a fused kernel cuts routing cost from $O(hM)$ to $O(r(h + M'))$, enabling $M' \gg M$ experts at matched active FLOPs. **(Bottom Left)** Inference (prefilling) wall-clock time of one layer at h=512, split into the router and the rest (setup as in Figure 4). The standard router's cost grows linearly with M and dominates at large M, while MoRE's fused low-rank router grows far more slowly. **(Bottom Right)** At matched FLOPs, MoRE memorizes more on the synthetic phonebook task, and the gap widens with h, as predicted by the $\Theta(h/r)$ scaling of Section 4.1.

weights in recent attention architectures (DeepSeek-AI, 2024a) as well as the low-rank design of LoRA adapters (Hu et al., 2021).

Expressivity guarantees. To determine whether the rank bottleneck limits MoRE's routing capacity, we study which expert assignments it can realize and how evenly it can use the experts. In Section 3, we first prove general upper and lower bounds on the rank needed to realize any top-k assignment. Then, we prove that **low-rank routing in MoRE layers preserves load-balancing ability**, by constructing a router with $r = \Theta(\log M)$ that balances expert assignment in a Gaussian memorization model extending that of Jelassi et al. (2025). By spreading inputs across experts, balanced routing helps use the memorization capacity of the whole expert pool. We also show empirically on a synthetic memorization task, Phonebook, that low rank does not hurt load balancing and memorization (Section 3.2).

Scaling experts efficiently. At matched active FLOPs, a MoRE layer with rank-r routing and hidden dimension h supports a factor of $\Theta(h/r)$ more experts than an MoE layer with the standard router (Section 4.1). However, realizing this gain in wall-clock time requires more than a FLOP reduction: materializing the score tensor in HBM (High Bandwidth Memory) becomes expensive when M is large. In this paper, we focus on inference, since a model is trained once but runs inference on every request after deployment and during RL post-training. We therefore design a fused kernel that keeps scores in on-chip SRAM (Section 4). The kernel computes scores in blocks and retains only the running top-k values and expert indices. Notably, the **inference time** (prefilling) of MoRE matches that of MoE at matched active FLOPs, with a much larger number of experts.

Experimental validation. Finally, we validate our architecture through pretraining experiments on both the phonebook task and real-world Internet data. We show that **MoRE models consistently improve performance in knowledge-intensive** tasks such as TriviaQA (Joshi et al., 2017) and Natural Questions (Kwiatkowski et al., 2019), compared with standard MoEs at equal active FLOPs. Additionally, we find that MoRE matches performance on reasoning tasks.

1.2 Related Work

Routing in MoE layers. The standard router in modern MoE is a learned linear layer that scores all M experts per token (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022). *Hash routing* fixes the assignment to a deterministic hash (Roller et al., 2021), but Clark et al. (2022); Dikkala et al. (2023) show that deterministic assignment underperforms learned routing at scale, motivating the search for *parameter-efficient* learned routers rather than discarding learning. Boix-Adsera (2025) suggests low-rank routing but evaluates it only in a limited distillation setting. In the $M \ll h$ regime, X-MoE (Chi et al., 2022) and L2R (Yang et al., 2026) also introduce low-rank projections, but to address issues in token representations and routing geometry. MoRE instead targets the fine-grained regime where M is large and $M \ll h$ no longer holds, using low-rank routing to reduce the cost of scoring many experts. He (2024) introduces *PEER*, a product-key retrieval layer with cost $O(h\sqrt{M})$. At matched active FLOPs, MoRE takes much less wall-clock time than PEER (see Appendix B).

Fine-grained MoE and scaling. Recent architectures are increasingly designed in the *fine-grained* regime, pairing a small expert intermediate dimension s with a large expert count M. Krajewski et al. (2024) introduce fine-grained MoE and show that it outperforms coarse-grained MoE at a fixed training budget, and that higher granularity is out of reach because of the routing cost. Li et al. (2026) confirm through extensive experiments that larger granularity and expert counts result in better performance. Since MoE memorization capacity scales with *total* rather than active parameters (Jelassi et al., 2025), designs with large M are directly motivated. Nevertheless, these works do not address the routing bottleneck that emerges as M grows, and we introduce MoRE to reduce the routing cost.

Efficient MoE systems. Existing fused MoE kernels target *expert dispatch*: ScatterMoE (Tan et al., 2024), MegaBlocks (Gale et al., 2023), and follow-up IO-aware kernels (Costin et al., 2025; Guo et al., 2025) fuse scatter-gather with the per-expert grouped matmul, but leave open the question of how to implement the router efficiently in the many-expert regime. vLLM (Kwon et al., 2023) provides a `FusedTopKRouter` that fuses top-k selection with softmax normalization, but does not fuse the router computation with top-k. At inference, expert offloading (Hwang et al., 2024; Xue et al., 2024; He et al., 2024) reduces serving cost by moving inactive experts to CPU memory, but does not reduce the router computation. MoRE instead fuses both low-rank matmuls and top-k selection into one router kernel (Section 4.2).

We refer the reader to Appendix B for a more detailed discussion of related work.

2 Preliminaries

Notation. We write h for the hidden dimension, s for the expert intermediate dimension, M for the number of experts per layer, k for the number of active experts per token (with $k < M$), r for the rank of the low-rank router, and L for the number of transformer layers. With batch size B and sequence length T, each iteration processes $N = BT$ tokens. We write $[m] = \{1, \dots, m\}$, and use P for the total parameter count and τ for the per-token active FLOPs.

Mixture-of-experts layer. An MoE layer has M experts $\{f_i\}_{i=1}^{M}$ and a router $g : \mathbb{R}^h \to \mathbb{R}^M$. Each expert is usually a SwiGLU MLP (Shazeer, 2020) with three weight matrices $W_i^{\text{gate}}, W_i^{\text{up}} \in \mathbb{R}^{s\times h}$ and $W_i^{\text{down}} \in \mathbb{R}^{h\times s}$:

$$f_i(x) \;=\; W_i^{\text{down}}\big(\sigma(W_i^{\text{gate}}x) \odot W_i^{\text{up}}x\big), \qquad f_{\text{MoE}}(x) \;=\; \sum_{i=1}^{M} g_i(x) f_i(x),$$

with σ the SiLU activation and $\odot$ the elementwise product. Each expert thus has $3hs$ parameters. The **standard router** (Shazeer et al., 2017; Fedus et al., 2022) is parametrized by a linear projection $R \in \mathbb{R}^{M\times h}$ and applies softmax over the k largest scores: $g(x; R) = \text{softmax}\big(\text{TopK}(Rx)\big)$, where $\text{TopK}(x)$ denotes the k largest values of $x \in \mathbb{R}^M$. We write its configuration as $\text{MoE}(M, k, h, s)$.

MoRE layer. In a MoRE layer, we replace R with R_1R_2 where $R_1 \in \mathbb{R}^{M\times r}$ and $R_2 \in \mathbb{R}^{r\times h}$, scoring through an r-dimensional bottleneck:

$$g(x; R_1, R_2) = \text{softmax}\big(\text{TopK}(R_1R_2x)\big).$$

Routing parameters drop from Mh to $(M + h)r$. We write the low-rank configuration as $\text{MoRE}(M, k, h, s, r)$.

Total parameters and active cost. The total MLP parameter count per MoE/MoRE layer is $3Mhs$, and the per-token active FLOPs per layer are

$$\tau \;=\; \begin{cases} C_e\, khs + C_r\, Mh & \text{(standard router)}, \\ C_e\, khs + C_r\, (M+h)\, r & \text{(low-rank router)}, \end{cases} \tag{1}$$

where C_e, C_r collect per-operation FLOP factors. Following Krajewski et al. (2024, App. E), at inference we have $(C_e, C_r) = (6, 6)$. Training gives a similar ratio, $(C_e, C_r) = (18, 14)$.

3 Low-rank routing preserves memorization capabilities

MoRE replaces the standard router with a rank-r factorization, which raises two related theoretical questions. First, can the router still express the needed expert assignments through the rank bottleneck? Second, when there are many inputs to memorize, can the low-rank router still use the experts' parameters efficiently by balancing the inputs assigned to them? We answer both questions positively in Sections 3.1 and 3.2, and confirm the answers with experiments on the synthetic Phonebook memorization task.

3.1 Routing expressivity

Here an assignment is a subset of k experts. We show that, for fixed k, a *single* router can select every assignment in a given family for some input, with rank growing only logarithmically in the number of experts and assignments. This dependence on the number of assignments is necessary up to a factor depending on numerical precision.

Setup. The rank-r router has a simple geometric form: the rows of $R_1 \in \mathbb{R}^{M\times r}$ define the M expert vectors $u_i \in \mathbb{R}^r$, and each input $x \in \mathbb{R}^h$ reaches R_1 only through its projected representation $z(x) := R_2 x \in \mathbb{R}^r$ where $R_2 \in \mathbb{R}^{r\times h}$. The router then selects the k experts with the largest routing scores $\langle u_i, z(x)\rangle$, $1 \le i \le M$. For exposition, throughout this section we normalize the scale by assuming $\|u_i\|_2 \le 1$ for every expert vector and $\|z(x)\|_2 \le 1$ for every input x under consideration.[1]

Margin and precision. Because top-k is implemented at finite precision, selecting a subset robustly requires a positive gap between selected and unselected scores. We formalize this as follows.

Definition 1. *Fix a rank-r router $R = R_1 R_2$. A subset $S \in \binom{[M]}{k}$ is* realized at input x with margin $\eta > 0$ *if $S = \mathrm{TopK}(Rx)$ and*

$$\min_{i\in S} (Rx)_i - \max_{j\notin S} (Rx)_j \ge \eta.$$

Moreover, we say a family $\mathcal{F} \subseteq \binom{[M]}{k}$ is realizable at margin η *by R if every $S \in \mathcal{F}$ has some input $x_S \in \mathbb{R}^h$ that realizes it at margin at least η.*

Note that for a p-bit implementation, a margin η is resolvable when $\eta = \Omega(2^{-p})$, which is equivalent to $p = \Omega(\log(1/\eta))$. We first present an upper bound based on the Johnson–Lindenstrauss lemma (Johnson & Lindenstrauss, 1984) by providing a router construction:

Theorem 1 (Upper bound on required rank)**.** *There exists a universal constant $C > 0$ such that the following holds. For $1 \le k \le M-1$, any collection of top-k assignments $\mathcal{F} \subseteq \binom{[M]}{k}$, and any rank*

$$r \ge Ck \log(M + |\mathcal{F}|), \qquad h \ge r,$$

there exists a rank-r router $R = R_1 R_2$ with $\|u_i\|_2 \le 1$ that realizes every $S \in \mathcal{F}$ at margin $\Omega(1/\sqrt{k})$, using inputs $\{x_S\}_{S\in\mathcal{F}}$ with $\|R_2 x_S\|_2 \le 1$.

Proof idea. In the uncompressed case $r = M$, take the expert vectors to be the standard basis vectors $e_1, \ldots, e_M \in \mathbb{R}^M$. The representation $z_S \propto \frac{\mathbf{1}_S}{k} - \frac{\mathbf{1}_{[M]\setminus S}}{M-k}$ is positive exactly on S, so it routes to S with margin $\Theta(1/\sqrt{k})$. The Johnson–Lindenstrauss lemma maps these $M + |\mathcal{F}|$ vectors to $r = \Theta(\log(M + |\mathcal{F}|)/\varepsilon^2)$ dimensions while preserving inner products up to additive error ε, and $\varepsilon = \Theta(1/\sqrt{k})$ keeps a constant fraction of the margin, which gives $r = O(k \log(M + |\mathcal{F}|))$. The full proof is in Appendix I.

[1] We state the general result in Appendix I, which only requires them to be bounded by some constants B_u and B_z.

Next, we use a volume-packing argument to prove a lower bound on the rank. The intuition is that if two different assignments $S, T \in \mathcal{F}$ are both realized with margin η, their projected representations must be separated.

Theorem 2 (Lower bound on required rank)**.** *Let $\mathcal{F} \subseteq \binom{[M]}{k}$. If a rank-$r$ router $R = R_1 R_2$ with $\|u_i\|_2 \le 1$ and $\|R_2 x_S\|_2 \le 1$ realizes every top-k assignment $S \in \mathcal{F}$ at margin at least η, then*

$$|\mathcal{F}| \le (1 + 2/\eta)^r, \qquad r \ge \frac{\log |\mathcal{F}|}{\log(1 + 2/\eta)}.$$

Proof idea. If two assignments $S \neq T$ are realized at x_S and x_T, pick $i \in S \setminus T$ and $j \in T \setminus S$. Expert i beats expert j by at least η at x_S and loses by at least η at x_T, so $\|u_i - u_j\|_2 \le 2$ forces $\|R_2 x_S - R_2 x_T\|_2 \ge \eta$. Packing $|\mathcal{F}|$ such points into the unit ball of $\mathbb{R}^r$ gives $|\mathcal{F}| \le (1 + 2/\eta)^r$.

Finally, taking $\mathcal{F}$ to be the collection of all top-k subsets gives the following consequence:

Corollary 1 (Full expressivity)**.** *At precision $p = \Omega(\log k)$, rank $r = O(k^2 \log(M/k))$ suffices to realize every top-k assignment over M experts. Conversely, any rank-r router that realizes every top-k assignment at precision scale p must have $r = \Omega(k \log(M/k)/p)$.*

Remark. The extra factor of k comes from preserving the routing margin in the Johnson–Lindenstrauss projection. Our bounds do not establish whether this factor is necessary.

3.2 Load balancing and memorization capacity

Memorization, a core function of MoE layers (Du et al., 2022; Geva et al., 2021; Jelassi et al., 2025), concerns how many facts can be stored, not which expert configurations can be addressed. Extending the Gaussian model of Jelassi et al. (2025), we prove that logarithmic rank does not affect load balancing, and then explain how load balancing relates to memorization capacity.

Proposition 1 (Gaussian load balancing with logarithmic rank)**.** *Let $x_1, \ldots, x_n \sim \mathcal{N}(0, I_h)$ be independent, and suppose $M = 2^r$ ($r = \Theta(\log_2 M)$) with $h \ge r$. For any $1 \le k \le M$, there exists a rank-r top-k router with M experts that selects every expert with probability exactly k/M on each example. Consequently, let N_a be the number of examples routed to expert a, and write $L := \log \frac{2M}{\delta}$. For any $\delta \in (0, 1)$, the following holds with probability at least $1 - \delta$:*

$$\max_{a \in [M]} \left| N_a - \frac{nk}{M} \right| \le \sqrt{\frac{2nkL}{M}} + \frac{2}{3} L.$$

The key intuition is that, under this construction, a standard Gaussian input has no preference for one expert over another in expectation. We defer the full proof to Appendix I. Thus, the low-rank factorization preserves the load-balanced mechanism of standard MoE, while reducing routing complexity from $O(Mh)$ to $O((M + h) \log M)$.

Why balanced load maximizes the number of stored facts. The number of facts an MLP expert can memorize is determined by its parameter count, as shown for two-layer networks (Daniely, 2020) and for language models (Allen-Zhu & Li, 2024; 2023). An MoE layer reaches the total capacity of its experts only under balanced load: an overloaded expert receives more facts than it can memorize, and an underutilized expert leaves its parameters unused.

Experimental validation with trained models. We verify in training that sufficient rank does not affect memorization, using the phonebook task of Jelassi et al. (2025). Each example is a five-letter name followed by an eight-digit number, such as "alicel12345678". We pretrain on these pairs and report the largest number of training examples for which the model attains $\ge 90\%$ exact-match accuracy on the numbers given the names, evaluated on $1{,}000$ training examples sampled uniformly at random. To approach the model's full memorization capacity, we train for up to $1{,}500$ epochs, in line with Allen-Zhu & Li (2024).

Figure 2(Left) varies the router rank r while keeping the expert configuration fixed: MoRE matches the memorization capacity of standard MoE once r is sufficiently large. To compare load balancing, Figure 2(Middle, Right) tracks normalized routing entropy, $H_{\text{norm}} = -\sum_{a=1}^{M} p_a \log p_a / \log M$, where p_a is expert a's share of routed tokens. Both models reach high entropy by the end of training.

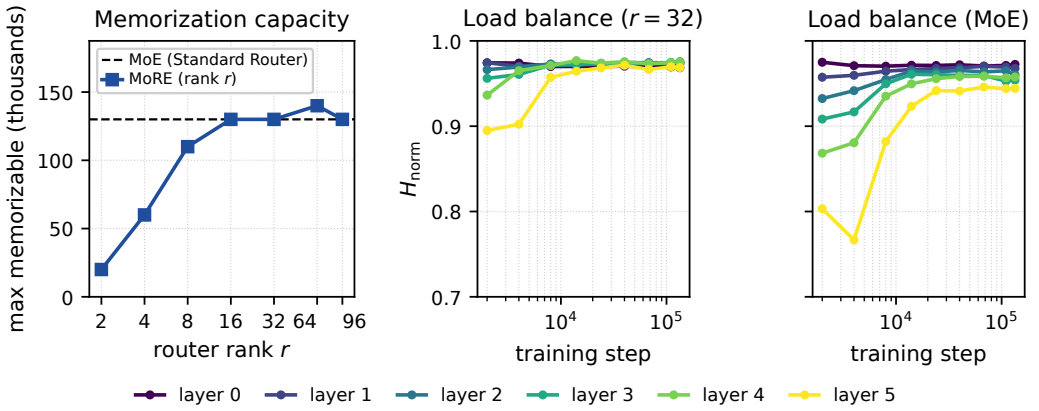


Figure 2: **(Left) Sufficient rank does not affect memorization. (Middle, Right) MoRE with $r = 32$ and MoE have similar load balancing at the end of training.** On the Phonebook task, at M=192, h=128, s=8, k=4, L=6. **Left:** maximum phonebook size MoRE memorizes against the router rank. The dashed line is MoE (standard router) at the same configuration, which MoRE already matches at $r = 16$. **Middle, Right:** routing entropy H_{norm} per layer against training step for MoRE (r=32) and MoE. 1 means equal load balancing and 0 means collapse onto one expert. We define H_{norm} and show similar results for other setups in Appendix G.1.

4 Scaling experts efficiently with fused implementation of MoRE

Since logarithmic rank keeps the load balanced and does not hurt memorization (Section 3), we ask: *at a fixed active-FLOP budget, how much additional capacity does the low-rank router unlock?* Section 4.1 shows a $\Theta(h/r)$ gain in the expert count.[2] The resulting large-M regime creates a memory-bandwidth bottleneck on the routing score tensor, which we resolve with a fused Triton (Tillet et al., 2019) kernel for the router (Section 4.2).

4.1 Scaling under fixed active-FLOPs

Fix the active-FLOP budget τ, the hidden dimension h, and the active expert count k, so that all non-MoE components have identical parameter counts across the two settings. We compare $\text{MoE}(M, k, h, s)$ and $\text{MoRE}(M', k, h, s', r)$ on total MLP parameters $P = 3Mhs$. The full calculation is in Proposition 2 (Appendix H).

The active budget τ decomposes into an expert cost $C_e khs$ and a router cost. For MoE, the router cost is $C_r Mh$, i.e. $C_r h$ FLOPs per expert. For MoRE, the router cost is $C_r M'r + C_r hr$, where the per-expert term $C_r r$ is an h/r reduction over MoE, and $C_r hr$ is a fixed projection overhead independent of M'. Since $P = 3Mhs \propto Ms$ is a product, and the budget trades off linearly between scaling M (via the router) and scaling s (via the experts), P is maximized when the budget is split equally between the router and the experts. This gives

$$M = \frac{\tau}{2C_r h} \quad \text{(MoE)}, \qquad M' = \frac{\tau - C_r hr}{2C_r r} \quad \text{(MoRE)}.$$

In the regime $\tau \gg C_r hr$ (equivalently $M, M' \gg r$), the projection overhead is negligible, and $M'/M \approx h/r$: each MoRE expert costs only a fraction r/h of the routing cost of an MoE expert. For example, at $h = 2048$ and $r = 16$, MoRE admits roughly $128\times$ more experts than MoE at the same active cost. In Section 5, we choose configurations according to this derivation, with $C_e = C_r$.[3] We find that the same $\Theta(h/r)$ conclusion also holds under multi-GPU expert parallelism (Appendix E.3.1).

[2] In Appendix G.7, we find that adding experts monotonically reduces MoRE's validation perplexity.

[3] Since $C_e = C_r = 6$ for inference and $C_e \approx C_r$ in training, the active FLOPs roughly match in both.

