EQUIMOE: A MIXTURE-OF-EXPERTS ARCHITECTURE WITH LOAD BALANCE BY CONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse mixture-of-experts (MoE) routing can concentrate assignments on a few expert groups, creating uneven work across expert-parallel ranks. EquiMoE fixes each group's assignment budget in the forward operator: every token enters every learned subspace branch, where an independent router selects a fixed number of experts from a disjoint bank. This gives exactly equal group assignment counts for any input and router parameters, while expert loads within each group may remain unequal. With one branch per rank, EquiMoE sends one reduced representation per token to each rank before local routing and returns one aggregated result per branch. The nominal per-call all-to-all payload is 4.19 MB, versus 33.55 MB for an assignment-expanded Standard MoE baseline: an 87.5% reduction under this dispatch convention. In four C4 runs of 20,000 steps (2.62 billion processed token positions), EquiMoE has equal rank assignment counts, constant measured payload, and training-loss trajectories close to the baselines. A LatentMoE-style scale-up comparison shows that reduced expert width with assignment-expanded dispatch does not yield the same payload saving. The guarantee concerns group assignments, not equal per-expert load. While EquiMoE's per-call communication time is approximately 26% lower than the baseline, this does not translate to proportional end-to-end speedup due to fixed all-to-all latency.

1 INTRODUCTION

Sparse mixture-of-experts (MoE) models separate total parameter capacity from the computation used for each token: a router activates only a small subset of available experts (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Riquelme et al., 2021; Jiang et al., 2024). This flexibility creates a familiar difficulty. Tokens can favor the same experts, concentrating assignments on a few expert groups while others receive little work. When groups reside on different devices, the resulting imbalance causes the heaviest rank to determine step completion time while others idle at the all-to-all barrier, wasting compute resources (Figure 1(a)). The assignment pattern is therefore both a modeling decision and a determinant of distributed workload.

A common response is to encourage balanced routing through auxiliary objectives or load-dependent adjustments (Fedus et al., 2022; DeepSeek-AI, 2024; Dai et al., 2024; Qiu et al., 2025). An alternative is to build the desired workload constraint into the model. This leads to the question we study: *can an MoE layer learn input-dependent expert specialization while keeping the amount of computation assigned to each expert group fixed?* The relevant distinction is between choosing *which experts* process a token and choosing *how much work* each group receives. These decisions need not be coupled.

We introduce **EquiMoE**, a mixture-of-experts architecture organized as a sum of parallel subspace MoEs. Each branch learns a projection of the full token representation, routes that representation among its own experts, and maps the result back to the model width. All branches participate in every token, but only a fixed subset of experts within each branch is active. With G branches, N total experts, and k total activations per token, each branch contains N/G experts and selects exactly k/G of them. Thus every branch receives the same assignment count for any batch and any router parameters. Equal group workload follows from the forward operator itself.

This structure leaves substantial conditional computation inside each branch. The projection determines which features a branch exposes to its router and experts; the local router chooses how to

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.

process those features; and the output projection integrates the result with the other branches. The projections are learned from the entire input, avoiding fixed coordinate slices. No orthogonality or diversity constraint is imposed; complementary specialization may emerge from training.

The key distinction from prior MoE designs is that EquiMoE combines branch-specific learned representations with *disjoint expert banks and independent subspace routers*. Both expert selection and expert computation take place inside each learned subspace, and a branch cannot redirect its assignments to another branch's experts. Prior work has explored related components—equal per-group activation quotas (Pangu Team, 2025) and projected multi-head expert representations (Huang et al., 2024)—but not their combination with disjoint expert banks. We discuss these connections in Section 2.

The architecture also determines a natural distributed execution. Assigning one branch to each device makes device assignment totals equal by construction. Each token sends one reduced representation to every branch before expert selection; selected expert outputs are combined locally before returning. Communication therefore does not replicate a full-width token for each selected expert. This execution is a consequence of the model definition, and we analyze it separately from the architecture.

Our contributions are: (1) an explicit MoE forward operator combining learned subspace branches, local expert selection, and equal activation budgets; (2) an exact group-level assignment-balance guarantee with parameter, compute, and communication accounting; and (3) an implementation in MindSpeed-LLM[1] and a 20,000-step C4 comparison against Standard MoE and LatentMoE configurations. Under the evaluated assignment-expanded dispatch convention, the implementation reduces logical per-call activation payload by 87.5% and makes that payload independent of router decisions; the experiments assess training loss at one seed and scale.

2 Related Work

Sparse conditional computation. Sparse MoEs scale conditional computation through learned expert routing (Bengio & LeCun, 2007; Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). EquiMoE organizes this computation into parallel subspace mixtures with fixed group participation.

Balancing through routing. Auxiliary losses encourage distributed expert use, while bias-based balancing changes routing preferences in response to load (Fedus et al., 2022; DeepSeek-AI, 2024). Expert Choice instead allocates a token budget to each expert (Zhou et al., 2022; Guo et al., 2025; Cheng et al., 2025). These methods constrain different assignment marginals. EquiMoE fixes the number of selections per token *and per group*, while retaining learned token-choice routing inside each group; the evaluated model additionally keeps the standard auxiliary and z-loss regularizers.

Equal-quota grouped experts. MoGE in Pangu Pro MoE partitions experts into groups and selects a fixed number from every group, already providing equal group assignment totals (Pangu Team, 2025). Its routing scores are computed from the full hidden representation with global normalization. EquiMoE instead assigns a learned reduced representation and an independent router to each disjoint expert bank, then reconstructs and adds the branch outputs. The distinction lies in the representation and computation attached to the quota.

Multi-head and latent MoEs. MH-MoE projects a token, splits it into sub-token representations, routes those through a shared expert pool, and merges the outputs (Huang et al., 2024). EquiMoE also uses learned input/output projections, but permanently associates each representation branch with a separate router and expert bank. This association enforces equal branch totals even when local preferences differ. LatentMoE reduces the width of expert computation through latent projections (Elango et al., 2026); reduced dimensionality alone does not specify EquiMoE's branch quotas. Our LatentMoE comparison probes reduced-width expert execution with conventional dispatch, but also changes the routing structure and capacity budget, so it does not isolate a single architectural component.

[1] https://gitcode.com/Ascend/MindSpeed-LLM

Efficient expert execution. Sparse and grouped execution systems such as MegaBlocks address variable expert workloads (Gale et al., 2023). Such kernels are complementary: EquiMoE fixes total group work, but does not make every local expert matrix the same size. Efficient handling of local skew remains relevant.

3 The EquiMoE Architecture

3.1 From a global expert pool to parallel subspace mixtures

For a row-vector token representation $x \in \mathbb{R}^d$, a standard token-choice MoE has the form

$$f_{\mathrm{MoE}}(x) = \sum_{i \in \mathrm{TopK}_k(xW)} a_i(x) F_i(x), \qquad W \in \mathbb{R}^{d \times N}. \tag{1}$$

All experts compete in one selection problem. If the experts are partitioned into groups, Equation 1 does not constrain how many selected experts belong to each group. Figure 1(a) illustrates this: the global router selects k experts from all N, then dispatches via variable-length all-to-all, so different ranks receive different token counts.

