Adaptive Inverted-Index Routing for Granular Mixtures-of-Experts

Klaus-Rudolf Kladny[1,2] **Maximilian Mordig**[1,2,3] **Bernhard Schölkopf**[1,2,3,4]

Michael Muehlebach[1]

[1] Max Planck Institute for Intelligent Systems, Tübingen
[2] Tübingen AI Center
[3] ETH Zurich
[4] ELLIS Institute Tübingen
{kkladny,mmordig,bs,michaelm}@tue.mpg.de

Abstract

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. We introduce the *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 efficient granular MoE routers, with consistent perplexity improvements up to 10 % over the best 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 together 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 becomes non-negligible as the amount of experts increase. In recommender systems, this problem is known as maximum inner product search (MIPS; Shrivastava & Li (2014); Abuzaid et al. (2019)):

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

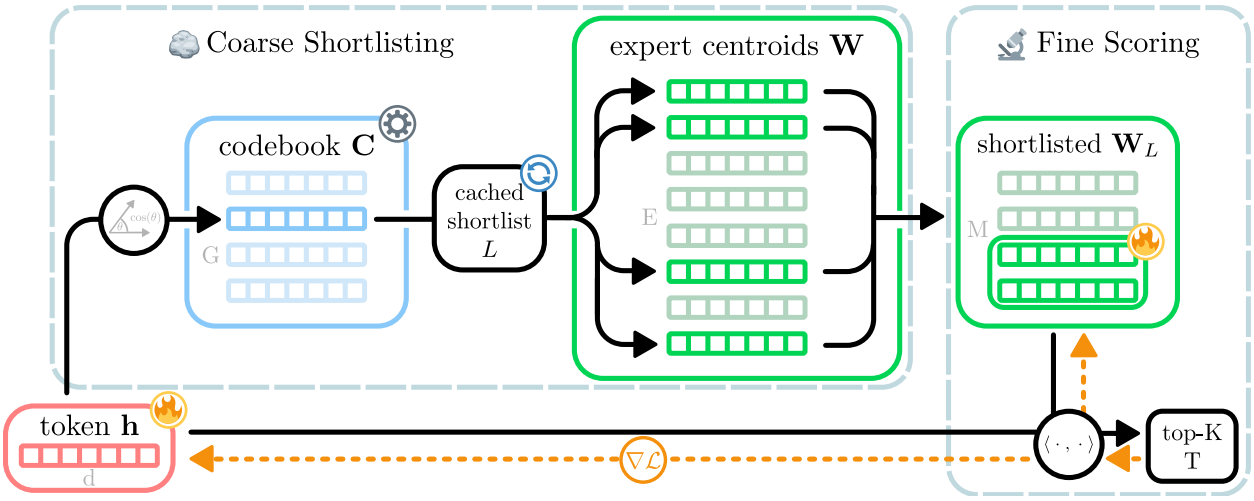

Figure 1: **Overview of AIR-MoE**. The method consists of two parts: The coarse shortlisting stage uses vector quantization to select a codeword (blue) that stores a pre-computed expert shortlist L that references specific expert centroids (green) and is updated after each optimizer step. The fine scoring 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.

find the maximum (or top-K) inner product value(s) between a query vector (token representation) and a large amount 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 costs, they either limit routing flexibility or introduce restrictive assumptions (see Sec. F.2) that typically degrade 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. 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 a FLOP-efficient 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 spherical 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 spherical 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 existing routing methods for granular MoE (Sec. 5).

Overview. Sec. 2 covers the main preliminaries of the manuscript, namely mixtures-of-expert 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:

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

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

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

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,

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

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 $\mathbf{w}_e^\top \mathbf{h}$ across many experts.

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

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 $\epsilon > 0$

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

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

Coarse Shortlisting

for $s \leftarrow 1$ **to** S **do**
  $g_s \leftarrow \arg\max_{g \in [G]} \mathrm{sim}_{\cos}(\mathbf{h}_s, \mathbf{c}_g)$

if *shortlist cache is invalid* **then** `// refresh only after optimizer update`
  for $g \leftarrow 1$ **to** G **do**
    $\boldsymbol{\eta}_1 \sim \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I})$
    $L_g \leftarrow \mathrm{TopM}_{e \in [E]}\big(\langle \mathbf{c}_g, \mathbf{w}_e \rangle + \boldsymbol{\eta}_1\big)$

Fine Scoring

for $s \leftarrow 1$ **to** S **do**
  $\boldsymbol{\eta}_2 \sim \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I})$
  $T_s, \mathbf{z}_s \leftarrow \mathrm{TopK}_{e \in L_{g_s}}\big(\langle \mathbf{h}_s, \mathbf{w}_e \rangle + \boldsymbol{\eta}_2\big)$ `// gradients do not pass to` $\mathbf{C}$

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

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

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).

