ADAPTIVE INVERTED-INDEX ROUTING FOR GRANULAR MIXTURES-OF-EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse Mixture-of-experts (MoE) models enable scalable transformer architectures by activating only a subset of experts per token. Recent evidence suggests that performance improves with increasingly granular experts, i.e., many small experts instead of a few large ones. However, this regime substantially increases routing cost, which can dominate computation and memory. We introduce *adaptive inverted-index routing for MoE* (AIR-MoE), an inverted-index-inspired routing architecture based on vector quantization (VQ). In a first stage, AIR-MoE performs *coarse shortlisting* by assigning tokens to VQ codewords to construct a candidate set of experts. In a second stage, *fine scoring* computes exact routing scores restricted to this shortlist. This two-stage procedure *approximates* true top-K routing while avoiding full expert scoring and, in contrast to prior work, imposing no structural constraints on expert parameters. AIR-MoE serves as a drop-in replacement for standard routers and requires no modifications to the model architecture or loss function. We further provide a lower bound on the mass recall achieved by AIR-MoE that yields insights into the inner workings. Empirically, AIR-MoE improves the perplexity–FLOPs trade-off over existing MoE routers in regimes with up to more than a million tiny experts, and achieves consistent perplexity improvements over the best existing baseline.

1 INTRODUCTION

Mixture-of-Experts (MoE; Jacobs et al. (1991)) is an ensemble learning technique where multiple so-called expert models are jointly trained with a routing mechanism that, for a given input, predicts weights to create a linear combination over expert outputs. Sparse MoE (Shazeer et al., 2017) is a variant of MoE that has become popular in recent LLM architectures (e.g., Du et al. (2022; 2025); Jiang et al. (2024)). In a sparse MoE, only the top-K experts (in terms of router scores) are considered in the linear combination, instead of all of them. This way, computational demands during inference are kept nearly constant as the number of experts grows, while increasing the expressivity of the model.

Empirical scaling laws suggest that the ideal regime uses *many* experts with *few* parameters each—so-called *granular experts* (Krajewski et al., 2024).[1] In practice, training such models is prohibitive because the routing overhead (computational and memory) becomes non-negligible as the number of experts increases. In recommender systems, this problem is known as maximum inner product search (MIPS; Shrivastava & Li (2014); Abuzaid et al. (2019)): find the maximum (or top-K) inner product value(s) between a query vector (token representation) and a large number of candidate vectors (expert centroids). Prior work in MoE research addresses this challenge either by restricting routing to predefined groups (hierarchical MoE; Shazeer et al. (2017)) or by imposing structural constraints on expert representations (He, 2024). While these approaches reduce computational and memory costs, they limit routing flexibility (see Sec. F.3), which typically degrades performance.

In the present work, we propose to use vector quantization to construct an inverted-index-like routing architecture, visualized in Fig. 1. For every token, in parallel, the top-K prediction procedure consists of two steps:

[1] We note that similar observations have been made earlier by Clark et al. (2022).

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.

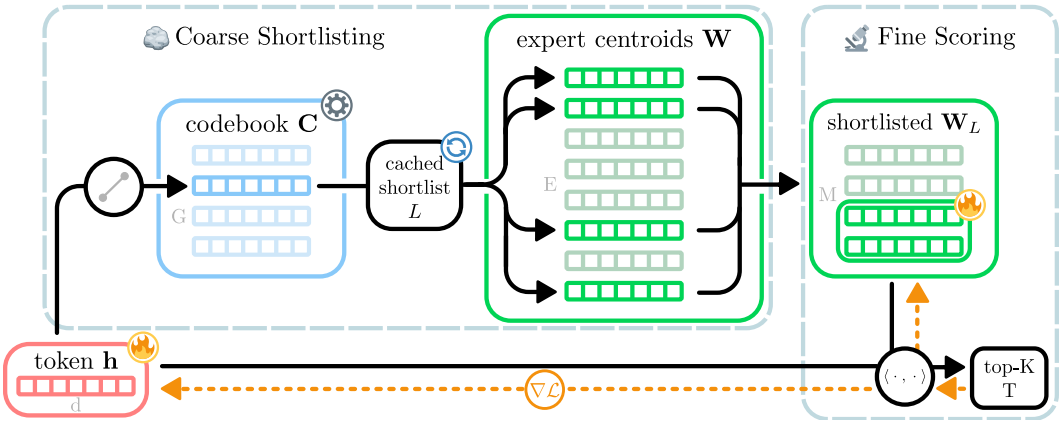

Figure 1: **Overview of AIR-MoE**. The method consists of two parts: The coarse shortlisting stage uses vector quantization to select a codeword (blue) that stores a pre-computed expert shortlist L that references specific expert centroids (green) and is updated after each optimizer step. The fine scoring stage takes the shortlisted expert weights and scores them exactly. Notably, the codebook is learned via gradient-free optimization (gear symbol) and only token representations and expert centroids are trained using the downstream gradient (dashed orange) *without* straight-through estimation trick.

1. Coarse Shortlisting: Retrieve the closest codeword and gather expert indices from the corresponding pre-evaluated shortlist.
2. Fine Scoring: Score all inner products for token and experts within the retrieved shortlist, then use the top-K expert indices and scores for routing.

This procedure avoids structural constraints: experts are neither partitioned (shortlists can be overlapping) nor parametrically restricted. Instead of enforcing exact top-K routing via constraints, we deliberately pursue an efficient and flexible approximation, akin to approximate nearest neighbor search (Indyk & Motwani, 1998).

A central challenge that arises from the two-step procedure sketched above is that the coarse shortlisting step is not differentiable. To address this issue, we propose a bi-level optimization approach where the codebook training is gradient-free and decoupled from the training of all other model parameters. Specifically, the codebook is trained via an adaptive k-means that gradually adapts to the internal distribution shift.

In addition to the methodological contribution, we explore the theoretical assumptions under which the top-K approximation of AIR-MoE is reasonable in terms of retaining probability mass in the routed shortlist (mass recall). Specifically, we show a lower bound on the mass recall that depends on the quantization error and the routing mass that lies outside of the top-M codeword scores.

We summarize our main contributions as follows:

- We introduce the *adaptive inverted-index for MoE* (AIR-MoE), an inverted-index-like routing architecture for granular mixture-of-experts based on adaptive vector quantization that approximates the underlying top-K experts without imposing restrictions (Sec. 3).
- To address the non-differentiability of the codebook, we propose a bi-level optimization strategy that updates the router parameters via gradient descent while learning the codebook using gradient-free adaptive k-means (Sec. 3.2).
- We derive a lower bound on the retained total probability mass of AIR-MoE, thereby providing an intuition for why and under what conditions the methods can be expected to work well (Sec. 3.3).
- We demonstrate empirically that AIR-MoE tends to perform favorably in comparison to coarse-grained MoE and existing routing methods for granular MoE (Sec. 5).

Overview. Sec. 2 covers the main preliminaries of the manuscript, namely mixture-of-experts as used in modern LLMs (Sec. 2.1) and granular MoE (Sec. 2.2), the motivating scaling law for

AIR-MoE. Thereafter, in Sec. 3, we propose AIR-MoE, specifically forward pass (Sec. 3.1), training (Sec. 3.2) and theory (Sec. 3.3). We then cover related work (Sec. 4) and experiments (Sec. 5). Finally, we discuss limitations in Sec. 6 and conclude with Sec. 7.

2 Preliminaries

2.1 Mixture-of-Experts in LLMs

In transformer-based LLMs (Vaswani et al., 2017), MoEs are typically applied to feed-forward neural network (FFN) layers with top-K inference (Shazeer et al., 2017) for compute & memory efficiency:

$$\mathrm{MoE}(\mathbf{h}) \;:=\; \sum_{e\in \mathrm{TopK}\{\boldsymbol{\gamma}(\mathbf{h})\}} \gamma_e(\mathbf{h})\,\mathrm{FFN}_{\phi}^{(e)}(\mathbf{h}), \tag{1}$$

where TopK denotes the indices of the $k \in \mathbb{N}$ largest router weights in $\boldsymbol{\gamma}(\mathbf{h}) \in \mathbb{R}^E$. The router weight for expert e is defined as

$$\gamma_e(\mathbf{h}) \;:=\; \frac{\exp\{z_e(\mathbf{h})\}}{\sum_{e'\in \mathrm{TopK}\{\mathbf{z}(\mathbf{h})\}} \exp\{z_{e'}(\mathbf{h})\}}, \quad z_e(\mathbf{h}) \;:=\; \langle \mathbf{w}_e, \mathbf{h}\rangle, \tag{2}$$

where $\mathbf{w}_e \in \mathbb{R}^d$ for all $e \in \{1, 2, \ldots, E\}$ are learnable expert centroids. A common choice for the FFN parameterization in (1) is the gated linear unit architecture of Shazeer (2020).

2.2 Granular MoE

Empirical scaling laws suggest that increasing expert granularity (using many small experts instead of a few large ones) can improve performance (Krajewski et al., 2024). In particular,

$$\mathcal{L}(Q) \;=\; \frac{a}{Q^b} + c, \qquad Q \;:=\; \frac{d_{\text{standard}}}{d_{\text{expert}}}, \tag{3}$$

where Q denotes the granularity with reference dimension d_{standard} and expert dimension d_{expert}. The variables a, b, c are positive constants. However, this regime substantially increases routing cost, as the router must evaluate inner products $\langle \mathbf{w}_e, \mathbf{h}\rangle$ across many experts.

The next section introduces our approach for reducing this routing overhead.

3 Adaptive Inverted-Index Routing for Mixture-of-Experts (AIR-MoE)

We proceed to elaborate on both the AIR-MoE forward pass in Sec. 3.1 and how the router is trained in Sec. 3.2.

3.1 Forward Pass

The full forward pass of an AIR-MoE router is shown in Alg. 1. We stress that in practice, we parallelize Alg. 1 over tokens and codewords and *do not require looping* (see Sec. C).

Two-Stage Retrieval. The essence is a retrieval-style routing that first shortlists a set of candidate experts L_{g_s} ("coarse") before evaluating the final expert scores within the shortlist set T ("fine"), thereby reducing computational cost. Specifically, this leads to the subset hierarchy

$$\underbrace{T}_{\text{top-}K\text{ index set}} \subset \underbrace{L}_{\text{shortlist}} \subset \underbrace{[E]}_{\text{full set of experts}}. \tag{4}$$

Coarse Shortlisting (Alg. 1, lines 2-7). To obtain a shortlist for an input token $\mathbf{h}_s \in \mathbb{R}^d$ within a batch of tokens $\mathbf{H} := \{\mathbf{h}_s\}_{s=1}^S$, we employ vector quantization (Gray, 1984; Linde et al., 1980).