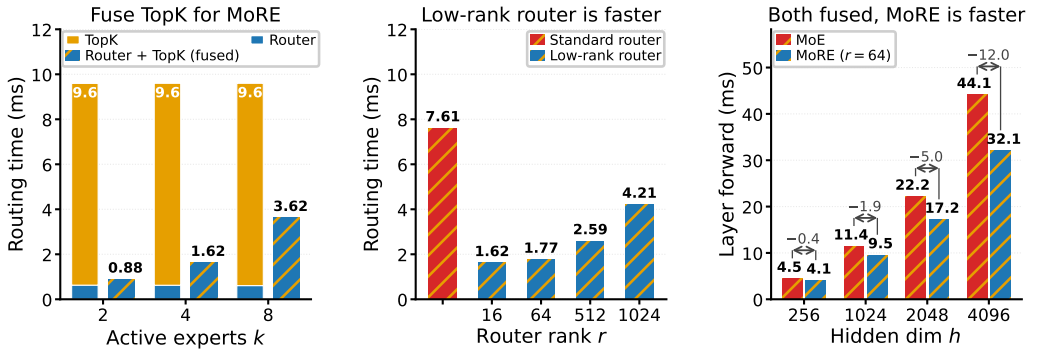


Figure 3: **(Left)** Fusing with top-k eliminates the need to materialize the scores for MoRE. **(Middle)** The low-rank router is faster than the standard router, while its cost grows with r. **(Right)** The gap between MoRE and MoE widens as h grows. M=4,096, N = 262,144 (B=256, T=1024), single NVIDIA B200, mean of 200 trials after 50 warm-ups. Left and middle time the router alone at h=2048 (left: r=16, middle: k=4). Right times the whole layer at s=64, k=4, r=64.

4.2 Realizing the speedup at large M

When the expert count M and the number of tokens N are large, running a top-k over the full (N, M) routing-score tensor requires writing the tensor to HBM (High Bandwidth Memory, see Appendix E.1) and reading it back, and dominates the routing time (Figure 3, Left). This requires *fusing operations* in addition to the low-rank factorization.

Below, we sketch how MoRE can be fused and leave the full details to Appendix F.1. Instead of writing the routing scores for every token to HBM and reading them back to compute the top-k scores, we compute $R_1 R_2 x$ and maintain a running top-k buffer (Figure 3, Left)[4]. We find that the low-rank router in MoRE is faster than the standard router across different ranks (Figure 3, Middle). MoE can be fused in a similar way, but we find that its speedup from fusing becomes small when h is large, while the speedup for MoRE remains visible, because for MoE the cost of computing Rx and maintaining the top-k buffer scales with h instead of r (Figure 3, Right). We also show similar speedups on multiple GPUs with expert and data parallelism, where communication costs are involved (Appendix E.3).

Wall-clock time at inference in the large-M regime. As shown in Figure 4, the fused MoRE layer outperforms the standard MoE layer across the whole grid, and the speedup grows with M along every row, reaching 1.83 to 4.54× at M = 8,192. Equivalently, at matched wall-clock time, MoRE supports about eight times more experts: at $h = 512$, the standard layer reaches $M = 512$ in 6.90 ms, while MoRE reaches $M = 4{,}096$ in 5.37 ms, realizing the scaling advantage of Section 4.1 in practice. Fusing also reduces peak memory by up to 14× for the whole layer (Figure 17). We find that MoE and MoRE have similar decoding speed (see Appendix E.4).

5 Experiments: MoRE improves performance at matched active FLOPs

In this section, we show that replacing MoE layers with MoRE layers in transformers can improve performance on memorization and knowledge-intensive tasks while keeping active FLOPs fixed.

5.1 Synthetic task: increased memorization capabilities at matched active FLOPs

We first test on the phonebook task (Section 3.2) whether MoRE's parameter advantage at matched active FLOPs translates into memorization capacity. MoRE memorizes substantially more pairs than MoE at every scale we sweep. In Figure 5, we sweep the hidden dimension $h \in \{64, 128, 256, 512\}$ at expert width $s = 8$, and the expert width $s \in \{8, 16, 32, 64\}$ at $h = 64$. At each point, we choose

[4]Our algorithm works well for common choices of k. For a discussion of our top-k algorithm and a comparison with other implementations, see Appendix E.2.

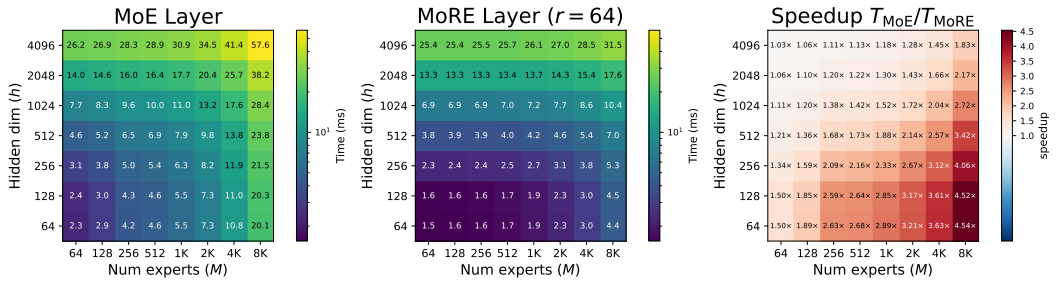


Figure 4: **Inference (prefilling): At a fixed configuration the MoRE layer is** 1.83 **to** $4.54\times$ **faster at** M=8,192**, and at matched active FLOPs it has comparable speed to the standard MoE layer while holding** h/r **times more experts.** Forward pass of a single layer. **Left:** MoE layer. **Middle:** MoRE layer (r=64). **Right:** speedup $T_{\text{MoE}}/T_{\text{MoRE}}$. s=64, k=4, $N = 262{,}144$ (B=256, T=1024), mean of 200 trials after 50 warm-ups on a single NVIDIA B200. The comparison where both layers are fused is in Figure 20. We provide additional measurements on an A100, with similar speedups, in Appendix G.3.

the optimal MoE and MoRE configurations by Proposition 2 so that they have matched active FLOPs. Notably, the gap widens with larger h, in agreement with the $\Theta(h/r)$ trend of Proposition 2.[5]

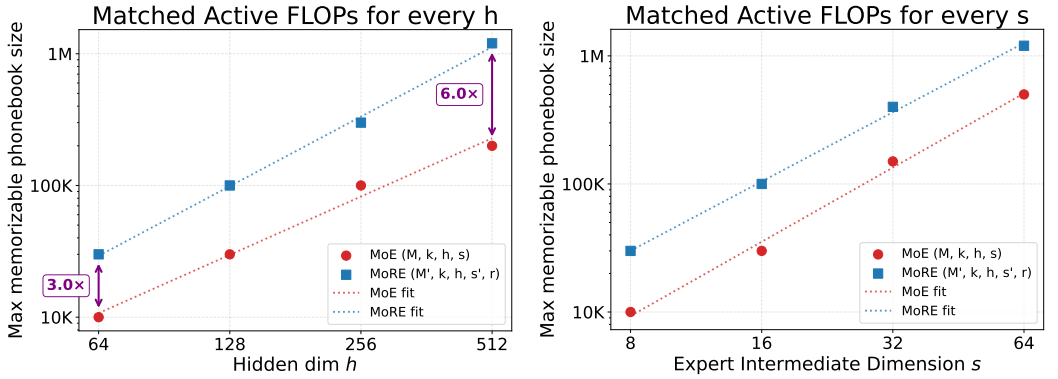


Figure 5: **At fixed active FLOPs** τ**, MoRE memorizes more phonebook entries than MoE, and the gap widens with** h **as Proposition 2 predicts. Left:** sweep over hidden dimension $h \in \{64, 128, 256, 512\}$ at expert width $s = 8$. **Right:** sweep over expert width $s \in \{8, 16, 32, 64\}$ at $h = 64$. Dotted lines are log-log linear fits. Configurations and sizes are in Tables 5 and 6 (Appendix C.1).

5.2 Real-world pretraining: improved performance on knowledge-intensive tasks

For the real-world pretraining, we construct a 40B-token mixture dataset following the natural-language pretraining recipe of Jelassi et al. (2025): a mixture of FineWeb-edu (HuggingFaceFW, 2024), Cosmopedia (Ben Allal et al., 2024), and Wikipedia, together with the training sets of the downstream benchmarks, so that the model is familiar with the downstream tasks at test time (Jelassi et al., 2025; Allen-Zhu & Li, 2023).

We select two MoE/MoRE pairs with matched active FLOPs by Section 4.1, a third pair with wider experts (s and s'), and a larger pair with $M = 1024$ and $M' = \frac{h}{r} \cdot M = 8192$, trained with expert parallelism. By construction, MoRE has 5 to 6$\times$ more total parameters. On the first three pairs, MoRE reduces validation perplexity by 7% to 15% (Table 1).[6]

[5]We stop at $h = 512$ and $s = 64$ because larger models memorize more than 2 million examples, and training on such datasets for up to 1,500 epochs exceeds our compute budget.

[6]The gap is smaller for the last pair. A possible reason is that a larger model needs >40B tokens, which is beyond our budget. In Appendix G.9, we find that the perplexity gap increases with the token budget.

s, h	Mo(R)E-M	Active MoE FLOPs / layer	Total Param	Val PPL↓	TriviaQA F1↑	NQ-Open F1↑	Hella. acc_norm↑	ARC-e acc↑	SciQ acc↑	Wino. acc↑
32, 1024	MoE-128	1.57M	287.0M	14.01	16.57 ±0.24	6.88 ±0.32	36.91 ±0.48	56.31 ±1.02	79.60 ±1.27	50.20 ±1.41
	MoRE-1536	1.57M	1.50B	**11.96**	**20.21 ±0.25**	**7.63 ±0.34**	**42.28 ±0.49**	**63.30 ±0.99**	**80.20 ±1.26**	**53.51 ±1.40**
64, 512	MoE-256	1.57M	362.9M	14.01	16.29 ±0.24	5.72 ±0.31	36.95 ±0.48	57.32 ±1.01	75.80 ±1.36	**51.46 ±1.40**
	MoRE-1792	1.57M	1.91B	**12.66**	**17.12 ±0.23**	**6.88 ±0.32**	**40.15 ±0.49**	**60.44 ±1.00**	**77.70 ±1.32**	50.28 ±1.41
128, 512	MoE-256	2.36M	664.9M	12.81	18.09 ±0.24	5.99 ±0.29	40.00 ±0.49	59.68 ±1.01	76.30 ±1.35	51.22 ±1.40
	MoRE-1792	2.26M	3.76B	**11.92**	**20.79 ±0.26**	**7.62 ±0.34**	**42.07 ±0.49**	**60.27 ±1.00**	**78.70 ±1.30**	**54.70 ±1.40**
64, 512	MoE-1024	3.93M	1.27B	12.43	21.67 ±0.29	**8.75 ±0.40**	40.82 ±0.49	60.86 ±1.00	77.10 ±1.33	**53.67 ±1.40**
	MoRE-8192	3.98M	7.92B	**11.92**	**22.15 ±0.29**	8.68 ±0.39	**42.33 ±0.49**	**61.74 ±1.00**	**81.50 ±1.23**	51.70 ±1.40

Table 1: Pretraining results for MoE Transformers on 40B tokens. Each MoRE configuration ($r = 64$) matches the active FLOPs of its MoE counterpart, while total parameters are scaled up. All tasks are evaluated 0-shot, and ± denotes the standard error over test samples. Active MoE FLOPs shown are at inference and we have matched FLOPs at training too. See Appendices C.2 and C.3 for details.

We evaluate 0-shot via lm-evaluation-harness (Gao et al., 2023) on **knowledge-intensive** tasks (TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019)) and **commonsense and science reasoning** (HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-Easy (Clark et al., 2018), SciQ (Welbl et al., 2017)). MoRE matches or improves knowledge-intensive performance while preserving commonsense and reasoning scores.

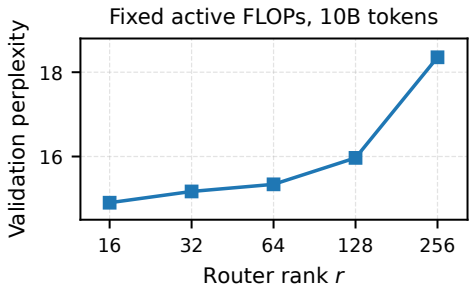


Figure 6: **With active FLOPs fixed, a smaller rank allows more experts, and validation perplexity falls as the rank shrinks.** The number of experts M grows from 64 to 8704 as r falls from 256 to 16, at $h = 512$, $k = 4$, $L = 12$ and 10B training tokens.

End-to-end wall-clock cost comparison. At fixed active FLOPs (Table 1), MoE and MoRE have similar inference time in both prefilling and decoding.[7] With the same configuration, training MoRE is 1.10–1.23× faster than training MoE (Appendix E.5).

Ablation of the router rank at matched active FLOPs. Reducing r from 256 to 16 at fixed active FLOPs raises MoRE's expert count from 64 to 8,704 and lowers validation perplexity monotonically, from 18.35 to 14.90 (Figure 6). As the benefit plateaus at $r = 64$, we keep $r = 64$ throughout our pretraining experiments.

6 Conclusion and Discussion

We introduce *MoRE*, a principled solution to the MoE routing bottleneck that scales experts by a factor of $\Theta(\frac{h}{r})$ while matching active FLOPs. We provide theoretical guarantees for its expressivity, design a hardware-efficient implementation, and demonstrate improved performance on knowledge-intensive benchmarks such as TriviaQA (Joshi et al., 2017) and Natural Questions (Kwiatkowski et al., 2019), without degrading reasoning.

Limitations and future work. Our real-world training experiments are still limited in scale ($h \leq 2048$, fewer than 10B total parameters) due to our limited computational budget. We also mainly compare MoE and MoRE under the principle of Section 4.1 for improving memorization, and future work could study the scaling laws of the two architectures. While we find that matching the standard router's initialization variance can already improve upon the default initialization of R_1, R_2 (Appendix D), a LoRA-like (Hu et al., 2021) initialization might improve performance further.

AI use statement

We used Claude Opus 4.8 and Claude Opus 5 to polish the writing and to create or modify scientific figures. The authors identified the memory bottleneck of the router, and Claude Opus 4.8 assisted

[7] At matched active FLOPs, training MoRE can be slower because of work that scales with total parameters, such as the AdamW update. A more careful implementation may shorten this gap (Appendix E.5).

with implementing the Triton kernels, measuring wall-clock time, and running experiments. We also used these models to refine mathematical proofs. We did not use them to analyze or interpret results. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

Reproducibility statement

We provide details for reproducing our experiments in Appendix C. The supplementary material includes our code for dataset generation and pretraining experiments, as well as our inference kernel. We will publicly release the kernel upon acceptance and hope it will benefit the community.

References

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin Mohamed Elnouby Ali, Josh Susskind, and Vimal Thilak. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models. *arXiv preprint arXiv:2501.12370*, 2025.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction, 2023. URL https://arxiv.org/abs/2309.14316.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws, 2024. URL https://arxiv.org/abs/2404.05405.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Cosmopedia: how to create large-scale synthetic data for pre-training. https://huggingface.co/blog/cosmopedia, 2024.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of EMNLP*, 2013.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale, 2024. URL https://arxiv.org/abs/2412.09764.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of AAAI*, 2020.

Enric Boix-Adsera. Secret mixtures of experts inside your llm. *arXiv preprint arXiv:2512.18452*, 2025.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models, 2026. URL https://arxiv.org/abs/2601.07372.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, Heyan Huang, and Furu Wei. On the representation collapse of sparse mixture of experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2204.09179.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models, 2022. URL https://arxiv.org/abs/2202.01169.

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.

Bobby Costin, Timor Averbuch, Dhruv Pai, Nathan Chen, and Ben Keigwin. Momoe: Memory-optimized mixture of experts. https://blog.tilderesearch.com/blog/momoe, 2025. Tilde Research blog, July 25, 2025.

Amit Daniely. Memorizing gaussians with no over-parameterizaion via gradient decent on neural networks, 2020. URL https://arxiv.org/abs/2003.12895.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024a. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI. Deepseek-v3 technical report, 2024b. URL https://arxiv.org/abs/2412.19437.

Nishanth Dikkala, Nikhil Ghosh, Raghu Meka, Rina Panigrahy, Nikhil Vyas, and Xin Wang. On the benefits of learning to route in mixture-of-experts models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=QV79qiKAjD.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022. URL https://arxiv.org/abs/2101.03961.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems*, volume 5, 2023.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, 2021.

Wentao Guo, Mayank Mishra, Xinle Cheng, Ion Stoica, and Tri Dao. Sonicmoe: Accelerating moe with io and tile-aware optimizations, 2025. URL https://arxiv.org/abs/2512.14080.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models, 2024. URL https://arxiv.org/abs/2402.12354.

Xin He, Shunkang Zhang, Yuxin Wang, Zihao Zeng, Kaijie Tang, Shaohuai Shi, Zhenheng Tang, Haiyan Yin, Ivor W. Tsang, Yew Soon Ong, and Xiaowen Chu. Expertflow: Optimized expert activation and token allocation for efficient mixture-of-experts inference, 2024. URL https://arxiv.org/abs/2410.17954.

Xu Owen He. Mixture of a million experts, 2024. URL https://arxiv.org/abs/2407.04153.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for LLM training, 2024. URL https://arxiv.org/abs/2410.10989.

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, 2021. URL https://arxiv.org/abs/2106.09685.

Zihao Huang, Qiyang Min, Hongzhi Huang, Defa Zhu, Yutao Zeng, Ran Guo, and Xun Zhou. Ultra-sparse memory network, 2025. URL https://arxiv.org/abs/2411.12364.

HuggingFaceFW. fineweb-edu (revision 22b0aca), 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the 51st International Symposium on Computer Architecture (ISCA)*, 2024. URL https://arxiv.org/abs/2308.12066.

Samy Jelassi, Clara Mohri, David Brandfonbrener, Alex Gu, Nikhil Vyas, Nikhil Anand, David Alvarez-Melis, Yuanzhi Li, Sham M. Kakade, and Eran Malach. Mixture of parrots: Experts improve memorization more than reasoning, 2025. URL https://arxiv.org/abs/2410.19034.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, et al. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Tianze Jiang, Blake Bordelon, Cengiz Pehlevan, and Boris Hanin. Hyperparameter transfer with mixture-of-expert layers, 2026. URL https://arxiv.org/abs/2601.20205.

William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pp. 189–206. American Mathematical Society, Providence, RI, 1984. doi: 10.1090/conm/026/737400.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017. URL https://arxiv.org/abs/1705.03551.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts, 2024. URL https://arxiv.org/abs/2402.07871.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276.

Woosuk Kwon, Zhuohan Li, Sicheng Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. URL https://arxiv.org/abs/2309.06180.

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, 2020. URL https://arxiv.org/abs/2006.16668.

Margaret Li, Sneha Kudugunta, Danielle Rothermel, and Luke Zettlemoyer. Slicing and dicing: Configuring optimal mixtures of experts, 2026. URL https://arxiv.org/abs/2605.11689.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1711.05101.

Meta AI. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2025.

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 EMNLP*, 2018.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax, 2018. URL https://arxiv.org/abs/1805.02867.

Mistral AI. Cheaper, better, faster, stronger: Continuing to push the frontier of AI and making it accessible to all. https://mistral.ai/news/mixtral-8x22b, 2024.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023. URL https://arxiv.org/abs/2211.05102.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. https://qwenlm.github.io/blog/qwen3_next/, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models, 2021. URL https://arxiv.org/abs/2106.04426.

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*, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. SocialIQA: Commonsense reasoning about social interactions. In *Proceedings of EMNLP-IJCNLP*, 2019.

John Schulman and Thinking Machines Lab. Lora without regret. https://thinkingmachines.ai/blog/lora/, 2025. Thinking Machines Lab Connectionism blog, September 29, 2025.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of NAACL-HLT*, 2018.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. Scattered mixture-of-experts implementation, 2024. URL https://arxiv.org/abs/2403.08245.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, pp. 10–19. ACM, 2019. doi: 10.1145/3315508.3329973.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106. Association for Computational Linguistics, 2017.

xAI. Open release of Grok-1. https://x.ai/blog/grok-os, 2024.

Leyang Xue, Yao Fu, Zhan Lu, Chuanhao Sun, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache, 2024. URL https://arxiv.org/abs/2401.14361.