3.1 Forward Pass

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. To obtain a shortlist for an input token $\mathbf{h}_s \in \mathbb{R}^d$ within a batch of tokens $\mathbf{H} \coloneqq \{\mathbf{h}_s\}_{s=1}^{S}$, we employ vector quantization (Gray, 1984; Linde et al., 1980). 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 cosine similarity for the mapping:

$$g_s \coloneqq \operatorname*{argmax}_{g \in [G]} \mathrm{sim}_{\cos}(\mathbf{h}_s, \mathbf{c}_g), \qquad \mathrm{sim}_{\cos}(\mathbf{x}, \mathbf{y}) \coloneqq \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \, \|\mathbf{y}\|}. \tag{5}$$

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 using the same rule (5), after each optimizer step by scanning the entire set of experts. 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 this 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, making the arithmetic cost independent of the number of routed tokens for a fixed optimizer step, and therefore increasingly favorable in large effective-batch regimes.

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

Fine Scoring. 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 mixture weights, respectively.

Implementation Details. To prevent experts from starvation, we employ switch transformer style load balancing and jitter (Fedus et al., 2022). In order to encourage exploration both on shortlist and token level, we apply jitter twice to each stage (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 spherical k-means algorithm. We describe both components below and provide the full optimization loop in Sec. A.

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 spherical k-means algorithm (Alg. 2; adaptation of McQueen (1967)) to minimize the expected quantization error, decoupled from the training of model parameters. The proposed algorithm (Alg. 2) combines three ideas:

1. Exponential moving average (Van Den Oord et al. (2017); EMA; Alg. 2: lines 7, 8): 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. Spherical k-means (Dhillon & Modha (2001); Alg. 2: lines 2, 13): Spherical k-means updates codewords and data points after projecting them onto the unit sphere $\mathbb{S}^d$. We adopt this technique as it is well-known to yield better results in high dimensions (e.g., Lelu & Cadot (2019)).
3. Reinitialization of inactive codes (Williams et al. (2020); Alg. 2: line 11): 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?

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 distribution

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

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

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

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

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

for $s \leftarrow 1$ **to** S **do**
  $\mathbf{h}'_s \leftarrow \Pi_{\mathbb{S}^d}(\mathbf{h}_s)$ // normalize token state
  $g_s \leftarrow \arg\max_{g \in [G]} \mathrm{sim}_{\cos}(\mathbf{h}'_s, \mathbf{c}_g)$ // assign token to nearest codeword
for $g \leftarrow 1$ **to** G **do**
  $n_g^{\text{batch}} \leftarrow \sum_{s=1}^{S} \mathbf{1}[g_s = g]$ // batch assignment count
  $\mathbf{m}_g^{\text{batch}} \leftarrow \sum_{s=1}^{S} \mathbf{1}[g_s = g]\, \mathbf{h}'_s$ // batch sum of assigned token states
  $n_g \leftarrow \gamma n_g + (1 - \gamma) n_g^{\text{batch}}$ // EMA update of counts
  $\mathbf{m}_g \leftarrow \gamma \mathbf{m}_g + (1 - \gamma) \mathbf{m}_g^{\text{batch}}$ // EMA update of sums
  if $n_g < \tau$ **then**
    $u \sim \mathrm{Unif}([S])$ // sample replacement token
    $\mathbf{m}_g \leftarrow \mathbf{h}'_u$
    $n_g \leftarrow 1$
  $\mathbf{c}_g \leftarrow \Pi_{\mathbb{S}^d}(\mathbf{m}_g)$ // update codeword & normalize
return $\mathbf{C}, n, \mathbf{M}$

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). *Let* $\epsilon(\mathbf{h}) \coloneqq \|\mathbf{h} - c(\mathbf{h})\|$ *and let* $L(c(\mathbf{h}))$ *denote the top-*M *experts under* $\langle c(\mathbf{h}), \mathbf{w}_e \rangle$. *Then*