Algorithm 1: AIR-MoE Forward Pass

Input: Token batch $\mathbf{H} = \{\mathbf{h}_s\}_{s=1}^{S}$, codebook $\mathbf{C} = \{\mathbf{c}_g\}_{g=1}^{G}$, expert centroids $\mathbf{W}$, shortlist size M, K (for top-K selection), jitter $\sigma_{\text{jit}} > 0$

Output: Selected expert indices T_s and scores $\mathbf{z}_s$

1 $\mathbf{w}_e \leftarrow \Pi_{\mathbb{S}^{d-1}}(\mathbf{w}_e), \quad \forall e \in [E]$ `// normalize expert centroids`

Coarse Shortlisting

2 **for** $s \leftarrow 1$ **to** S **do**

3 $\quad g_s \leftarrow \arg\min_{g \in [G]} D(\mathbf{h}_s, \mathbf{c}_g)$

4 **if** *shortlist cache is invalid* **then** `// refresh only after optimizer update`

5 $\quad$ **for** $g \leftarrow 1$ **to** G **do**

6 $\quad\quad \boldsymbol{\eta}_1 \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{jit}}^2 \mathbf{I})$

7 $\quad\quad L_g \leftarrow \text{TopM}_{e \in [E]}\left(\langle \mathbf{c}_g, \mathbf{w}_e \rangle + \boldsymbol{\eta}_1\right)$

8

Fine Scoring

9 **for** $s \leftarrow 1$ **to** S **do**

10 $\quad \boldsymbol{\eta}_2 \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{jit}}^2 \mathbf{I})$

11 $\quad T_s, \mathbf{z}_s \leftarrow \text{TopK}_{e \in L_{g_s}}\left(\langle \mathbf{h}_s, \mathbf{w}_e \rangle + \boldsymbol{\eta}_2\right)$ `// gradients do not pass to` $\mathbf{C}$

12

13 **return** $\{T_s, \mathbf{z}_s\}_{s=1}^{S}$

Specifically, we map each input token $\mathbf{h}_s$ to a codeword from a codebook $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_G\}$ of size G.[2] We use Euclidean distance for the mapping (Alg. 1, line 3):

$$g_s \;\coloneqq\; \arg\min_{g \in [G]} \; D(\mathbf{h}_s, \mathbf{c}_g), \qquad D(\mathbf{x}, \mathbf{y}) \coloneqq \|\mathbf{x} - \mathbf{y}\|_2 \,. \tag{5}$$

The assigned codeword is then given by $c(\mathbf{h}_s) \coloneqq \mathbf{c}_{g_s}$. Each codeword $\mathbf{c}_g$ in turn stores a shortlist $L(\mathbf{c}_g)$, which is refreshed after each optimizer step by scanning the entire set of experts (Alg. 1, lines 5-7). This yields a learned inverted-index over experts, where codewords define coarse cells and each shortlist corresponds to a posting list of candidate experts. While refreshing shortlists incurs a computational cost of $\mathcal{O}(EGd)$, the computation is amortized across the effective batch because it is independent of the number of tokens being routed S, making the arithmetic cost increasingly favorable in large effective-batch regimes common in pre-training.

Fine Scoring (Alg. 1, lines 9-11). After a shortlist is retrieved, the final stage scores each token exactly within the shortlist. In this way, both an approximation $T_s \subset L_g$ of the top-K indices and corresponding scores $\mathbf{z}_s \in \mathbb{R}^k$ are obtained that represent the active experts and their selected router logits, respectively.

Implementation Details. To prevent experts from starvation, we employ switch-transformer-style load balancing and jitter (only at training time; Fedus et al. (2022)). In order to encourage exploration both on shortlist and token level, we apply independent jitter in both stages (Alg. 1, lines 6, 10). Another relevant aspect is the projection of expert centroids onto the unit sphere $\Pi_{\mathbb{S}^d}(\mathbf{w}_e) \coloneqq \mathbf{w}_e / \|\mathbf{w}_e\|$ (Alg. 1, line 1). Doing so has been shown to be beneficial by Nguyen et al. (2024). In addition, the normalization is important to ensure tightness of the centroid approximation (Sec. B, which proves Prop. 1).

3.2 Training

Training AIR-MoE separates the gradient-based optimization of model parameters from the gradient-free adaptation of the codebook. The token representations and expert centroids are updated through the downstream language-modeling objective, whereas the codebook tracks the evolving representation distribution via an adaptive k-means algorithm. We describe both components below and provide the full optimization loop in Sec. A.

[2] in practice, we note that $G \ll E$ to obtain computational gains.

Algorithm 2: Gradient-Free Codebook Update (Adaptive K-Means)

Input: Token batch $\mathbf{H} := \{\mathbf{h}_s\}_{s=1}^{S} \subset \mathbb{R}^d$, codebook $\mathbf{C} = \{\mathbf{c}_g\}_{g=1}^{G} \subset \mathbb{R}^d$, decay $\gamma \in [0, 1)$, running counts $n := \{n_g\}_{g=1}^{G}$, running sums $\mathbf{M} := \{\mathbf{m}_g\}_{g=1}^{G} \subset \mathbb{R}^d$, dead-code threshold τ

Output: Updated codebook $\mathbf{C}$, running counts n, running sums m

1 **for** $s \leftarrow 1$ **to** S **do**
2 $\quad g_s \leftarrow \arg\min_{g \in [G]} D(\mathbf{h}_s, \mathbf{c}_g)$ `// assign token to nearest codeword`
3 **for** $g \leftarrow 1$ **to** G **do**
4 $\quad n_g^{\text{batch}} \leftarrow \sum_{s=1}^{S} \mathbf{1}[g_s = g]$ `// batch assignment count`
5 $\quad \mathbf{m}_g^{\text{batch}} \leftarrow \sum_{s=1}^{S} \mathbf{1}[g_s = g]\, \mathbf{h}_s$ `// batch sum of assigned token states`
6 $\quad n_g \leftarrow \gamma n_g + (1-\gamma) n_g^{\text{batch}}$ `// EMA update of counts`
7 $\quad \mathbf{m}_g \leftarrow \gamma \mathbf{m}_g + (1-\gamma) \mathbf{m}_g^{\text{batch}}$ `// EMA update of sums`
8 $\quad$ **if** $n_g < \tau$ **then**
9 $\quad\quad u \sim \text{Unif}([S])$ `// sample replacement token`
10 $\quad\quad \mathbf{m}_g \leftarrow \mathbf{h}_u$
11 $\quad\quad n_g \leftarrow 1$
12 $\quad \mathbf{c}_g \leftarrow \mathbf{m}_g / n_g$ `// update codeword`
13 **return** $\mathbf{C}, n, \mathbf{M}$

Gradient-Free Codebook Training. A key aspect in learning the codebook $\mathbf{C}$ is that no gradients flow from the router output to $\mathbf{C}$, because $\mathbf{C}$ is only used for shortlisting experts (Alg. 1, line 7), while the actual score computation is done using the token representations $\mathbf{H}$ (Alg. 1, line 11). To make sure that a good codebook is learned, we use an adaptive k-means algorithm (Alg. 2; adaptation of McQueen (1967)) to minimize the expected quantization error. The proposed algorithm (Alg. 2) combines two ideas:

1. Exponential moving average (Van Den Oord et al. (2017); EMA; Alg. 2: lines 6, 7): To model the gradual drift of the distribution of internal representations, we use an exponential moving average approach. Notably, we perform EMA for both the codewords $\mathbf{M}$ and the counts n.
2. Reinitialization of inactive codes (Williams et al. (2020); Alg. 2: line 10): A challenge in learning codebooks is that codewords can die, because non-selected codewords are not updated.[3] We therefore replace codewords whose estimated average count falls below a threshold $\tau > 0$ by random token representations from the current batch.

Representation and Centroid Training. The internal representations $\mathbf{H}$ and the expert centroids $\mathbf{W}$, in contrast to the codebook, obtain a gradient signal from the downstream prediction loss. In contrast to common vector quantization approaches for machine learning (e.g., Van Den Oord et al. (2017); Esser et al. (2021); Razavi et al. (2019)), AIR-MoE does *not* use the straight-through estimation trick (Bengio et al., 2013). Instead, all gradients come directly from the downstream loss and do not pass the quantization stage, as shown in Fig. 1.

3.3 When does the shortlist preserve routing mass?

We analyze the deterministic router used for validation and inference. In these phases, routing jitter is disabled and shortlists are computed from the current codebook and expert centroids. Instead of capturing the exact top-K experts per token, we instead strive for selecting experts whose output probabilities are large. To this end, per token with embedding $\mathbf{h}$, we define the full routing

[3] This phenomenon is analogous to dying experts in sparse MoE architectures.

distribution

$$\pi_e(\mathbf{h}) \coloneqq \frac{\exp(z_e(\mathbf{h}))}{\sum_{j=1}^{E} \exp(z_j(\mathbf{h}))}, \qquad e \in [E],$$

and the retained routing mass

$$\mathrm{MassRecall}(\mathbf{h}) \coloneqq \sum_{e \in L(c(\mathbf{h}))} \pi_e(\mathbf{h}).$$

Proposition 1 (Routing Mass Preservation). *Under the centroid normalization in Alg. 1, let* $\epsilon(\mathbf{h}) \coloneqq D(\mathbf{h}, c(\mathbf{h}))$ *and let* $L(c(\mathbf{h}))$ *denote the top-*M *experts under* $\langle c(\mathbf{h}), \mathbf{w}_e \rangle$. *Then, during validation,*

$$\mathrm{MassRecall}(\mathbf{h}) \geq \exp(-2\epsilon(\mathbf{h}))\, \rho_M(c(\mathbf{h})),$$

where

$$\rho_M(c(\mathbf{h})) \coloneqq \sum_{e \in L(c(\mathbf{h}))} \pi_e(c(\mathbf{h}))$$

is the routing mass captured by the codeword shortlist.

The bound, which is proved in Sec. B, highlights two factors: (i) the quantization error $\epsilon(\mathbf{h})$, which should be small for accurate codebooks, and (ii) the shortlist mass $\rho_M(c(\mathbf{h}))$, which increases with M. Hence, AIR-MoE retains most routing mass when tokens are well quantized and the shortlist is sufficiently large.

