

IntBMoE: Integrating Block-Level Conditioning into Expert Composition for Full-Participation Mixture-of-Experts

Ran Cheng*, Longfei Xu*, Zheng Liu*, Kaikui Liu, Xiangxiang Chu
DreamX, Alibaba Group

Abstract

Mixture-of-Experts (MoE) scales capacity, but existing designs cannot set three quantities independently. For a single token, **participation** is how many experts contribute knowledge to its output, **execution** is how many are actually computed (compute cost), and **materialization** is how many expert-sized parameter sets must be built and stored (memory cost). Sparse routing keeps execution and materialization low, but shrinks participation: for each token, only a few experts contribute. Dense output-mixing restores full participation, but its execution grows with the number of experts. Parameter-merging keeps execution at one expert, but its materialization grows with the number of routing decisions.

We propose IntBMoE, a block-conditioned MoE that decouples all three by pairing dense expert composition with sparse block execution. Its blocks come from a small learned codebook, one per entry. At each internal layer, a lightweight hypernetwork merges all expert bases in that layer’s pool into one composed expert. **Participation is full**, because every composed expert draws on the entire pool. **Execution stays sparse**, because a router sends each token to only a few blocks. **Materialization is bounded**, because the codebook, not the input, fixes how many blocks exist. Dual-Path Residual Gating (DPRG) further couples two independently composed paths through multiplicative gating.

Experiments on image classification show consistent gains over representative sparse and dense MoE baselines. Additional experiments on language modeling and sequential recommendation validate its generalization beyond vision. IntBMoE is fully deployed in AMap’s generative recommendation system, serving hundreds of millions of users under a 60 ms latency budget, with a 2.4% relative UVCTR gain in online A/B testing. Our code is available at <https://github.com/AMAP-ML/DreamX-Rec/>.

1 Introduction

Mixture-of-Experts (MoE) architectures have been widely adopted in language models Lepikhin et al. (2020); Fedus et al. (2022); Jiang et al. (2024); Dai et al. (2024), vision models Riquelme et al. (2021); Fan et al. (2022), multimodal models Mustafa et al. (2022); Xue et al. (2023); Li et al. (2024), and recommendation systems Deng et al. (2025); Zhu et al. (2025). A standard MoE layer contains a pool of experts and a router that determines how the experts process each token. Existing designs follow three main strategies. *Sparse-routing methods* execute only a small number of selected experts Shazeer et al. (2017); Lepikhin et al. (2020); Fedus et al. (2022); Dai et al. (2024). *Dense output-mixing methods* execute every expert and combine their outputs Ma et al. (2018); Tang et al. (2020). *Parameter-merging methods* combine the parameters of all experts into a single composite expert and then execute it Muqeeth et al. (2023); Zhong et al. (2024).

*Equal contribution.

†Project lead.

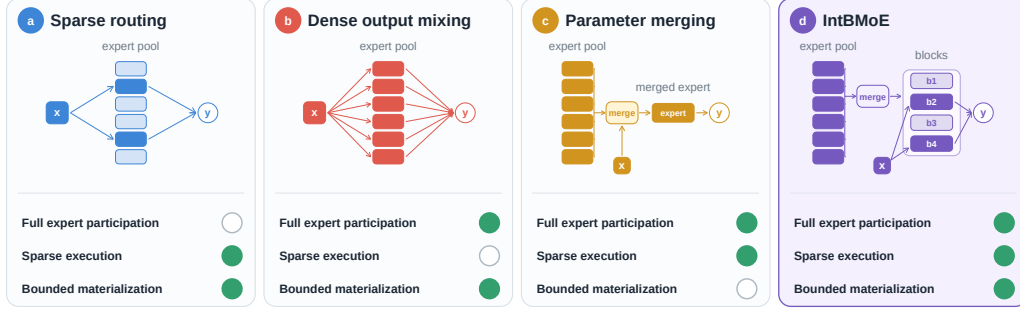


Figure 1: Comparison of MoE design strategies. Filled circles indicate that a strategy satisfies the corresponding property, while hollow circles indicate otherwise.