Zijie Yan, Hongxiao Bai, Xin Yao, Dennis Liu, Tong Liu, Hongbin Liu, Pingtian Li, Evan Wu, Shiqing Fan, Li Tao, Robin Zhang, Yuzhong Wang, Shifang Xu, Jack Chang, Xuwen Chen, Kunlun Li, Yan Bai, Gao Deng, Nan Zheng, Vijay Anand Korthikanti, Abhinav Khattar, Ethan He, Soham Govande, Sangkug Lym, Zhongbo Zhu, Qi Zhang, Haochen Yuan, Xiaowei Ren, Deyu Fu, Tailai Ma, Shunkang Zhang, Jiang Shao, Ray Wang, Vasudevan Rengasamy, Rachit Garg, Santosh Bhavani, Xipeng Li, Chandler Zhou, David Wu, Yingcan Wei, Ashwath Aithal, Michael Andersch, Mohammad Shoeybi, Jiajie Yao, and June Yang. Scalable training of mixture-of-experts models with megatron core, 2026. URL https://arxiv.org/abs/2603.07685.

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. *arXiv preprint arXiv:2505.09388*, 2025.

Minghao Yang, Ren Togo, Guang Li, Takahiro Ogawa, and Miki Haseyama. L2r: Low-rank and lipschitz-controlled routing for mixture-of-experts, 2026. URL https://arxiv.org/abs/2601.21349.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of EMNLP*, 2018.

Ted Zadouri, Markus Hoehnerbach, Jay Shah, Timmy Liu, Vijay Thakkar, and Tri Dao. Flashattention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling, 2026. URL https://arxiv.org/abs/2603.05451.

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 (ACL)*, pp. 4791–4800. Association for Computational Linguistics, 2019.

A Frontier MoE Model Configurations

For reference, Table 2 summarizes the configurations of recent frontier MoE models. The trend toward higher sparsity ($M \gg k$) and higher granularity ($s < h$) motivates the fine-grained regime studied throughout this paper.

Model	Total	Active	L	M	k	h	s
Mixtral 8x22B Mistral AI (2024)	141B	39B	56	8	2	6144	16384
Grok-1 xAI (2024)	314B	79B	64	8	2	6144	32768
DeepSeek-V2 DeepSeek-AI (2024a)	236B	21B	60	160	6	**5120**	**1536**
DeepSeek-V3 DeepSeek-AI (2024b)	671B	37B	61	256	8	**7168**	**2048**
Llama 4 Maverick Meta AI (2025)	400B	17B	48	128	1	5120	8192
Qwen3 235B-A22B Yang et al. (2025)	235B	22B	94	128	8	**4096**	**1536**
Kimi K2 Kimi Team (2025)	1.04T	32B	61	384	8	**7168**	**2048**
gpt-oss-120b OpenAI (2025)	117B	5.1B	36	128	4	2880	2880
Qwen3-Next-80B-A3B Qwen Team (2025)	80B	3B	48	512	10	**2048**	**512**

Table 2: Configurations of frontier MoE models. M: total experts; k: active experts per token; h: hidden dimension; s: expert FFN intermediate dimension. The trend shows greater sparsity and granularity is preferred.

B Additional Discussions in Related Work

Knowledge storage in language models. Two threads study how language models store factual knowledge. The first measures memorization in standard transformer and MoE architectures: Geva et al. (2021) interpret FFN layers as key-value memories, Allen-Zhu & Li (2024) establish knowledge capacity scaling laws for dense LMs, and Jelassi et al. (2025) show that for MoE, memorization capacity scales with *total* parameters rather than active parameters, directly motivating designs with

large M. A second thread replaces the FFN with a learnable key-value lookup: Memory Layers at Scale (Berges et al., 2024) scales product-key retrieval to billions of stored embeddings, Ultra-Sparse Memory (Huang et al., 2025) pushes to ultra-sparse activation patterns, and Cheng et al. (2026) extend retrieval-style storage to conditional access. MoRE is complementary to this thread: it preserves the MLP-expert structure of MoE and inherits the memorization analysis of Jelassi et al. (2025) while reducing only the routing cost.

Low-rank routing in the $M \ll h$ regime. X-MoE (Chi et al., 2022) scores experts by the dot product between a hidden vector $x \in \mathbb{R}^h$ and M learned expert embeddings. The gradient pulls x toward the span of those embeddings, so after training the hidden states entering a routed layer concentrate in a subspace of $\mathbb{R}^h$ of dimension at most M. When $M \ll h$, that subspace is much smaller than $\mathbb{R}^h$, a phenomenon the authors call *representation collapse*. To mitigate this, Chi et al. (2022) project both expert embeddings and the hidden vector to a lower dimension. L2R (Yang et al., 2026) identifies a related but distinct issue: when $M \ll h$, token representations and expert embeddings become nearly orthogonal, which narrows the margin between routing logits and produces ambiguous expert selection. L2R introduces low-rank routing to improve discrimination among experts.

Our work also uses low-rank routing and derives upper and lower bounds on the rank based on the margin argument, but it serves a different purpose. The mechanisms above require $M \ll h$ (e.g., $h = 4096$ with $M \leq 64$, as in Yang et al. (2026) and Mixtral (Jiang et al., 2024)). MoRE instead targets the fine-grained regime where M is scaled up and $M \ll h$ does not hold. In this regime, the low-rank router reduces routing cost rather than correcting the geometry of token representations.

PEER (He, 2024) reaches a very large expert pool by shrinking each expert to a single neuron and replacing the linear router with product-key retrieval, which has complexity $O(h\sqrt{M})$ and is asymptotically better than our $O((M + h)r)$ where $r = O_k(\log M)$. However, in the following comparison, we find that MoRE improves upon PEER based on the wall-clock speed in practice, with matched active FLOPs.

B.1 Wall-clock Comparison with PEER

A PEER layer holds M experts, each a single neuron $e_i(x) = \sigma(u_i^\top x)\, v_i$ with $u_i, v_i \in \mathbb{R}^h$, so an expert has width 1 and $2h$ parameters. Routing uses H query networks, each a matrix in $\mathbb{R}^{d_q \times h}$, together with two sub-key tables of shape $\sqrt{M} \times d_q/2$. Each head splits its query in half, matches the halves against the two tables, keeps the top κ on each axis, and takes the top κ of the κ^2 candidate sums, so the layer activates $k = H\kappa$ experts per token. The per-layer parameter counts and per-token active FLOPs (we count the query projections and the sub-key tables as active FLOPs too) are

$$P_{\text{experts}} = 2Mh, \qquad P_{\text{router}} = Hhd_q + \sqrt{M}\, d_q, \qquad \tau_{\text{PEER}} = 12\, kh + 6\, Hd_q\big(h + \sqrt{M}\big), \quad (2)$$

to be read against $3Mhs$, $(M + h)r$ and $C_e khs + C_r(M + h)r$ for MoRE.

For comparison, we implement PEER with reference to the public PEER-pytorch repository,[8] and change the code to make the heads share one set of product keys, following He (2024). We compare MoRE against it at matched total parameters and matched active FLOPs.

We match against the (s=64, h=512) row of Table 1, where MoRE-1792 uses $M' = 1792$, $s' = 56$, $k = 4$ and $r = 64$ over $L = 12$ layers. Matching expert parameters gives $M = \frac{3}{2}M's' = 150{,}528$, and since product-key retrieval needs M to be a perfect square we take $M = 388^2 = 150{,}544$. Matching active FLOPs at $H = 8$ then gives $d_q = 32$ and $\kappa = 56$, so $k = 448$ experts are active per token. We choose $d_q = 32$ rather than the $d_q = h$ of the default implementation in order to match the router FLOPs.

Table 3 measures one layer at the matched configuration. **At matched active FLOPs, a MoRE layer takes substantially less wall-clock time than a PEER layer.** Because M is large for PEER, we additionally implement a fused kernel for the expert application that never materializes the gathered weights. This fused kernel runs $1.90\times$ faster on that stage, but the wall-clock ratio remains large.

[8] https://github.com/lucidrains/PEER-pytorch

	Forward		Forward + backward	
Tokens per step	MoRE	PEER	MoRE	PEER
4,096	1.06	5.56 (5.2×)	2.94	16.30 (5.5×)
16,384	1.23	22.06 (18.0×)	3.01	60.96 (20.3×)
65,536	1.93	87.79 (45.6×)	4.75	244.01 (51.4×)
262,144	4.92	350.86 (71.4×)	14.19	974.41 (68.7×)

Table 3: **Single-layer wall-clock time at matched active FLOPs. The ratio grows with batch size.** Single-layer wall-clock in milliseconds at the matched configuration, on one NVIDIA B200 in bf16, median of 20 iterations after 5 warm-ups.

Table 4 reports validation perplexity for the three architectures trained for 1B tokens on mixture dataset specified in Appendix C. MoE and MoRE use the Switch load-balancing loss (Fedus et al., 2022).

Architecture	Validation PPL	Seconds per step	Training time
MoE-256	24.26	0.321	20.4 min
MoRE-1792	24.13	0.405	25.8 min
PEER	24.17	6.135	390.1 min

Table 4: Validation perplexity at 1B tokens, $h = 512$, $L = 12$, four B200s, 262,144 tokens per optimizer step.

PEER and MoRE reach similar perplexity under the 1B-token budget. The wall-clock cost differs by $15\times$: **MoRE trains in** 25.8 **minutes compared to** 390.1 **minutes for PEER** (Table 4).

C Experimental Details

Router initialization. We initialize each entry of the low-rank router factors as $\mathcal{N}(0, \sigma'^2)$ with $\sigma' = \sqrt{\sigma}/r^{1/4}$ and $\sigma = 0.02$, so that the variance of the router logits matches the standard router. See Appendix D for the derivation and an empirical comparison to the default initialization.

To make the speedup by the low-rank router more apparent, we seek an efficient implementation of the Mixture-of-Experts architecture and therefore build upon ScatterMoE (Tan et al., 2024) (refer to Appendix E for more details). We believe that further speedups are achievable on specific hardware by applying more careful optimization as in Guo et al. (2025).

We use load balancing coefficient as 0.001 for both MoE and MoRE setups and do not have evidence that MoRE is sensitive to its choice.

C.1 Phonebook

Tokenization. Each example is tokenized as "`$abcde|12345678.`", with a five-letter lowercase name, the separator "`|`", and an eight-digit number, padded with "$\star$" to a fixed sequence length of 16. The vocabulary has $V = 40$ symbols: 26 lowercase letters, 10 digits, and 4 specials ("`$`" BOS, "`.`" EOS, "`|`" separator, "$\star$" pad).

Architecture and sweep. All models are $L = 6$-layer Transformers with top-k routing at $k = 4$ and low-rank rank $r = 16$. The active budget τ has two natural scaling axes at fixed h: the hidden dimension itself and the expert width s. We sweep each in turn: (i) $h \in \{64, 128, 256, 512\}$ at fixed $s = 8$, and (ii) $s \in \{8, 16, 32, 64\}$ at fixed $h = 64$. For every (h, s), the standard baseline takes $M = 4s$ experts, and the low-rank counterpart takes the largest M' that matches the standard active FLOPs at $r = 16$, following Proposition 2.

Choice of s'. Under our convention $C_e = C_r$, Proposition 2 prescribes the optimal MoRE expert width as $s'^* = s - r/(2k) = s - 2$ at $r = 16, k = 4$. For $s \in \{8, 16, 32, 64\}$ this gives $s'^* \in$

$\{6, 14, 30, 62\}$, none of which are multiples of 8 – the kernel-friendly width for ScatterMoE. We round each s' to the nearest multiple of 8, which collapses to s itself across the entire sweep, and we use the prescribed $M'^* = h(2ks - r)/(2r) = h(s-2)/4$ unchanged. The rounding inflates the total MLP parameter count by a factor $s/(s-2)$, which is largest at the smallest scale ($8/6 \approx 1.33\times$ at $s = 8$) and shrinks to $\approx 1.03\times$ at $s = 64$, it does not change the qualitative trends of Figure 5.

Training and learning-rate selection. Each configuration is trained with batch size 1,024 under a constant-with-warmup schedule (warmup ratio 0.05, weight decay 0.01, $\beta_2 = 0.95$), with early stopping once exact-string accuracy on the digit numbers crosses 0.9. The learning rate is selected per configuration by a short calibration phase over $\{1\times10^{-4}, 3\times10^{-4}, 1\times10^{-3}\}$, picking the value with the highest validation character accuracy on a brief preliminary run; the calibrated rate is then used for the full 1,500-epoch run.

Per-dot configurations. Tables 5 and 6 list, for every dot in Figure 5, the full (M, k, h, s) and (M, k, h, s, r) configuration and the maximum memorizable phonebook size at the calibrated learning rate. Table 5 covers the vary-h sweep at $s = 8$, and Table 6 covers the vary-s sweep at $h = 64$; the (h=64, s=8) corner appears in both as the shared anchor between the two sweeps.

The number of experts M are small for Table 5 as the optimal $M = ks$ in our setup, when $s = 8$, $M = ks = 32$.

Configuration	M/M'	k	h	s	r	Max memorizable
MoE	32	4	64	8	—	10K
MoE	32	4	128	8	—	30K
MoE	32	4	256	8	—	100K
MoE	32	4	512	8	—	200K
MoRE	96	4	64	8	16	30K
MoRE	192	4	128	8	16	100K
MoRE	384	4	256	8	16	300K
MoRE	768	4	512	8	16	1.2M

Table 5: Phonebook per-dot configurations for the **vary-h sweep** at $s = 8$. M (M' for MoRE) is the number of experts; k active per token; h hidden dim; s expert intermediate dim; r router rank (MoRE only). "Max memorizable" is the largest phonebook size at which the calibrated training run reached the early-stopping threshold.

Configuration	M/M'	k	h	s	r	Max memorizable
MoE	32	4	64	8	—	10K
MoE	64	4	64	16	—	30K
MoE	128	4	64	32	—	150K
MoE	256	4	64	64	—	500K
MoRE	96	4	64	8	16	30K
MoRE	224	4	64	16	16	100K
MoRE	480	4	64	32	16	400K
MoRE	992	4	64	64	16	1.2M

Table 6: Phonebook per-dot configurations for the **vary-s sweep** at $h = 64$. Columns as in Table 5. The (h=64, s=8) row is shared with Table 5.

C.2 Real-world Pretraining

Data mixture. We roughly follow the natural-language pre-training recipe of Jelassi et al. (2025, Appendix B.2), which consists of a ≈ 60.4 B token mixture: (i) FineWeb-edu deduplicated (HuggingFaceFW, 2024) (37 B); (ii) Cosmopedia synthetic textbooks (Ben Allal et al., 2024) (14 B); (iii) Wikipedia (4 B unique, looped 3× for 12 B); and (iv) the training splits of 12 downstream knowledge-intensive and commonsense benchmarks (TriviaQA (Joshi et al., 2017), Natural Questions (Kwiatkowski et al., 2019), HotpotQA (Yang et al., 2018), WebQuestions (Berant et al., 2013),

Scale	Model	M	k	h	s	r
s=32, h=1024	Standard	128	4	1024	32	–
	Low-rank (r=64)	1536	4	1024	24	64
s=64, h=512	Standard	256	4	512	64	–
	Low-rank (r=64)	1792	4	512	56	64
s=128, h=512	Standard	256	4	512	128	–
	Low-rank (r=64)	1792	4	512	112	64

Table 7: Architectural configurations for each variant in Table 1. M: number of experts. k: experts active per token. h: model hidden dimension. s: expert FFN hidden dimension. r: router rank (Low-rank only). All variants use the GPT-2 tokenizer. Within each Scale group, the Low-rank variant scales M up while reducing s slightly so that active FLOPs approximately match the Standard reference; the (s=128, h=512) group is the exception, obtained by scaling s and s' of (s=64, h=512) by 2 with M and M' fixed.

ComplexWebQuestions (Talmor & Berant, 2018), ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), SocialIQA (Sap et al., 2019), and Winogrande (Sakaguchi et al., 2020)), each looped 3× to familiarize the model with the evaluation format (Allen-Zhu & Li, 2023).

Every source uses only its `train` split. We include the benchmark training splits in line with Jelassi et al. (2025), since Allen-Zhu & Li (2023) show that mixed training is required for successful knowledge extraction. We sample disjoint training (40 B tokens) and 5,000-sequence validation sets uniformly from this pool, following the convention of Jelassi et al. (2025). We conduct zero-shot evaluations only, since we did not include few-shot demonstrations in training.

Tokenization. We use the GPT-2 byte-pair-encoding tokenizer (Radford et al., 2019) with vocabulary size $V = 50{,}257$. Documents from each mixture source are concatenated and chunked to a fixed sequence length of $T = 1{,}024$ tokens, then materialized as a single `uint16` memory-mapped NumPy file for efficient reuse across runs.

Architecture. All models are $L = 12$-layer Transformers with top-k routing at $k = 4$, head dimension 64, and untied input/output embeddings. We compare three reference points $(h, s) \in \{(1024, 32),\ (512, 64),\ (512, 128)\}$. For the first two, the standard baseline takes $M = 4s$ experts, and the low-rank counterpart fixes router rank $r = 64$ and takes the largest M' that matches the standard active FLOPs, following Proposition 2. The third point $(512, 128)$ is constructed by scaling both s and s' of $(512, 64)$ by 2 while keeping $M = 256$ and $M' = 1792$ fixed. Full configurations are listed in Table 7.

Training and learning-rate selection. Each configuration is trained for $\approx$ 40 B tokens, with sequence length 1,024 on 4 GPUs in DDP, using AdamW (Loshchilov & Hutter, 2019) ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$), weight decay 0.01, max grad norm 1.0, and a cosine learning-rate schedule with warmup ratio 0.05, in bf16 precision. The peak (global) learning rate is selected per configuration from $\{5{\times}10^{-4},\ 1{\times}10^{-3},\ 3{\times}10^{-3},\ 5{\times}10^{-3}\}$ by best validation perplexity on a 2B-token run, which selects $1{\times}10^{-3}$ for every configuration. We set the router learning rate, for both R_1 and R_2 of the low-rank router, equal to the global learning rate, as we find that validation loss is not very sensitive to the router learning rate (Figure 7).

C.3 FLOPs Calculation and Total parameter accounting

For consistency with Proposition 2, we report *active FLOPs per token* (rather than active parameters) in Table 1, since the matched-active configuration grid is selected to fix this quantity. We use the standard convention that one fused multiply–add counts as 2 FLOPs. All h=512 models use n_h=8 attention heads; h=1024 uses n_h=16, n_{kv}=4. Head dimension is d=h/n_h=64 throughout. Vocabulary size is V=50,257 (GPT-2 BPE), and input/output embeddings are untied.

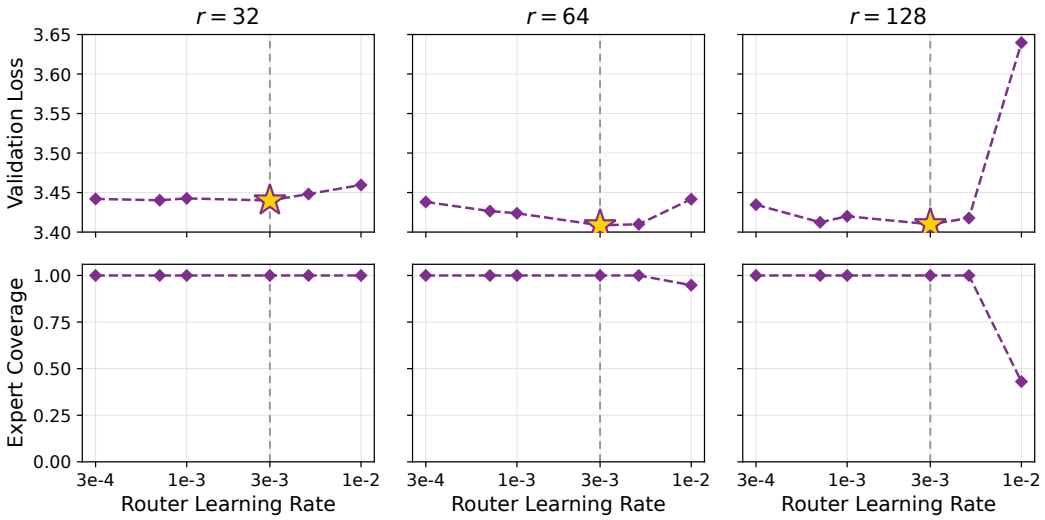


Figure 7: **Expert coverage is insensitive to the router learning rate, and the optimal router learning rate is the global learning rate at every rank. We therefore do not tune the router learning rate separately, and set it equal to the global learning rate.** Switch balancing loss, $M = 1792$, $s = 56$, $h = 512$, $k = 4$, $L = 12$, trained on 500M tokens of the mixture corpus at global learning rate $\eta = 3 \times 10^{-3}$. The dashed vertical line in every panel marks that global rate, and stars mark each rank's best cell. *Top:* the optimum coincides with the global learning rate at all three ranks. The 5×10^{-4} cell is omitted at every rank because its apparent advantage at this budget does not survive to 2B tokens. *Bottom:* coverage holds at 1.000 at every rank up to 5×10^{-3}, and at $r = 32$ across the entire grid. It falls only at 10^{-2}, and only for $r = 64$ and $r = 128$.