4 Related Work

Granular MoE. Prior work uses fixed, non-learned hash-table routers to improve routing efficiency (Roller et al., 2021; Nogueira Dos Santos et al., 2024), although such approaches are fundamentally limited both empirically and theoretically (Clark et al., 2022; Dikkala et al., 2023). More recently, He (2024) proposed a learnable router based on product-key retrieval (Lample et al., 2019), which decomposes expert centroids into two separately evaluated parts and reduces routing cost from $\mathcal{O}(SEd)$ to $\mathcal{O}(Sd(\sqrt{E}+K^2))$. Like our method, it uses two-stage retrieval to score experts. Unlike AIR-MoE, it retrieves the exact top-K experts rather than approximating them, but requires expert centroids $\mathbf{W}$ to be Cartesian products of two prototypes, restricting the space of admissible expert representations. The unrestricted centroids of AIR-MoE can therefore model token–expert dependencies more flexibly. Moreover, the method of He (2024) scales quadratically with the number of active experts K, requiring a multi-head MoE architecture with smaller per-head K. In contrast, AIR-MoE requires no such architectural modifications and can be used more easily as a drop-in replacement in existing MoEs.

Grouping in MoE. Our method belongs to a broader class of approaches that group experts using either overlapping (Su et al., 2024b; Tang et al., 2025) or non-overlapping groups (Jordan & Jacobs, 1991; 1994; Shazeer et al., 2017). Like AIR-MoE, Su et al. (2024b); Tang et al. (2025); Xie et al. (2023) construct expert groups, but do so to mitigate load imbalance (Su et al., 2024b; Tang et al., 2025) or "overfitting" (Xie et al., 2023), rather than to reduce computation. Hierarchical MoEs (Jordan & Jacobs, 1991; 1994; Shazeer et al., 2017) also use non-overlapping groups to reduce computation. Unlike AIR-MoE, however, they partition experts into fixed subgroups and restrict routing to experts within the selected groups. This avoids recomputing codebook–expert scores (Alg. 1, line 7) but limits expressivity by preventing experts from being reused globally. Furthermore, hierarchical MoEs typically learn group selection through additive group and expert scores, whereas AIR-MoE forms groups through vector quantization.

Vector Quantization in MoE. Do et al. (2024) also use vector quantization to encode token vectors in MoEs. Unlike AIR-MoE, however, they use codewords directly to score experts rather than to construct a retrieval index. Their method therefore provides no computational benefit.

Table 1: **Main results.** Perplexity (PPL) and improvement compared to the best non-AIR-MoE baseline across different model sizes and datasets. Best results are highlighted in bold. AIR-MoE consistently achieves favorable perplexity compared to existing granular MoE approaches, as well as the standard coarse approach. More metrics are shown in Sec. F.2

Size	Method	WikiText-103		C5		OpenWebText2	
		PPL ↓	Gain vs. best baseline ↑	PPL ↓	Gain vs. best baseline ↑	PPL ↓	Gain vs. best baseline ↑
Small	AIR	**19.79**	**+2.1%**	**546.87**	**+4.2%**	**49.16**	**+5.4%**
	PEER	20.30	-0.4%	570.88	+0.0%	51.95	+0.0%
	Hierarchical	20.21	+0.0%	918.47	-60.9%	67.21	-29.4%
	Std. Coarse	21.79	-7.8%	796.31	-39.5%	56.85	-9.4%
Medium	AIR	**18.92**	**+3.7%**	**43.85**	**+4.4%**	**20.64**	**+3.9%**
	PEER	19.64	+0.0%	45.88	+0.0%	21.49	+0.0%
	Hierarchical	24.05	-22.4%	51.45	-12.1%	25.68	-19.5%
	Std. Coarse	21.02	-7.0%	49.81	-8.6%	22.77	-5.9%
Large	AIR	–	–	**46.84**	**+2.1%**	**18.03**	**+0.7%**
	PEER	–	–	47.86	+0.0%	18.16	+0.0%
	Hierarchical	–	–	51.73	-8.1%	20.24	-11.5%
	Std. Coarse	–	–	49.06	-2.5%	18.67	-2.8%

Inverted File Indices & Differentiable Retrieval. Our method is closely related to inverted file indices (IVF; Salton et al. (1983); Sivic & Zisserman (2003); Johnson et al. (2019)), which perform approximate nearest-neighbor search through coarse-to-fine retrieval. A coarse quantizer partitions the space, restricting each query to a small number of cells before candidates within those cells are scored exactly. AIR-MoE applies the same principle to MoE routing: learned shortlist embeddings serve as coarse centroids, and each induces a candidate shortlist whose experts are then re-scored exactly. Unlike classical IVF systems, which are typically constructed offline for a fixed database, our method is trained end-to-end and jointly adapts its quantization and expert representations. Recent work similarly learns index representations jointly with retrieval models, blurring the distinction between indexing and retrieval (Wang et al., 2022; Zhuang et al., 2022). Our method advances this direction by enabling FLOP-efficient approximate MIPS in a gradient-compatible manner without restricting the expert space of MoE architectures.

5 Experiments

We aim to answer the following questions: **(1)** How does AIR-MoE compare with existing techniques for granular MoE? **(2)** How does AIR-MoE compare with a coarse MoE architecture? **(3)** How well does AIR-MoE scale in the amount of experts? **(4)** How do the different components of AIR-MoE affect performance?

We address questions **(1)** to **(3)** in the main text and refer to Sec. F.5 for ablation experiments (**(4)**).

5.1 Setup

We describe the core setup here and defer details to Sec. E.

Datasets. We evaluate on WikiText-103 (Merity et al., 2017), C5 (Common Crawl Foundation, 2025), and OpenWebText2 (Biderman et al., 2022).

Baselines. (1) Coarse MoE with standard routing and $K = 1$ (matched active parameters) and expert-choice gating, (2) granular MoE with standard routing (only shown in Sec. 5.3 and Sec. F.4 because of memory overflow), (3) granular PEER routing (He, 2024), (4) granular hierarchical MoE (Shazeer et al., 2017).

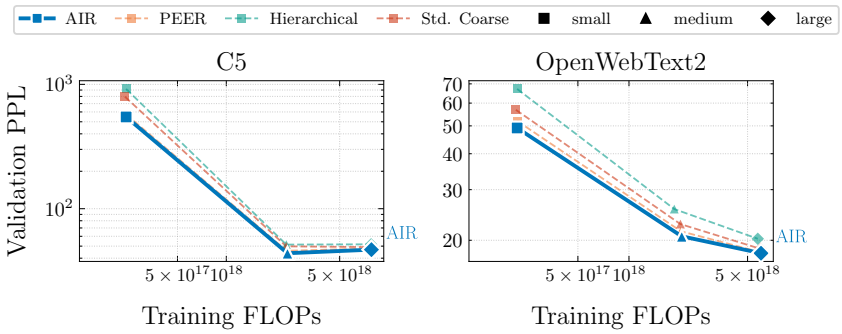

Figure 2: **Performance–efficiency trade-off.** Validation perplexity vs. training FLOPs. AIR-MoE (blue) occupies the lower-left region of the plot and thus achieves the strongest Pareto trade-off.

Metrics. Perplexity (PPL; ↓); entropy over expert usage $-\mathbb{E}_{\hat{p}}[\log \hat{p}]$ (Entropy); fraction of dead experts, i.e., experts that do not receive a single token (Dead Experts); overlap fraction between top-K estimate and true top-K (Overlap; only for AIR-MoE).

Architecture. We use the Llama3 transformer (Grattafiori et al., 2024), replacing the middle FFN with a MoE layer that consists of tiny experts with a single intermediate dimension (He, 2024) and adding post-MoE layer normalization (Ba et al., 2016; Krajewski et al., 2024). We report results for small (512^2 experts; $\approx$ 313M total parameters), medium (1024^2 experts; $\approx$ 2B parameters total), and large (1024^2 experts; $\approx$ 2.9B parameters total) models. For WikiText-103, a relatively small data set, we omit the large model due to strong overfitting.

Training. We use standard pre-training hyperparameters (see Sec. E.5) and align baselines by active parameters (details in Sec. E.5.2). For WikiText-103, we train for 10 (small) and 3 (medium) epochs, respectively. For C5 and OpenWebText2, we use $10\times$ the amount of the MoE-free (that is, without the MoE layer) model parameters in tokens.

5.2 Main Results

The main results, shown in Tab. 1 and Fig. 2, demonstrate the merits of the proposed routing architecture: AIR-MoE yields consistent gains over the considered MoE baselines across datasets and model sizes (multi-seed results are shown in Sec. F.1). Compared to hierarchical and PEER routers, AIR-MoE achieves persistent PPL reductions of up to 5.4% relative to the best baseline (Tab. 1). Due to the vast amount of expert parameters, even the routing cost of the standard coarse approach becomes non-negligible and therefore does not yield large FLOP benefits, while performing worse than both PEER and AIR-MoE in terms of perplexity (Fig. 2). The hierarchical router generally yields the weakest results, typically underperforming even the coarse-grained MoE baseline. We hypothesize that this poor performance results from the absence of overlap between groups: although some experts may be broadly useful across many tokens, they can only be selected when their corresponding coarse group is activated. In contrast, the shortlists constructed by AIR-MoE naturally overlap, and some experts are included substantially more often than others, as shown empirically in Sec. F.6.

5.3 Router Efficiency at Scale

In order to assess router efficiency beyond FLOPs, we run an additional experiment for a combined forward and backward pass of the routers in isolation. We compare router FLOPs, runtime per iteration, and peak VRAM usage as a function of the number of experts (details regarding setup are shown in Sec. E.4 and further scaling experiments are shown in Sec. F.4).

The results, shown in Fig. 3, demonstrate that AIR-MoE (blue) scales substantially better than the standard granular router (pink) and comparably to the hierarchical and PEER routers across all reported metrics.

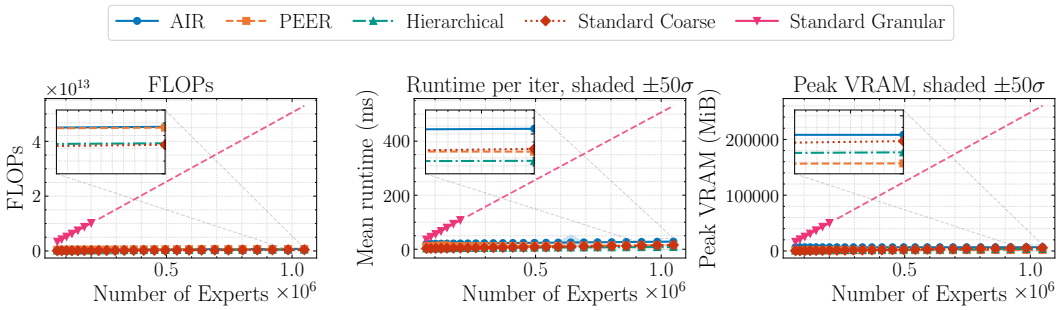

Figure 3: **Router efficiency.** Mean results ±50 standard deviations across 100 iterations. The dashed extension of the standard granular approach (pink) is a linear extrapolation because of memory overflow. Standard granular routing leads to substantial computational and memory overhead as the amount of experts is scaled up. AIR-MoE, in contrast, keeps resource demand at a similar level to existing efficient granular approaches.