EquiMoE replaces this global mixture with G parallel branches. Define

$$r = d/G, \qquad n = N/G, \qquad q = k/G, \quad G \mid d,\ G \mid N,\ G \mid k, \quad 1 \le q \le n. \tag{2}$$

Here G is an architectural hyperparameter that is in principle independent of device placement; our implementation sets $E = G$ so that each EP rank owns one branch, though other mappings are possible (Section 5). Branch g owns a down-projection $D_g \in \mathbb{R}^{d \times r}$, a router $W_g \in \mathbb{R}^{r \times n}$, a bank of n experts $F_{g,i} : \mathbb{R}^r \to \mathbb{R}^r$, and an up-projection $A_g \in \mathbb{R}^{r \times d}$. Expert banks and routers are distinct across branches. Figure 1(b) illustrates the architecture.

3.2 Learned representations and independent local routing

Every branch observes a learned representation of the entire token:

$$z_g = xD_g, \qquad \ell_g = z_g W_g, \qquad \mathcal{S}_g(x) = \mathrm{TopK}_q(\ell_g). \tag{3}$$

The D_g matrices can be concatenated into a single $d \times d$ projection whose output is partitioned into G representations. This dense implementation is mathematically equivalent to separate branch projections, yet the branch representations remain learned functions of the full input rather than fixed input-coordinate slices.

The local router selects q experts using only z_g. For the normalization used in our experiment, selected logits define the weights

$$a_{g,i}(x) = \frac{\exp(\ell_{g,i})}{\sum_{j \in \mathcal{S}_g(x)} \exp(\ell_{g,j})}, \qquad u_g(x) = \sum_{i \in \mathcal{S}_g(x)} a_{g,i}(x) F_{g,i}(z_g). \tag{4}$$

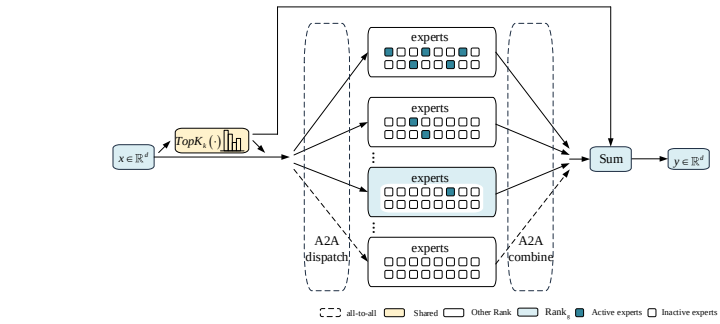
Each expert is a bias-free SwiGLU network with input/output width r and intermediate width h. The choice of h controls capacity separately from the subspace width. In particular, reducing r does not require reducing h.

3.3 Reconstruction and the complete forward operator

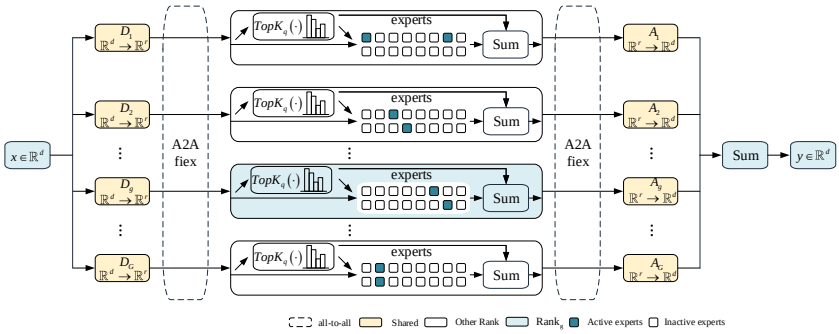
The branch outputs are projected back to the model space and added:

$$\boxed{f_{\mathrm{EquiMoE}}(x) = \sum_{g=1}^{G} \left[\sum_{i \in \mathcal{S}_g(x)} a_{g,i}(x) F_{g,i}(xD_g) \right] A_g.} \tag{5}$$

The A_g matrices are learned independently of the D_g matrices; they are not constrained to be inverses. Concatenating $u_1, \ldots, u_G$ and applying one dense output projection implements the same sum. The surrounding transformer supplies normalization and the residual connection.



(a) Standard MoE: route then communicate



(b) EquiMoE: communicate then route locally

Figure 1: **Standard MoE vs EquiMoE.** (a) The global router on $x \in \mathbb{R}^d$ selects k experts, then dispatches via variable-length all-to-all. Different ranks receive different token counts (L_g may differ), causing inter-rank load imbalance: the slowest rank determines step completion time, while others idle at the all-to-all barrier. In our measurements (Section 6.3), the worst-case max/min ratio reaches 1.22 for Standard MoE during early training, meaning the heaviest rank receives 22% more assignments than the lightest. (b) x is projected into G learned sub-representations of width $r = d/G$, sent one per branch via *fixed-volume* all-to-all, then routed locally. All ranks receive identical token counts ($L_g = Mq$ for all g), eliminating inter-rank imbalance by construction. The projection weights D_g and A_g are shared across ranks and synchronized via all-reduce.

Equation 5 defines the model without mentioning devices, collectives, or token dispatch. It preserves a total of $Gq = k$ expert activations per token while allocating them equally across branches. It also changes the hypothesis class: experts process learned width-r representations, and choices in one branch cannot consume another branch's quota. Each branch normalizes its selected weights to sum to one (Equation 4), after which the branch outputs are summed (Equation 5). Their resulting scale depends on the learned projections and branch-output correlations; it does not follow from the count G alone.

4 Load Balance as an Architectural Invariant

4.1 Exact balance across expert groups

Consider any batch of M token positions. Let $c_{g,i}$ count assignments to expert i in branch g, and let $L_g = \sum_i c_{g,i}$ denote the branch total. Define the normalized group imbalance

$$\rho_{\text{group}} = \frac{\max_g L_g}{G^{-1} \sum_g L_g}. \tag{6}$$

A value of one denotes equal group totals.

Proposition 1 (Group-level balance by construction)**.** *If each token selects exactly $q = k/G$ distinct experts in every branch, and no assignments are dropped or padded, then for all inputs and router*

parameters,

$$L_g = Mq = Mk/G \quad \textit{for every } g, \qquad \rho_{\text{group}} = 1. \tag{7}$$

Proof. For each token t and branch g, $\sum_{i=1}^{n} \mathbf{1}\{i \in \mathcal{S}_g(x_t)\} = q$. Summing this identity over the M tokens gives $L_g = Mq$. The expression is independent of g, so the maximum branch load equals the mean. □

The guarantee holds before training and throughout optimization, including under arbitrary changes in router preferences or token composition. It is a counting property of equal quotas, also exploited by MoGE (Pangu Team, 2025); learned subspaces determine how EquiMoE performs computation under that constraint. For identically shaped experts, the guarantee also fixes the leading expert matrix-multiplication work per branch.

4.2 Group balance and expert balance are different

The proposition does not require uniform expert usage within a branch. Since a token selects an expert at most once, $c_{g,i} \le M$, and therefore

$$\rho_{\text{expert},g} = \frac{\max_i c_{g,i}}{Mq/n} \le \frac{n}{q} = \frac{N}{k}. \tag{8}$$

Equality is possible if every token selects the same q experts. Thus EquiMoE eliminates *inter-group assignment imbalance*, while local expert imbalance remains a learning and execution concern. In particular, equal group totals do not prove equal wall-clock time: local histogram shapes and kernel efficiency can differ. Throughout this paper, structural load balance refers to the group-level guarantee in Proposition 1.