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

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 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. One line of research by Roller et al. (2021); Nogueira Dos Santos et al. (2024) uses fixed (i.e., not learned) hash-table routers in order to increase routing efficiency, though such approaches have been shown both empirically and theoretically to be fundamentally limited (Clark et al., 2022; Dikkala et al., 2023). One more recent method by He (2024) aims at reducing routing overhead, while keeping routers learnable: The proposed router architecture uses the product-key retrieval method (Lample et al., 2019) to split expert centroids into two parts (which can be evaluated separately), reducing routing overhead from $\mathcal{O}(SEd)$ to $\mathcal{O}(Sd(\sqrt{E} + K^2))$. Just as the present work, He (2024) use a two-stage retrieval method to score experts. In contrast to the present work, the approach by He (2024) allows for *exact* retrieval of the top-K

experts, while AIR-MoE is an *approximation* of the top-K. However, He (2024) pay the price of *imposing a specific structure* on the expert centroids $\mathbf{W}$, which restricts the space of admissible expert representations: $\mathbf{W}$ is computed as a Cartesian product over two prototypes. The expert centroids of AIR-MoE, in contrast, are not limited and can therefore model token-expert dependencies more flexibly. Another difference is that the method by He (2024) scales quadratically in the amount of active experts k, which necessitates modifying the MoE architecture to compute expert outputs over multiple heads with smaller k for compensation. AIR-MoE, in contrast, does not require such modifications of the routing architecture and is therefore simpler to use as a drop-in replacement for existing MoE architectures.

Grouping in MoE. The present work can be categorized into a broader class of approaches based on grouping experts, either with overlapping (Su et al., 2024b; Tang et al., 2025) or non-overlapping groups (Jordan & Jacobs, 1991; 1994; Shazeer et al., 2017). Similar to the present work, Su et al. (2024b); Tang et al. (2025); Xie et al. (2023) generate groups of experts. However, these groups are not created to reduce computational cost, but instead as a means for avoiding load imbalance Su et al. (2024b); Tang et al. (2025) or "overfitting" (Xie et al., 2023). Hierarchical MoEs (Jordan & Jacobs, 1991; 1994; Shazeer et al., 2017) also employ non-overlapping groups to reduce computation. In contrast to AIR-MoE, however, hierarchical MoEs partition experts into fixed sub-groups, restricting routing to experts within the selected groups. While this avoids the need to recompute codebook–expert scores (Alg. 1, line 7), it limits expressivity by preventing globally reusable experts. Furthermore, hierarchical MoEs typically learn group selection through additive group and expert scores, whereas AIR-MoE forms groups via vector quantization.

Vector Quantization in MoE. A prior work that studies vector quantization in MoE is by Do et al. (2024). Like AIR-MoE, Do et al. (2024) employ vector quantization to encode token vectors. In contrast to the present work, however, Do et al. (2024) do not use the codewords to create a retrieval index. Instead, vector quantization is used to score experts. Thus, the method by Do et al. (2024) comes with no computational benefits and does not serve the purpose of granular MoE.

Inverted File Indices & Differentiable Retrieval. Our approach 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 via a coarse-to-fine strategy. A coarse quantizer partitions the space, and queries are restricted to a small number of cells before performing exact scoring within the corresponding candidate sets. AIR-MoE follows the same principle for mixture-of-experts routing: learned shortlist embeddings act as coarse centroids, and each shortlist induces a shortlist of candidate experts that are re-scored exactly. In contrast to classical IVF systems, which are typically constructed offline over a fixed database, our method is trained end-to-end and jointly learns both the quantization and expert representations in an adaptive fashion. Recent work has therefore begun to blur the boundary between indexing and retrieval by learning index representations jointly with the retrieval model (Wang et al., 2022; Zhuang et al., 2022). Our method contributes to 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 coarse MoE architectures? **(3)** How do the different components of AIR-MoE affect performance?