Active MoE FLOPs in Table 1. Per token and per MoE layer, the active FLOPs in training (forward and backward) are $18khs + 14Mh$ for MoE and $18khs' + 14(M' + h)r$ for MoRE. At inference (forward only), they are $6khs + 6Mh$ for MoE and $6khs' + 6(M' + h)r$ for MoRE. The constants are derived below. In the computation of the configurations, we treat $C_e \approx C_r$. Table 1 reports the inference FLOPs, and Table 8 lists both the training and the inference FLOPs.

(s, h)	Model	Training FLOPs	Inference FLOPs
(32, 1024)	MoE-128	4.19M	1.57M
	MoRE-1536	4.06M	1.57M
(64, 512)	MoE-256	4.19M	1.57M
	MoRE-1792	4.13M	1.57M
(128, 512)	MoE-256	6.55M	2.36M
	MoRE-1792	6.19M	2.26M
(64, 512)	MoE-1024	9.70M	3.93M
	MoRE-8192	9.72M	3.98M

Table 8: Active MoE FLOPs per token per layer for the configurations in Table 1. Training FLOPs are $18khs + 14Mh$ for MoE and $18khs' + 14(M' + h)r$ for MoRE. Inference FLOPs are $6khs + 6Mh$ and $6khs' + 6(M' + h)r$.

FLOPs constants in Proposition 2. The forward-pass formula above tabulates the empirical FLOPs we report. For the analytical scaling argument, Proposition 2 writes the active-FLOP budget as $\tau = C_e \cdot khs + C_r \cdot Mh$, with $C_r r(h + M)$ in place of $C_r Mh$ for the rank-r low-rank router, abstracting per-operation FLOPs into two scalar constants. We follow the full-training (forward + backward) accounting of Krajewski et al. (2024, Appendix E), which charges 6 FLOPs per pair of (active parameter, processed token) for each linear projection (forward 2 + backward input-grad 2 +

backward weight-grad 2). A SwiGLU expert performs three projections, giving $C_e = 3\times 6 = 18$. The standard router charges 6 for its linear projection plus 2 FLOPs per pair for each of four token-routing operations (forward and backward dispatches of inputs and outputs), giving $C_r = 6 + 4 \times 2 = 14$ as in Krajewski et al. (2024). The forward-only ratio is similar: $C_e^{\text{fwd}}/C_r^{\text{fwd}} = 6/6 = 1$, compared to $C_e/C_r = 18/14 \approx 1.29$ for full training. We set $C_e = C_r$, which simplifies the configurations (for example, the optimal MoE then has $M = ks$), and use the same C_r for the low-rank router. Only the ratio C_e/C_r enters the matching of an MoE and a MoRE configuration. The token-routing operations cost $O(kh)$ per token in ScatterMoE, so a stricter accounting charges each router only for its matrix multiplications ($C_r = 2$ at inference). Under this accounting, MoRE uses 4.5% to 12.5% fewer active FLOPs than its MoE counterpart in every pair of Table 1. The $\Theta(h/r)$ ratio in Proposition 2 depends only on the leading $C_r Mh$ term, so it is insensitive to the precise value of C_r within the defensible range $[6, 20]$ noted by Krajewski et al. (2024).

Total parameter count. Total parameters include the embedding matrix, all M experts (active and inactive), the router, attention, layer norms, and the unembedding head:

$$P_{\text{total}} = \underbrace{2\,V\,h}_{\text{embed+LM head}} + \underbrace{h}_{\text{final LN}} + L \cdot \Big[\underbrace{\boldsymbol{P}_{\text{router}}}_{\text{router}} + \underbrace{\boldsymbol{3\,M\,h\,s}}_{\text{all experts}} + \underbrace{2h^2 + 2\,h\,n_{kv}\,d}_{\text{attention}} + \underbrace{2\,h}_{\text{2 RMSNorm}} \Big], \tag{3}$$

where $P_{\text{router}} = h\,M$ for the standard router and $r\,(h + M)$ for the low-rank variant. (RMSNorm has no bias, so each contributes h scale parameters.) The two bold terms are the only ones that depend on the expert configuration: once h is fixed, the embedding, attention, and norm terms are also fixed, and the parameter count varies only through the router and expert terms, which are set by M and s. The depth-independent embedding and LM head term is amortized over layers, so as the depth L grows the router and the experts account for a larger fraction of the total parameters.

D Initialization

The Switch Transformer (Fedus et al., 2022) sets the router weights elementwise to $\mathcal{N}(0, \sigma^2)$ with $\sigma = 0.02$, and prior work has reported that the precise router initialization variance has limited impact on standard-router performance. The natural default for the low-rank router carries the same per-entry standard deviation over to both factors, i.e. each entry of $R_1 \in \mathbb{R}^{M\times r}$ and $R_2 \in \mathbb{R}^{r\times h}$ is drawn i.i.d. from $\mathcal{N}(0, \sigma^2)$ with the same $\sigma = 0.02$. We refer to this as the *default* (or *standard*) initialization.

Empirically, however, we find that the default initialization noticeably degrades validation perplexity and shifts the optimal router learning rate significantly. We instead choose σ_1, σ_2 so that the variance of the router logits matches the standard baseline. Lemma 1 makes this matching condition precise.

D.1 Setup

We treat $x \in \mathbb{R}^h$ as a fixed nonzero input and condition on it throughout. The standard router $R \in \mathbb{R}^{M\times h}$ has entries i.i.d. $\mathcal{N}(0, \sigma^2)$, and the low-rank factors $R_1 \in \mathbb{R}^{M\times r}$, $R_2 \in \mathbb{R}^{r\times h}$ have independent entries i.i.d. $\mathcal{N}(0, \sigma_1^2)$ and $\mathcal{N}(0, \sigma_2^2)$ respectively, with R_1 independent of R_2. We denote the corresponding learning rates by η for the standard router, and η_1, η_2 for R_1, R_2. Our goal is to derive σ_1, σ_2 (and, in future work, η_1, η_2) by matching the variance and the logit update of the standard router.

D.2 Matching variance

Lemma 1 (Variance matching)**.** *Under the setup above, for any fixed $x \neq 0$,* $\mathrm{Var}([R_1 R_2 x]_k) = \mathrm{Var}([Rx]_k)$ *for every $k \in [M]$ if and only if*

$$\sigma_1 \sigma_2 = \frac{\sigma}{\sqrt{r}}.$$

The matching condition does not depend on x.

Proof. All variances below are conditional on the fixed input x.

For the standard router, $[Rx]_k = \sum_{j=1}^{h} R_{kj}\, x_j$ is a linear combination of i.i.d. $\mathcal{N}(0, \sigma^2)$ entries with deterministic coefficients x_j, so

$$\mathrm{Var}\big([Rx]_k\big) \;=\; \sigma^2 \|x\|_2^2.$$

For the low-rank router, write $z = R_2 x \in \mathbb{R}^r$, so $[R_1 R_2 x]_k = \sum_{i=1}^{r} (R_1)_{ki}\, z_i$. The same linear-combination argument applied to R_2 gives $\mathrm{Var}(z_i) = \sigma_2^2 \|x\|_2^2$. Since R_1 is independent of R_2 and zero-mean, each summand satisfies

$$\mathrm{Var}\big((R_1)_{ki}\, z_i\big) \;=\; \mathbb{E}\big[(R_1)_{ki}^2\big]\, \mathbb{E}\big[z_i^2\big] \;=\; \sigma_1^2\, \sigma_2^2\, \|x\|_2^2,$$

and the r summands are mutually independent because they share no entry of R_1. Summing,

$$\mathrm{Var}\big([R_1 R_2 x]_k\big) \;=\; r\, \sigma_1^2\, \sigma_2^2\, \|x\|_2^2.$$

Equating the two variances gives $r\, \sigma_1^2\, \sigma_2^2\, \|x\|_2^2 = \sigma^2 \|x\|_2^2$, and since $\|x\|_2^2 > 0$ this is equivalent to $\sigma_1 \sigma_2 = \sigma / \sqrt{r}$, independent of x. □

In all the experiments in this paper, we set $\sigma_1 = \sigma_2$ and $\eta_1 = \eta_2$. Combined with Lemma 1, this gives

$$\sigma_1 \;=\; \sigma_2 \;=\; \frac{\sqrt{\sigma}}{r^{1/4}}, \qquad \sigma = 0.02.$$

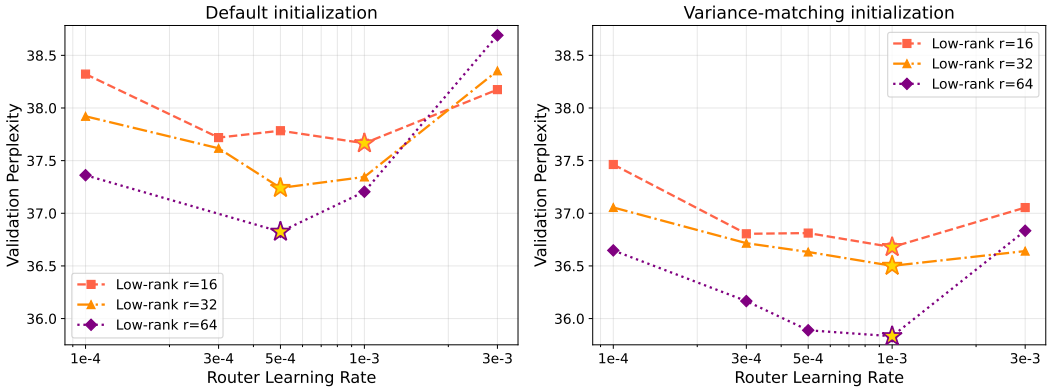


Figure 8: Router learning-rate sweep on FineWeb-edu at the optimal global learning rate $\eta = 10^{-3}$, comparing the default per-entry initialization (*left*) against the variance-matching initialization of Lemma 1 (*right*) for low-rank ranks $r \in \{16, 32, 64\}$. Each curve trains for 400M FineWeb-edu tokens. The variance-matching initialization achieves lower validation perplexity across ranks, and also has stabler optimum across this range, although it might also shift leftwards when r is larger.

D.3 Discussion

Figure 8 presents our empirical comparison. We hypothesize that a theoretical analysis analogous to those developed for LoRA (Hu et al., 2021) and LoRA+ (Hayou et al., 2024; Schulman & Thinking Machines Lab, 2025) could provide a principled approach to selecting both the initialization and learning rates for routers in the low-rank setting, potentially eliminating the need to tune η_1 and η_2 individually for each rank. Although a key distinction from LoRA is that LoRA initializes its output to zero to preserve finetuning performance, whereas initializing the router's output to zero is undesirable in our context: doing so would impede the ability to break ties when selecting the top k experts.

In a complementary direction, Jiang et al. (2026) derive a hyperparameter-transfer parameterization for MoE routing weights via a dynamical mean-field analysis; we believe their framework can be adapted to the low-rank router setting as well.

E EFFICIENCY

For the MoE architecture, we build upon ScatterMoE (Tan et al., 2024) and implement a new fused Triton kernel for the router (refer to Figure 9 for the detailed operations in the forward pass). We believe that further speedups are achievable on specific hardware by applying more careful optimization as in Guo et al. (2025).

Why we build on ScatterMoE. Figure 9 illustrates the operations in one MoE layer forward pass. Beyond the router, the expert MLP involves token dispatch (sorting by expert assignment), two grouped linear transformations with a SwiGLU activation, and a scatter-back step that combines expert outputs. Each of these steps is a potential source of memory traffic. For example, a naive grouped MoE implementation pays $O(Nkh)$ memory traffic to materialize the gathered tokens (sorted by expert assignment) before the per-expert grouped GEMM, and pays the same again on the scatter-back.

ScatterMoE (Tan et al., 2024) fuses scatter-gather into the per-expert grouped matmul, removing both round-trips and producing a kernel whose only HBM traffic is the input X, the weight stack, and the output Y. With this dispatch overhead eliminated, the standard router's $O(NhM)$ matmul is the only remaining cost that grows linearly with M. Our low-rank router is designed precisely to address this last bottleneck, while reusing the rest of ScatterMoE's machinery unchanged.

E.1 GPU MEMORY HIERARCHY AND RELATED ANALYSIS

Bandwidth tiers. Modern GPU architectures expose a multi-level memory hierarchy with sharply varying bandwidth. Off-chip high-bandwidth memory (HBM) provides the largest capacity but the lowest bandwidth. On-chip, the L2 cache offers an intermediate level, while each streaming multiprocessor (SM) contains a small pool of SRAM (registers and shared memory) with the highest bandwidth. Any data that does not fit in on-chip memory must be materialized to HBM, incurring an order-of-magnitude bandwidth penalty.

Our fusing avoids writing the full (N, M) logits and the intermediate tensors of low-rank router to HBM, and computes the Top-K operation in SRAM, which is much faster than the HBM round-trip.

Comparison to other fused-MoE kernels. Existing efficient MoE kernels, ScatterMoE (Tan et al., 2024), MegaBlocks (Gale et al., 2023), MoMoE (Costin et al., 2025) and SonicMoE (Guo et al., 2025), do top-k computation on the materialized (N, M) logits on the HBM and have to pay expensive read and write costs. Our kernel fuses the router with top-k and do sequential top-k operation on each tile, this avoid expensive memory operations between HBM and SRAM, but the cost would also scale with k, making large k infeasible in our setup.

E.2 TOP-K ALGORITHM ANALYSIS

Our implementation is faster than `torch.topk` when $K < 16$ and is common across frontier model choices (Table 2). But it is slower than `torch.topk` for $K \geq 16$. For our fused kernel, the per-token cost is $\Theta(K \cdot M) + \Theta(K^2 \cdot M/\texttt{BLOCK_M})$.

For each of the $M/\texttt{BLOCK_M}$ tiles, we run K iterations to find the K largest values and insert them to the K-element running buffer. The $\Theta(K \cdot M)$ term aggregates the K iterations of max/argmax reductions over the `BLOCK_M` new candidates, across $M/\texttt{BLOCK_M}$ tiles. The $\Theta(K^2 \cdot M/\texttt{BLOCK_M})$ term is for computing the minimum over the K-element buffer that happen inside the same K iterations.

We suspect that a more carefully engineered fused topk along the lines of SonicMoE (Guo et al., 2025) (Section D and Appendix F.4), could outperform `torch.topk` at larger K and close this gap, which we leave to future work.

E.3 EXPERT PARALLELISM

We explained how MoRE can be implemented efficiently on a single GPU, here we describe how we implement it under expert parallelism, a common setting in large-scale MoE training. We consider

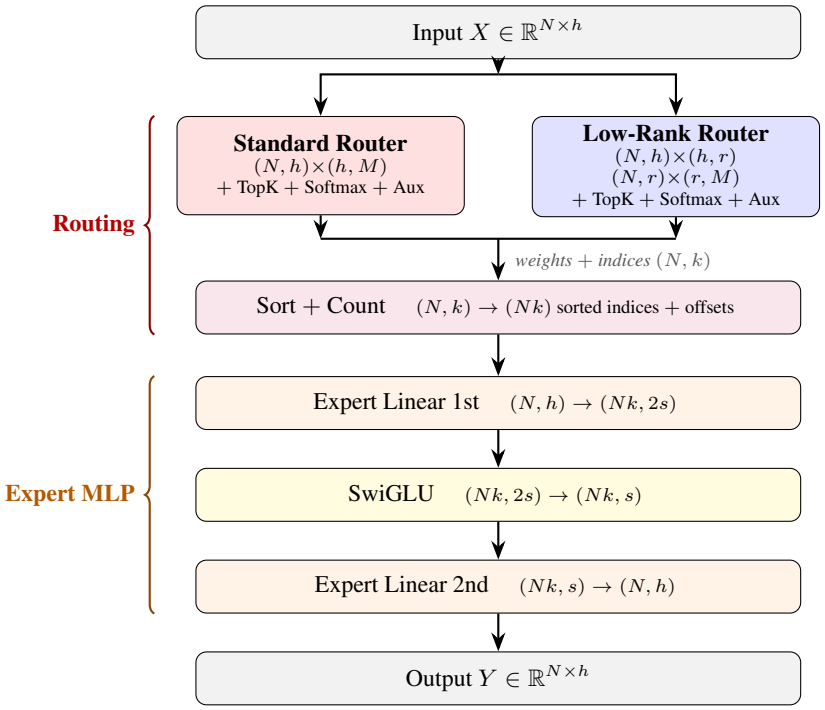


Figure 9: Operations in the forward pass of one MoE layer. The first expert linear projects to $2s$ because SwiGLU uses separate gate and up projections: $\text{SiLU}(Xw_{\text{gate}}) \odot Xw_{\text{up}}$. Expert linear layers use fused scatter-gather kernels (Tan et al., 2024) that eliminate $O(Nkh)$ memory traffic from token dispatch, making the standard router the dominant remaining overhead and motivating the low-rank factorization.

there are E GPUs where each GPU contains M/E experts, for simplicity we assume all GPUs are inside one node, and we defer inter-node communication analysis to future work.

Following popular design choices (Lepikhin et al., 2020; Fedus et al., 2022; Yan et al., 2026), we replicate non-expert components including router in all GPUs, which our fusing technique still carries over as every GPU has a local router. We could also shard the router across GPUs, i.e. each GPU holds M/E experts and a $1/E$ fraction of the router, by computing a top-k locally inside each GPU and then a global top-k across GPUs, which is a hierarchical top-k which we leave to future work. It has the same scaling argument as in the main paper.

E.3.1 Scaling under Expert Parallelism

Here we show that the $\Theta(h/r)$ advantage of Section 4.1 still holds under expert parallelism, and that replicating the router makes the routing bottleneck more severe.

Active cost with a replicated router. Because the router is replicated, every device scores all tokens against all experts and computes the global top-k locally. Each token-expert pair is therefore still computed exactly once across the system, while the router runs E times. Per-token active FLOPs across GPUs become

$$\tau = \begin{cases} C_e\, khs + E\, C_r\, Mh & \text{(standard router)}, \\ C_e\, khs + E\, C_r\, (M + h)\, r & \text{(low-rank router)}. \end{cases} \tag{4}$$

Equivalently, since each device holds M/E experts and therefore computes k/E of the token-expert pairs in expectation, the active FLOPs *per GPU* are $C_e(k/E)hs + C_r Mh$ for the standard router. Either way the factor E falls on the router term and not the expert term, so routing occupies a larger share of the budget than on a single device: replicating the router sharpens exactly the bottleneck that MoRE is designed to remove.

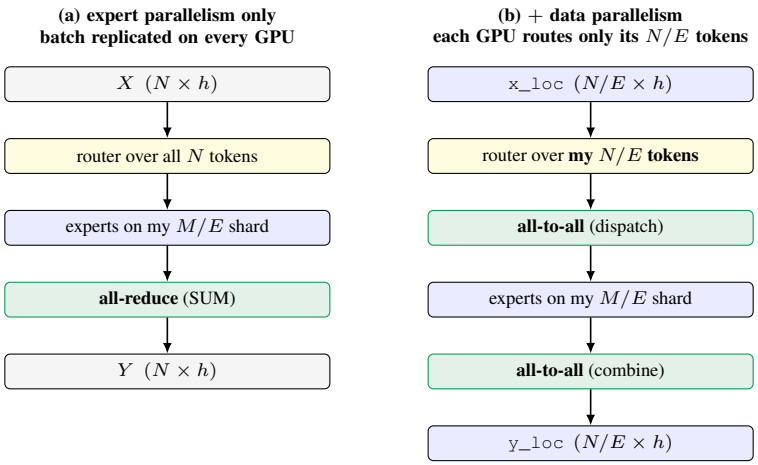


Figure 10: The two implementations measured in Table 9. Both shard the experts M/E per GPU and hold a full copy of the router weights. In (a) the batch is replicated, so every GPU scores all N tokens and one all-reduce sums the partial outputs; the routing work is therefore duplicated E times. In (b) each GPU keeps only its own N/E tokens and scores only those, so the routing work is divided by E, at the cost of shipping each token to the GPU owning its selected expert and back with two all-to-alls.