4.3 The modeling cost of equal participation

The balance guarantee comes at the cost of restricting the routing space. For a fixed expert partition, unrestricted top-k routing can choose from $\binom{N}{k}$ expert subsets. EquiMoE permits $\binom{n}{q}^G$ subsets, each satisfying the same branch quotas. This restriction prevents a token from placing all of its computation in one preferred group. Learned projections provide a way for each branch to develop useful features under mandatory participation, but do not recover unrestricted global routing.

The number of branches also controls the subspace width and the size of the local selection problem. Increasing G reduces r, n, and q at fixed d, N, and k. Since $q \ge 1$, $G \le k$; the present construction does not offer multiple branches for a globally top-1 model. These are architectural choices tied to the capacity budget; changing them after training would change parameter shapes and invalidate existing checkpoints.

5 Expert-Parallel Realization and Cost

5.1 Communicate representations before selecting experts

We implement EquiMoE in MindSpeed-LLM with one branch on each of E expert-parallel ranks, setting $E = G$ so that each rank owns exactly one branch. This one-branch-per-rank mapping is an implementation choice, not an architectural constraint: G defines the model structure (branch count, subspace width, expert partition), while E defines the device placement; they need not coincide in general. Each source rank s holds T token positions $X_s \in \mathbb{R}^{T \times d}$, giving $M = ET$ positions in an EP group. Shared projection weights are replicated across sources; branch routers and experts reside with their branch owners.

A source computes $Z_{s,g} = X_s D_g$ for every branch and stacks the results as $[E, T, r]$. The first all-to-all sends each block to its branch owner. Owner g receives

$$R_g = \text{concat}_{s=1}^{E} Z_{s,g} \in \mathbb{R}^{ET \times r}. \tag{9}$$

It then applies its local router and experts, combining selected outputs into one width-r row per received token. A second all-to-all returns those rows to their source, where the up-projections produce Equation 5. Algorithm 1 specifies the forward pass.

Each owner processes *all* ET token positions in its own subspace. Its assignment count is $Mq = Tk$, exactly equal across ranks. Both collectives use fixed-size splits of T rows per peer; expert choices affect only local dispatch. The current implementation assumes equal source token counts, dropless routing, homogeneous expert shapes, and tensor and context parallel sizes of one in the evaluated configuration.

Algorithm 1 EquiMoE forward pass on rank s

Require: $X_s : [T, d]$; $E{=}G$ (one branch per rank), $r = d/G$, $n = N/G$, $q = k/G$; equal T across ranks

$Z_s \leftarrow \mathrm{stack}_{g=1}^{G}(X_s D_g)$ ▷ $[E, T, r]$

$R_s \leftarrow \mathrm{AllToAll}(\mathrm{reshape}(Z_s, [ET, r]), [T]^E)$ ▷ Receive every source's tokens in branch s: $[ET, r]$

$(P, I) \leftarrow \mathrm{LocalRouter}(R_s, W_s, q)$ ▷ weights and indices: $[ET, q]$

$(H, c, \mathcal{C}) \leftarrow \mathrm{LocalDispatch}(R_s, P, I)$ ▷ $H : [ETq, r]$

$O \leftarrow \mathrm{LocalExperts}(H, c)$ ▷ c is the local expert histogram

$U_s \leftarrow \mathrm{LocalCombine}(O, \mathcal{C})$ ▷ weighted sum: $[ET, r]$

$V_s \leftarrow \mathrm{AllToAll}(U_s, [T]^E)$

$V_s \leftarrow \mathrm{reshape}(V_s, [E, T, r])$

$Y_s \leftarrow \sum_{g=1}^{G} V_s[g] A_g$; cache $X_s, R_s, H, P, I, \mathcal{C}, O, V_s$

return $Y_s : [T, d]$

The backward pass reverses the two exchanges and differentiates the local mixture; the full algorithm, local derivatives, and loss-scaling convention are given in Appendix D.

5.2 Activation payload and computation

Each forward exchange sends $ETr = Td$ logical activation elements per rank, of which $(E-1)Tr$ are off-rank. This payload is independent of k: a representation is sent to a branch once, and expert outputs are combined there before returning. A conventional assignment-expanded dispatcher sends Tkd elements at dispatch, giving a factor-k logical payload ratio. A dispatcher that deduplicates tokens per destination can have a smaller baseline volume; this distinction is made explicit in Appendix E. Payload ratios do not directly predict latency ratios. Table 3 (Appendix E) separates the analytic costs.

The two projection banks add $2d^2$ unique weights and $4Td^2$ forward FLOPs. Reduced-width SwiGLU experts contain $3rh$ weights each. Local routing produces $Mn = TN$ logits per rank, the same number as global routing on source-local tokens, but its matrix multiplication uses the smaller input width. At fixed h, EquiMoE reduces expert capacity along with expert computation; widening h can recover capacity, while the projection cost remains. A fair budget comparison must account for the complete branch, including projection cost.

6 Experiments

6.1 Setup and model budgets

We evaluate four MoE configurations as replacement layers in an OLMoE-inspired language model (Muennighoff et al., 2024), using MindSpeed-LLM and C4 (Raffel et al., 2020). All configurations use 16 transformer layers, hidden width 2048, 8 Ascend 910C NPUs with EP size 4 and DP size 2, BF16, and grouped expert GEMMs. Table 2 (Appendix E) summarizes the full configuration.

The four configurations are:

- **Standard MoE** (baseline): 64 experts, top-8 routing, expert intermediate width 1024, full hidden-state input.
- **EquiMoE**: 64 experts, top-8 routing, four branches with 16 experts each, width-512 representations, two selections per branch, expert intermediate width 3760.

- **EquiMoE ScaleUp**: LatentMoE-style scale-up with α=4. 256 experts, top-32 routing, four branches with 64 experts each, width-512 representations, expert intermediate width 848. Activated parameters per token are matched to the baseline (-0.5%).
- **LatentMoE**: Dimension reduction without fixed-volume dispatch. 256 experts, top-32 routing, width-512 latent projections, expert intermediate width 976. Uses standard variable-length all-to-all communication.

Each run uses a single seed and completes 20,000 updates with global batch size 128 and sequence length 1024: 2.62 billion processed token positions. All runs use the same 50 preprocessed C4 shards with a 90/10 train/validation split (no test set), an OLMoE tokenizer, and are trained on 8 Ascend 910C NPUs. The configuration is not a reproduction of the released OLMoE model.

Capacity and compute. EquiMoE widens its expert intermediate dimension to compensate for the reduced input width, achieving per-rank parameter storage close to the baseline (EquiMoE ≈1.87 B, Standard MoE ≈1.93 B under EP = 4, DP = 2 with a distributed optimizer). EquiMoE ScaleUp and LatentMoE both use α=4 scale-up: 256 experts with top-32 routing, where the increased expert count and activation diversity compensate for the dimension reduction, following the methodology of Elango et al. (2026). The activated parameters per token for EquiMoE ScaleUp are within 0.5% of the baseline. LatentMoE uses a different intermediate width (976 vs 848 for ScaleUp); the per-layer MoE parameter counts are not identical across configurations (Standard ≈402.8 M, ScaleUp ≈342.0 M, LatentMoE ≈386.4 M, each including projections and routers), and the full-model totals differ accordingly. The 976 and 848 widths are chosen so that ScaleUp and LatentMoE are close in per-rank storage and activated parameters per token, but they are not exactly matched. Loss comparisons between them should therefore be interpreted qualitatively.