We analyze these approaches along three dimensions. **Participation:** how many experts contribute knowledge to a token’s output? **Execution:** how many of them must actually be computed, and so how much compute? **Materialization:** how many expert-sized parameter sets must be built and stored (one for every distinct routing decision), and so how much memory? In principle the three are independent, yet existing designs couple them, as summarized in Figure 1, producing three common trade-offs:

- **Sparse execution limits participation.** Sparse-routing methods bound execution by activating only a few experts for each token. The decision narrows participation: non-selected experts neither affect the token’s output nor receive a learning signal from it.
- **Full participation requires dense execution.** Dense output-mixing methods combine the outputs of all experts for each token. Because every expert must be evaluated, execution cost grows with the number of experts.
- **Single-expert execution increases materialization.** Parameter-merging methods combine the full expert pool into one composed expert and execute it once. Every distinct routing decision, however, needs its own copy of those composed weights, so the number of expert-sized parameter sets built at run time grows with the number of routing units.

Taken together, these trade-offs leave a central question: **Can every token benefit from the full expert pool without requiring dense execution or unbounded parameter materialization?** Existing MoE formulations cannot satisfy all three requirements because they couple the construction of expert transformations with their execution on tokens. To break this coupling, we propose IntBMoE, a block-conditioned MoE architecture that separates the two stages. The full expert pool first constructs a bounded set of reusable transformations, after which each token independently selects and executes only a few of them. IntBMoE thereby achieves pool-wide expert participation, sparse execution, and bounded parameter materialization.

Our main contributions are summarized as follows:

- **Decoupling participation, execution, and materialization.** We propose IntBMoE to set the three quantities independently. A small codebook of learned embeddings defines the blocks a module can use, and a shared hypernetwork builds each one layer by layer, merging that layer’s expert bases into a single composed expert. A router then sends each token to only a few of these blocks. Participation is therefore pool-wide, execution stays sparse, and materialization is bounded by the codebook.
- **Expressive expert composition.** We introduce Dual-Path Residual Gating (DPRG), which merges each block’s expert bases twice, into a value path and a gate path whose product is nonlinear in those bases. DPRG therefore adds expressiveness without enlarging the expert pool.

- **Visual evaluation and cross-domain generalization.** We compare IntBMoE with representative sparse and dense MoE baselines on ImageNet-1K and observe consistent improvements. Results on language modeling and sequential recommendation further show that the same architecture is effective across modalities and application domains. IntBMoE has also been fully deployed in AMap’s generative recommendation system, serving hundreds of millions of users under a strict 60 ms latency budget and delivering a 2.4% relative UVCTR improvement in large-scale online A/B testing.

2 Related Work

2.1 Sparse Mixture-of-Experts

Sparsely gated MoEs activate only a subset of experts for each token [Shazeer et al. \(2017\)](#). GShard [Lepikhin et al. \(2020\)](#) combines Top-2 expert routing with automatic sharding for large-scale Transformer training, while Switch Transformer [Fedus et al. \(2022\)](#) simplifies the routing rule to Top-1 selection. V-MoE [Riquelme et al. \(2021\)](#) extends token-choice sparse routing to Vision Transformers by routing image patch tokens to a small number of experts. Expert Choice [Zhou et al. \(2022\)](#) reverses the assignment direction. Each expert selects a fixed-capacity set of tokens, balancing expert workloads while allowing a variable number of assignments per token. DeepSeekMoE [Dai et al. \(2024\)](#) improves expert organization through fine-grained expert segmentation and shared-expert isolation. DeepSeek-V3 [Liu et al. \(2024\)](#) retains this fine-grained architecture and introduces an auxiliary-loss-free routing bias to improve load balance.