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

5.1 Setup

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

Table 1: **Main results.** Perplexity (PPL) and training FLOPs for models with 65,536 experts across different model sizes and datasets. FLOPs correspond to the amount of FLOPs at PPL minimum. Best results are highlighted in bold. AIR-MoE consistently achieves a favorable trade-off compared to existing granular MoE approaches, as well as the coarse approach. While the standard granular baseline attains the lowest PPL, it incurs prohibitive computational cost and is therefore shown in gray. All other metrics are shown in Sec. F.1

		WikiText-103		C5		OpenWebText2	
Size	Method	PPL ↓	FLOPs ↓	PPL ↓	FLOPs ↓	PPL ↓	FLOPs ↓
Small	AIR	**21.82**	**324.1P**	**131.81**	625.5P	**32.14**	505.0P
	PEER	22.25	352.8P	145.32	626.3P	33.10	505.6P
	Hierarchical	22.55	350.4P	147.99	622.6P	36.05	502.6P
	Std. Granular	21.34	467.5P	132.34	817.2P	31.56	648.3P
	Std. Coarse	23.84	342.5P	151.99	**611.7P**	36.05	**493.9P**
Medium	AIR	**18.62**	755.9P	**30.39**	4.1E	**20.51**	3.6E
	PEER	18.71	**751.0P**	31.60	4.1E	21.30	**3.5E**
	Hierarchical	19.27	954.7P	33.08	4.1E	23.07	**3.5E**
	Std. Coarse	18.79	836.7P	32.10	**4.0E**	21.25	**3.5E**
Large	AIR	–	–	**41.25**	14.3E	**16.65**	**11.2E**
	PEER	–	–	45.37	14.0E	17.88	**11.2E**
	Hierarchical	–	–	45.65	14.4E	18.17	11.3E
	Std. Coarse	–	–	43.13	**13.2E**	16.98	11.3E

Datasets. We evaluate on WikiText-103 (Merity et al., 2017), C5 (Common Crawl Foundation, 2025), and OpenWebText2 (Gao et al., 2020).

Baselines. (1) Coarse MoE with $K = 1$ (matched active parameters), (2) granular MoE with standard routing, (3) granular PEER routing (He, 2024), (4) granular hierarchical MoE (Shazeer et al., 2017).

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 (≈ 61M parameters), medium (≈ 0.27B parameters), and large (≈ 0.45B parameters) 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.4) and align baselines by active parameters (details in Sec. E.4.2). For WikiText-103, we train for 10 epochs. For C5 and OpenWebText2, we use 10× the amount of 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 efficient granular MoE baselines across datasets and model sizes.[4] While AIR-MoE performs slightly worse than the standard granular approach, which scores all expert scores exactly, the latter is substantially more expensive in terms of training FLOPs; for

[4]On C5, the large models obtain worse PPL than the medium models across all routing methods. Inspection of the validation curves suggests that this is an optimization effect under the fixed token budget and learning-rate schedule.

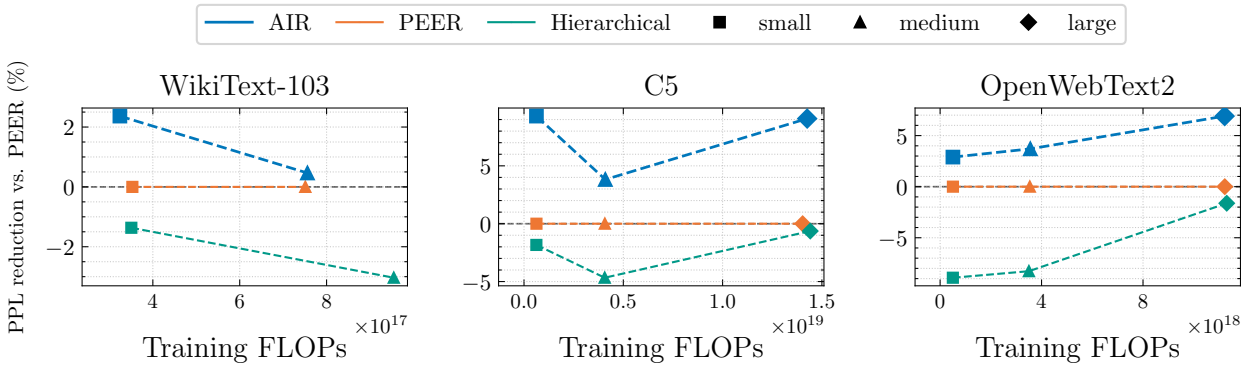

Figure 2: **Relative Reduction in PPL.** The plots show the relative improvement in PPL with respect to the best baseline (PEER). Larger means better. AIR-MoE achieves consistent PPL improvements (up to 10%) in comparison to the PEER baseline, across model sizes and data sets.