The ratio is unchanged. Applying the budget allocation of Proposition 2 to (4) gives

$$M^* = \frac{\tau}{2C_r Eh}, \qquad M' = \frac{\tau - E\,C_r hr}{2C_r Er}, \qquad \frac{M'}{M^*} = \frac{h}{r}\left(1 - \frac{E\,C_r hr}{\tau}\right).$$

The factor E enters both configurations identically and cancels, so the $\Theta(h/r)$ advantage carries over. The correction term is close to 1 whenever $\tau > E\,C_r Mh \gg E\,C_r rh$, that is whenever $M \gg r$, which is the same condition as on a single device.

Table 9 measures both implementations of Figure 10 while sweeping M from 2,048 to 16,384. The standard MoE router dominates the layer time under expert parallelism, and communication does not, so replacing it with the low-rank router of MoRE speeds the layer up by up to $3.1\times$.

E.4 Decoding

In terms of decoding, it is memory-bound rather than compute-bound, since each step processes a single token per sequence and the cost is dominated by reading the weights and the KV cache (Shazeer, 2019; Pope et al., 2023).

We initially found the decoding speedup to degrade sharply as the expert count M grows, which we found the reason is the `scatter2scatter` kernel of ScatterMoE (Tan et al., 2024): it loops over the entire contiguous range of expert ids the block spans from the smallest to the largest. A decoding step carries so few tokens whose expert ids are scattered across the pool, and the loop then walks almost the entire expert range, which makes decoding significantly slower at large M.

We therefore modify the kernel to step directly from one expert present in the block to the next and make it significantly faster. Table 10 and Figure 11 show the resulting decode times.

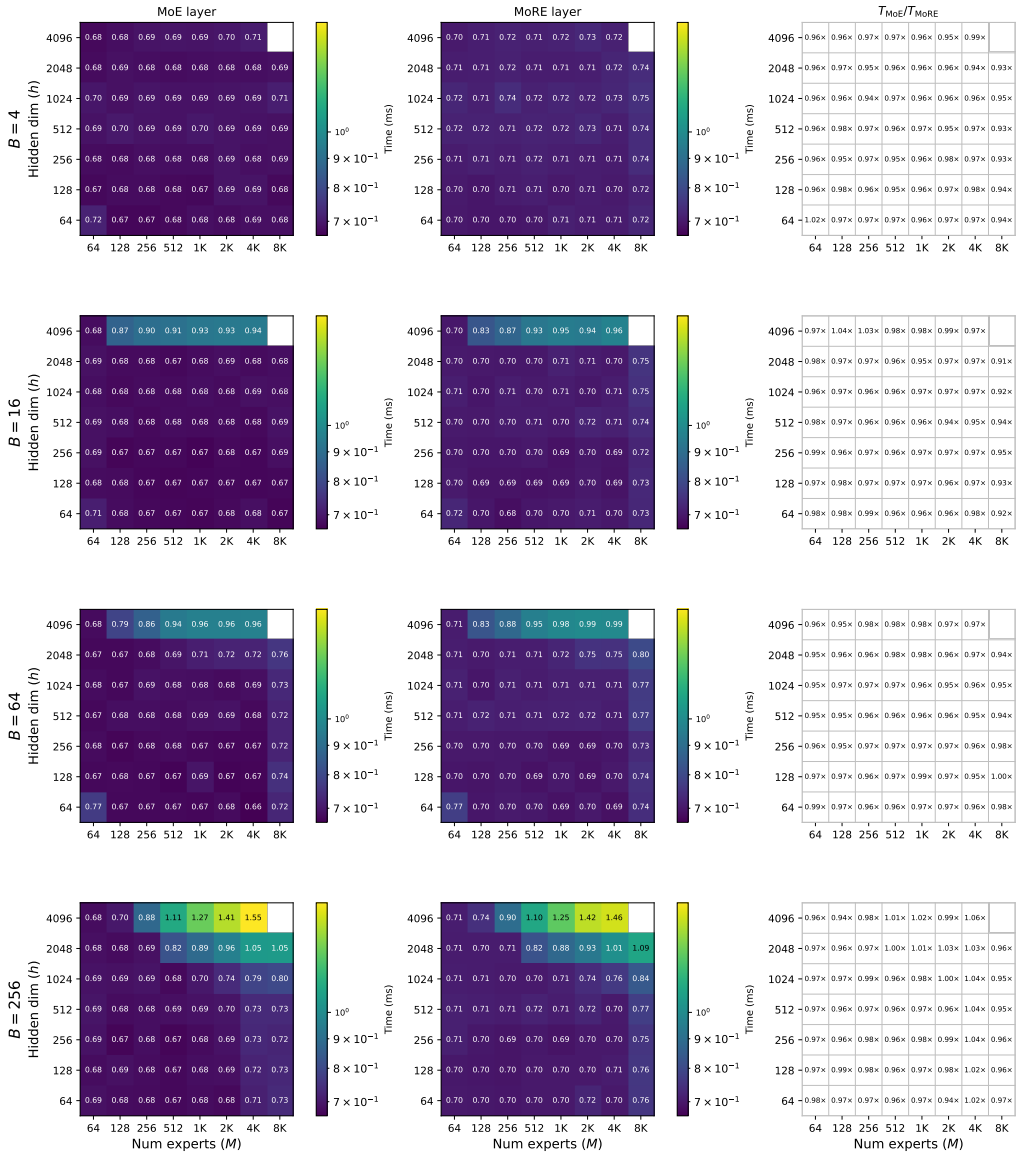


Figure 11: **MoRE has comparable decoding time with MoE for the same configuration.** Decode wall-clock of one MoE layer on the (h, M) grid of Figure 4, one row per decode batch size B. **Left:** MoE layer, standard router. **Middle:** MoRE layer, fused low-rank router. **Right:** the ratio of the two. Both arms use the dispatch of this section. One NVIDIA B200 in `bf16`, s=64, k=4, r=64, one token per sequence, median of ten timed calls after five warm-ups. The h=4096, M=8192 cell is blank because the expert stride overflows the `int32` offsets ScatterMoE uses.

		Router		Shared phases		Layer total		
E	M	**MoE**	**MoRE**	experts	comm.	MoE	MoRE	speedup
(a) expert parallelism only								
2	2,048	**8.20**	**1.68**	3.86	0.61	12.68	6.15	2.1×
2	8,192	**27.90**	**6.09**	4.15	0.61	32.66	10.85	3.0×
2	16,384	**45.30**	**11.95**	4.41	0.62	50.33	16.98	3.0×
4	2,048	**9.01**	**1.68**	3.81	0.96	13.77	6.45	2.1×
4	8,192	**29.01**	**6.09**	4.05	0.86	33.93	11.00	3.1×
4	16,384	**45.34**	**11.95**	4.25	0.79	50.38	17.00	3.0×
(b) + data parallelism, each GPU routes its own N/E tokens								
2	2,048	**4.25**	**0.97**	8.59	4.12	16.96	13.68	1.2×
2	8,192	**13.04**	**3.30**	8.70	4.42	26.17	16.43	1.6×
2	16,384	**22.81**	**6.60**	9.08	4.43	36.32	20.11	1.8×
4	2,048	**2.31**	**0.65**	4.73	2.23	9.27	7.61	1.2×
4	8,192	**6.73**	**1.93**	4.75	2.22	13.70	8.89	1.5×
4	16,384	**10.55**	**3.62**	4.85	2.18	17.59	10.66	1.7×

Table 9: **The standard MoE router is the bottleneck of one layer under expert parallelism and dominates communication, and MoRE achieves a notable speedup by replacing it with a low-rank router.** Forward pass of one MoE layer. The *experts* column includes the local dispatch bookkeeping, which moves nothing between devices: it turns expert ids into a sorted pair list, and in (b) it also gathers the rows that each of the other GPUs will receive. *comm.* is the collectives themselves, a single all-reduce of the (N, h) output in (a) against two all-to-alls in (b), the second of which also carries the (M) balancing statistic. All runs use the Switch load-balancing loss. $h = 512$, $s = 64$, $k = 4$, $r = 64$, `bf16`, $N = 262{,}144$, mean of 30 trials on B200 GPUs in one node.

Scale	Model	#Experts	B=4	B=16	B=64	B=256
s=32, h=1024	MoE	128	17.9	17.9	17.9	18.2
	MoRE (r=64)	1,536	18.5	18.3	18.4	18.6
s=64, h=512	MoE	256	18.0	17.8	17.9	18.1
	MoRE (r=64)	1,792	18.5	18.5	18.4	18.6
s=128, h=512	MoE	256	17.9	17.8	17.9	18.0
	MoRE (r=64)	1,792	18.5	18.3	18.5	18.5

Table 10: **Decoding: MoRE matches MoE in wall-clock time even with a much larger number of experts.** Per-step decode wall-clock (ms) for MoE/MoRE pairs used in our pretraining experiments with matched active FLOPs (Table 1), across decode batch size B. Each MoRE row holds 6–12× the experts of the MoE it is matched against, and stays within 3% of it at every scale and batch size. One NVIDIA B200 in `bf16`, 12 layers, k=4, prefill 128, median of the last six of twelve decode steps. Both use the expert dispatch of this section.

E.5 Wall-clock measurements for configurations in Table 1

Table 11 reports inference (prefilling) and training wall-clock time for the pairs of Table 1. The training column is measured over the 40 B-token runs. At inference, which runs only the forward pass, MoRE is as fast as its MoE counterpart or faster. In training, MoRE costs 15 to 42% more per step, although it has 5 to 6 times as many parameters and 7 to 12 times as many experts.

Since the two models have the same active FLOPs, we attribute the extra time to work whose cost grows with the number of parameters and experts rather than with active FLOPs. First, the AdamW optimizer updates every parameter and its first and second moments at every step, and gradient clipping and the data-parallel gradient all-reduce also process every parameter. Second, the backward pass computes a weight gradient for every expert by summing over the tokens routed to that expert. We expect that storing the AdamW states and the backward gradients in lower precision could help close this gap.

Scale	Model	#Experts	Total params	Prefill (ms)	Training (ms)	Ratio
s=32, h=1024	MoE	128	0.29 B	115.4	241.0	
	MoRE (r=64)	1,536	1.50 B	115.1	312.5	1.30×
s=64, h=512	MoE	256	0.36 B	101.4	165.0	
	MoRE (r=64)	1,792	1.91 B	95.3	221.1	1.34×
s=128, h=512	MoE	256	0.66 B	102.0	182.7	
	MoRE (r=64)	1,792	3.76 B	98.0	259.3	1.42×
s=64, h=512	MoE	1,024	1.27 B	105.4	313.8	
	MoRE (r=64)	8,192	7.92 B	104.0	360.6	1.15×

Table 11: **MoRE prefills as fast as the MoE it matches in active FLOPs and costs 15–42% more per training step.** Wall-clock (ms) for the pairs of Table 1. Prefill is the forward pass of the whole model on one NVIDIA B200 at the per-device batch the training runs use, T=1024, mean of 200 iterations after 100 warm-ups.

In Table 12, we also compare MoRE with a standard MoE of the same configuration, that is, with the same number and width of experts and therefore about the same total parameters. We find MoE trains 1.10 to 1.23× slower than MoRE. This implies that the extra training time of MoRE in Table 11 comes from its larger total number of parameters, not from the low rank.

Configuration	Model	Total params	Training (ms)	MoE / MoRE
M=1,536, s=24, h=1024	MoRE (r=64)	1.50 B	312.5	
	MoE	1.51 B	344.2	1.10×
M=1,792, s=56, h=512	MoRE (r=64)	1.91 B	221.1	
	MoE	1.92 B	249.6	1.13×
M=1,792, s=112, h=512	MoRE (r=64)	3.76 B	259.3	
	MoE	3.77 B	289.6	1.12×
M=8,192, s=52, h=512, EP	MoRE (r=64)	7.92 B	383.8	
	MoE	7.96 B	472.2	1.23×

Table 12: **MoRE and MoE at the same configuration: MoE is 1.10 to 1.23× slower than MoRE.** Training step time (ms) of each MoRE model of Table 11 and of a standard MoE with the same number M and width s of experts, on 4×NVIDIA B200. The last pair is trained with expert parallelism (EP).

F PSEUDOCODE

We give Triton-style pseudocode for the two fused router kernels. Neither materializes the (N, M) logits tensor in HBM. The inference kernel returns the top-k selection alone. The training kernel also returns the statistic that the load-balancing loss needs, and pays a second sweep over the expert dimension for it.

Notation. Write the router logits as

$$\ell \;=\; X R_2^\top R_1^\top \;\in\; \mathbb{R}^{N\times M}, \qquad \ell_{n,j} \in \mathbb{R} \tag{5}$$

for token $n \in [N]$ and expert $j \in [M]$, where $[M] = \{1, \dots, M\}$, with $R_2 \in \mathbb{R}^{r\times h}$ and $R_1 \in \mathbb{R}^{M\times r}$ as in Section 2. Both kernels evaluate (5) one $B_N \times B_M$ tile at a time, so the (N, M) logits exist only in on-chip SRAM. An overbar denotes the gradient of the total loss with respect to that tensor and always has the same shape as the tensor: $\bar{A} = \partial\mathcal{L}/\partial A$.

F.1 INFERENCE (PREFILLING)

At inference the router must name the k experts each token will use and supply the k logits that produce the gate weights. There is no loss, so nothing reads a softmax over all M experts. Algorithm 1

therefore walks the expert dimension once, keeping a running top-k buffer in SRAM, and writes $2Nk$ numbers instead of NM. In Figure 18, we fuse the standard MoE using a similar way, but does not have intermediate z which has size in terms of r. This makes the complexity of fusing MoE scale with h instead with r, and makes the fusing benefit for MoRE diminishes at large h.

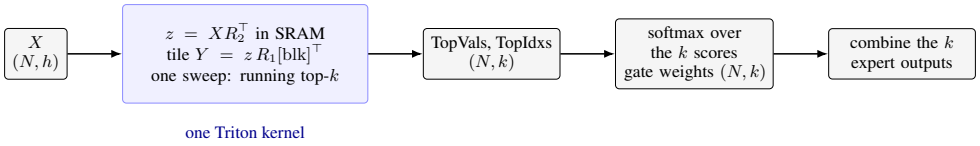


Figure 12: **Flowchart for inference (prefilling) for low-rank router.** The blue block is one Triton kernel, holding one $B_N \times B_M$ tile of ℓ at a time, so neither the (N, r) intermediate z nor the (N, M) logits reaches HBM. TopIdxs names the k experts to run for each token; a softmax over the k entries of TopVals gives the weights that combine their outputs. Nothing here depends on all M experts, which is why one sweep is enough.

Algorithm 1 Fused low-rank router + Top-k (inference, prefilling)

Require: Input $X \in \mathbb{R}^{N\times h}$, router weights $R_1 \in \mathbb{R}^{M\times r}$, $R_2 \in \mathbb{R}^{r\times h}$, top-$k$ size k, block sizes B_N, B_M, B_H
Ensure: TopVals $\in \mathbb{R}^{N\times k}$, TopIdxs $\in \mathbb{Z}^{N\times k}$ in HBM

for each token block $b \in \{0, \ldots, \lceil N/B_N \rceil - 1\}$ **in parallel do**
  ▷ *Phase 1: $z = X_{\text{block}} R_2^\top$ accumulated in SRAM*
  $z \leftarrow 0$ ▷ (B_N, r) in `fp32` registers
  for each hidden tile $i \in \{0, B_H, 2B_H, \ldots\}$ **do**
    Load $X_{\text{tile}} \leftarrow X[bB_N{:}(b{+}1)B_N,\ i{:}i{+}B_H]$ from HBM ▷ (B_N, B_H)
    Load R_2 tile from HBM ▷ (r, B_H)
    $z \mathrel{+}= X_{\text{tile}} \cdot R_{2,\text{tile}}^\top$ ▷ tile matmul in SRAM
  Narrow z to `bf16` ▷ tensor-core operands are narrow; the accumulator is `fp32`
  ▷ *Phase 2: one sweep of $zR_1^\top$ over M with a running top-k*
  Allocate top_vals $\leftarrow -\infty$, top_idxs $\leftarrow 0$ ▷ (B_N, k) in SRAM
  for each expert tile $m_0 \in \{0, B_M, 2B_M, \ldots\}$ **do**
    Load R_1 tile from HBM ▷ (B_M, r), cached in L2 across blocks
    $y_{\text{tile}} \leftarrow z \cdot R_1[m_0{:}m_0{+}B_M, :]^\top$, out-of-range experts set to $-\infty$ ▷ (B_N, B_M) in SRAM
    for $c = 1, \ldots, k$ **do** ▷ merge tile logits into the running top-k
      $(v_{\max}, j_{\max}) \leftarrow (\max, \arg\max)$ of y_{tile} along axis 1
      $(v_{\min}, j_{\min}) \leftarrow (\min, \arg\min)$ of top_vals along axis 1
      If $v_{\max} > v_{\min}$: write $(v_{\max},\ m_0 + j_{\max})$ into slot $j_{\min}$
      Mask $y_{\text{tile}}[:, j_{\max}] \leftarrow -\infty$ ▷ prevent re-selection
  Write top_vals, top_idxs to HBM

The kernel reads X once, R_1 once and R_2 once, and writes $2Nk$ numbers, so its HBM traffic is $O(Nh + Mr + Nk)$ rather than $O(NM)$. The inner merge loop runs k times per expert tile and is sequential, which is why the cost grows linearly in k in Figure 3(Left) and the cost is mild within the range of commonly used k.

F.2 Training

Training adds the load-balancing loss, and with it a quantity that depends on all M experts rather than on the k selected ones. Figure 13 shows its effect on the forward and on the backward.

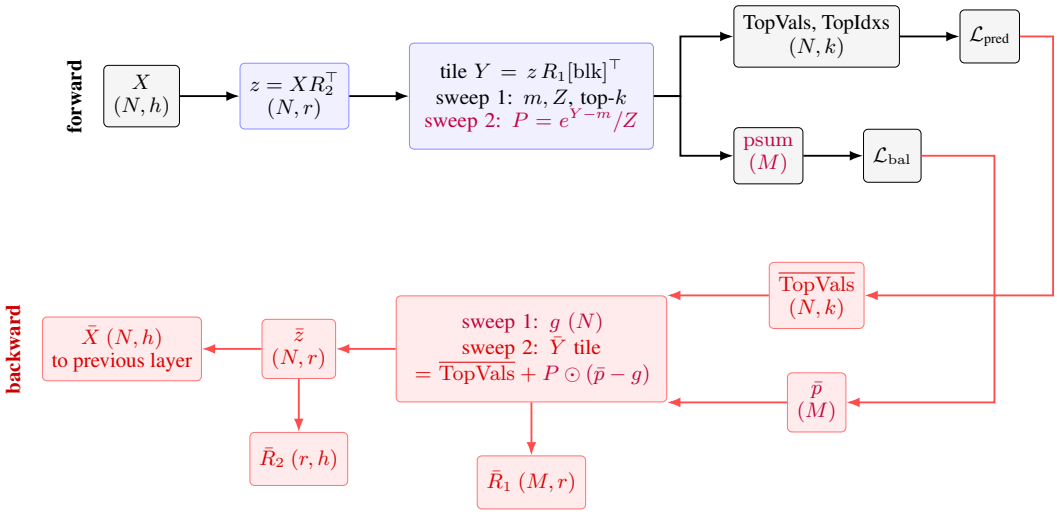


Figure 13: **Fused low-rank router for training.** Blue blocks run inside the kernel, which holds one $B_N \times B_M$ tile of ℓ at a time and never forms the full $N \times M$ matrix. Purple marks what the Switch loss adds. Sweep 2 computes the quantity required for the load-balancing loss: $\text{psum}_i = \sum_n P_{n,i}$, the softmax probability of expert i summed over tokens. $\overline{\text{TopVals}}$ is the gradient of the prediction loss with respect to the k selected scores, and is sparse in M. $\bar{p}$ is the gradient of the balancing loss with respect to psum, one number per expert and dense in M.