Subsequent work modifies expert structure and routing while preserving sparse execution. D²-MoE [Gu et al. \(2025\)](#) decomposes pretrained experts into a shared base and compressed expert-specific deltas, after which sparse routing activates only the selected deltas. ReLU-routing ReMoE [Wang et al. \(2024\)](#) replaces discontinuous Top- k selection with continuous ReLU gates and regularizes their sparsity and load balance. LapSum SoftMoE [Zasada et al. \(2026\)](#) instead uses a truncated soft Top- k relaxation and learns how to allocate an overall expert-computation budget across layers. Dense2MoE [Zheng et al. \(2025\)](#) extends sparse selection to model depth. Its Mixture of Blocks executes only a subset of existing Transformer blocks.

Recent adaptive sparse MoEs relax fixed choices for the number of experts maintained per layer and activated per token. DynMoE [Guo et al. \(2025\)](#) adjusts the expert pool based on token-expert routing coverage and uses Top-any routing. MASS [Park & Park \(2026\)](#) expands the expert pool using gradient-based semantic drift detection and uses Top- p routing.

Together, these methods improve scalability, routing efficiency, and expert organization. However, expert participation remains coupled to execution. Each token can benefit only from the experts selected and evaluated for it.

2.2 Dense Output Mixing

Dense output-mixing methods evaluate all experts and combine their outputs with learned routing weights. MMoE [Ma et al. \(2018\)](#) shares an expert pool across tasks and learns a task-specific gate to combine the expert outputs. PLE [Tang et al. \(2020\)](#) stacks multiple extraction layers containing shared and task-specific experts, progressively separating shared knowledge from task-specific information. Both allow every expert in the relevant pool to contribute to an output, but doing so requires computing every expert’s output, causing the execution cost to grow with the number of experts.

Soft MoE [Puigcerver et al. \(2024\)](#) provides a distinct slot-based variant. It softly aggregates input tokens into a fixed set of slots, processes each slot with its assigned expert, and maps the processed slots back to individual tokens. Because each slot mixes all input tokens, the original formulation does not preserve causality and is not directly applicable to autoregressive prediction. μ MoE [Oldfield et al. \(2024\)](#) instead takes a factorized approach.

It represents expert weights as a tensor and computes their mixture using CP or Tensor Ring factorization. This avoids materializing the full tensor and evaluating experts separately.

2.3 Parameter Merging

Parameter-merging methods achieve full expert participation in a different way. SMEAR Muqeeth et al. (2023) constructs one composite expert by taking a routing-weighted average of all experts and then executes the merged expert on the input. In its example-level form, all tokens in a sequence share the same merged expert. This prevents the composition from adapting to individual tokens. A composition derived from the complete sequence also uses future information, so it cannot be applied directly to causal prediction. In its token-level form, SMEAR produces a separate composition for each token. This enables token-specific adaptation but requires a full-pool parameter merge per token, substantially increasing materialization cost. Lory Zhong et al. (2024) reduces the cost of this operation through causal segment-level routing. A composition derived from the preceding segment is reused by all tokens in the current segment. During generation, a prompt-conditioned composition is reused within the request. This reduces the frequency of parameter merging, but all tokens within a segment share the same composition, limiting token-level adaptation.

DSFNet Yu et al. (2025) performs input-conditioned parameter merging to construct scenario-specific network parameters, using gates to linearly combine parameter sets from disentangled factor-scenario branches. In contrast, IntBMoE preconstructs a finite set of input-independent blocks and adapts to each token through sparse block routing.

2.4 Hypernetworks and Parameter Generation

Hypernetworks Ha et al. (2017) generate the parameters of a target network from learned or dynamically produced conditioning embeddings. Directly generating full weight matrices can be memory-intensive. When the conditioning signal is input-dependent, the parameters must also be regenerated as the signal changes.