Table 2: **Ablation results.** Results shown for WikiText-103, small model. The results confirm that the different components of AIR-MoE are relevant.

Ablation	PPL ↓	FLOPs ↓	Dead Experts ↓	Overlap ↑	Entropy ↑
AIR	**21.72**	**324.1P**	**0.0%**	0.64	10.80
Euclidean	21.83	**324.1P**	0.3%	0.64	**10.81**
Expert Choice	22.77	357.5P	78.6%	**0.74**	9.53
Missing Normalization	21.96	**324.1P**	0.7%	0.66	10.73
Static Code	23.27	360.1P	42.4%	0.07	10.24

example, on WikiText-103 Small, standard granular routing requires 467.5P FLOPs compared to 324.1P for AIR-MoE. Compared to hierarchical and PEER routers, AIR-MoE achieves persistent PPL reductions of up to 10% relative to PEER at similar FLOP cost (Fig. 2).

5.3 Ablation Studies

We ablate key components of AIR-MoE on the small model using WikiText-103.

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) Euclidean assignment. We replace spherical k-means with Euclidean clustering by removing normalization in Alg. 2.

Results. As shown in Tab. 2, AIR-MoE achieves the best perplexity and lowest fraction of dead experts. Removing normalization has some negative effect on perplexity, while having no considerable effect in terms of FLOPs. Using Euclidean distance similarly slightly decreases perplexity and expert usage at comparable FLOPs. Expert-choice gating attains the highest overlap, but at the cost of severe under-utilization (78.6% dead experts), showing that overlap alone is not a reliable proxy for effective routing. The static codebook achieves the worst overall results, confirming the effectivity of the adaptive codebook.

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 FLOP savings depend strongly on G, E, and the effective batch size S.

Beyond Router FLOPs. Following prior work (He, 2024), this work primarily studies the trade-off between perplexity and training FLOPs. We emphasize, however, that FLOP reductions alone do not guarantee proportional wall-clock speedups without appropriate hardware support and optimized implementations (Fedus et al., 2022; Lepikhin et al., 2021). This is particularly relevant in the granular MoE regime, where using a large number of experts can introduce substantial router-independent overhead due to memory traffic, indexing, and irregular expert usage. Our current implementation does not fully eliminate these systems-level costs. Bridging this gap through hardware-aware implementations and optimized kernels is an important direction for future work.

Dimensionality Reduction. AIR-MoE works in the original token space without dimensionality reduction. Another orthogonal direction is to reduce the dimensionality of tokens either via learned (e.g., Chi et al. (2022)) or randomized (e.g., Achlioptas et al. (2001)) dimensionality reduction before feeding them into the router.

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 FLOP-efficient routing 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 mixture-of-experts architectures.

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

Dimitris Achlioptas, Frank McSherry, and Bernhard Schölkopf. Sampling Techniques for Kernel Methods. *Advances in Neural Information Processing Systems*, 14, 2001. 10

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. 21

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. 20

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Bo Li, Saksham Singhal, Prakhar Bajaj, Xia Song, and Furu Wei. On the Representation Collapse of Sparse Mixture of Experts. *Advances in Neural Information Processing Systems*, 2022. 10

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. 8

Inderjit S. Dhillon and Dharmendra S. Modha. Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1):143–175, 2001. 5

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. 7

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. 5, 9, 10

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. 8, 20

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. 4

Xu Owen He. Mixture of a Million Experts. *arXiv preprint arXiv:2407.04153*, 2024. 2, 6, 7, 8, 10, 21

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. 7

Michael I. Jordan and Robert A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2):181–214, 1994. 7

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

Alain Lelu and Martine Cadot. Evaluation of text clustering methods and their dataspace embeddings: an exploration. *International Federation of Classification Societies*, pp. 131–139, 2019. 5

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. *International Conference on Learning Representations*, 2021. 10

Yoseph Linde, Andres Buzo, and Robert Gray. An Algorithm for Vector Quantizer Design. *IEEE Transactions on communications*, 28(1):84–95, 1980. 4