6 Discussion & Limitations

Shortlist Updates. A distinguishing feature of AIR-MoE is that, unlike prior approaches, it updates all shortlists after each optimizer step (Sec. 3.1). This incurs a computational cost of $\mathcal{O}(EGd)$ per update, where G denotes the codebook size and E the number of experts. In practice, this overhead is typically tolerable because it is independent of the effective batch size (that is, the micro batch size multiplied by the amount of gradient accumulation steps). Nevertheless, the resulting compute savings depend strongly on G, E, and the effective batch size S (see Sec. F.4 and F.7).

Expert Parallelism. A key open question for making granular MoEs viable for large-scale foundation-model training is how to design efficient expert-parallel systems, such as DeepSpeed-MoE (Rajbhandari et al., 2022) and Megatron-LM (Shoeybi et al., 2019). In the granular regime, expert parallelism can substantially increase inter-device memory traffic and degrade runtime. Although we do not study this direction, architectures such as AIR-MoE may be well suited to distributed training because expert shortlists create routing locality. If each shortlist is assigned to one device, its expert centroids and FFN weights can be colocated there, requiring each token to be routed only to the device hosting its shortlist and potentially reducing cross-device communication.

Custom Kernel Implementations. All implementations used to obtain the runtime and peak-VRAM results reported in Sec. 5.3 and Sec. F.4, including both AIR-MoE and the considered baselines, rely exclusively on functionality provided by PyTorch. We do not develop method-specific custom kernels, which could substantially affect the observed hardware-level performance. Designing and carefully optimizing custom kernels represents an important direction for future work.

7 Conclusion

We introduced AIR-MoE, an inverted-index-inspired routing architecture for granular mixture-of-experts. By using a vector-quantization-based coarse-to-fine scheme, AIR-MoE approximates top-K routing without fixed grouping or imposing structural constraints on expert representations. We further proposed a bi-level training strategy for learning the adaptive codebook and provided a lower bound on the retained routing mass. Empirically, AIR-MoE achieves a favorable perplexity–FLOPs trade-off in granular MoE settings, comparing favorably to existing MoE approaches while remaining substantially cheaper than exact granular routing. Overall, our results suggest that inverted-index structures provide an effective mechanism for routing in sparse and highly granular mixture-of-experts architectures.

Reproducibility Statement

All source code, including instructions in the `README.md`, is available at https://anonymous.4open.science/r/adaptive_inverted_index_routing_code-D6FE/.

REFERENCES

Firas Abuzaid, Geet Sethi, Peter Bailis, and Matei Zaharia. To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. *International Conference on Data Engineering*, pp. 1250–1261, 2019. 1

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023. 19

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016. 8

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*, 2013. 5

Stella Biderman, Kieran Bicheno, and Leo Gao. Datasheet for the Pile. *arXiv preprint arXiv:2201.07311*, 2022. 7, 19

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified Scaling Laws for Routed Language Models. *International Conference on Machine Learning*, pp. 4057–4086, 2022. 1, 6

Common Crawl Foundation. Common crawl creative commons corpus (c5). *Common Crawl*, 2025. Available at https://huggingface.co/datasets/BramVanroy/CommonCrawl-CreativeCommons, accessed 2025-08-11. 7

Inderjit S. Dhillon and Dharmendra S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1):143–175, 2001. 24

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. *Conference on Empirical Methods in Natural Language Processing*, pp. 9376–9396, 2023. 6

Giang Do, Kha Pham, Hung Le, and Truyen Tran. On the Role of Discrete Representation in Sparse Mixture of Experts. *arXiv preprint arXiv:2411.19402*, 2024. 6

Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, et al. Kimi-VL Technical Report. *arXiv preprint arXiv:2504.07491*, 2025. 1

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. *International Conference on Machine Learning*, pp. 5547–5569, 2022. 1

Patrick Esser, Robin Rombach, and Björn Ommer. Taming Transformers for High-Resolution Image Synthesis. *Conference on Computer Vision and Pattern Recognition*, pp. 12873–12883, 2021. 5

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. 4, 24

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. 8

Robert Gray. Vector Quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984. 3

Xu Owen He. Mixture of a Million Experts. *arXiv preprint arXiv:2407.04153*, 2024. 1, 6, 7, 8, 20

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *ACM Symposium on Theory of Computing*, pp. 604–613, 1998. 2

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, 1991. 1

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. 1

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019. 7

Michael I. Jordan and Robert A. Jacobs. Hierarchies of Adaptive Experts. *Advances in Neural Information Processing Systems*, 4, 1991. 6

Michael I. Jordan and Robert A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2):181–214, 1994. 6

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling Laws for Fine-Grained Mixture of Experts. *International Conference on Machine Learning*, 2024. 1, 3, 8

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large Memory Layers with Product Keys. *Advances in Neural Information Processing Systems*, 32, 2019. 6, 20

Pingzhi Li, Xiaolong Jin, Zhen Tan, Yu Cheng, and Tianlong Chen. QuantMoE-Bench: Examining Post-Training Quantization for Mixture-of-Experts. *arXiv preprint arXiv:2406.08155*, 2024a. 24

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. Merge, Then Compress: Demystify Efficient SMoE with Hints from Its Routing Policy. *International Conference on Learning Representations*, 2024b. 24

Yoseph Linde, Andres Buzo, and Robert Gray. An Algorithm for Vector Quantizer Design. *IEEE Transactions on communications*, 28(1):84–95, 1980. 3

James B. McQueen. Some Methods of Classification and Analysis of Multivariate Observations. *Proc. of 5th Berkeley Symposium on Math. Stat. and Prob.*, pp. 281–297, 1967. 5

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. *International Conference on Learning Representations*, 2017. 7, 19

Huy Nguyen, Pedram Akbarian, Trang Pham, Trang Nguyen, Shujian Zhang, and Nhat Ho. Statistical advantages of perturbing cosine router in sparse mixture of experts. *arXiv preprint arXiv:2405.14131*, 6, 2024. 4

Cicero Nogueira Dos Santos, James Lee-Thorp, Isaac Noble, Chung-Ching Chang, and David Uthus. Memory Augmented Language Models through Mixture of Word Experts. *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4425–4438, 2024. 6

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. 19

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. pp. 18332–18346, 2022. 9

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating Diverse High-Fidelity Images with VQ-VAE-2. *Advances in Neural Information Processing Systems*, 32, 2019. 5

Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Hash Layers For Large Sparse Models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021. 6

Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983. 7

Noam Shazeer. GLU Variants Improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020. 3

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. *International Conference on Learning Representations*, 2017. 1, 3, 6, 7, 24

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2019. 9

Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in Neural Information Processing Systems*, 27, 2014. 1

Sivic and Zisserman. Video Google: A text retrieval approach to object matching in videos. *International Conference on Computer Vision*, pp. 1470–1477, 2003. 7

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063, 2024a. 19

Zhenpeng Su, Zijia Lin, Xue Bai, Xing Wu, Yizhe Xiong, Haoran Lian, Guangyuan Ma, Hui Chen, Guiguang Ding, Wei Zhou, et al. MaskMoE: Boosting Token-Level Learning via Routing Mask in Mixture-of-Experts. *arXiv preprint arXiv:2407.09816*, 2024b. 6

Yehui Tang, Xiaosong Li, Fangcheng Liu, Wei Guo, Hang Zhou, Yaoyuan Wang, Kai Han, Xianzhi Yu, Jinpeng Li, Hui Zang, et al. Pangu Pro MoE: Mixture of Grouped Experts for Efficient Sparsity. *arXiv preprint arXiv:2505.21411*, 2025. 6

Aaron Van Den Oord, Oriol Vinyals, et al. Neural Discrete Representation Learning. *Advances in Neural Information Processing Systems*, 30, 2017. 5

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017. 3

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, et al. A Neural Corpus Indexer for Document Retrieval. *Advances in Neural Information Processing Systems*, 35:25600–25614, 2022. 7

Will Williams, Sam Ringer, Tom Ash, David MacLeod, Jamie Dougherty, and John Hughes. Hierarchical Quantized Autoencoders. *Advances in Neural Information Processing Systems*, 33: 4524–4535, 2020. 5

Yuan Xie, Shaohan Huang, Tianyu Chen, and Furu Wei. MoEC: Mixture of Expert Clusters. *AAAI Conference on Artificial Intelligence*, 37(11):13807–13815, 2023. 6

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-Experts with Expert Choice Routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022. 24

Shengyao Zhuang, Houxing Ren, Linjun Shou, Jian Pei, Ming Gong, Guido Zuccon, and Daxin Jiang. Bridging the Gap Between Indexing and Retrieval for Differentiable Search Index with Query Generation. *arXiv preprint arXiv:2206.10128*, 2022. 7

Appendix

Table of Contents

A Full Optimization Loop **14**

B Proof for Prop. 1 **14**

C Further Implementation Details **16**

D FLOP Analysis **17**
D.1 Elementwise and Reduction Operations 18
D.2 Softmax 18
D.3 Normalization Layers 18
D.4 Attention 18
D.5 Routing and Indexing Operations 18

E Experimental Setup **19**
E.1 Data 19
E.2 Training 19
E.3 Architectures 19
E.4 Efficiency at Scale Experiments 20
E.5 Hyperparameters 20
E.6 Hardware 21

F Additional Experimental Results **21**
F.1 Multi-Seed Runs 21
F.2 Additional Metrics 21
F.3 Qualitative Differences in Expert Usage 22
F.4 Additional Efficiency at Scale Experiments 22
F.5 Ablation Studies 24
F.6 Empirical Inspection of Shortlist Overlap 24
F.7 On the Choice of Codebook Size and Shortlist Size 24

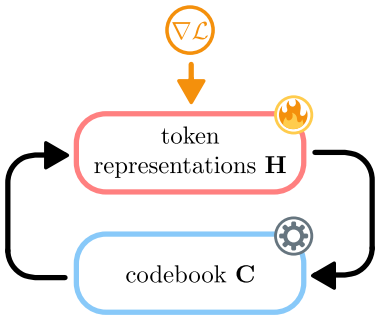

Figure 4: **Feedback Loop**. AIR-MoE forms a bi-level optimization loop between gradient-learned token representations and a codebook updated via adaptive clustering.