HyperMoE Zhao et al. (2024) applies this idea to sparse MoE. It encodes information associated with a token’s unselected experts and generates a HyperExpert for that token. The HyperExpert is executed alongside the selected experts, providing an additional token-conditioned path while the unselected experts remain inactive. IntBMoE conditions its hypernetwork on a finite codebook of learned, input-independent block embeddings. The hypernetwork generates compact coefficients that combine a shared pool of expert bases, rather than directly generating full weight matrices. Because the conditioning set is finite and input-independent, the composed blocks can be precomputed and reused across routing decisions. IntBMoE can therefore be viewed as a basis-constrained hypernetwork with bounded parameter materialization.

3 Preliminaries

Let $\mathbf{x}_t \in \mathbb{R}^d$ denote the representation of input element t , and let $f_e(\cdot; \theta_e)$ denote expert e with parameters θ_e . A conventional sparse MoE maintains E experts and uses a router to select k of them:

$$\mathbf{r}_t = g_{\text{router}}(\mathbf{x}_t), \quad \mathbf{p}_t = \text{softmax}(\mathbf{r}_t), \quad \mathbf{y}_t^{\text{sparse}} = \sum_{e \in \text{TopK}(\mathbf{r}_t, k)} p_{t,e} f_e(\mathbf{x}_t; \theta_e). \quad (1)$$

Here, $g_{\text{router}} : \mathbb{R}^d \rightarrow \mathbb{R}^E$ is the routing network, $\mathbf{r}_t$ contains its scores for the E experts, and $p_{t,e}$ is the routing probability assigned to expert e . $\text{TopK}(\mathbf{r}_t, k)$ returns the indices of the k highest-scoring experts. Sparse routing evaluates only k expert networks for the token.

A direct dense output mixture instead aggregates all expert outputs,

$$\mathbf{y}_t^{\text{dense}} = \sum_{e=1}^E p_{t,e} f_e(\mathbf{x}_t; \theta_e). \quad (2)$$

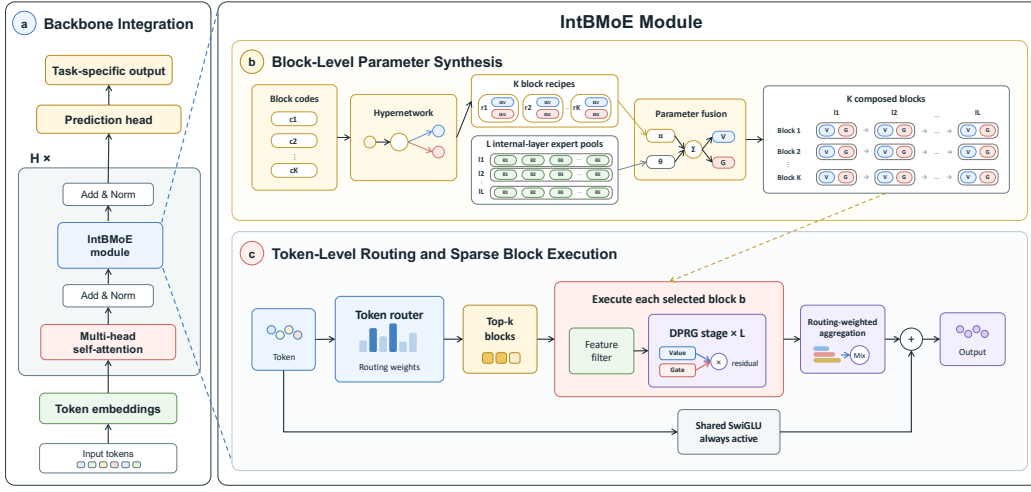


Figure 2: Overview of IntBMoE. The left panel shows independent IntBMoE modules integrated into a multi-layer Transformer backbone. The right panel shows block-level expert parameter synthesis and token-level Top- k block routing and execution.

All E experts can therefore affect the token, but every expert must be executed. Slot-based variants change the unit of expert computation from individual tokens to learned token mixtures, but still process the full set of expert-associated slots.