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. 8, 20

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. 5

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. 20

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, 2, 3, 7, 8

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. 21

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. 7

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. 7

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. 7

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. 9

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

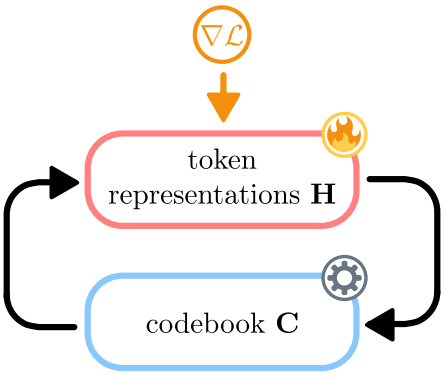

Figure 3: **Feedback Loop**. AIR-MoE forms a bi-level optimization loop between gradient-learned token representations and a codebook updated via adaptive clustering.

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 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 during the forward pass. Instead, the forward pass uses the currently cached shortlists. The cache is refreshed only after an optimizer step, using the updated expert centroids and the updated codebook.

B Proof for Prop. 1

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

$$\epsilon(\mathbf{h}) \;\coloneqq\; \|\mathbf{h} - \mathbf{c}\|.$$

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 micro-batch`

    Compute token representations $\mathbf{H}_t = \{\mathbf{h}_s\}_{s=1}^{S}$ using parameters θ_t

    `// gradient-free codebook update`

    $(\mathbf{C}_t, n_t, \mathbf{M}_t, \{g_s\}_{s=1}^{S}) \leftarrow \textsc{AdaptiveSphericalKMeans}(\mathbf{H}_t, \mathbf{C}_t, n_t, \mathbf{M}_t)$

    `// codeword assignment`

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

      $g_s \leftarrow \arg\max_{g \in [G]} \mathrm{sim}_{\cos}(\mathbf{h}_s, \mathbf{c}_g)$

    `// 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**

        $\boldsymbol{\eta}_1 \sim \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I})$

        $L_g \leftarrow \mathrm{TopM}_{e \in [E]}(\langle \mathbf{c}_g, \mathbf{w}_e \rangle + \boldsymbol{\eta}_1)$ `// cache integer expert indices`

    `// fine scoring within cached shortlists`

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

      $\boldsymbol{\eta}_2 \sim \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I})$

      $T_s, \mathbf{z}_s \leftarrow \mathrm{TopK}_{e \in L_{g_s}} \left(\langle \mathbf{h}_s, \mathbf{w}_e \rangle + \boldsymbol{\eta}_2 \right)$

    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. Sec. C shows a simplified PyTorch-style implementation of the proposed AIRRouter. The goal of the router is to reduce routing cost by restricting the set of experts considered for each token using a vector-quantization (VQ) based coarse-to-fine selection strategy.

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 a discrete shortlist index per token as well as a vector quantization loss that encourages balanced usage of shortlists and stable codebook learning.

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. Each token is only scored against experts belonging to its selected shortlist. This avoids computing scores for all experts and reduces complexity from $\mathcal{O}(TE)$ to $\mathcal{O}(TM)$, where T is the number of tokens, E the total number of experts, and $M \ll E$.

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, Sec. C omits several implementation details that are present in the full version used in experiments:

- load-balancing losses for experts and shortlists,
- expert-choice routing with capacity constraints,
- jitter noise for exploration,
- caching of shortlist-to-expert assignments,
- chunked computation for memory efficiency,
- overlap estimation with exact routing,
- logging and diagnostic metrics.

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):
    """Ultra-minimal readable version."""
    def __init__(self, hidden_size, num_experts, num_shortlists,
                 top_k=2, experts_per_shortlist=128):
        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)

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

        # ---- 1) Select shortlist per token (VQ) ----
        shortlist_idx = self.shortlist_codebook(hidden_states)

        # ---- 2) Top experts per shortlist ----
        shortlist_centroids = self.shortlist_codebook.get_embedding_weights()
        shortlist_expert_scores = shortlist_centroids @ self.expert_features.t()
        top = torch.topk(shortlist_expert_scores,
                         k=self.experts_per_shortlist, dim=-1)
        top_experts_per_shortlist = top.indices               # [S, M]

        # ---- 3) Score token against shortlist candidates ----
        shortlist_idx = shortlist_idx.view(-1)
        cand_experts = top_experts_per_shortlist[shortlist_idx]
        cand_features = self.expert_features[cand_experts]    # [T, M, H]
        scores = torch.einsum("th,tmh->tm", hidden_states, cand_features)

        # ---- 4) 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)         # [T, k]

        # ---- 5) Normalize routing weights ----
        top_weights = torch.softmax(top_scores, dim=-1)       # [T, 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:

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

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 (Gao et al., 2020; Biderman et al., 2022) is a large-scale English web text corpus and a major component of *The Pile*. 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). Logging is performed every 500 steps, with at most 2 checkpoints retained.