A Full Optimization Loop

Alg. 3 summarizes the full training procedure of AIR-MoE. The algorithm can be viewed as a bi-level optimization loop between gradient-trained model parameters and a gradient-free adaptive codebook. Let θ denote all parameters trained by backpropagation, including the transformer parameters, expert FFN parameters, and expert centroids $\mathbf{W}$. In contrast, the codebook $\mathbf{C}$ and its exponential-moving-average statistics $(n, \mathbf{M})$ are auxiliary router state and are not updated by the optimizer.

Importantly, the codeword-to-expert shortlists are not recomputed after every codebook update. Instead, a cached shortlist table is reused throughout each gradient-accumulation window. The codebook therefore continues to adapt across microbatches while the shortlist cache remains fixed and may become slightly stale. This deliberate approximation amortizes the $\mathcal{O}(EGd)$ shortlist-construction cost over the effective batch. After each optimizer step, the cache is invalidated and lazily reconstructed during the subsequent forward pass using the updated expert centroids and codebook.

B Proof for Prop. 1

Proof. Let $\mathbf{c} = c(\mathbf{h})$ denote the assigned codeword and define

$$\epsilon(\mathbf{h}) \;\coloneqq\; D(\mathbf{h}, \mathbf{c}).$$

Further, let

$$L(\mathbf{c}) \;=\; \mathrm{TopM}_{e\in[E]}\langle \mathbf{c}, \mathbf{w}_e\rangle$$

denote the shortlist induced by the codeword. By definition,

$$\mathrm{MassRecall}(\mathbf{h}) \;=\; \sum_{e\in L(\mathbf{c})} \pi_e(\mathbf{h}) \;=\; \frac{\sum_{e\in L(\mathbf{c})} \exp\{z_e(\mathbf{h})\}}{\sum_{j=1}^{E} \exp\{z_j(\mathbf{h})\}}.$$

We first compare token and codeword logits. By the Cauchy-Schwarz inequality, for any expert $e \in [E]$,

$$|z_e(\mathbf{h}) - z_e(\mathbf{c})| \;=\; |\langle \mathbf{w}_e, \mathbf{h} - \mathbf{c}\rangle| \;\le\; \|\mathbf{w}_e\|\,\|\mathbf{h} - \mathbf{c}\| \;\le\; \epsilon(\mathbf{h}),$$

where the last step uses the assumption $\|\mathbf{w}_e\| \le 1$. Hence, for every $e \in [E]$,

$$z_e(\mathbf{h}) \;\ge\; z_e(\mathbf{c}) - \epsilon(\mathbf{h}) \qquad \text{and} \qquad z_e(\mathbf{h}) \;\le\; z_e(\mathbf{c}) + \epsilon(\mathbf{h}).$$

Exponentiating yields

$$\exp\{z_e(\mathbf{h})\} \;\ge\; \exp(-\epsilon(\mathbf{h}))\,\exp\{z_e(\mathbf{c})\},$$

and

$$\exp\{z_e(\mathbf{h})\} \;\le\; \exp(\epsilon(\mathbf{h}))\,\exp\{z_e(\mathbf{c})\}.$$

Algorithm 3: Full Optimization Loop for AIR-MoE

Input: Training data $\mathcal{D}$; initial parameters θ_0; initial codebook $\mathbf{C}_0$; EMA counts n_0; EMA sums $\mathbf{M}_0$; optimizer `optim`; training horizon T; gradient accumulation steps A; shortlist size M; top-K

Output: Trained parameters θ_T and codebook $\mathbf{C}_T$

Initialize empty shortlist cache $\{L_g\}_{g=1}^{G} \leftarrow \varnothing$

for $t \leftarrow 0, 1, \ldots, T-1$ **do**
  `optim.zero_grad()`
  for $a \leftarrow 1, \ldots, A$ **do**
    $\mathcal{D}_\mathrm{b} \sim \mathcal{D}$ `// sample microbatch`
    Compute token representations $\mathbf{H}_t = \{\mathbf{h}_s\}_{s=1}^{S}$ using parameters θ_t
    `// gradient-free codebook update`
    $(\mathbf{C}_t, n_t, \mathbf{M}_t, \{g_s\}_{s=1}^{S}) \leftarrow \textsc{AdaptiveKMeans}(\mathbf{H}_t, \mathbf{C}_t, n_t, \mathbf{M}_t)$
    `// codeword assignment`
    for $s \leftarrow 1$ **to** S **do**
      $g_s \leftarrow \arg\min_{g \in [G]} D(\mathbf{h}_s, \mathbf{c}_g)$
    `// normalize expert centroids for routing`
    $\mathbf{w}_e \leftarrow \Pi_{\mathbb{S}^d}(\mathbf{w}_e), \quad \forall e \in [E]$
    `// lazily construct codeword-to-expert shortlists`
    if *shortlist cache is empty* **then**
      for $g \leftarrow 1$ **to** G **do**
        $\boldsymbol{\eta}_1 \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{jit}}^2 \mathbf{I})$
        $L_g \leftarrow \mathrm{TopM}_{e \in [E]}(\langle \mathbf{c}_g, \mathbf{w}_e \rangle + \boldsymbol{\eta}_1)$ `// cache expert indices`
    `// fine scoring within cached shortlists`
    for $s \leftarrow 1$ **to** S **do**
      $\boldsymbol{\eta}_2 \sim \mathcal{N}(\mathbf{0}, \sigma_{\mathrm{jit}}^2 \mathbf{I})$
      $T_s, \mathbf{z}_s \leftarrow \mathrm{TopK}_{e \in L_{g_s}}\big(\langle \mathbf{h}_s, \mathbf{w}_e \rangle + \boldsymbol{\eta}_2\big)$
    Compute prediction loss $\mathcal{L}_{\mathrm{LM}}$
    Compute load balancing $\mathcal{L}_{\mathrm{router}}$
    $\mathcal{L} \leftarrow \mathcal{L}_{\mathrm{LM}} + \mathcal{L}_{\mathrm{router}}$
    Backpropagate $\mathcal{L}/A$ `// gradients update` θ`, not` $\mathbf{C}_t$
  `// gradient update of model parameters`
  $\theta_{t+1} \leftarrow$ `optim.step`(θ_t)
  `// invalidate shortlist cache after optimizer step`
  $\{L_g\}_{g=1}^{G} \leftarrow \varnothing$
return $\theta_T, \mathbf{C}_T$

Applying the lower bound in the numerator and the upper bound in the denominator gives

$$\mathrm{MassRecall}(\mathbf{h}) = \frac{\sum_{e \in L(\mathbf{c})} \exp\{z_e(\mathbf{h})\}}{\sum_{j=1}^{E} \exp\{z_j(\mathbf{h})\}} \;\geq\; \frac{\exp(-\epsilon(\mathbf{h})) \sum_{e \in L(\mathbf{c})} \exp\{z_e(\mathbf{c})\}}{\exp(\epsilon(\mathbf{h})) \sum_{j=1}^{E} \exp\{z_j(\mathbf{c})\}}.$$

Therefore,

$$\mathrm{MassRecall}(\mathbf{h}) \;\geq\; \exp(-2\epsilon(\mathbf{h})) \frac{\sum_{e \in L(\mathbf{c})} \exp\{z_e(\mathbf{c})\}}{\sum_{j=1}^{E} \exp\{z_j(\mathbf{c})\}}.$$

It remains to identify the fraction on the right-hand side. By the definition of the full routing distribution at the codeword,

$$\pi_e(\mathbf{c}) = \frac{\exp\{z_e(\mathbf{c})\}}{\sum_{j=1}^{E} \exp\{z_j(\mathbf{c})\}},$$

hence

$$\frac{\sum_{e \in L(\mathbf{c})} \exp\{z_e(\mathbf{c})\}}{\sum_{j=1}^{E} \exp\{z_j(\mathbf{c})\}} = \sum_{e \in L(\mathbf{c})} \pi_e(\mathbf{c}) = 1 - \sum_{e \notin L(\mathbf{c})} \pi_e(\mathbf{c}).$$

Using the definition

$$\rho_M(\mathbf{c}) \;\coloneqq\; \sum_{e \in L(\mathbf{c})} \pi_e(\mathbf{c}),$$

we obtain

$$\mathrm{MassRecall}(\mathbf{h}) \;\geq\; \exp(-2\epsilon(\mathbf{h}))\, \rho_M(\mathbf{c}),$$

which proves the desired result. □

C Further Implementation Details

Overview. Listing 1 shows a simplified PyTorch-style implementation of the proposed AIR Router.

Step 1: Shortlist selection. Each token representation $\mathbf{h} \in \mathbb{R}^H$ is assigned to a shortlist centroid using a vector-quantization module (`AdaptiveKmeans`). This produces one discrete shortlist index per token and updates the codebook's exponential-moving-average statistics without backpropagating through the quantization operation.

Step 2: Experts per shortlist. For each shortlist centroid, a fixed subset of experts is selected. This is implemented by computing similarities between shortlist centroids and expert feature vectors, followed by a top-M selection. This step defines a coarse shortlist of candidate experts per shortlist.

Step 3: Candidate scoring. After shortlist selection, each token is scored only against the experts contained in its selected shortlist. This avoids dense scoring against all experts and reduces the routing cost from $\mathcal{O}(SE)$ to $\mathcal{O}(SM)$. A practical challenge is that tokens are assigned to shortlists irregularly, so different shortlists may receive different numbers of tokens. To handle this efficiently, our implementation of AIR-MoE uses PyTorch's `grouped_mm` operation, which evaluates the per-shortlist matrix multiplications as a grouped GEMM. This allows all non-empty shortlists to be processed in parallel without materializing a token-expanded candidate tensor of shape $S \times M \times H$ and without padding each shortlist to a fixed maximum capacity.

Step 4: Final expert selection. From the candidate set, the router performs a standard top-K selection per token. The selected experts and their normalized softmax weights define the routing decision.

Omitted components. For clarity, Listing 1 omits several implementation details that are present in the full version used in experiments:

- load-balancing losses for experts,
- jitter noise for exploration,
- caching of shortlist-to-expert assignments,
- chunked computation for memory efficiency,
- overlap estimation with exact routing.