Optimization. All runs use Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.1, gradient clipping at 1.0, and a cosine learning rate schedule from 2×10^{-4} to 2×10^{-5} without warmup. All configurations retain auxiliary balancing and router z-loss. For EquiMoE, the G branch-level auxiliary losses are summed (not averaged); the effective regularization strength per token thus differs from a standard MoE with a single global router at the same coefficient (see Appendix A).

6.2 Language-model learning

Figure 2 shows training loss for EquiMoE versus Standard MoE, sampled every 200 steps. The two trajectories are closely matched throughout training, with final losses within 0.02 nats of each other. Figure 4 (Appendix B) compares EquiMoE ScaleUp with LatentMoE under the same α=4 scale-up configuration. Both use 256 experts and top-32 routing; their trajectories are also close in these runs. Because their intermediate widths and routing structures differ, this comparison cannot attribute any loss difference to dispatch alone.

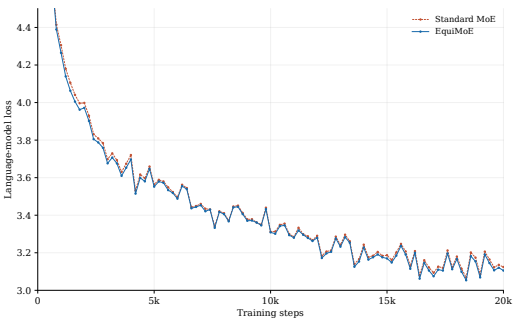


Figure 2: **Training loss: EquiMoE vs Standard MoE.** Loss sampled every 200 steps in one run per configuration.

6.3 Load balance

Rank-level balance. The architectural guarantee of Proposition 1 ensures that every expert-parallel rank receives exactly the same number of token assignments, regardless of routing outcomes. We verify this empirically by computing per-rank token totals from the router output: for each rank, we sum the per-expert token counts over the experts assigned to that rank's EP partition.

Figure 6 (Appendix B) shows the resulting rank-level load heatmaps for all four configurations. EquiMoE and EquiMoE ScaleUp exhibit perfectly uniform color (all ranks receive identical token counts, CV = 0.00%), confirming the architectural guarantee. Standard MoE and LatentMoE both

show variation across ranks (CV = 1.77% and 1.27%, respectively); these values are small in absolute terms, but the key distinction is that EquiMoE eliminates the variance entirely, while variable-length dispatch cannot guarantee zero imbalance for any routing configuration.

Expert-level load distribution. Figure 7 (Appendix B) shows expert-level load heatmaps, visualizing how tokens are distributed among individual experts. Each row is a layer; each column is an expert; color denotes normalized load.

6.4 Communication efficiency

The all-to-all communication volume per dispatch call is determined by the number of token representations sent and their width. For Standard MoE, each token is replicated k_0 times at full width d, giving a payload of $T \times k_0 \times d \times 2$ bytes (BF16). For EquiMoE, each token is split into G sub-representations of width $r = d/G$, one sent to each branch, giving $G \times T \times r \times 2 = T \times d \times 2$ bytes—independent of k_0. The reduction factor is therefore k_0. For LatentMoE with scale-up factor α, each token is replicated $k = \alpha k_0$ times at reduced width $r = d/\alpha$, giving $T \times k \times r \times 2 = T \times \alpha k_0 \times (d/\alpha) \times 2 = T \times k_0 \times d \times 2$ bytes—the same as the baseline. These comparisons assume assignment-expanded dispatch; a dispatcher that deduplicates tokens per destination can have a smaller baseline volume (see Appendix E).

Table 1 compares nominal payloads and logged per-call values. EquiMoE and EquiMoE ScaleUp both have a 4.19 MB nominal payload—an 87.5% reduction from the baseline's 33.55 MB under assignment-expanded dispatch. LatentMoE's nominal payload is also 33.55 MB: its fourfold reduction in representation width is offset by fourfold more selected experts. This comparison concerns the specified model configurations and dispatchers, not dimension reduction in every possible MoE implementation.

Table 1: **All-to-all comm. volume per call.** T=1024, d=2048, k_0=8, G=4, α=4, r=512, k=32.

Config	Theory	MB	Measured	CV_t
Standard	$Tk_0d\times2$	33.55	33.74±0.58	0.34%
EquiMoE	$GTr\times2$	4.19	4.19±0.00	0.00%
ScaleUp	$GTr\times2$	4.19	4.19±0.00	0.00%
LatentMoE	$Tkr\times2$	33.55	33.56±0.62	0.31%

EquiMoE's communication volume is *constant* in the measured steps (range = 0.00 MB), because the fixed-volume payload depends only on tensor shapes, not routing outcomes. Standard MoE and LatentMoE show ranges of 1.16 MB and 1.25 MB in the logged per-call metric. The logged values record the number of bytes sent per all-to-all call (dispatch direction); return volumes are symmetric under balanced routing but may differ under variable-length dispatch. Communication statistics cover steps 1–1000; the fixed-size EquiMoE tensor shapes imply the same logical payload at later steps when the batch shape remains unchanged. We do not infer network latency or end-to-end training speed from these byte counts. The 87.5% reduction applies to assignment-expanded dispatch; a dispatcher that deduplicates tokens per destination would have a smaller baseline volume, narrowing the gap (see Appendix E).

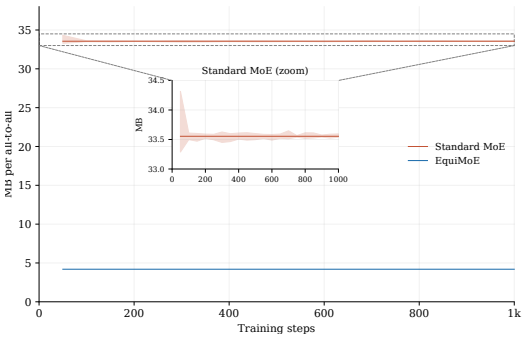


Figure 3: **Communication volume: EquiMoE vs Standard MoE.** Shaded bands show per-rank min/max across 8 ranks. EquiMoE has a fixed 4.19 MB payload with zero band; Standard MoE fluctuates around 33.55 MB with inter-rank variance. Inset zooms into 33–34.5 MB.

Figure 3 shows the communication volume over training steps. EquiMoE's volume is a flat line at 4.19 MB; Standard MoE fluctuates around 33.55 MB. The same pattern holds for the scale-up comparison (Figure 5, Appendix B): EquiMoE ScaleUp is flat at 4.19 MB, while LatentMoE fluctuates at 33.55 MB. Figure 8 (Appendix B) shows the corresponding per-call communication *time*. EquiMoE's time stabilizes at approximately 0.49 ms after kernel warmup, while Standard MoE fluc-

tuates around 0.67 ms and LatentMoE around 0.62 ms. The time reduction (approximately 26%) is smaller than the volume reduction (87.5%) because all-to-all has a fixed per-call latency that does not scale with payload size. We report these times as supplementary evidence; the primary claim is the payload reduction and its zero variance, not wall-clock speedup.

6.5 Scope of the evidence

The experiments evaluate one seed per configuration at a single scale; no statistical significance claim is made. They support three conclusions:

1. **EquiMoE achieves perfect rank-level balance and reduces communication by 87.5%.** The rank-level CV is exactly zero (Figure 6), and the all-to-all volume is constant at 4.19 MB (Table 1).
2. **The tested reduced-width model with assignment-expanded dispatch retains the baseline payload.** LatentMoE's nominal communication volume (33.55 MB) equals that of Standard MoE because its fourfold increase in selected experts offsets its fourfold decrease in representation width. EquiMoE sends one representation per branch before routing and has a 4.19 MB nominal payload.
3. **Scale-up is compatible with fixed-volume dispatch.** EquiMoE ScaleUp (256 experts, top-32) maintains zero rank-level CV and the same 87.5% communication reduction, while LatentMoE at the same scale has 1.27% rank-level CV and no communication savings.