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	fp16
OpenWebText2	medium	1024	16	steps	500 steps	linear (5% warmup)	3e−4	fp16
OpenWebText2	large	1024	32	steps	500 steps	linear (5% warmup)	3e−4	fp16
C5	small	1024	16	steps	500 steps	linear (5% warmup)	3e−4	fp16
C5	medium	1024	16	steps	500 steps	linear (5% warmup)	3e−4	fp16
C5	large	1024	32	steps	500 steps	linear (5% warmup)	3e−4	fp16
WikiText-103	small	256	16	epoch	every epoch	linear (5% warmup)	3e−4	fp16
WikiText-103	medium	256	16	epoch	every epoch	linear (5% warmup)	3e−4	fp16

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 and is trained with a maximum sequence length of 2048 tokens. 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) with a 4:1 ratio of query heads to key/value heads, 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 Hyperparameters

E.4.1 Scale-invariant Hyperparameters

AIR-MoE. Across all experiments, we set $\gamma = 0.95$, $\tau = 1.0$, load balancing weight $\lambda = 5 \cdot 10^{-5}$ and noise variance $\epsilon = 0.01$. We stress that these parameters are used *across all experiments* without separate tuning.

Hierarchical Router. Analogously to AIR-MoE, we set $\epsilon = 0.01$ and apply it at both the cluster selection and expert selection level.

Standard Granular. Across all experiments, we set $\epsilon = 0.01$ and use dropless expert-choice gating.

Standard Coarse. Across all experiments, we set $\epsilon = 0.01$ and use dropless expert-choice gating.

E.4.2 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 the total amount of active experts matches the amount of active experts of AIR-MoE (that is, $K_{\text{PEER}} = \frac{K_{\text{AIR}}}{P}$). For each model size, we use the same with a fixed effective active-K budget, specifically $K = 512$.

Intra-method constraints. Each router must satisfy a set of structural feasibility conditions:

- **Adaptive Inverted-Index (AIR) router.** The candidate pool must cover at least K experts: $l \cdot |\mathcal{E}_{\text{shortlist}}| \geq K$, where $|\mathcal{E}_{\text{shortlist}}|$ is the number of experts per VQ code. Additionally, no code may cover more experts than exist: $|\mathcal{E}_{\text{shortlist}}| \leq E$.
- **Hierarchical router.** Experts must partition evenly into G clusters: $E \bmod G = 0$. The selected clusters must suffice to cover K active experts: $l \cdot (E/G) \geq K$.

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 all other methods it is K directly. This ensures that the total compute per token (in terms of expert FLOPs) is comparable.

B. Equal coarse structure. The VQ 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: $l \cdot |\mathcal{E}_{\text{shortlist}}|$ for VQ 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 d, the coarse model uses $k = 1$ with $E_{\text{coarse}} = \lceil E/k \rceil$ experts each of intermediate size $d \cdot k$. This ensures the active parameter count per forward pass is matched.

E.5 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 A100-SXM4-80GB	4	16	4	120
OpenWebText2	medium	NVIDIA A100-SXM4-80GB	4	16	4	120
OpenWebText2	large	NVIDIA H100 80GB HBM3	4	32	4	240
C5	small	NVIDIA A100-SXM4-80GB	4	16	4	200
C5	medium	NVIDIA A100-SXM4-80GB	4	16	4	200
C5	large	NVIDIA H100 80GB HBM3	4	32	4	240
WikiText-103	small	NVIDIA A100-SXM4-80GB	4	16	4	120
WikiText-103	medium	NVIDIA A100-SXM4-80GB	4	16	4	120

F Additional Experimental Results

F.1 Additional Metrics

The additional metrics, shown in Tab. 3, provide more information about AIR-MoE: the dying expert problem is effectively alleviated and routing entropy is similar to the routing entropy of other techniques.