Parameter-merging methods obtain full expert participation without separately executing every expert. Let u denote a routing unit, which may be a token, segment, or sequence. We use $a_{u,e}$ for the coefficient assigned to expert e . These methods construct

$$\bar{\theta}_u = \sum_{e=1}^E a_{u,e} \theta_e, \quad \mathbf{y}_t^{\text{merge}} = f(\mathbf{x}_t; \bar{\theta}_u). \quad (3)$$

The token is processed by one composite expert, but each routing unit requires its own expert-sized parameter set, constructed by combining all experts. If a layer contains G distinct routing units, parameter synthesis costs $O(GE|\theta|)$, where $|\theta|$ denotes the size of one expert. Here, *materialization* refers to constructing these derived parameter sets at runtime.

4 Method

4.1 Architecture Overview

IntBMoE integrates into a Transformer backbone Vaswani et al. (2017) by replacing its FFN sublayer. Each IntBMoE module composes reusable multi-layer blocks from shared expert pools and sparsely routes each token to a few blocks, as illustrated in Figure 2. At the block level, K learned codebook embeddings produce composition coefficients. These coefficients combine layer-specific expert pools into K token-independent L -layer blocks. At the token level, a router independently selects the Top- k blocks for each token. Each selected block applies block-conditioned feature filtering and processes the token sequentially through its L composed experts. The selected outputs are weighted by their routing probabilities and combined with the output of an always-active shared SwiGLU expert. The following subsections describe these components in detail.

4.2 Block-Level Parameter Synthesis

Block synthesis starts from K learned codebook embeddings, one per candidate block. A shared hypernetwork maps each embedding to value and gate composition coefficients, which

combine the layer-wise expert pools into K reusable multi-layer blocks. The token router then selects among these blocks, as described in the next subsection.

Concretely, each IntBMoE module maintains a codebook of K learned embeddings,

$$\mathcal{C} = \{\mathbf{c}_b \in \mathbb{R}^{d_c}\}_{b=1}^K. \quad (4)$$

Each embedding $\mathbf{c}_b$ identifies one L -layer block and is used to synthesize its parameters. The codebook therefore defines the K blocks available to the token router.

We use column vectors throughout; linear maps act by left multiplication. Let $d_0 = d_L = d$, with d_ℓ denoting the output dimension of internal layer ℓ . For each $\ell \in \{1, \dots, L\}$, the module maintains a layer-specific pool of E expert bases shared across the K blocks,

$$\mathcal{P}^{(\ell)} = \{(W_e^{(\ell)}, \mathbf{b}_e^{(\ell)})\}_{e=1}^E, \quad W_e^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \quad \mathbf{b}_e^{(\ell)} \in \mathbb{R}^{d_\ell}. \quad (5)$$

Here, $\mathcal{P}^{(\ell)}$ denotes the expert pool at internal layer ℓ , and $(W_e^{(\ell)}, \mathbf{b}_e^{(\ell)})$ is its e -th expert. These routed experts are distinct from the always-active shared expert introduced later. The pools are independent across internal layers and backbone layers.

The block hypernetwork h_ϕ is shared across the K codebook entries within one IntBMoE module. It takes only the learned block embedding $\mathbf{c}_b$ as input and uses a linear–LayerNorm–ReLU trunk followed by two linear output heads. Its hidden representation is

$$\mathbf{q}_b = \text{ReLU}(\text{LN}(A\mathbf{c}_b + \mathbf{a})), \quad (6)$$

where $A \in \mathbb{R}^{d_h \times d_c}$ and $\mathbf{a} \in \mathbb{R}^{d_h}$ are the trainable weight matrix and bias of the input projection. The resulting representation satisfies $\mathbf{q}_b \in \mathbb{R}^{d_h}$. Two linear output heads map $\mathbf{q}_b$ to the value and gate composition coefficients α_b^v and α_b^g , respectively:

$$\alpha_b^v = A_v \mathbf{q}_b + \mathbf{a}_v, \quad \alpha_b^g = A_g \mathbf{q}_b + \mathbf{a}_g, \quad \alpha_b^v, \alpha_b^g \in \mathbb{R}^E. \quad (7)$$

Here, $A_v, A_g \in \mathbb{R}^{E \times d_h}$ and $\mathbf{a}_v, \mathbf{a}_g \in \mathbb{R}^E$ are the trainable weights and biases of the two output heads.