The experiments do not establish statistical significance across seeds, downstream-task performance, or superiority over other MoE architectures. The exact group-count guarantee is analytic (Proposition 1) and does not depend on these empirical results.

7 Limitations and Conclusion

EquiMoE trades unrestricted global selection for predictable group participation. Mandatory participation may allocate computation to a less useful branch, and learned subspaces are not guaranteed to specialize or remain distinct. Group balance does not prevent local expert collapse or ensure identical device latency. Moreover, G determines parameter shapes and cannot be changed by simply repartitioning experts. The current one-branch-per-rank implementation couples deployment size to G and assumes equal source token counts; alternative placements and irregular inference batches require additional implementation work.

Within these limits, EquiMoE shows that an MoE can learn conditional expert computation while giving every expert group an exactly equal assignment budget. Learned subspace branches provide the representation, fixed local quotas supply the invariant, and branch-local aggregation enables a compact distributed execution. In the tested C4 setting, its assignment-expanded per-call payload is 87.5% below the Standard MoE baseline and has zero measured variation, while training losses remain close. Matched-capacity ablations, validation loss, and end-to-end timing are needed to establish when this architecture improves model quality or training efficiency.

Reproducibility Statement

Sections 3–5 define the forward operator, assumptions, proof, and costs. The appendices specify distributed execution, implementation details, and the logged training configuration. Empirical measurements come from training logs of four configurations: Standard MoE, EquiMoE, EquiMoE ScaleUp, and LatentMoE. Exact environment and training-code provenance remain incomplete, as detailed in Appendix A.3.

AI Use Statement

During manuscript preparation, OpenAI Codex assisted with literature review, manuscript polishing, and automated experiment scripts; AI-assisted analysis did not generate new training measurements. The authors are responsible for verifying the final claims, references, and artifacts.

REFERENCES

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

Anzhe Cheng, Shukai Duan, Shixuan Li, Chenzhong Yin, Mingxi Cheng, Heng Ping, Tamoghna Chattopadhyay, Sophia I Thomapos, Shahin Nazarian, Paul Thompson, and Paul Bogdan. ER-MoE: Eigen-reparameterized mixture-of-experts for stable routing and interpretable specialization. *arXiv preprint arXiv:2511.10971*, 2025.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. J. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Zhao, Y. K. Liu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, Nave Assaf, Maximilian Golub, Joey Guman, Tiyasa Mitra, Ritchie Zhao, Ritika Borkar, Ran Zilberstein, Mostofa Patwary, Mohammad Shoeybi, and Bita Rouhani. LatentMoE: Toward optimal accuracy per FLOP and parameter in mixture of experts. *arXiv preprint arXiv:2601.18089*, 2026.

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

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. In *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

Hongcan Guo, Haolang Lu, Guoshun Nan, Bolun Chu, Jialin Zhuang, Yuan Yang, Wenhao Che, Xinye Cao, Sicong Leng, Qimei Cui, and Xudong Jiang. Advancing expert specialization for better MoE. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Shaohan Huang, Xun Wu, Shuming Ma, and Furu Wei. MH-MoE: Multi-head mixture-of-experts. *arXiv preprint arXiv:2411.16205*, 2024.

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

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

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Pangu Team. Pangu Pro MoE: Mixture of grouped experts for efficient sparsity. *arXiv preprint arXiv:2505.21411*, 2025.

Zihan Qiu, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Xu, et al. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models. *arXiv preprint arXiv:2501.11873*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, Andreas Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

A Implementation and Training Configuration

A.1 Model components and tensor layouts

The implementation class is `EquiMoEEPLayer`, selected in the MoE layer specification by `use_eqmoe`. The constructor copies the router configuration, setting its input width to r, expert count to n, and number of selections to q. The expert configuration changes its input/output hidden width to r while retaining the intermediate width h.

`FastGroupedProjection` concatenates the down-projection weights for a dense source-local multiplication. The resulting destination-major $[E, T, r]$ tensor is flattened to $[ET, r]$ for all-to-all. A destination receives the same shape, now ordered by source. Local dispatch creates token–expert pairs, sorts by expert index, and counts assignments with `bincount`. Local combination restores pair order, multiplies by gating weights, and uses `index_add_` to sum contributions per token. Following the return collective, `forward_up` implements the concatenation-and-projection equivalent of Equation 5.

The code asserts that N and k are divisible by E, but uses integer division for the hidden width. Equation 2 additionally requires divisibility of d. The permutation expects valid expert indices; dropping sentinels and capacity padding are outside the analyzed configuration. The CPU verification script checks extracted projection and dispatch routines against the direct mathematical operator, including forward and gradient agreement under simulated exchanges. These checks do not substitute for a full distributed integration test.

A.2 Local regularization and initialization

For a branch processing M token positions, let

$$f_{g,i} = \frac{1}{Mq} \sum_{t=1}^{M} \mathbf{1}\{i \in \mathcal{S}_g(x_t)\}, \qquad p_{g,i} = \frac{1}{M} \sum_{t=1}^{M} [\mathrm{softmax}(\ell_g(x_t))]_i. \tag{10}$$

A conventional local auxiliary loss is $\mathcal{L}_{\mathrm{aux},g} = \alpha n \sum_i f_{g,i} p_{g,i}$. Dividing assignment frequencies by q makes them sum to one. The G branch-level auxiliary losses are summed to produce the total auxiliary contribution; with the same nominal coefficient α, the effective regularization strength per token differs from a standard MoE with a single global router, since candidate sets, token populations, and distributed gradient aggregation differ. The runs use $\alpha = 0.01$ and router z-loss coefficient 0.001.

Projection initialization uses Xavier-uniform weights and a fixed seed sequence $123 + 10g$ for zero-based branch index g. The core grouped-router initialization uses the training seed plus EP rank and marks router weights for expert-style gradient reduction. MindSpeed-LLM also patches router methods, so runtime configuration and patch versions matter.

Projection replicas must synchronize gradients across source ranks, while routers and experts synchronize only with replicas of the same branch. The projection parameters (D_g, A_g) are unmarked dense parameters, so the distributed optimizer reduces their gradients over the full data-parallel group (which spans all EP ranks); expert and router parameters carry expert-style reduction marks and reduce only over branch replicas. Correct collective backward permutations, mixed precision, recomputation, optimizer reduction groups, and checkpoint restoration are necessary to realize one shared model.

A.3 Run manifest and provenance

The logs record eight processes with expert parallel size four and tensor, pipeline, and context parallel sizes one. The eight ranks comprise two EP groups of four ranks each (EP replication factor two across data-parallel groups). A source contributes one 1024-token sequence per microbatch, giving $T = 1024$ and $M = ET = 4096$ tokens in each EP group. Global batch size 128 gives 16 accumulation microbatches across the eight ranks.

All four models have 16 attention heads, key/value channel width 16 with one GQA group (the reported MindSpeed-LLM setting for this configuration), RMSNorm, SwiGLU, untied input/output

embeddings, padded vocabulary size 151,936, and rotary base 10^6. Attention and hidden dropout are zero. Full uniform activation recomputation uses one layer per recomputation unit; training uses a distributed optimizer. The argument dictionaries differ in the MoE layer type (`use_eqmoe`, `use_latent_moe`), expert count, top-k, and expert intermediate width.