These components do not change the conceptual routing mechanism, but are included in the full implementation for improved training stability and scalability.

```python
import torch
import torch.nn as nn


class AIRRouter(nn.Module):
    """Minimal PyTorch-like AIR-Router."""
    def __init__(self, hidden_size, num_experts, num_shortlists, top_k,
    ↪  experts_per_shortlist):
        super().__init__()
        self.top_k = top_k
        self.experts_per_shortlist = experts_per_shortlist

        self.shortlist_codebook = AdaptiveKmeans(
            num_embed=num_shortlists,
            embed_dim=hidden_size
        )

        self.expert_features = nn.Parameter(
            torch.empty(num_experts, hidden_size)
        )
        nn.init.xavier_uniform_(self.expert_features)

        self._step = 0
        self._cached_step = None
        self._cached_experts_per_shortlist = None

    def on_optimizer_step(self):
        self._step += 1
        self.shortlist_codebook.on_optimizer_step()
        self._cached_step = None

    @torch.no_grad()
    def _get_experts_per_shortlist(self):
        if self._cached_step != self._step:
            centroids = self.shortlist_codebook.get_embedding_weights()
            scores = centroids @ self.expert_features.t()
            top = torch.topk(scores,
                             k=self.experts_per_shortlist,
                             dim=-1)
            self._cached_experts_per_shortlist = top.indices
            self._cached_step = self._step
        return self._cached_experts_per_shortlist

    def forward(self, hidden_states):
        B, L, H = hidden_states.shape
        hidden_states = hidden_states.reshape(-1, H)             # [S, H]

        # 1) Assign tokens to codewords and update the codebook.
        #    The cached expert shortlists are refreshed after optimizer steps.
        shortlist_idx = self.shortlist_codebook(hidden_states)  # [S]

        # 2) Reuse the current cached shortlist-to-expert table.
        experts_per_shortlist = self._get_experts_per_shortlist()     # [G, M]
        cand_experts = experts_per_shortlist[shortlist_idx]      # [S, M]

        # 3) Score candidates
        scores, cand_experts = self._score_candidates_grouped_mm(
            hidden_states,
            shortlist_idx,
            experts_per_shortlist,
        )

        # 4) Select final top-K experts.
        top_scores, top_pos = torch.topk(scores,
                                         k=self.top_k,
                                         dim=-1)
        top_experts = cand_experts.gather(1, top_pos)            # [S, k]
        top_weights = torch.softmax(top_scores, dim=-1)          # [S, k]

        return top_weights, top_experts
```

Listing 1: Simplified PyTorch-like code for the AIR-Router.

D FLOP Analysis

We estimate computational cost using `torch.utils.flop_counter` and extend the default registry with custom formulas for operations that are either not covered or insufficiently modeled by PyTorch. All results in the paper report analytical FLOP counts obtained from traced operator graphs.

Counting convention. We count one floating-point addition, subtraction, multiplication, division, exponential, logarithm, or square root as one FLOP. Elementwise operators therefore contribute one FLOP per output element. Our goal is consistency across methods rather than exact hardware-level instruction counts.

D.1 Elementwise and Reduction Operations

For standard elementwise arithmetic (`add`, `sub`, `mul`, `div`) we count

$$\text{FLOPs} = \text{numel}(x). \tag{6}$$

For common nonlinearities we use fixed per-element costs:

$$\texttt{exp},\ \texttt{log},\ \texttt{sqrt},\ \texttt{rsqrt} : 1, \tag{7}$$

$$\texttt{sigmoid},\ \texttt{silu} : 3, \tag{8}$$

$$\texttt{gelu} : 6. \tag{9}$$

Reductions are approximated as

$$\texttt{mean} : \text{numel}(x) + 1, \tag{10}$$

$$\texttt{sum},\ \texttt{var_mean} : 2\,\text{numel}(x). \tag{11}$$

D.2 Softmax

For softmax over dimension size d_{sm} with $N = \text{numel}(x)$ we use

$$\text{FLOPs} = 2N + \frac{N}{d_{\text{sm}}}, \tag{12}$$

corresponding to one exponential and one division per element plus the reduction cost. The backward pass is approximated as $5\,\text{numel}(x)$ FLOPs.

D.3 Normalization Layers

Let d denote the normalized dimension and V the number of normalized vectors.

LayerNorm.

$$\text{FLOPs} = V\,(4d + d + \mathbb{1}_{\text{weight}}d + \mathbb{1}_{\text{bias}}d). \tag{13}$$

The backward pass is approximated as $8Vd$.

RMSNorm.

$$\text{FLOPs} = V\,(4d + \mathbb{1}_{\text{weight}}d), \tag{14}$$

with the same backward approximation $8Vd$.

BatchNorm. For $N = \text{numel}(x)$ and C channels we use

$$\text{FLOPs} = (2N + 2C) + pN, \tag{15}$$

where $p = 2$ plus optional affine operations.

D.4 Attention

For scaled dot-product attention with query shape (b, h, s_q, d) and key length s_k, we count

$$\text{FLOPs} = 4bhs_q s_k d + 2bhs_q s_k, \tag{16}$$

covering $QK^\top$, softmax, and attention–value multiplication. This formula is applied to both standard and fused attention kernels.

D.5 Routing and Indexing Operations

We also account for routing-related operators.

top-K. For selection over size n with batch size B:

$$\text{FLOPs} = B\,n\log_2(k+1), \tag{17}$$

corresponding to an $O(n \log k)$ approximation.

Gather and scatter. For `gather`, `index_add`, `scatter`, and related operators, we count one unit per affected element:

$$\text{FLOPs} = \text{numel}(\text{affected elements}). \tag{18}$$

E Experimental Setup

E.1 Data

WikiText-103 (Merity et al., 2017) is a language modeling dataset derived from verified *Good* and *Featured* articles on Wikipedia. It contains approximately 103 million tokens in total, with about 103M tokens for training, 217K for validation, and 245K for testing. The dataset preserves the original article structure and long-range dependencies, making it suitable for evaluating long-context language modeling performance.

C5 (Raffel et al., 2020) is a large-scale web text corpus constructed as an extension of the Colossal Clean Crawled Corpus (C4). It contains approximately 87 billion tokens after filtering and deduplication. Similar to C4, the data is obtained from Common Crawl and cleaned using heuristic filtering rules to remove low-quality, boilerplate, and non-natural language content.

OpenWebText2 (**?**Biderman et al., 2022) is a large-scale English web text corpus and a major component of *The OpenWebText2*. It comprises approximately 65 GiB of raw text (corresponding to roughly 15–20 billion tokens), collected from outbound Reddit links and filtered to resemble the distribution of high-quality web text.

E.2 Training

All runs use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1, gradient clipping at 1.0, seed 42, and best-checkpoint selection by validation loss (lower is better).

Dataset	Size/Config	Block	Eff. BS/dev	Eval Strat.	Eval Freq.	LR Sched.	Peak LR	Prec.
OpenWebText2	small	1024	16	steps	500 steps	linear (5% warmup)	3e−4	bf16
OpenWebText2	medium	1024	16	steps	500 steps	linear (5% warmup)	3e−4	bf16
OpenWebText2	large	1024	32	steps	500 steps	linear (5% warmup)	3e−4	bf16
C5	small	1024	16	steps	500 steps	linear (5% warmup)	3e−4	bf16
C5	medium	1024	16	steps	500 steps	linear (5% warmup)	3e−4	bf16
C5	large	1024	32	steps	500 steps	linear (5% warmup)	3e−4	bf16
WikiText-103	small	256	16	epoch	every epoch	linear (5% warmup)	3e−4	bf16
WikiText-103	medium	256	16	epoch	every epoch	linear (5% warmup)	3e−4	bf16

E.3 Architectures

Across all three scales, we maintain a consistent foundational setup to ensure fair comparisons. Every model utilizes the LLaMA-3 tokenizer with a vocabulary size of 128,256. We employ Rotary Positional Embeddings (Su et al. (2024a); RoPE) with a base frequency of $\theta = 500,000$ to handle positional information, and we tie the token embedding weights with the pre-softmax output projection to improve parameter efficiency. Furthermore, each variant leverages Grouped-Query Attention (Ainslie et al. (2023); GQA), optimizing inference throughput without sacrificing representational power.

The primary axes of scaling across our models are depth (number of hidden layers), width (hidden dimension d_{model} and FFN intermediate dimension d_{ffn}), and attention capacity. The specific hyperparameter configurations for each scale are detailed below:

Hyperparameter	**Small**	**Medium**	**Large**
Hidden Dimension (d_{model})	256	512	768
Intermediate Dimension (d_{ffn})	768	1536	2048
Number of Layers	16	24	24
Attention Heads (n_{heads})	4	8	12
Key/Value Heads ($n_{\text{kv_heads}}$)	1	2	4
Vocabulary Size	128,256		
Max Sequence Length	2048		
RoPE Base (θ)	5×10^5		
Tied Embeddings	True		

E.4 Efficiency at Scale Experiments

The following setup is used for Sec. 5.3 and Sec. F.4: Unless stated otherwise, the experiments use the standard configuration corresponding to the large-architecture setting and a batch size of 32 (corresponding to 32768 tokens), with $E = 1024^2$ for the middle panels and $E = 256^2$ for the bottom panel. We use the smaller expert count in the latter case because the standard granular baseline runs out of memory even with routing dimension 2. The experiments are run on a single NVIDIA H100 80GB HBM3 GPU. All router implementations rely exclusively on existing PyTorch operations and *do not use custom kernel implementations*. Further performance improvements may therefore be achievable through dedicated kernel optimization.

E.5 Hyperparameters

E.5.1 Scale-invariant Hyperparameters

AIR-MoE. We set $\gamma = 0.95$, $\tau = 1.0$, jitter standard deviation $\sigma_{\text{jit}} = 0.005$, load balancing weight $\lambda = 10^{-4}$ for C5 and wikitext, and load balancing weight $\lambda = 10^{-5}$ for OpenWebText2.

Hierarchical Router. Analogously to AIR-MoE, we set $\sigma_{\text{jit}} = 0.005$ and apply it at both the cluster selection and expert selection level. We furthermore apply identical load balancing as for AIR-MoE.

Standard Coarse. Across all experiments, we set $\sigma_{\text{jit}} = 0.005$ and use dropless expert-choice gating.

E.5.2 MoE Configurations

To ensure fair and meaningful comparisons across all MoE routing methods, we enforce a set of rules that govern how each method's hyperparameters are set relative to one another. These rules are applied uniformly across all model sizes.

Notation. Let E denote the total number of experts, K the number of experts activated per token (*active-K*), and l the number of coarse-level candidates selected before fine-grained routing (needed for the hierarchical router, AIR-MoE always has $l = 1$). For the PEER router, which maintains P independent routing heads, the effective active-K is $K \cdot P$. In practice, in line with the original paper (He, 2024), we choose $P = 8$ and choose K such that the total amount of active experts matches the amount of active experts of AIR-MoE (that is, $K_{\text{PEER}} = \frac{K_{\text{AIR}}}{P}$). We also use query BatchNorm (Lample et al., 2019) in PEER, as described by He (2024). For each model size, we use the same with a fixed effective active-K budget, specifically $K = 128$.