F.2.1 Forward-Pass Pseudocode

At forward pass in training, we leverage the techniques in online softmax of Milakov & Gimelshein (2018) to compute the load-balancing loss. Let $P \in \mathbb{R}^{N\times M}$ be the row-wise softmax of the logits,

$$P_{n,j} \;=\; \frac{e^{\,\ell_{n,j}}}{Z_n}, \qquad Z_n \;=\; \sum_{j'=1}^{M} e^{\,\ell_{n,j'}} \in \mathbb{R}. \tag{6}$$

The Switch load-balancing loss (Fedus et al., 2022) is $\mathcal{L}_{\text{bal}} = M \sum_{i=1}^{M} f_i p_i$, where f_i is the fraction of dispatch slots that went to expert i and carries no gradient, and $p_i = \frac{1}{N}\sum_n P_{n,i}$ is that expert's mean routing probability over *all* M experts. The only quantity in it that a tiled kernel cannot produce for free is the column sum

$$\text{psum}_i \;=\; \sum_{n=1}^{N} P_{n,i} \;\in\; \mathbb{R}^M, \qquad p_i = \text{psum}_i/N, \tag{7}$$

because $P_{n,i}$ needs the normalizer Z_n, which is not known until every expert has been visited. Returning psum is therefore the whole of what the kernel must add, and it is M numbers rather than NM.

Building the softmax denominator without the (N, M) tensor. The kernel maintains the row maximum $m_n = \max_{j\in[M]} \ell_{n,j}$ and the denominator $Z_n = \sum_{j'} e^{\ell_{n,j'} - m_n}$ across the expert tiles, and uses $P_{n,j} = e^{\ell_{n,j} - m_n}/Z_n$. Neither m_n nor Z_n is final until every expert has been seen, so both are built incrementally. After t tiles the kernel holds $m^{(t)}$, the largest logit so far, and $Z^{(t)}$, the sum over those experts. When tile $t{+}1$ raises the maximum to m', every term already in $Z^{(t)}$ carries the factor $e^{-m^{(t)}}$ and now needs $e^{-m'}$. That correction is one constant shared by all of them, so a single multiply rescales the whole accumulated sum, and the calculation is shown as (8):

$$Z^{(t+1)} = Z^{(t)}\, e^{\,m^{(t)} - m'} \;+\; \textsc{rowsum}\big(e^{\,y_{\text{tile}} - m'}\big), \qquad m^{(t+1)} = m'. \tag{8}$$

Here $e^{\,y_{\text{tile}} - m'}$ is elementwise on the (B_N, B_M) tile and ROWSUM adds each row across the B_M experts of this tile, so both sides of (8) are vectors of length B_N: one running normalizer per token.

Once m and Z are final, a second sweep regenerates each tile and accumulates $\text{COLSUM}(e^{y_{\text{tile}}-m}/Z)$ into psum, where COLSUM reduces along tokens and so returns one number per expert. The second sweep is needed because the normalizer is only complete at the end of the first, and because the top-k merge overwrites y_{tile} with $-\infty$ as it consumes columns. It costs one extra $zR_1^\top$ matmul, $2NMr$ FLOPs, and no extra HBM traffic.

In the implementation the first matmul $z = XR_2^\top$ is left to cuBLAS and z is passed into the kernel, so the training kernel fuses the second matmul, the top-k and the softmax statistics, while the inference kernel of Algorithm 1 fuses all three.

Algorithm 2 Fused low-rank router + Top-k (training, forward). Purple lines are what the Switch load-balancing loss adds.

Require: $z = XR_2^\top \in \mathbb{R}^{N\times r}$, router weight $R_1 \in \mathbb{R}^{M\times r}$, top-$k$ size k, block sizes B_N, B_M
Ensure: TopVals $\in \mathbb{R}^{N\times k}$, TopIdxs $\in \mathbb{Z}^{N\times k}$, psum $\in \mathbb{R}^M$, $m \in \mathbb{R}^N$, $Z \in \mathbb{R}^N$ in HBM

for each token block $b \in \{0, \dots, \lceil N/B_N \rceil - 1\}$ **in parallel do**
  Load z_{block} from HBM ▷ (B_N, r) in SRAM registers
  ▷ *Sweep 1: tile $zR_1^\top$ over M with running top-k and running m, Z*
  Allocate top_vals $\leftarrow -\infty$, top_idxs $\leftarrow 0$ ▷ (B_N, k) in SRAM
  Allocate $m \leftarrow -\infty$, $Z \leftarrow 0$ ▷ (B_N) each, in SRAM
  for each expert tile $m_0 \in \{0, B_M, 2B_M, \dots\}$ **do**
    Load R_1 tile from HBM ▷ (B_M, r), cached in L2 across blocks
    $y_{\text{tile}} \leftarrow z \cdot R_1^\top$, out-of-range experts set to $-\infty$ ▷ (B_N, B_M) in SRAM
    $m' \leftarrow \max\big(m,\ \text{ROWMAX}(y_{\text{tile}})\big)$ ▷ new running maximum, (B_N)
    $Z \leftarrow Z\, e^{m-m'} + \text{ROWSUM}\big(e^{y_{\text{tile}}-m'}\big);\ m \leftarrow m'$ ▷ eq. (8)
    for $c = 1, \dots, k$ **do** ▷ merge tile logits into running top-k
      $(v_{\max}, j_{\max}) \leftarrow$ (max, arg max) of y_{tile} along axis 1
      $j_{\min} \leftarrow$ arg min of top_vals along axis 1
      If $v_{\max} >$ top_vals$[:, j_{\min}]$: write $(v_{\max}, m_0 + j_{\max})$ into slot $j_{\min}$
      Mask $y_{\text{tile}}[:, j_{\max}] \leftarrow -\infty$ ▷ destroys y_{tile}; prevent re-selection
  ▷ *Sweep 2: normalized column sums; m and Z are still in registers*
  for each expert tile $m_0 \in \{0, B_M, 2B_M, \dots\}$ **do**
    $y_{\text{tile}} \leftarrow z \cdot R_1^\top;\ P \leftarrow e^{y_{\text{tile}}-m}/Z$ ▷ (B_N, B_M), recomputed
    $\text{ATOMICADD}\big(\text{psum}[m_0{:}m_0{+}B_M],\ \text{COLSUM}(P)\big)$
  Write top_vals, top_idxs, m, Z to HBM

Forming the loss. The balancing loss itself stays outside the kernel and costs $O(M)$:

$$\mathcal{L}_{\text{bal}} \;=\; M\sum_{i=1}^{M} f_i\, p_i \;=\; \frac{M}{N}\, f \cdot \text{psum}, \qquad f_i = \frac{1}{Nk}\sum_{n=1}^{N}\sum_{j=1}^{k} \mathbf{1}\big[\text{TopIdxs}[n,j] = i\big], \tag{9}$$

with f obtained by a `bincount` on TopIdxs and detached, as is standard. Because the kernel returns psum rather than a loss value, the same outputs serve the Switch form of (9) or a different balancing objective, without touching the kernel.

F.2.2 Backward-Pass Pseudocode

Two gradients reach the kernel, and they have different shapes. $\overline{\text{TopVals}} \in \mathbb{R}^{N\times k}$ arrives from the prediction loss on the k selected scores; it is sparse in M, touching only the selected experts. $\bar{p} \in \mathbb{R}^M$ arrives from $\mathcal{L}_{\text{bal}}$ through psum, and by (9) it is $\bar{p}_i = (M/N) f_i$, one number per expert and dense in M. The kernel must return $\bar{X} \in \mathbb{R}^{N\times h}$, $\bar{R}_1 \in \mathbb{R}^{M\times r}$ and $\bar{R}_2 \in \mathbb{R}^{r\times h}$, as in Figure 13. Everything else follows from the chain rule on $\ell = zR_1^\top$ and $z = XR_2^\top$, so the only new quantity to derive is the logit gradient $\bar{y}$.

The logit gradient of the balancing loss. Fix a token n and an expert $j \in [M]$. Since $\text{psum}_i = \sum_{n'} P_{n',i}$ and only row n of P depends on ℓ_n, the chain rule contracts over the expert index alone:

$$\frac{\partial \mathcal{L}_{\text{bal}}}{\partial \ell_{n,j}} = \sum_{i=1}^{M} \bar{p}_i\, \frac{\partial P_{n,i}}{\partial \ell_{n,j}}. \tag{10}$$

The softmax Jacobian follows from the quotient rule on $P_{n,i} = e^{\ell_{n,i}}/Z_n$ with $Z_n = \sum_{j'} e^{\ell_{n,j'}}$. The numerator depends on $\ell_{n,j}$ only when $i = j$, contributing $\delta_{ij} e^{\ell_{n,i}}$ with $\delta_{ij} = 1$ if $i = j$ and 0 otherwise; the denominator always does, contributing $\partial Z_n / \partial \ell_{n,j} = e^{\ell_{n,j}}$. Hence

$$\frac{\partial P_{n,i}}{\partial \ell_{n,j}} = \frac{\delta_{ij}\, e^{\ell_{n,i}} Z_n - e^{\ell_{n,i}} e^{\ell_{n,j}}}{Z_n^2} = \delta_{ij} \frac{e^{\ell_{n,i}}}{Z_n} - \frac{e^{\ell_{n,i}}}{Z_n} \frac{e^{\ell_{n,j}}}{Z_n} = P_{n,i}\big(\delta_{ij} - P_{n,j}\big). \tag{11}$$

Substituting (11) into (10), the δ_{ij} collapses the first sum to its $i = j$ entry and the second factors out $P_{n,j}$:

$$\frac{\partial \mathcal{L}_{\text{bal}}}{\partial \ell_{n,j}} = \underbrace{\bar{p}_j P_{n,j}}_{\text{from } \delta_{ij}} - P_{n,j} \sum_{i=1}^{M} \bar{p}_i P_{n,i} = P_{n,j}\big(\bar{p}_j - g_n\big), \qquad g_n \equiv \sum_{i=1}^{M} \bar{p}_i P_{n,i} \in \mathbb{R}. \tag{12}$$

So $g \in \mathbb{R}^N$ is one scalar per token: the mean of $\bar{p}$ under that token's own routing distribution. Because g_n sums over all M experts, it must be complete before any single expert's gradient can be written, so the backward takes two sweeps for the same reason the forward does.

Algorithm 3 Low-rank router (training, backward) with activation recompute. Purple lines are what the Switch load-balancing loss adds.

Require: X, $R_1 \in \mathbb{R}^{M\times r}$, $R_2 \in \mathbb{R}^{r\times h}$, TopIdxs $\in \mathbb{Z}^{N\times k}$, $\overline{\text{TopVals}} \in \mathbb{R}^{N\times k}$ (top-k logit gradients), $\bar{p} \in \mathbb{R}^M$, saved $m, Z \in \mathbb{R}^N$
Ensure: $\bar{X} \in \mathbb{R}^{N\times h}$, $\bar{R}_1 \in \mathbb{R}^{M\times r}$, $\bar{R}_2 \in \mathbb{R}^{r\times h}$ in HBM

for each token block b **in parallel do**
  ▷ *Phase 1: recompute $z = X_{\text{block}} R_2^\top$ in SRAM (saves (N, r) activation memory)*
  $z \leftarrow 0$ ▷ (B_N, r) in SRAM
  for each hidden tile i **do**
    $z \mathrel{+}= X_{\text{tile}} \cdot R_2^\top$
  ▷ *Phase 2, sweep 1: the per-token scalar g_n of eq. (12)*
  $g \leftarrow 0$ ▷ (B_N) in SRAM
  for each expert tile $m_0 \in \{0, B_M, 2B_M, \ldots\}$ **do**
    $y_{\text{tile}} \leftarrow z \cdot R_1^\top$; $P \leftarrow e^{\,y_{\text{tile}} - m}/Z$
    $g \mathrel{+}= \textsc{rowsum}\big(P \odot \bar{p}[m_0{:}m_0{+}B_M]\big)$
  ▷ *Phase 2, sweep 2: the logit gradient, contracted into $\bar{z}$ and $\bar{R}_1$*
  $\bar{z} \leftarrow 0$ ▷ (B_N, r) in SRAM
  for each expert tile $m_0 \in \{0, B_M, 2B_M, \ldots\}$ **do**
    $\bar{y} \leftarrow 0$; scatter $\overline{\text{TopVals}}[n, j]$ into column TopIdxs$[n, j]$ of this tile ▷ sparse: k of B_M
    $y_{\text{tile}} \leftarrow z \cdot R_1^\top$; $P \leftarrow e^{\,y_{\text{tile}} - m}/Z$
    $\bar{y} \mathrel{+}= P \odot \big(\bar{p}[m_0{:}m_0{+}B_M] - g\big)$ ▷ eq. (12), dense in M
    $\bar{z} \mathrel{+}= \bar{y} \cdot R_1[m_0{:}m_0{+}B_M, :]$; $\bar{R}_1[m_0{:}m_0{+}B_M, :] \mathrel{+}= \bar{y}^\top z$
  ▷ *Phase 3: $\bar{X}_{\text{block}} = \bar{z} \cdot R_2$ tiled over h*
  for each hidden tile i **do**
    Load R_2 tile, write $\bar{X}_{\text{tile}} \leftarrow \bar{z} \cdot R_2$
  $\bar{R}_2 \mathrel{+}= \bar{z}^\top X_{\text{block}}$ ▷ $(r, B_N) \times (B_N, h) \to (r, h)$

G Additional Plots

G.1 Routing entropy

We report the routing entropy of the per-expert token counts $c_1, \ldots, c_M$ of one layer. Writing $p_a = c_a / \sum_b c_b$ for the share of the layer's routed tokens that expert a receives, the normalized entropy is $H_{\text{norm}} = \big(-\sum_a p_a \log p_a\big)/\log M$, accumulated over the full training set. It is 1 when every expert carries the same load, and falls towards 0 as routing concentrates on fewer experts. The counts are read from saved checkpoints: for each checkpoint we pass the full training set through that fixed model once and count the tokens each expert receives.

Figure 14 repeats Figure 2 at h=64, s=8, M=1024, where the expert pool is more than five times larger, and we find similar results.

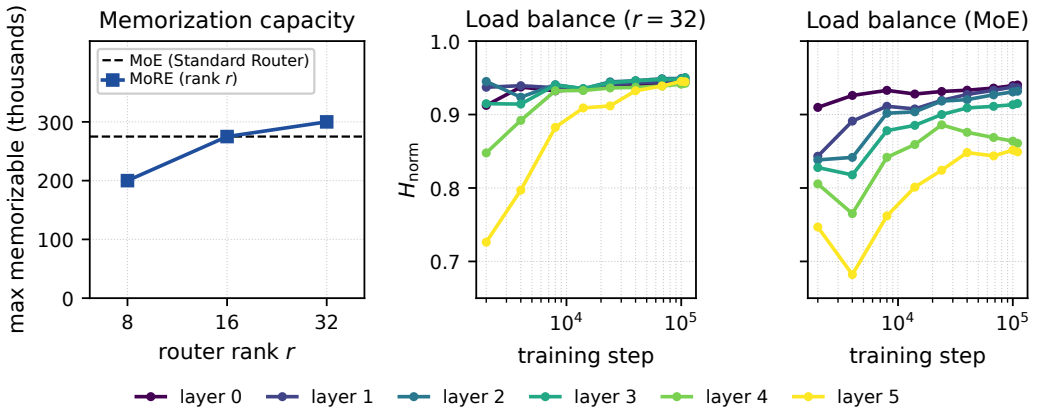


Figure 14: **Similar results at a larger expert pool.** Phonebook, h=64, s=8, M=1024, k=4, L=6, panels as in Figure 2. **Left:** largest phonebook memorized, searched on a 25K grid. **Middle, Right:** H_{norm} per layer against training step, for r=32 and for MoE, both trained on the 200K phonebook. This configuration is off the matched-FLOP curve of Proposition 2: it places several ranks below $\log_2 M$, and both arms hold the same experts, so only the router differs.

G.2 Forward pass of a single layer at larger expert widths

Figure 15 repeats Figure 4 at $s \in \{128, 256, 512\}$ with the same protocol: inference (prefilling) of the whole layer, k=4, r=64, N = 262,144 (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups on a single NVIDIA B200. The fused MoRE layer is faster in every cell that fits in memory. The speedup still grows with M along every row, and it shrinks as s grows, because wider experts make the expert computation, which both layers share, a larger part of the layer. At M = 8,192 it is 1.7 to 4.2× at s=128, 1.6 to 3.6× at s=256 and 1.6 to 3.0× at s=512.

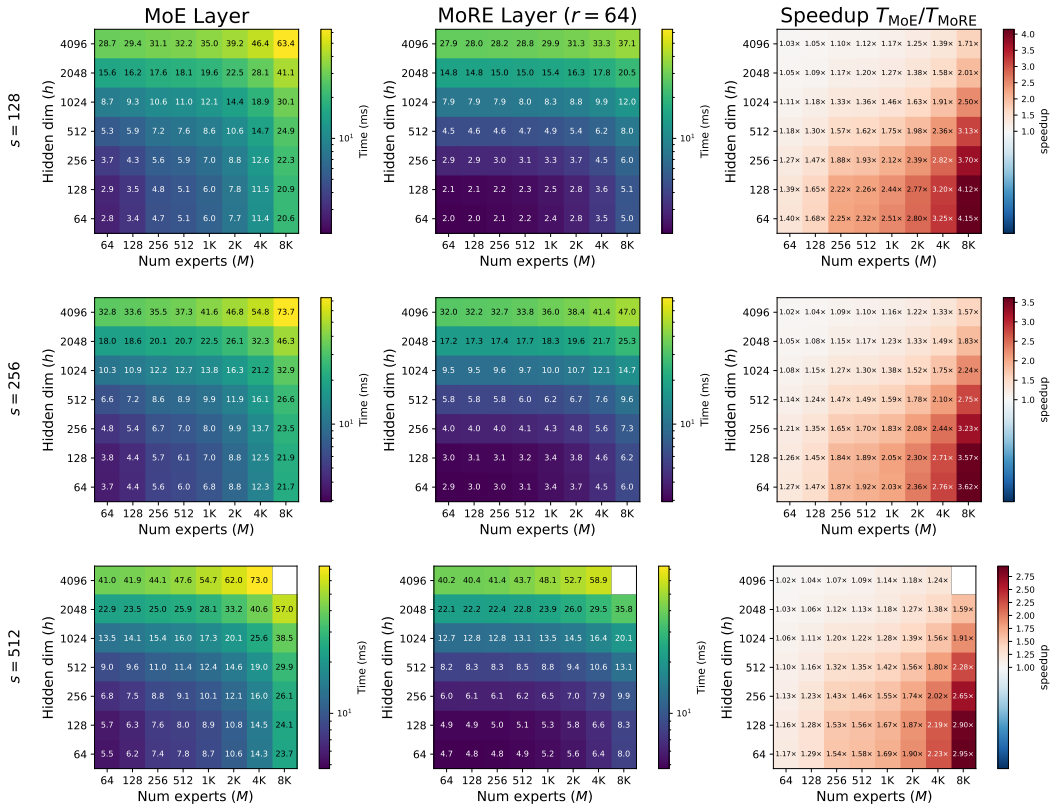


Figure 15: **The fused MoRE layer stays faster at larger expert widths.** Inference (prefilling) of a single layer at s=128, 256 and 512 (rows). **(a)** MoE Layer. **(b)** MoRE Layer (r=64). **(c)** speedup $T_{\text{MoE}}/T_{\text{MoRE}}$. The blank cell at s=512 ran out of memory on one B200. k=4, r=64, N = 262,144 (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups on a single NVIDIA B200.

G.3 The same grid on an A100

Figure 16 repeats the inference (prefilling) measurement of Figure 4 on a single NVIDIA A100-80GB, over the same (h, M) grid: hidden dimension $h \in \{64, 128, 256, 512, 1{,}024, 2{,}048, 4{,}096\}$ against expert count $M \in \{64, 128, 256, 512, 1{,}024, 2{,}048, 4{,}096, 8{,}192\}$, at s=64, k=4 and r=64. The fused MoRE layer is faster in every cell, by 1.04 to 3.07×, and the speedup grows with M along every row, reaching 2.60 to 3.07× at M = 8,192.