All runs use post-selection softmax with top-k probability normalization, no capacity factor, and no capacity padding. Data shard prefixes run from 00000 through 00049. Exact tokenizer and data checksums, the training-code revision, and the full hardware/software manifest are not recorded, limiting exact replication.

B Load-Balance Heatmaps and Communication Time

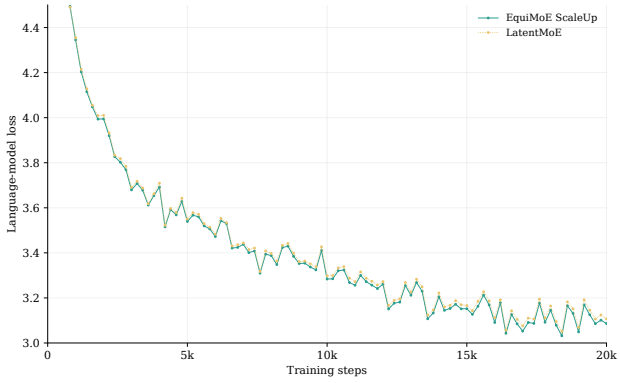


Figure 4: **Training loss: EquiMoE ScaleUp vs LatentMoE.** Both scale-up configurations use 256 experts and top-32 routing; their losses are close, although their intermediate widths differ.

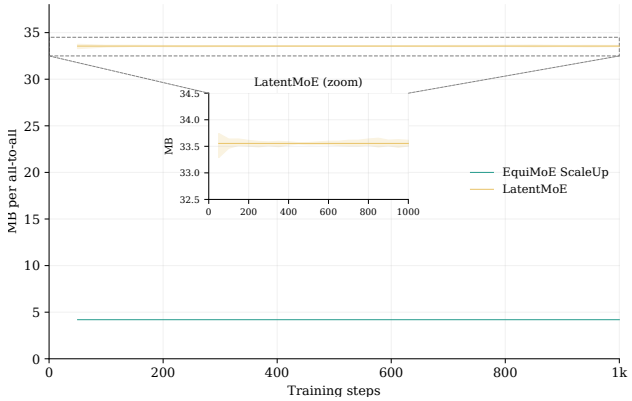


Figure 5: **Communication volume: EquiMoE ScaleUp vs LatentMoE.** Shaded bands show per-rank min/max. EquiMoE ScaleUp has a fixed 4.19 MB payload; LatentMoE's nominal payload is 33.55 MB with similar variance. Inset zooms into 33–34.5 MB.

C Measurement Details

Training logs contain one record per step for each of the four configurations. All runs complete 20,000 steps with consumed-sample increments of 128 and zero skipped or NaN iterations. Load-balance and communication statistics are collected via instrumentation hooks that record per-layer router outputs and all-to-all payload sizes at regular intervals. Summary statistics are computed from these dump files, and figure smoothing is used only for display.

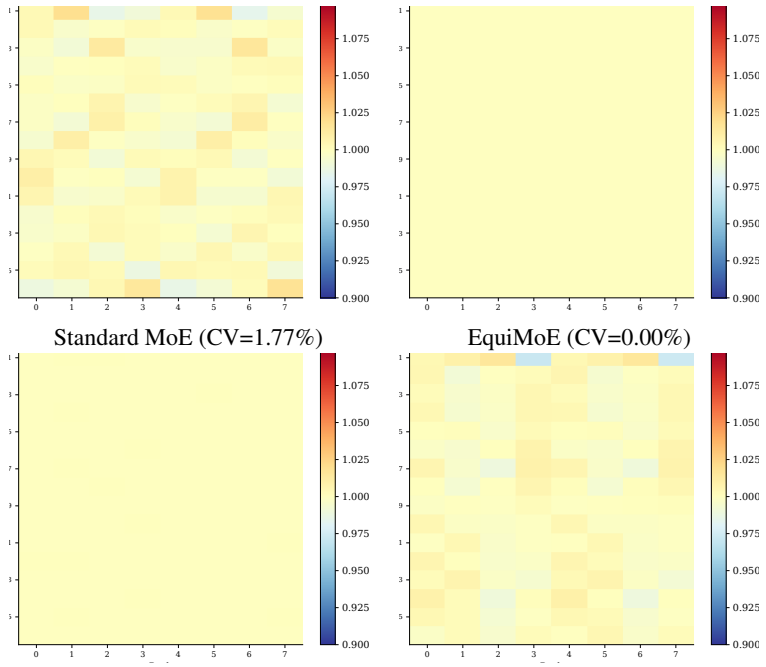


Figure 6: **Rank-level load heatmaps.** Each row is a transformer layer; each column is one of 8 ranks (EP = 4, DP = 2). Color denotes normalized token load per rank (1.0 = layer average). EquiMoE and EquiMoE ScaleUp achieve perfect rank-level balance (CV = 0.00%); Standard MoE and LatentMoE exhibit inter-rank variation. CV here is computed spatially across ranks, whereas Table 1 reports the temporal CV across training steps.

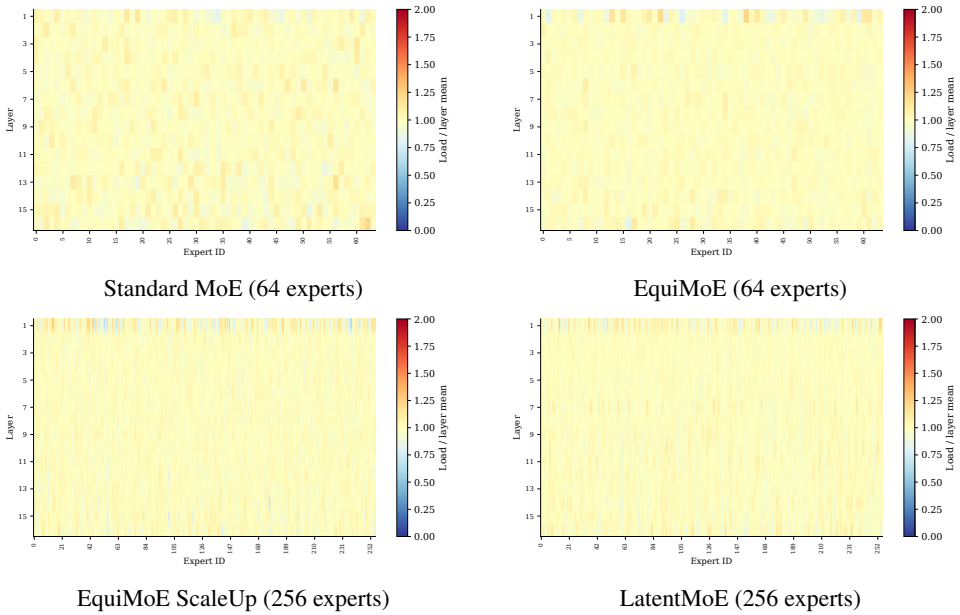


Figure 7: **Expert-level load heatmaps.** Color denotes normalized load per expert (1.0 = layer average; red = overloaded, blue = underutilized).