Choice of M and G for AIR-MoE. As motivated empirically (see Sec. F.7), we generally choose M and G as a function of E according to the following rule

$$G = \frac{\sqrt{E}}{2}, \quad M = \frac{4E}{G}.$$

Cross-method fairness. All methods compared within the same experiment are constrained to be equivalent along three dimensions:

A. Equal active-K. The effective number of experts activated per token is held fixed across all granular methods. For PEER this is $K \cdot P$; for the standard coarse router it is K directly. This ensures that the total compute per token (in terms of expert FLOPs) is comparable.

B. Equal coarse structure. The AIR router uses G codebook shortlists and the hierarchical router uses G clusters. We enforce this to be equal so that the coarseness of the two-stage selection process is matched.

C. Equal candidate pool. Beyond equal active-K, we additionally match the number of candidate experts scored before final selection: $|\mathcal{E}_{\text{shortlist}}|$ for AIR and $l \cdot (E/G)$ for hierarchical routing.

Coarse baseline equivalence. The standard coarse MoE baseline (top-1 routing over large experts) is constructed to be parameter-equivalent to the granular model. Specifically, given granular active-K and E granular experts each of intermediate size 1, the coarse model uses $K = 1$ with $E_{\text{coarse}} = \lceil E/K \rceil$ experts each of intermediate size K. This ensures the active parameter count per forward pass is matched.

E.6 HARDWARE

All experiments are run on an internal HTCondor cluster. The individual experiments are run on the following hardware setups:

Dataset	Size/Config	GPU Name	# GPUs	# CPUs	# processes	host RAM
OpenWebText2	small	NVIDIA H100 80GB HBM3	4	16	4	120
OpenWebText2	medium	NVIDIA H100 80GB HBM3	4	16	4	120
OpenWebText2	large	NVIDIA H100 80GB HBM3	4	32	4	240
C5	small	NVIDIA H100 80GB HBM3	4	16	4	200
C5	medium	NVIDIA H100 80GB HBM3	4	16	4	200
C5	large	NVIDIA H100 80GB HBM3	4	32	4	240
WikiText-103	small	NVIDIA H100 80GB HBM3	4	16	4	120
WikiText-103	medium	NVIDIA H100 80GB HBM3	4	16	4	120

F ADDITIONAL EXPERIMENTAL RESULTS

F.1 MULTI-SEED RUNS

To provide evidence that the performance improvement of AIR-MoE in comparison to existing methods is invariant to the choice of seed, we run two additional seeds for the medium-sized AIR and PEER configurations on C5 and OpenWebText2. The results are given in Table 2.

Table 2: Results across three random seeds for the medium-sized AIR and PEER configurations on C5 and OpenWebText2. Relative improvements are computed with respect to PEER for the corresponding seed.

Seed	Method	PPL ↓	Rel. Impr. ↑
C5			
Seed 42	AIR	**43.85**	**+4.4%**
	PEER	45.88	–
Seed 43	AIR	**24.75**	**+3.0%**
	PEER	25.53	–
Seed 44	AIR	**32.02**	**+1.2%**
	PEER	32.40	–
OpenWebText2			
Seed 42	AIR	**20.64**	**+3.9%**
	PEER	21.49	–
Seed 43	AIR	**21.40**	**+3.0%**
	PEER	22.06	–
Seed 44	AIR	**21.49**	**+3.3%**
	PEER	22.23	–

Across all three seeds and both datasets, AIR consistently outperforms PEER, demonstrating that the observed performance gains are robust to the choice of random seed.

F.2 ADDITIONAL METRICS

The additional metrics, shown in Tab. 3, provide more information about AIR-MoE: It comes at a similar FLOP cost to PEER, but slightly more than the hierarchical router due to the necessity to refresh the shortlist cache.

Table 3: Results across datasets and model sizes.

Method	PPL ↓	FLOPs ↓	Dead Experts ↓	Entropy ↑
Dataset: wikitext-103 — Size: small				
AIR	**19.79**	363.4P	51.3%	11.39
PEER	20.30	364.4P	8.0%	11.02

Continued on next page

Method	PPL ↓	FLOPs ↓	Dead Experts ↓	Entropy ↑
Hierarchical	20.21	**322.8P**	**0.0%**	**12.06**
Std. Coarse	21.79	396.7P	0.1%	6.97
Dataset: wikitext-103 — Size: medium				
AIR	**18.92**	370.7P	19.3%	12.50
PEER	19.64	366.5P	53.1%	10.90
Hierarchical	24.05	364.8P	1.0%	**12.66**
Std. Coarse	21.02	**364.1P**	**0.9%**	7.95
Dataset: c5 — Size: small				
AIR	**546.87**	241.3P	38.8%	8.80
PEER	570.88	241.9P	**0.0%**	**9.45**
Hierarchical	918.47	241.1P	**0.0%**	8.66
Std. Coarse	796.31	**237.0P**	**0.0%**	6.43
Dataset: c5 — Size: medium				
AIR	**43.85**	2.4E	13.7%	10.48
PEER	45.88	2.4E	3.5%	**10.53**
Hierarchical	51.45	**2.3E**	85.5%	8.93
Std. Coarse	49.81	**2.3E**	**0.0%**	7.59
Dataset: c5 — Size: large				
AIR	**46.84**	**7.8E**	16.5%	**10.67**
PEER	47.86	**7.8E**	0.3%	10.03
Hierarchical	51.73	**7.8E**	97.9%	7.75
Std. Coarse	49.06	**7.8E**	**0.0%**	7.35
Dataset: OpenWebText2 — Size: small				
AIR	**49.16**	219.3P	63.8%	9.55
PEER	51.95	219.9P	1.2%	**10.16**
Hierarchical	67.21	219.2P	**0.0%**	9.73
Std. Coarse	56.85	**215.5P**	**0.0%**	6.67
Dataset: OpenWebText2 — Size: medium				
AIR	**20.64**	2.0E	39.1%	10.48
PEER	21.49	2.0E	53.8%	10.02
Hierarchical	25.68	**1.8E**	5.3%	**10.83**
Std. Coarse	22.77	2.0E	**0.0%**	7.69
Dataset: OpenWebText2 — Size: large				
AIR	**18.03**	6.0E	34.4%	10.57
PEER	18.16	6.0E	29.4%	10.19
Hierarchical	20.24	**5.7E**	2.3%	**10.94**
Std. Coarse	18.67	6.0E	**0.0%**	7.42

F.3 Qualitative Differences in Expert Usage

The greatest difference between AIR-MoE and the considered baselines for granular MoE lies in the fact that AIR-MoE does not impose restrictions, as can be seen in Fig. 5: Both PEER and the hierarchical router achieve FLOP-efficient routing via coupling neighboring expert indices via weight sharing (PEER) or grouping (hierarchical). AIR-MoE, in contrast, does not impose such restrictions and therefore provides a more flexible model.

F.4 Additional Efficiency at Scale Experiments

For the experimental setup, see Sec. E.4. The results, which can be seen in Fig. 6, show that AIR-MoE (blue) scales substantially better than the standard granular router (pink) and comparably to the hierarchical and PEER routers across all reported metrics as the microbatch size and routing dimension increase. For the latter, AIR-MoE has slightly lower runtime than PEER.

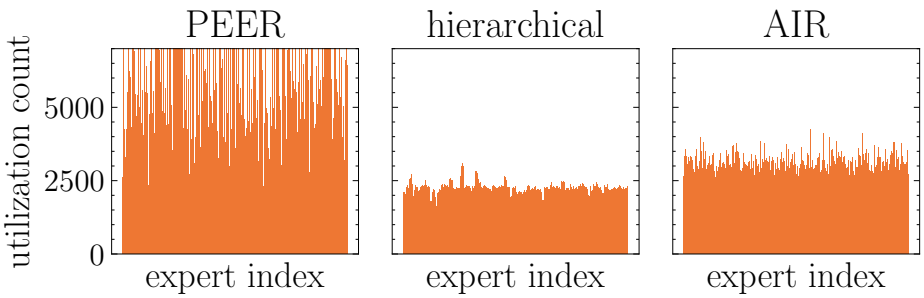

Figure 5: **Qualitative Comparison of Expert Usage.** Existing routing methods impose hard structural constraints on the expert selection mechanism, either via structural constraints (PEER) or grouping constraints. These constraints, in turn, lead to routing artifacts depicted in **(a)** and **(b)**, respectively. AIR-MoE, in contrast, imposes no assumptions by computing an approximation of the top-K expert scores whose quality is based on how well the internal representation cluster naturally.