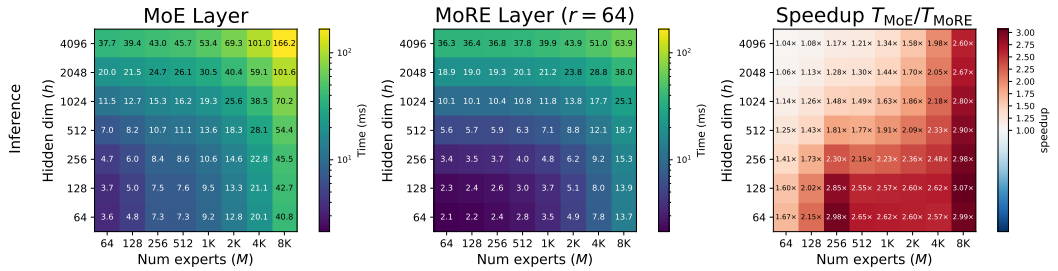


Figure 16: **The A100 reproduces the B200 result, with the fused MoRE layer** 2.60 **to** 3.07× **faster at** M=8,192**.** Inference (prefilling) of a single layer, the two arms at equal M. **(a)** MoE Layer. **(b)** MoRE Layer (r=64). **(c)** speedup $T_{\text{MoE}}/T_{\text{MoRE}}$. Single NVIDIA A100-80GB, s=64, k=4, N = 262,144 (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups.

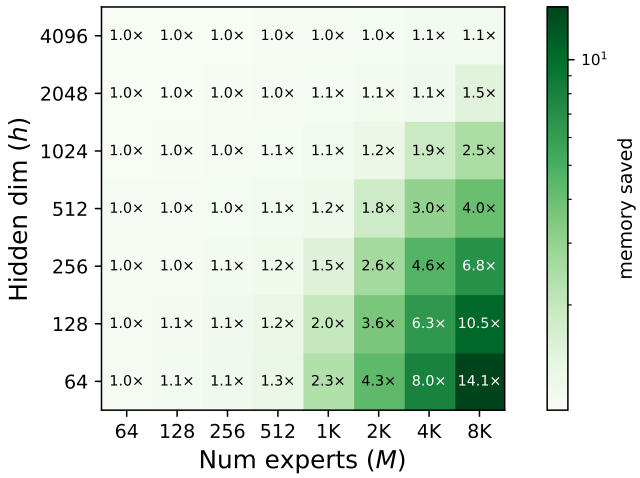


Figure 17: **Fusing the low-rank router reduces peak memory by up to $14\times$ for the whole layer.** Ratio of peak memory over a forward and a backward, unfused MoRE against fused MoRE, on the same grid.

G.4 Fusing a standard MoE layer

The kernel of Section 4.2 can also be written for a standard router. In Figure 18, we find the benefit of fusing diminishes as h grows larger.

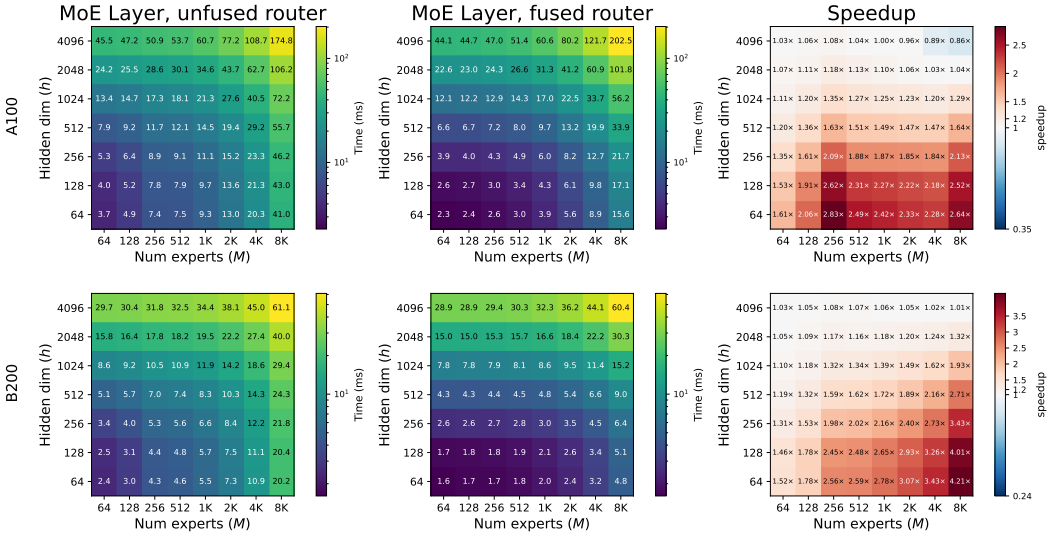


Figure 18: **On both A100 and B200, fusing Standard MoE does not get much acceleration at large h.** Inference (prefilling) of a single layer, both arms a standard MoE layer at the same M. **(a)** unfused router. **(b)** fused router. **(c)** speedup. **(Top)** A100, 0.86 to $2.83\times$. **(Bottom)** B200, 1.01 to $4.21\times$. At h=4,096, M=8,192 fusing takes the layer from 174.8 to 202.5 ms on the A100 and from 61.1 to 60.4 ms on the B200. s=64, k=4, $N = 262{,}144$ (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups.

Figure 18 holds M fixed along each row. Reading one row of it, M=4,096, against the low-rank router at the same shape gives Figure 19: fusing takes the same work off both layers, and only the standard one gives it back as h grows.

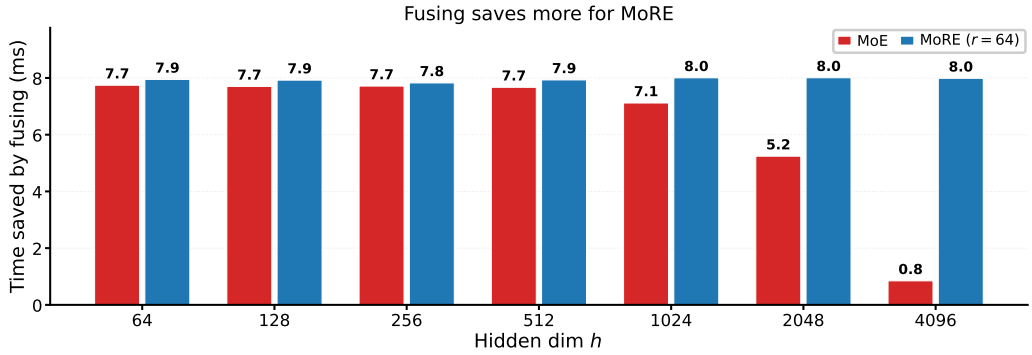


Figure 19: **Fusing keeps saving time for MoRE as h grows, but stops saving time for MoE.** Forward pass of a single layer, the unfused-router time minus the fused-router time, against the hidden dimension h at a fixed pool of M=4,096 experts. What fusing removes is the round trip of the (N, M) scores, which depends on N and M but not on h. The standard router gives it back because its fused kernel reloads a (B_M, h) tile of the router weight for every token block. s=64, k=4, r=64, N = 262,144 (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups on a single NVIDIA B200.

G.5 The low-rank router once both layers are fused

Figure 20 compares MoE and MoRE (r=64) where both are fused.

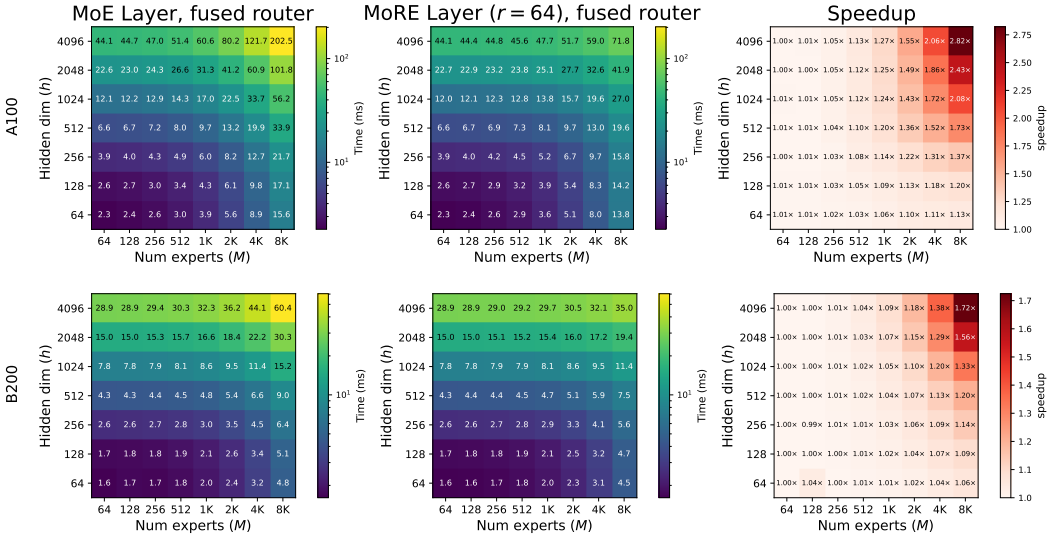


Figure 20: **Comparing fused MoE and fused MoRE, the low-rank factorization gives speedup up to 2.82× on an A100 and 1.72× on a B200.** Inference (prefilling) of a single layer, both arms fused, at the same M. **(a)** MoE Layer with the fused router. **(b)** MoRE Layer (r=64) with the fused router. **(c)** speedup. **(Top)** A100, 1.00 to 2.82×. **(Bottom)** B200, 0.99 to 1.72×. s=64, k=4, N = 262,144 (B=256, T=1024), `bf16`, mean of 30 trials after 10 warm-ups.

G.6 End-to-End Forward-Pass Breakdown at Large M

Figure 21 measures the end-to-end wall-clock time of a 12-layer transformer with 8 attention heads (FlashAttention-4 (Zadouri et al., 2026)), vocabulary size V=50,257, h=512, s=64, k=4, B=64, T=1024, sweeping $M \in \{1{,}024, 2{,}048, 4{,}096, 8{,}192\}$.

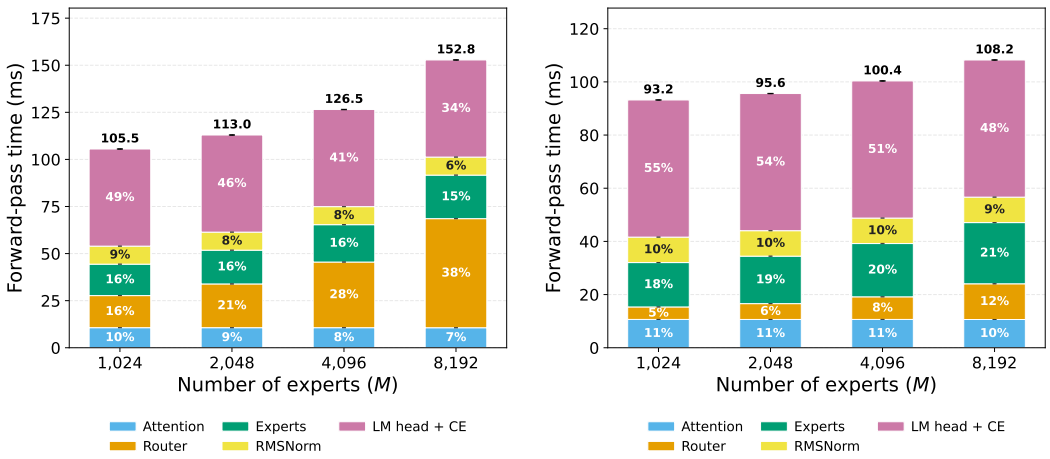


Figure 21: **End-to-end forward-pass breakdown at h=512, sweeping M.** Each bar decomposes the wall-time of a 12-layer transformer forward pass into Attention, Router, Experts (dispatch + grouped MLP), RMSNorm, and the Liger-fused (Hsu et al., 2024) LM head + cross-entropy. **(Left)** Standard **MoE** router (unfused $h \times M$ matmul + top-k + softmax + load-balancing-loss as separate kernels): the router slice grows from 16% of total at M=1,024 to **38% at** M=8,192, becoming the largest single component and the dominant driver of the rising end-to-end cost. **(Right)** Fused low-rank **MoRE** router (r=64): the router slice stays at 5–12% across the same sweep, and total wall-time grows much more slowly with M. At M=8,192, MoRE is **30% faster end-to-end** (108.2 ms vs. 152.8 ms), with router cost reduced from 57.9 ms to 13.4 ms. Mean of 200 trials with 50 warm-ups, 95% C.I. on a single NVIDIA B200.

G.7 Adding More Experts Help

We sweep the size of the expert pool over $M \in \{112, 224, 448, 896, 1792\}$ while holding the rest of the architecture fixed at h=512, s=56, r=64, k=4 and L=12. All five configurations train on the same 5B tokens. Figure 22 shows that validation perplexity decreases monotonically over the whole range, from 19.10 at M=112 to 17.05 at M=1792, demonstrating the benefits of using more experts.

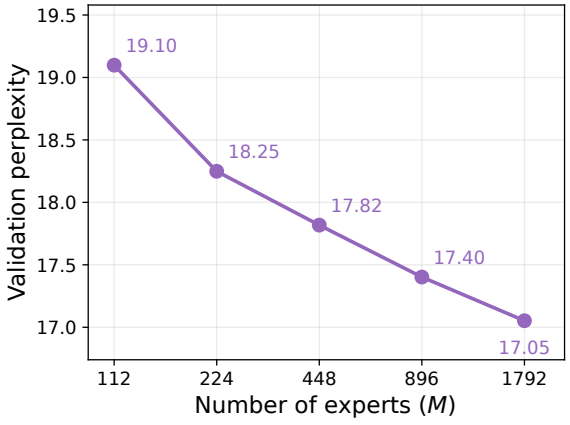


Figure 22: **Adding more experts on MoRE keeps improving quality.** Validation perplexity of MoRE ($r = 64$) after 5B training tokens, against the number of experts M. The curve is monotone across a 16× increase in the pool size.

G.8 Fusion benefit at different batch sizes

In Figure 23, we find that the benefit of fusing grows with the batch size. The fully fused MoRE router is $1.4\times$ faster than the unfused one at $B{=}2$ and $6.3\times$ faster at $B{=}256$. At $B{=}2$ both routers take about 0.1 ms. As B grows, most of the unfused router's time goes to the top-k over the (N, M) score tensor that it stores in HBM, whose size grows with the number of tokens $N = B \cdot T$, while the fused kernel never stores this tensor.

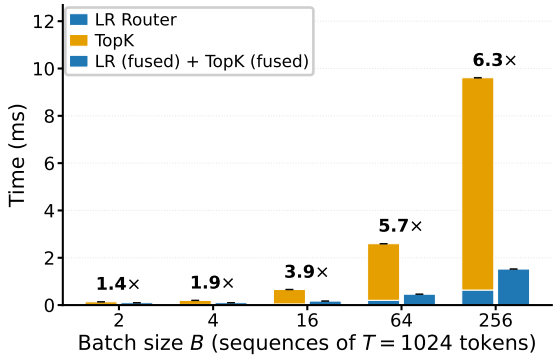


Figure 23: **The benefit of fusing the MoRE router grows with the batch size.** Forward-pass time of the low-rank router, unfused and fully fused, with the speedup of the fused kernel above each pair. $h{=}2048$, $M{=}4{,}096$, $r{=}16$, $k{=}4$, $T{=}1024$, `bf16`, mean of 200 trials after 50 warm-ups on a single NVIDIA B200.

G.9 The perplexity gap grows with the token budget

In Figure 24, we find that the perplexity gap between MoRE and MoE grow with token budget, which we think the gap would grow larger if more tokens are trained. This also matches our intuition, since MoRE's benefits of memorizing more information would be more apparent when more tokens are trained.

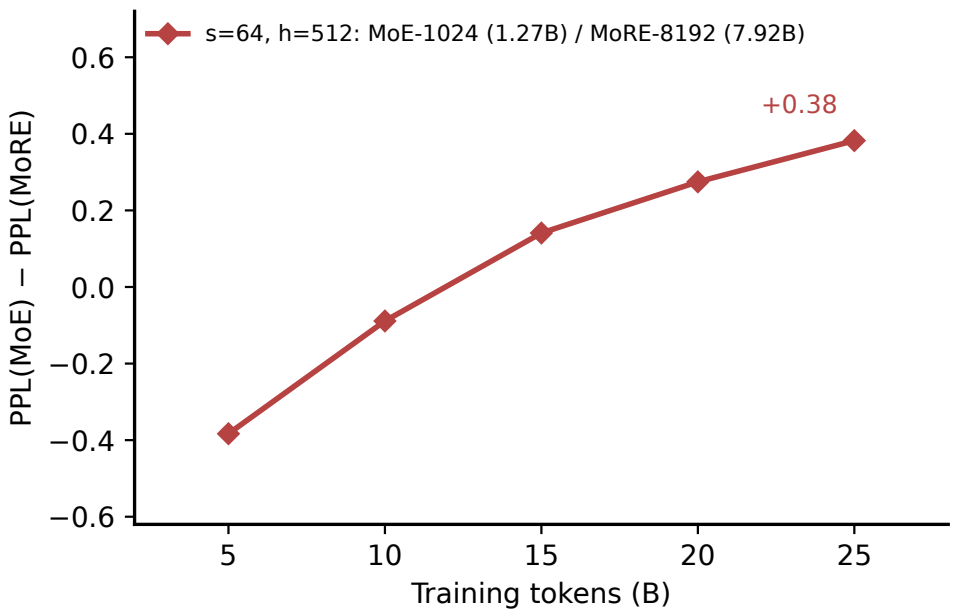


Figure 24: **The perplexity gap between the two configurations grows with the token budget.** Validation perplexity of MoE-1024 minus that of MoRE-8192 on the 5,000 held-out mixture sequences, against training tokens. Positive values mean MoRE has the lower perplexity. Both models have matched active FLOPs and 1.27B against 7.92B total parameters. We plot the first 25B tokens before the cosine schedule enters its annealing tail.

H Scaling Total Parameters

Proposition 2 (Optimal scaling under fixed active-FLOPs)**.** *Fix the active-FLOP budget τ, the hidden dimension h, and the active expert count k. The MoE configuration (M, k, h, s) and the MoRE configuration (M', k, h, s', r) that maximize total MLP parameters under matched τ are*

$$M^* = \frac{\tau}{2C_r h}, \qquad s^* = \frac{\tau}{2C_e k h}, \qquad T^*_{\mathrm{MoE}} = \frac{3\tau^2}{4C_e C_r\, hk},$$
$$M' = \frac{\tau'}{2C_r r}, \qquad s' = \frac{\tau'}{2C_e k h}, \qquad T^*_{\mathrm{MoRE}} = \frac{3(\tau')^2}{4C_e C_r\, rk},$$

where $\tau' := \tau - C_r hr$. Furthermore, in the regime $M, M' \gg r$ (equivalently $\tau \gg C_r hr$),

$$\frac{M'}{M^*} \to \frac{h}{r}, \qquad \frac{T^*_{\mathrm{MoRE}}}{T^*_{\mathrm{MoE}}} \to \frac{h}{r}.$$

Proof. We use the same budget-allocation argument in both settings.

MoE. Allocate fraction $\alpha \in [0, 1]$ of the budget to the router and $1 - \alpha$ to the experts:

$$C_r Mh = \alpha\tau, \qquad C_e khs = (1-\alpha)\tau,$$

so $M = \alpha\tau/(C_r h)$ and $s = (1-\alpha)\tau/(C_e kh)$. Substituting into $T_{\mathrm{MoE}} = 3Mhs$,

$$T_{\mathrm{MoE}}(\alpha) = \frac{3\tau^2}{C_e C_r\, hk} \cdot \alpha(1-\alpha),$$

which is maximized at $\alpha = 1/2$, yielding the stated $M^*, s^*, T^*_{\mathrm{MoE}}$.

MoRE. The router cost expands as $C_r(M' + h)r = C_r M' r + C_r hr$, where the second term is a fixed overhead independent of M'. Subtracting it leaves a variable budget $\tau' := \tau - C_r hr$, which we split the same way:

$$C_r M' r = \alpha\tau', \qquad C_e khs' = (1-\alpha)\tau',$$

so $M' = \alpha\tau'/(C_r r)$ and $s' = (1-\alpha)\tau'/(C_e kh)$. Substituting into $T_{\mathrm{MoRE}} = 3M'hs'$,

$$T_{\mathrm{MoRE}}(\alpha) \;=\; \frac{3(\tau')^2}{C_e C_r\, rk} \cdot \alpha(1-\alpha),$$

again maximized at $\alpha = 1/2$, yielding the stated $M', s', T^*_{\mathrm{MoRE}}$.