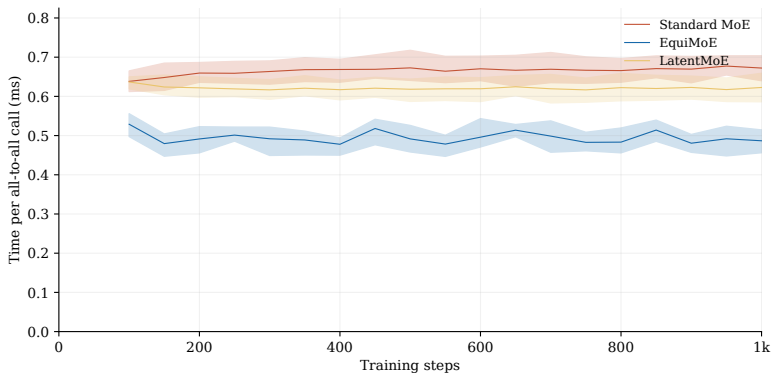


Figure 8: **All-to-all communication time per call.** EquiMoE stabilizes at approximately 0.49 ms after kernel warmup; Standard MoE fluctuates around 0.67 ms and LatentMoE around 0.62 ms. The time reduction is smaller than the volume reduction because all-to-all has a fixed per-call latency. Shaded bands show the per-rank minimum and maximum across 8 ranks at each step. Measurements cover steps 100–1000 (steps 1–50 excluded due to kernel warmup).

D Backward Derivatives and Architectural Analysis

Algorithm 2 EquiMoE backward pass on rank s

Require: Output gradient $\bar{Y}_s : [T, d]$ and Algorithm 1's cache

for $g = 1, \dots, G$ **do**
  $\bar{A}_g^{(s)} \leftarrow V_s[g]^\top \bar{Y}_s$; $\bar{V}_s[g] \leftarrow \bar{Y}_s A_g^\top$
end for
$\bar{U}_s \leftarrow \mathrm{AllToAll}^*(\mathrm{reshape}(\bar{V}_s, [ET, r]), [T]^E)$
$(\bar{O}, \bar{P}) \leftarrow \mathrm{CombineVJP}(\bar{U}_s, O, P, I, \mathcal{C})$
$(\bar{H}, \bar{\theta}_s) \leftarrow \mathrm{ExpertsVJP}(\bar{O}, H, \theta_s)$
$\bar{R}_s^{\mathrm{expert}} \leftarrow \mathrm{DispatchVJP}(\bar{H}, I, \mathcal{C})$
$(\bar{R}_s^{\mathrm{router}}, \bar{W}_s) \leftarrow \mathrm{RouterVJP}(\bar{P}, R_s, W_s, I)$
$\bar{R}_s \leftarrow \bar{R}_s^{\mathrm{expert}} + \bar{R}_s^{\mathrm{router}}$ ▷ add regularizer gradients if used
$\bar{Z}_s \leftarrow \mathrm{reshape}(\mathrm{AllToAll}^*(\bar{R}_s, [T]^E), [E, T, r])$
for $g = 1, \dots, G$ **do**
  $\bar{D}_g^{(s)} \leftarrow X_s^\top \bar{Z}_s[g]$
end for
$\bar{X}_s \leftarrow \sum_{g=1}^{G} \bar{Z}_s[g] D_g^\top$
Synchronize parameter gradients over their respective replica groups
return $\bar{X}_s$ and parameter gradients

Algorithm 2 reverses the two exchanges and differentiates the local mixture. Bars denote gradients and VJP denotes a vector–Jacobian product. The adjoint $\mathrm{AllToAll}^*$ reverses the corresponding forward layout; with equal splits it uses the same all-to-all primitive. Expert dispatch accumulates gradients from repeated token assignments. The router differentiates selected softmax weights while holding discrete top-q indices fixed. Both paths contribute to the projected-input gradient. Auxiliary and z-loss gradients are added through their router graphs when enabled. Shared projection gradients combine contributions from source ranks; router and expert gradients synchronize only across replicas of their own branch.

D.1 Local mixture derivatives

Let $b_t = \partial \mathcal{L}/\partial u_g(x_t)$, and write $e_{t,i} = F_{g,i}(z_t)$ for each selected expert output. The weighted combination yields $\bar{e}_{t,i} = a_{t,i} b_t$ and $\bar{a}_{t,i} = \langle b_t, e_{t,i} \rangle$. For selected-logit softmax,

$$\bar{\ell}_{t,i} = a_{t,i} \left(\bar{a}_{t,i} - \sum_{j \in \mathcal{S}_g(x_t)} a_{t,j} \bar{a}_{t,j} \right) \quad (i \in \mathcal{S}_g(x_t)), \tag{11}$$

with zero task-loss logit gradient on unselected entries. The task-loss derivatives are

$$\bar{W}_g = R_g^\top \bar{\ell}_g, \qquad \bar{z}_t = \sum_{i \in \mathcal{S}_g(x_t)} a_{t,i} b_t J_{F_{g,i}}(z_t) + \bar{\ell}_t W_g^\top, \tag{12}$$

where $J_{F_{g,i}}$ is the output-by-input Jacobian and all gradient vectors use the row-vector convention. Expert VJPs accumulate contributions from all selected experts. Auxiliary losses may also produce gradients on unselected logits. The discrete expert indices have no derivative. At selection boundaries the algorithm uses the branch selected by the forward pass, as in ordinary top-k MoE autograd.

The selected-logit normalization in Equation 4 requires care when $q = 1$: the selected weight is then exactly one and carries no task-loss gradient to its logit. Our experiment uses $q = 2$. A top-1 variant should specify a different gating convention if a nonconstant selected weight is desired.

The backward pass uses two activation exchanges, each carrying Td logical elements per rank, just as in the forward pass. This accounting excludes parameter-gradient reductions. Fixed group quotas also fix the number of expert assignments differentiated on each owner, although recomputation and the shape of local expert batches can still affect backward execution time.

D.2 Caches and synchronization

Algorithm 2 uses expert forward activations internally; activation recomputation may regenerate them. The symbols θ_s denote all expert weights of branch s. Projection gradients $\bar{A}_g^{(s)}$ and $\bar{D}_g^{(s)}$ are source-local contributions; they are summed across sources for a sum loss. Mean-loss normalization and data-parallel averaging must be applied consistently, without averaging the same contribution twice. Routers and experts already receive every source token within their EP group and reduce only across replicas of that branch. The algorithm specifies the mathematical backward pass, not a separately measured custom backward kernel.

D.3 Representation and quota design

Concatenating the down-projections gives a dense input map; stacking the up-projections gives a dense output map. Between them lie G disjoint conditional nonlinear functions with equal activation budgets. Dense mixing allows each branch to access the entire token representation and contribute to a shared output space, while the nonlinear blocks retain separate expert parameters. The input and output maps need not be orthogonal, invertible, or mutually inverse. This construction permits full-width reconstruction without guaranteeing that learned branch representations are distinct.

D.4 Comparison with unrestricted group assignments

For comparison, partition a standard top-k MoE into G equally sized groups of $n = N/G$ experts. A token can place at most $\min(k, n)$ assignments in one group, while the mean group total is Mk/G. Its worst-case group imbalance therefore satisfies

$$\rho_{\text{group}}^{\text{standard}} \leq \frac{G \min(k, n)}{k} = \min(G, N/k). \tag{13}$$

This bound is attainable when every token concentrates as many choices as possible in one group. With $G = 4$, $N = 64$, and $k = 8$, all eight choices can fall in one group, yielding a group imbalance of four. EquiMoE instead assigns two choices to each group, giving imbalance one for the same tokens. Both designs still permit the same worst-case normalized imbalance among individual experts. The structural change therefore removes variability in the distribution of work across groups. It does not prove a stronger uniform-utilization property for the entire expert population.