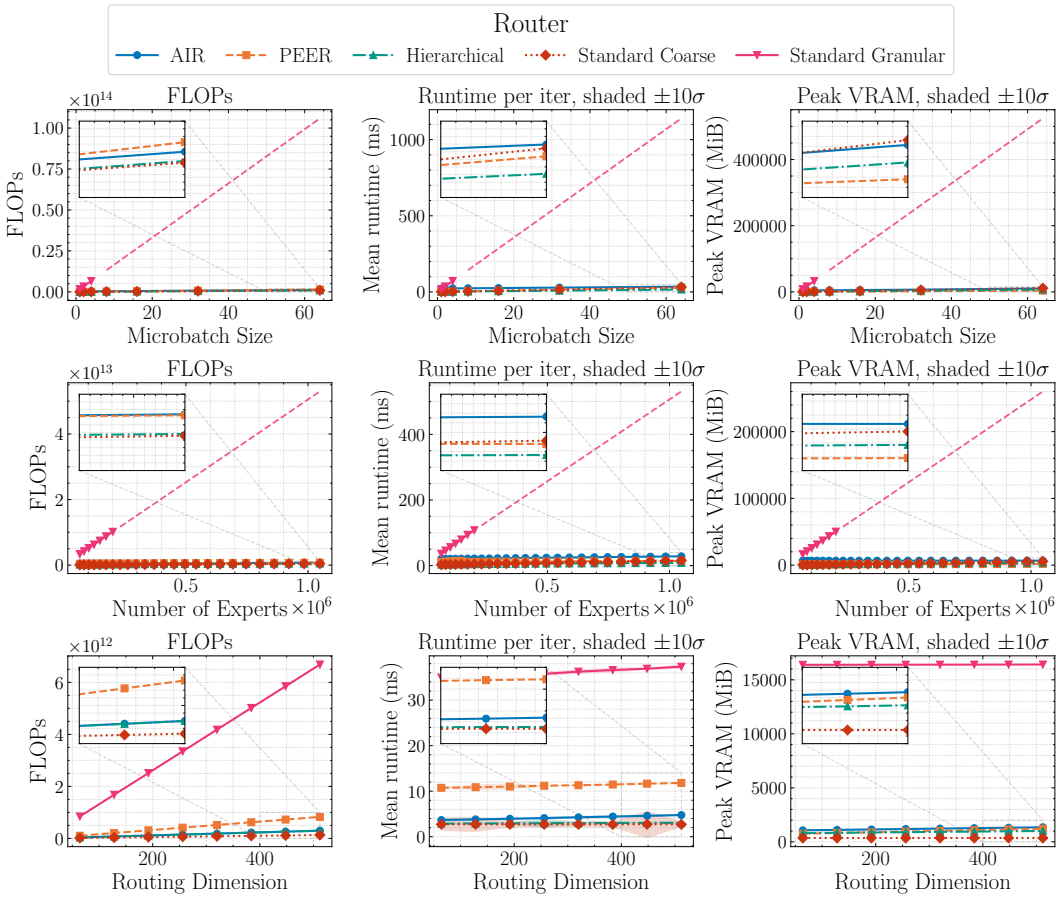

Figure 6: **All Efficiency at Scale Experiments.** Each row shows results of different routers as a function of different parameters $\pm 10\sigma$, where σ is standard deviation. The dashed line for the standard granular baseline corresponds to a linear extrapolation that had to be done due to memory overflow. The plots show that AIR-MoE effectively reduces routing overhead in comparison to the standard granular routing approach beyond being FLOP-efficient.

Table 4: **Ablation results.** Results shown for C5, medium model. The results confirm that the different components of AIR-MoE are relevant.

Ablation	PPL ↓	FLOPs ↓	Dead Experts ↓	Overlap Approx. ↑	Entropy ↑
AIR	**43.85**	**2.4E**	13.7%	0.58	10.48
Cosine	44.11	**2.4E**	**8.4%**	0.65	10.45
Expert Choice	48.77	**2.4E**	68.1%	**0.81**	**11.95**
Missing Normalization	46.20	**2.4E**	85.9%	0.72	9.85
Static Code	47.81	**2.4E**	89.1%	0.05	10.01

F.5 Ablation Studies

We ablate key components of AIR-MoE on the medium model using C5.

Ablations. We consider the following variants: (1) No centroid projection. We remove the projection of expert centroids onto the unit sphere in Alg. 1, thereby decoupling routing from cosine geometry. (2) Non-adaptive codebook. The codebook is initialized from randomly selected training tokens and kept fixed, removing adaptivity to the evolving representation space. (3) Expert-choice gating. We replace standard load balancing (Fedus et al., 2022) with expert-choice routing (Zhou et al., 2022). Experts outside the shortlist receive a routing score of $-\infty$. (4) Cosine similarity assignment. We replace Euclidean k-means with spherical clustering (Dhillon & Modha, 2001) in Alg. 2.

Results. As shown in Tab. 4, AIR-MoE achieves the best perplexity. Removing normalization has some negative effect on perplexity, while having no considerable effect in terms of FLOPs. Using cosine similarity slightly increases perplexity, but slightly increases expert usage and overlap at comparable FLOPs. Expert-choice gating attains the highest overlap, but at the cost of severe underutilization (68.1% dead experts), showing that overlap alone is not a reliable proxy for effective routing. The static codebook achieves the second-worst overall results, confirming the effectiveness of the adaptive codebook.

F.6 Empirical Inspection of Shortlist Overlap

In addition to dynamic grouping of experts, another major difference between AIR-MoE and hierarchical routing architectures (Shazeer et al., 2017) lies in the fact that expert groups are generally overlapping, which we deem to be a major contributor to the improved performance of AIR-MoE in comparison to the hierarchical approach. In this section, we provide some insight into how experts empirically spread across shortlists.

As can be seen in Fig. 7, for the large model trained on C5 with $E = 1024^2$, the distribution over shortlist coverages is highly skewed: while most experts appear only in a small number of shortlists, a smaller number of experts are included much more frequently. This suggests that the learned shortlists can naturally accommodate broadly useful "hot" experts. We view this as a desirable property, since some experts may learn transformations that are useful across multiple regions of the input space. [4] Overlapping shortlists allow such experts to be reused flexibly, without forcing all tokens that benefit from them to pass through the same coarse routing decision. Hierarchical routers, in contrast, impose a more rigid partitioning structure: every expert is assigned to one particular branch and it can only be reached through that branch, which limits the sharing of broadly useful experts across otherwise distinct groups of tokens.

F.7 On the Choice of Codebook Size and Shortlist Size

It is easy to see that increasing the amount of experts per shortlist M has a similar effect to increasing the codebook size G (i.e., the amount of shortlists). In order to find a good trade-off between the two, we set a fixed budget $G \cdot M = 4E$, where we recall that E is the total amount of experts. We then vary the value of M as we assess different values of G.
The results, which can be seen in Fig. 8, show that there exists a trade-off between codebook size and shortlist size: if the codebook is very small, the codebook update step is cheap, but the cost of evaluating token-expert scores increases. If the codebook is too large, however, the shortlist

[4] The importance of such experts has been explored in earlier compression works by Li et al. (2024b;a)

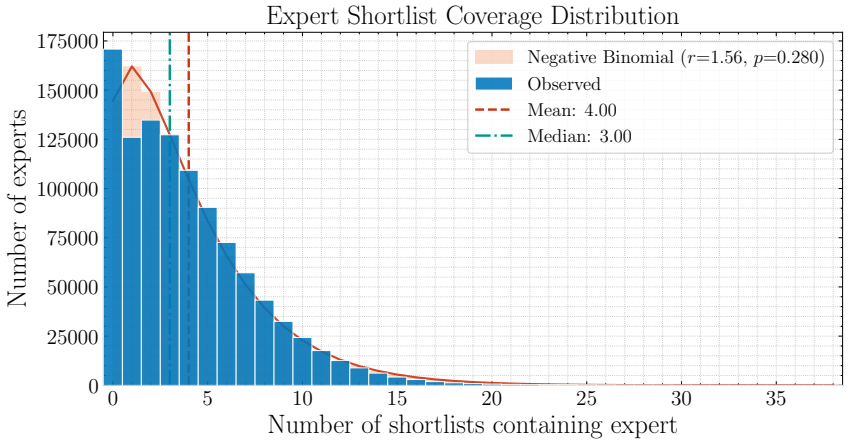

Figure 7: **Shortlist Overlap Analysis.** Histogram about how many times experts are covered my different shortlists. The results shows that the distribution is highly skewed and has a similar shape to a negative binomial distribution (plotted in the background), which reflects the fact that AIR-MoE actively models "hot" experts that represent broadly useful transformations.

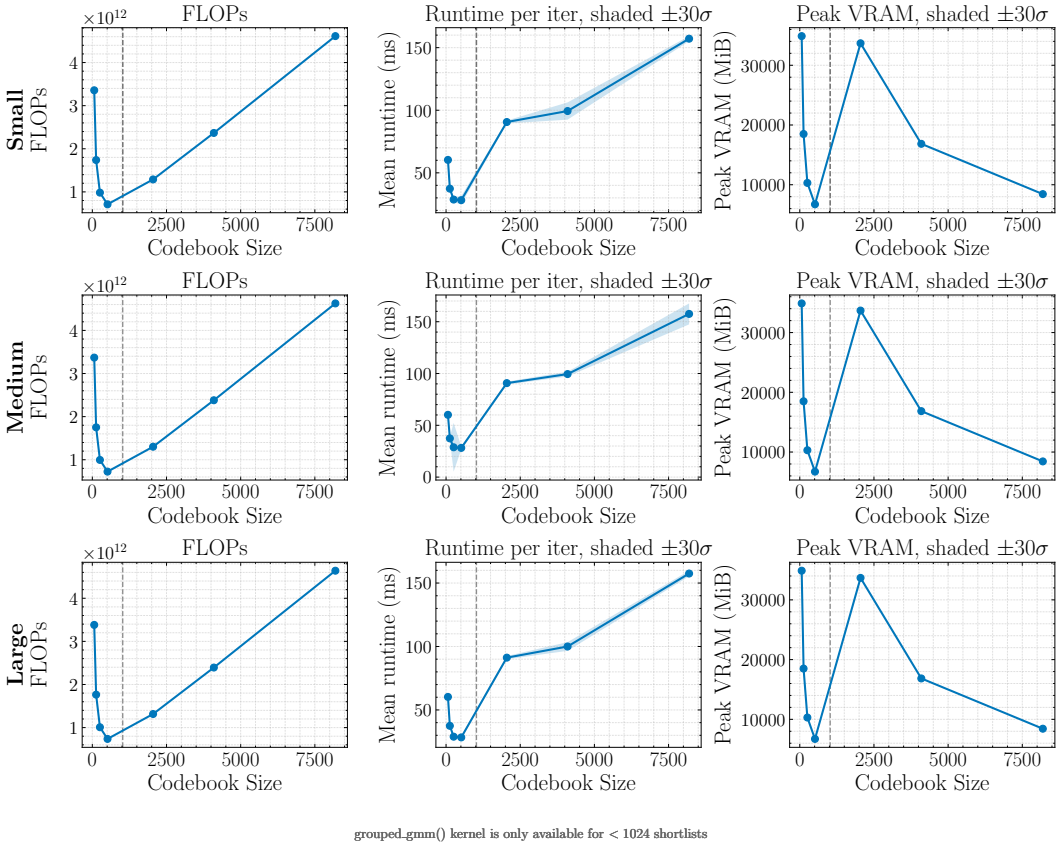

Figure 8: **Performance of AIR-MoE for Different Codebook Sizes.** Each row reports results for a different model size, parameterized by the hidden dimension. For each codebook size G, the shortlist size M is chosen such that $G \cdot M = 4E$. Overall, smaller codebooks yield more efficient routing. The kink at a codebook size of 1024 arises because the `grouped_mm()` kernel supports dimensions only up to 1024. For larger codebooks, we therefore use a more memory-bandwidth-intensive implementation that gathers the corresponding expert centroids separately for each token.

update dominates overall routing cost. In addition, the `grouped_mm()` kernel is only supported for dimensions up to 1024. Therefore, larger codebook sizes are implemented via replicating all expert centroids per token (no grouping), which leads to high memory demand and increased memory latency (leading to the observed kink) for large shortlists. We find that codebook size $G = 512$ for $E = 1024^2$ experts provides the best trade-off among the tested configurations.