Limit. The ratios are

$$\frac{M'}{M^*} \;=\; \frac{h}{r}\left(1 - \frac{C_r hr}{\tau}\right), \qquad \frac{T^*_{\mathrm{MoRE}}}{T^*_{\mathrm{MoE}}} \;=\; \frac{h}{r}\left(1 - \frac{C_r hr}{\tau}\right)^2.$$

Since $\tau = C_e khs + C_r Mh > C_r Mh \gg C_r hr$ when $M \gg r$, the correction $C_r hr/\tau \ll 1$, and both ratios converge to h/r. □

I THEORETICAL RESULTS

Section 3 states the rank bounds under the normalization $\|u_i\|_2 \le 1$ and $\|R_2 x\|_2 \le 1$. The proofs below are given in a slightly more general form with arbitrary norm bounds B_u and B_z. In this form, it is convenient to measure margins in normalized units: a normalized margin η corresponds to a raw logit margin $\eta B_u B_z$. Setting $B_u = B_z = 1$ recovers the statements used in Section 3.

We first prove the lower bound. The argument shows that realizing many distinct top-k assignments forces many separated projected representations to fit inside an r-dimensional ball.

Theorem 3 (Lower bound on required rank, general norm bounds)**.** *Let $\mathcal{F} \subseteq \binom{[M]}{k}$ and let $B_u, B_z > 0$. If a rank-r router $R = R_1 R_2$ with $\|u_i\|_2 \le B_u$ and $\|R_2 x_S\|_2 \le B_z$ realizes every $S \in \mathcal{F}$ at margin at least $\eta B_u B_z$, then*

$$|\mathcal{F}| \le (1 + 2/\eta)^r, \qquad r \ge \frac{\log|\mathcal{F}|}{\log(1 + 2/\eta)}.$$

Proof. The proof is a volume packing argument: distinct assignments force their projections z_S to be far apart in $\mathbb{R}^r$, and we count how many such well-separated points fit in a ball.

For each $S \in \mathcal{F}$, choose an input x_S realizing S with margin $\eta B_u B_z$, and write $z_S = R_2 x_S$. Take any two distinct $S, T \in \mathcal{F}$: since $|S| = |T| = k$, there exist $i \in S \setminus T$ and $j \in T \setminus S$. At x_S expert i wins over j, while at x_T expert j wins over i, so

$$\langle u_i - u_j,\, z_S\rangle \;\ge\; \eta B_u B_z, \qquad \langle u_j - u_i,\, z_T\rangle \;\ge\; \eta B_u B_z.$$

Adding the inequalities and applying Cauchy–Schwarz with $\|u_i - u_j\|_2 \le 2B_u$ gives $\|z_S - z_T\|_2 \ge \eta B_z$.

Hence $\{z_S : S \in \mathcal{F}\}$ are $|\mathcal{F}|$ points pairwise separated by at least ηB_z, all lying in the radius-B_z ball of $\mathbb{R}^r$. The open balls of radius $\eta B_z/2$ around these points are therefore disjoint and contained in the ball of radius $B_z(1 + \eta/2)$. Comparing volumes,

$$|\mathcal{F}| \cdot \left(\frac{\eta B_z}{2}\right)^r \;\le\; \left(B_z\left(1 + \frac{\eta}{2}\right)\right)^r,$$

which rearranges to $|\mathcal{F}| \le (1 + 2/\eta)^r$. □

Proof of Theorem 2. Apply Theorem 3 with $B_u = B_z = 1$. □

We next prove the upper bound. The construction first realizes the desired assignments in dimension M using the standard basis, then applies a Johnson–Lindenstrauss projection (Johnson & Lindenstrauss, 1984) to reduce the dimension while preserving the inner products that determine the top-k ordering.

Theorem 4 (Upper bound on required rank, general norm bounds)**.** *Fix $1 \le k \le M - 1$, $\mathcal{F} \subseteq \binom{[M]}{k}$, and norm bounds $B_u, B_z > 0$. Set $\rho := \sqrt{1/k + 1/(M-k)} > 1/\sqrt{k}$. For any target normalized margin $\eta \in (0, \rho)$, $h \ge r$, and*

$$r \;\ge\; \frac{C \log(M + |\mathcal{F}|)}{(\rho - \eta)^2}$$

with universal $C > 0$, there is a rank-r router with $\|u_i\|_2 \le B_u$ realizing every $S \in \mathcal{F}$ at some input with $\|R_2 x_S\|_2 \le B_z$ and margin at least $\eta B_u B_z$.

Proof. We first solve the problem in the uncompressed regime $r = M$, then use a Johnson–Lindenstrauss projection to compress down to $r = O(\log(M + |\mathcal{F}|)/(\rho - \eta)^2)$ while preserving the inner products that determine routing.

To begin, let $e_1, \dots, e_M$ be the standard basis of $\mathbb{R}^M$, which serve as expert vectors. For each $S \in \mathcal{F}$, define

$$z_S \;:=\; \frac{1}{\rho}\left(\frac{\mathbf{1}_S}{k} - \frac{\mathbf{1}_{[M]\setminus S}}{M-k}\right).$$

A direct computation gives

$$\|z_S\|_2^2 \;=\; \frac{1}{\rho^2}\Big(k \cdot \tfrac{1}{k^2} + (M-k)\cdot \tfrac{1}{(M-k)^2}\Big) \;=\; \frac{1}{\rho^2}\Big(\tfrac{1}{k} + \tfrac{1}{M-k}\Big) \;=\; 1.$$

For $i \in S$ and $j \notin S$,

$$\langle e_i, z_S\rangle \;=\; \frac{1}{k\rho}, \qquad \langle e_j, z_S\rangle \;=\; -\frac{1}{(M-k)\rho}, \qquad \langle e_i, z_S\rangle - \langle e_j, z_S\rangle \;=\; \rho.$$

So in dimension M, the trivial choice of expert vectors e_i realizes every $S \in \mathcal{F}$ with margin exactly ρ, which is strictly larger than the target η.

We now compress this construction via Johnson–Lindenstrauss. Let $\mathcal{V} = \{e_1, \dots, e_M\} \cup \{z_S : S \in \mathcal{F}\}$, so $|\mathcal{V}| \le M + |\mathcal{F}|$. By the Johnson–Lindenstrauss lemma, for $r = O(\log|\mathcal{V}|/\varepsilon^2)$ there is a linear map $\Phi : \mathbb{R}^M \to \mathbb{R}^r$ preserving squared Euclidean distances on $\mathcal{V} \cup \{0\}$ multiplicatively to within $(1 \pm \varepsilon)$, i.e. $(1-\varepsilon)\|u-v\|_2^2 \le \|\Phi u - \Phi v\|_2^2 \le (1+\varepsilon)\|u-v\|_2^2$ for all $u, v \in \mathcal{V} \cup \{0\}$. Since every $u \in \mathcal{V}$ has unit norm, $\|u-v\|_2^2 \le 4$ for all such pairs, so this is equivalent to additive preservation up to $O(\varepsilon)$; absorbing the constant, we assume preservation up to additive error ε in what follows. The polarization identity

$$\langle u, v\rangle \;=\; \tfrac{1}{2}\big(\|u\|_2^2 + \|v\|_2^2 - \|u-v\|_2^2\big)$$

then preserves all pairwise inner products of vectors in $\mathcal{V}$ up to additive error $O(\varepsilon)$, and absorbing the constant we may assume preservation up to additive ε. Choose

$$\varepsilon \;:=\; c_0\,(\rho - \eta)$$

for a sufficiently small universal constant $c_0 > 0$, which gives $r = O\big(\log(M + |\mathcal{F}|)/(\rho-\eta)^2\big)$.

To extract the rank-r router from Φ, define normalized compressed vectors

$$\tilde{u}_i \;:=\; \frac{\Phi e_i}{\|\Phi e_i\|_2}, \qquad \tilde{z}_S \;:=\; \frac{\Phi z_S}{\|\Phi z_S\|_2},$$

and set $u_i = B_u \tilde{u}_i$, $\bar{z}_S = B_z \tilde{z}_S$. The norm bounds are immediate. Since normalization changes the preserved inner products by at most another constant multiple of ε, for every $i \in S$ and $j \notin S$,

$$\langle u_i, \bar{z}_S\rangle - \langle u_j, \bar{z}_S\rangle = B_u B_z(\langle \tilde{u}_i, \tilde{z}_S\rangle - \langle \tilde{u}_j, \tilde{z}_S\rangle) \ge B_u B_z(\rho - C'\varepsilon) \ge \eta B_u B_z$$

for a universal constant $C' > 0$, by the choice of c_0. Hence every $S \in \mathcal{F}$ is realized with margin at least $\eta B_u B_z$.

Finally, we lift to input space: since $h \ge r$, set $R_2 = [I_r \mid 0] \in \mathbb{R}^{r \times h}$ and choose x_S whose first r coordinates are $\bar{z}_S$, so that $R_2 x_S = \bar{z}_S$. This realizes the desired router with the claimed parameters. □

Proof of Theorem 1. Apply Theorem 4 with $B_u = B_z = 1$ and target margin $\eta = c/\sqrt{k}$ for a sufficiently small universal constant $c > 0$. Since $\rho > 1/\sqrt{k}$, we have $(\rho - \eta)^2 = \Omega(1/k)$, and therefore $r = O(k \log(M + |\mathcal{F}|))$. The constructed margin is $\eta = \Omega(1/\sqrt{k})$, as claimed. □

Proof of Corollary 1. For $\mathcal{F} = \binom{[M]}{k}$, Stirling's bounds $(M/k)^k \le |\mathcal{F}| \le (eM/k)^k$ give

$$\log|\mathcal{F}| \;=\; \Theta(k\log(M/k)).$$

Since $M \le |\mathcal{F}|$ for $1 \le k \le M-1$, we also have $\log(M+|\mathcal{F}|) = \Theta(\log|\mathcal{F}|) = \Theta(k\log(M/k))$.

Upper bound. By Theorem 1, any rank $r \ge Ck\log(M+|\mathcal{F}|)$ admits the desired router at margin $\Omega(1/\sqrt{k})$. Substituting $\log(M+|\mathcal{F}|) = \Theta(k\log(M/k))$ yields rank $r = O(k^2\log(M/k))$ as a sufficient condition. This margin is resolvable under $p = \Omega(\log k)$ precision.

Lower bound. At p-bit top-k precision, $\eta = \Omega(2^{-p})$ implies $\log(1+2/\eta) = O(p)$. Theorem 2 then gives

$$r \;\ge\; \frac{\log|\mathcal{F}|}{\log(1+2/\eta)} \;=\; \Omega\!\left(\frac{k\log(M/k)}{p}\right).$$

□

Proposition 1 (restated). Let $x_1,\dots,x_n \sim \mathcal{N}(0,I_h)$ be independent, and suppose $M = 2^r$ ($r = \Theta(\log_2 M)$) with $h \ge r$. For any $1 \le k \le M$, there exists a rank-r top-k router with M experts that selects every expert with probability exactly k/M on each example. Consequently, if N_a is the number of examples routed to expert a, then writing $L := \log\frac{2M}{\delta}$, for any $\delta \in (0,1)$ the following holds with probability at least $1-\delta$:

$$\max_{a\in[M]}\left|N_a - \frac{nk}{M}\right| \le \sqrt{\frac{2nkL}{M}} \;+\; \frac{2}{3}L.$$

Construction. The construction follows the routing step in the Gaussian memorization argument of Jelassi et al. (2025). Let $R_2 \in \mathbb{R}^{r\times h}$ select the first r coordinates, so $z_i := R_2 x_i \sim \mathcal{N}(0,I_r)$ independently for $i \in [n]$. Index the $M = 2^r$ experts by sign vectors $a \in \{\pm1\}^r$ and set the corresponding expert vector to $u_a = a/\sqrt{r}$.

At $k = 1$ the winner is available in closed form: the score $\langle u_a, z\rangle = \frac{1}{\sqrt{r}}\sum_\ell a_\ell z_\ell$ is maximized coordinate-wise by $a_\ell = \mathrm{sign}(z_\ell)$, so the selected expert is the sign pattern of z, which is uniform on $\{\pm1\}^r$. At $k > 1$ no such closed form is available, since the identity of the runner-up experts depends on the magnitudes $|z_\ell|$ and not only on their signs. We therefore argue from a symmetry of the construction rather than from an explicit description of the selected set. The symmetry specializes to the sign-pattern argument at $k = 1$, so the proof below also covers the top-1 case.

Definition 2 (Score vector)**.** *The* score vector *at a projected input $z \in \mathbb{R}^r$ is the M-dimensional vector $s(z) \in \mathbb{R}^M$ that collects the routing scores of all M experts, with entries indexed by the experts themselves:*

$$s(z)_a \;:=\; \langle u_a, z\rangle \;=\; \frac{1}{\sqrt{r}}\sum_{\ell=1}^{r} a_\ell z_\ell, \qquad a \in \{\pm1\}^r. \tag{13}$$

The router selects the k experts whose entries in $s(z)$ are largest.

The key point is that under this construction the standard Gaussian has no special preference for one expert over another in expectation. Each expert is a vertex of the hypercube $\{\pm1\}^r$, and negating a subset of the input coordinates is the same as negating the corresponding entries of the expert index, because the two negations cancel in the inner product $\langle a, z\rangle$. Negating input coordinates therefore only relabels the experts. We capture this intuition using the reflection action.

Definition 3 (Reflection action)**.** *Let $\mathcal{G} := \{\pm1\}^r$, a group under coordinatewise multiplication $\odot$. It acts on $\mathbb{R}^r$ by $\sigma\cdot z := \sigma\odot z$ and on the expert set $\{\pm1\}^r$ by $\pi_\sigma(a) := \sigma\odot a$. For a score vector $s \in \mathbb{R}^M$, write $s\circ\pi_\sigma$ for the relabelled vector with entries $(s\circ\pi_\sigma)_a = s_{\sigma\odot a}$.*

The lemma below states the three properties of this action that the proof uses.

Lemma 2 (Symmetry properties)**.** *For every $\sigma \in \mathcal{G}$:*

(G1) (Invariance of the input distribution.) *If $z \sim \mathcal{N}(0,I_r)$ then $\sigma\cdot z$ has the same distribution as z.*

(G2) (Equivariance of the router.) $s(\sigma \cdot z) = s(z) \circ \pi_\sigma$ *for all* $z \in \mathbb{R}^r$. *Reflecting the input relabels the score vector by* π_σ *and changes nothing else.*

(G3) (Transitivity.) *The map* $\sigma \mapsto \pi_\sigma$ *is a group action of* $\mathcal{G}$ *on the expert set which is simply transitive: for all* $a, b \in \{\pm 1\}^r$ *there is exactly one* $\sigma \in \mathcal{G}$ *with* $\pi_\sigma(a) = b$, *namely* $\sigma = a \odot b$. *Moreover* $\pi_\sigma^{-1} = \pi_\sigma$.

Proof. (G1) The coordinates of z are independent and symmetric about 0, so negating any subset of them preserves the joint distribution.

(G2) By (13), $s(\sigma \cdot z)_a = \frac{1}{\sqrt{r}} \sum_\ell a_\ell \sigma_\ell z_\ell = \frac{1}{\sqrt{r}} \sum_\ell (\sigma \odot a)_\ell z_\ell = s(z)_{\sigma \odot a} = (s(z) \circ \pi_\sigma)_a$.

(G3) Since $\pi_\sigma \pi_\tau = \pi_{\sigma \odot \tau}$ and $\pi_{\mathbf{1}} = \mathrm{id}$, the map is a group action. If $\pi_\sigma(a) = b$ then $\sigma = \sigma \odot a \odot a = b \odot a$, which is unique, and $\pi_{b \odot a}(a) = b \odot a \odot a = b$ confirms that it works. Finally $\pi_\sigma(\pi_\sigma(a)) = \sigma \odot \sigma \odot a = a$. □

Top-k selection reads the score vector only through the ordering of its entries, never through which expert happens to carry which entry. Relabelling the experts therefore relabels the selected set in the same way.

Lemma 3 (Top-k selection is equivariant and almost surely well defined)**.** *Let* $s \in \mathbb{R}^M$ *be a score vector whose* M *entries are pairwise distinct, and let* $\mathrm{TopK}(s)$ *be the set of the* k *experts whose entries in* s *are largest. Relabelling does not create equal entries, so* $\mathrm{TopK}(s \circ \pi_\sigma)$ *is defined for every* $\sigma \in \mathcal{G}$, *and the selection is* $\mathcal{G}$-equivariant*:*

$$\mathrm{TopK}(s \circ \pi_\sigma) \;=\; \pi_\sigma\big(\mathrm{TopK}(s)\big).$$

Moreover, for $z \sim \mathcal{N}(0, I_r)$ *the entries of* $s(z)$ *are almost surely pairwise distinct, so* $\mathrm{TopK}(s(z))$ *is almost surely well defined.*

Proof. Relabelling the entries of a vector relabels its order statistics. The entry of $s \circ \pi$ at index a is the entry of s at index $\pi(a)$, so the indices carrying the k largest entries of $s \circ \pi$ are exactly π^{-1} applied to those for s. With $\pi = \pi_\sigma$ and $\pi_\sigma^{-1} = \pi_\sigma$ from (G3), this is the stated equivariance.

For distinctness, if $a \neq b$ then $u_a - u_b = (a - b)/\sqrt{r} \neq 0$, so $\{z : s(z)_a = s(z)_b\}$ is a hyperplane through the origin and therefore has Lebesgue measure zero. A union over the finitely many pairs is still null. □

Lemma 4 (Uniform marginals)**.** *Let* $z \sim \mathcal{N}(0, I_r)$ *and let* $1 \leq k \leq M$. *Then*

$$\Pr\big[\, a \in \mathrm{TopK}(s(z)) \,\big] \;=\; \frac{k}{M} \qquad \textit{for every expert } a \in \{\pm 1\}^r.$$

Proof. By Lemma 3 we may work on the almost sure event that the entries of $s(z)$ are pairwise distinct. Fix $\sigma \in \mathcal{G}$ and an expert a. Using (G1) in the first step, (G2) in the second, and equivariance in the third,

$$\begin{aligned}\Pr[\, a \in \mathrm{TopK}(s(z)) \,] &= \Pr[\, a \in \mathrm{TopK}(s(\sigma \cdot z)) \,] = \Pr[\, a \in \mathrm{TopK}(s(z) \circ \pi_\sigma) \,] \\ &= \Pr[\, a \in \pi_\sigma(\mathrm{TopK}(s(z))) \,] = \Pr[\, \pi_\sigma(a) \in \mathrm{TopK}(s(z)) \,],\end{aligned}$$

where the last step uses $\pi_\sigma^{-1} = \pi_\sigma$ from (G3). By transitivity (G3), for any two experts a and b there is a σ with $\pi_\sigma(a) = b$, so the function $a \mapsto \Pr[a \in \mathrm{TopK}(s(z))]$ takes the same value at every expert. Since $|\mathrm{TopK}(s(z))| = k$ always, we have $\sum_{a=1}^M \mathbf{1}\{a \in \mathrm{TopK}(s(z))\} = k$, and taking expectations gives $\sum_{a=1}^M \Pr[a \in \mathrm{TopK}(s(z))] = k$. A constant function on M points summing to k equals k/M at each point. □

Proof of Proposition 1. Lemma 4 gives selection probability k/M for every expert. Fix an expert a. The indicators $\mathbf{1}\{a \text{ selected at } x_i\}$, $i \in [n]$, are independent because the x_i are independent, and each is Bernoulli(k/M). Hence $N_a \sim \mathrm{Binomial}(n, k/M)$ with mean nk/M and variance $v := n\frac{k}{M}(1 - \frac{k}{M})$.

Bernstein's inequality with increments bounded by 1 gives

$$\Pr\left[\left|N_a - \frac{nk}{M}\right| \geq t\right] \;\leq\; 2\exp\!\left(-\frac{t^2}{2v + \frac{2}{3}t}\right).$$

Setting the right-hand side to δ/M and solving $t^2 - \frac{2}{3}Lt - 2vL = 0$ yields $t = \frac{L}{3} + \sqrt{\frac{L^2}{9} + 2vL} \leq \sqrt{2vL} + \frac{2}{3}L$, using $\sqrt{x+y} \leq \sqrt{x} + \sqrt{y}$. Bounding $v = n\frac{k}{M}(1 - \frac{k}{M}) \leq \frac{nk}{M}$ and taking a union bound over the M experts gives the claimed bound.

This proposition isolates only the routing step. The dependence on n enters through the number of examples assigned to each expert, approximately nk/M. The router rank only needs to distinguish the M expert labels, which requires $r = \log_2 M$ in this construction, independently of k. The expert MLPs then memorize their assigned examples, as in the memorization upper bound of Jelassi et al. (2025). □