D.5 Selecting a branch capacity budget

The intermediate width can be derived from the analytic budget instead of copying the full-width expert's value. Let h_0 be the standard-MoE intermediate width. Equating the unique branch parameter counts in Table 3 gives

$$h_{\text{param}} = Gh_0 - \frac{2Gd}{3N} + \frac{G-1}{3}. \tag{14}$$

Equating the leading forward matrix FLOPs, including routers and projections, instead gives

$$h_{\text{compute}} = Gh_0 - \frac{2Gd}{3k} + \frac{N(G-1)}{3k}. \tag{15}$$

These expressions assume the same d, N, k and omit elementwise operations. Real implementations must round to supported kernel dimensions. For the evaluated configuration, $G = 4$, $d = 2048$, $N = 64$, $k = 8$, and $h_0 = 1024$, the two targets are approximately 4011.7 and 3421.3. The actual width 3760 lies between them: it yields fewer unique branch parameters but more leading matrix FLOPs. The difference arises because dense projections execute for every token, whereas only selected experts contribute to per-token expert compute. Consequently, matching stored capacity and matching conditional compute are separate controls.

E Additional Cost Accounting

Table 2: Architecture and training configuration for the four evaluated models. EquiMoE and EquiMoE ScaleUp use learned subspace branches with fixed group quotas; LatentMoE uses dimension reduction with variable-length dispatch.

Setting	Standard MoE	EquiMoE	EquiMoE ScaleUp	LatentMoE
Layers / hidden width	16 / 2048	16 / 2048	16 / 2048	16 / 2048
Experts / activations	64 / 8	64 / 8	256 / 32	256 / 32
Subspace branches	–	4	4	–
Expert input width	2048	512	512	512
Expert intermediate width	1024	3760	848	976
Expert-parallel size	4	4	4	4
Fixed-volume dispatch	No	Yes	Yes	No
Micro / global batch size	1 / 128	1 / 128	1 / 128	1 / 128
Sequence length	1024	1024	1024	1024

Table 3: Analytic costs with $E{=}G$ (one branch per EP rank), fixed expert intermediate width h, and multiply-add counted as two FLOPs. Standard MoE quantities are per source rank before dispatch; EquiMoE quantities are per owner rank after the first all-to-all, where each owner processes ET tokens. The equality $Mn = TN$ for router logits holds because $M = ET$, $n = N/G$, and $E = G$. Standard MoE uses assignment-expanded EP, with balanced assignment counts for the expert-FLOP row.

Quantity	Standard MoE	EquiMoE
Logical dispatch elements	Tkd	Td
Router logit elements	TN	TN
Router matrix FLOPs	$2TdN$	$2TdN/G$
Projection matrix FLOPs	0	$4Td^2$
Expert matrix FLOPs	$6Tkdh$	$6Tkdh/G$
Expert weights	$3Ndh$	$3Ndh/G$
Projection weights	0	$2d^2$
Router weights	dN	dN/G

Communication conventions. For $E = G$ and $M = ET$, one EquiMoE exchange has logical payload $Td = Md/E$ per rank and Md across the group. Its off-rank component is $Td(E-1)/E$ per rank. Dispatch and return double these forward quantities; bytes additionally depend on dtype. Metadata, backward communication, collective transport overhead, and padding are excluded.

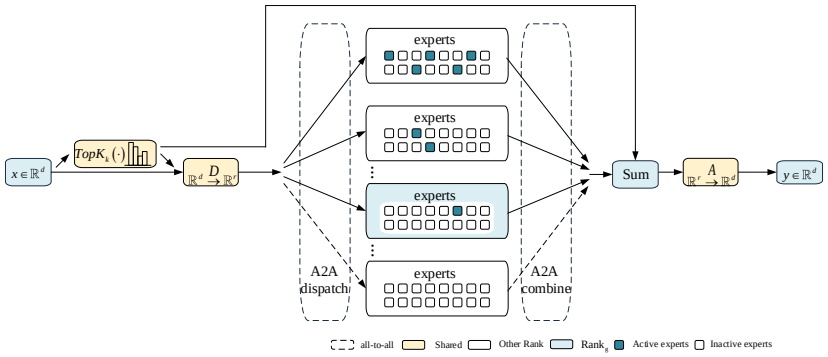


Figure 9: **LatentMoE architecture.** A global router operates on the original token $x \in \mathbb{R}^d$. A learned projection D then reduces the token to width r for expert computation. Two variable-length all-to-all exchanges dispatch and combine tokens based on routing results, so different EP ranks receive different token counts. An up-projection A restores the output to width d.

An assignment-expanded standard dispatcher sends Tkd elements per source. Balanced destination loads give the same return volume per rank, while arbitrary routing changes return sizes. With destination deduplication, a token visiting $u_t \leq \min(k, E)$ ranks contributes $u_t d$ dispatch elements, giving $d\sum_t u_t$ per source. If all tokens visit every rank, this is TEd; if each visits one rank, it is Td. Return deduplication also requires local aggregation. The factor-k payload comparison in the main text applies specifically to assignment-expanded dispatch.

Parameter and FLOP conventions. For bias-free SwiGLU experts, the unique weights in a standard MoE branch and in EquiMoE are

$$P_{\text{standard}} = 3Ndh + dN, \tag{16}$$

$$P_{\text{EquiMoE}} = 3Ndh/G + 2d^2 + dN/G. \tag{17}$$

These expressions exclude transformer attention, embeddings, normalizations, and optimizer state. Physical per-rank storage depends on replication and sharding. At $G = 1$, EquiMoE still has input/output projections and is not generally identical to standard MoE. Setting $h_{\text{EquiMoE}} = Gh_{\text{standard}}$ matches expert weights and leading expert FLOPs, but leaves additional projection cost and a smaller router.

Temporary tensors. Each EquiMoE owner materializes $Mq = Tk$ assignments, with routed activations of $Tkr = Tkd/E$ elements. Index arrays and inverse permutations scale with Tk independently of feature width. Router logits have $Mn = TN$ elements per rank. Reduced activation width therefore does not reduce all temporary storage by the same factor, and no empirical peak-memory claim is made.

F LatentMoE Architecture

LatentMoE (Elango et al., 2026) reduces the expert computation width through a learned projection without imposing branch-level quotas. A global router $W \in \mathbb{R}^{d\times N}$ selects k experts from the full pool of N experts using the original token representation:

$$\ell = xW, \qquad \mathcal{S}(x) = \text{TopK}_k(\ell). \tag{18}$$

A single down-projection $D \in \mathbb{R}^{d\times r}$ then maps the token to a latent representation $z = xD \in \mathbb{R}^r$, where $r = d/\alpha$ for scale-up factor α. The selected experts process z, and an up-projection $A \in \mathbb{R}^{r\times d}$ maps the result back to the model space:

$$f_{\text{LatentMoE}}(x) = \left[\sum_{i\in\mathcal{S}(x)} a_i(x)F_i(xD)\right] A. \tag{19}$$

Unlike EquiMoE, LatentMoE uses a single projection (not G disjoint branch projections), a global router on the full-width token (not G independent local routers on reduced representations), and variable-length all-to-all dispatch (not fixed-volume dispatch). Figure 9 illustrates the architecture.