Table 3: Results across datasets and model sizes.

Method	PPL ↓	FLOPs ↓	Dead Experts ↓	Entropy ↑
Dataset: C5 | Size: small				
AIR	**131.81**	625.8P	0.1%	9.51
PEER	145.32	626.5P	**0.0%**	9.96
Hierarchical	147.99	622.8P	**0.0%**	**10.15**
Std. Coarse	151.99	**612.0P**	**0.0%**	3.90
Dataset: C5 | Size: medium				
AIR	**30.39**	4.1E	**0.0%**	10.09
PEER	31.60	4.1E	**0.0%**	10.16
Hierarchical	33.08	4.1E	**0.0%**	**11.04**
Std. Coarse	32.10	**4.0E**	**0.0%**	4.16
Dataset: C5 | Size: large				
AIR	**41.25**	14.3E	**0.0%**	9.98
PEER	45.37	14.0E	**0.0%**	9.95
Hierarchical	45.65	14.4E	**0.0%**	**11.05**
Std. Coarse	43.13	**13.2E**	**0.0%**	4.16
Dataset: Wikitext-103 | Size: small				
AIR	**21.82**	**324.1P**	0.1%	10.78
PEER	22.25	352.9P	**0.0%**	10.76
Hierarchical	22.55	350.5P	**0.0%**	**10.97**
Std. Coarse	23.84	342.6P	**0.0%**	4.28
Dataset: Wikitext-103 | Size: medium				
AIR	**18.62**	756.0P	**0.0%**	10.84
PEER	18.71	**751.2P**	**0.0%**	10.68
Hierarchical	19.27	954.8P	**0.0%**	**11.04**
Std. Coarse	18.79	836.8P	**0.0%**	4.51
Dataset: OpenWebText2 | Size: small				
AIR	**32.14**	505.0P	1.4%	10.02
PEER	33.10	505.6P	**0.0%**	10.25
Hierarchical	36.05	502.6P	**0.0%**	**10.48**
Std. Coarse	36.05	**493.9P**	**0.0%**	4.11
Dataset: OpenWebText2 | Size: medium				
AIR	**20.51**	3.6E	**0.0%**	10.32
PEER	21.30	**3.5E**	**0.0%**	10.19
Hierarchical	23.07	**3.5E**	**0.0%**	**11.00**
Std. Coarse	21.25	**3.5E**	**0.0%**	4.24
Dataset: OpenWebText2 | Size: large				
AIR	**16.65**	**11.2E**	**0.0%**	10.33
PEER	17.88	**11.2E**	**0.0%**	10.18
Hierarchical	18.17	11.3E	**0.0%**	**11.01**
Std. Coarse	16.98	11.3E	**0.0%**	4.27

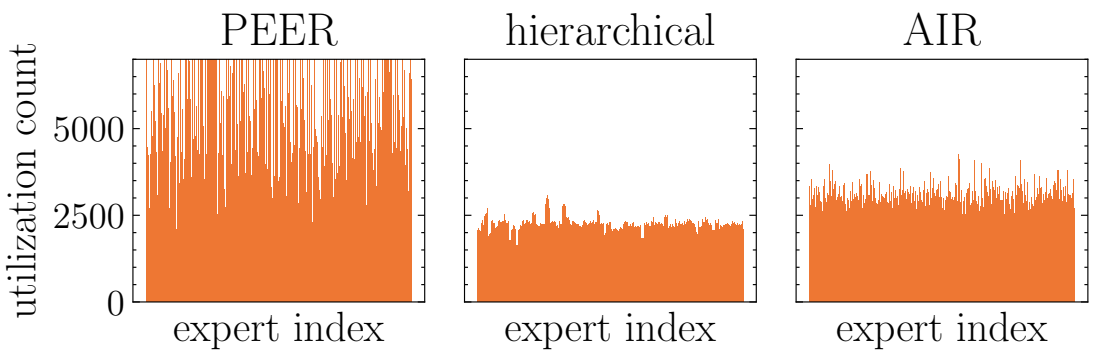

Figure 4: **Qualitative Comparison of Expert Usage.** Existing routing methods impose hard structural constraints on the expert selection mechanism, either via structural constraints (PEER) or grouping constraints. These constraints, in turn, lead to routing artifacts depicted in **(a)** and **(b)**, 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.

F.2 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. 4: 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.