The coefficients are not normalized by softmax or sigmoid. They may be negative and need not sum to one, allowing composition over the linear span of the expert bases rather than restricting it to their convex hull. We instead apply a variance-preserving factor of $1/\sqrt{E}$ when composing the expert bases, keeping the scale of the composed parameters approximately stable as the expert pool grows. For path $p \in \{v, g\}$, the composed parameters at every internal layer are

$$W_{b,p}^{(\ell)} = \frac{1}{\sqrt{E}} \sum_{e=1}^E \alpha_{b,e}^p W_e^{(\ell)}, \quad \mathbf{b}_{b,p}^{(\ell)} = \frac{1}{\sqrt{E}} \sum_{e=1}^E \alpha_{b,e}^p \mathbf{b}_e^{(\ell)}. \quad (8)$$

Thus, the hypernetwork produces one recipe (α_b^v, α_b^g) for block b . At each internal layer, this recipe combines that layer’s expert pool into separate value and gate parameter sets. Repeating this process over all L internal layers constructs the complete L -layer block. Because h_ϕ does not depend on token representations, the K composed blocks can be precomputed and shared across all tokens.

4.3 Token-Level Block Routing

Routing is performed independently for each token. For token t , the router uses its representation $\mathbf{x}_t$ to produce one score for each block and selects the Top- k blocks:

$$\mathbf{r}_t = g_{\text{block-router}}(\mathbf{x}_t), \quad \mathcal{B}_t = \text{TopK}(\mathbf{r}_t, k), \quad (9)$$

where $\mathcal{B}_t$ is the set of indices of the k blocks selected for token t . The block router $g_{\text{block-router}} : \mathbb{R}^d \rightarrow \mathbb{R}^K$ outputs a score vector $\mathbf{r}_t \in \mathbb{R}^K$ and is implemented as a two-layer ReLU MLP by default. The routing weight of a selected block b is

$$\pi_{t,b} = \text{softmax}(\mathbf{r}_t)_b, \quad b \in \mathcal{B}_t. \quad (10)$$

4.4 Block-Conditioned Feature Filtering

Before entering a selected block, the token representation $\mathbf{x}_t$ is filtered using that block’s codebook embedding $\mathbf{c}_b$:

$$\mathbf{m}_{t,b} = \text{sigmoid}(\mathbf{W}_f[\mathbf{x}_t \parallel \mathbf{c}_b] + \mathbf{b}_f), \quad \mathbf{z}_{t,b}^{(0)} = \mathbf{x}_t \odot \mathbf{m}_{t,b}. \quad (11)$$

Here, $[\cdot \parallel \cdot]$ denotes vertical concatenation of column vectors, $\mathbf{W}_f \in \mathbb{R}^{d \times (d+d_c)}$ and $\mathbf{b}_f \in \mathbb{R}^d$ are the learnable parameters of a feature-filtering layer shared across blocks, and $\mathbf{m}_{t,b} \in (0,1)^d$ is a soft feature-wise mask. The filtered representation $\mathbf{z}_{t,b}^{(0)} \in \mathbb{R}^d$ is the input to the first internal layer of block b for token t . This gives different blocks distinct views of the same token before their composed transformations are applied.

4.5 Dual-Path Residual Gating

Although each parameter path in equation 8 is composed linearly from the expert bases, DPRG introduces a nonlinear interaction between two independently composed paths. It couples the value and gate paths through residual multiplicative modulation.

For internal layer ℓ and input $\mathbf{z}_{t,b}^{(\ell-1)}$, the DPRG transformation is

$$\mathbf{v}_{t,b}^{(\ell)} = W_{b,v}^{(\ell)} \mathbf{z}_{t,b}^{(\ell-1)} + \mathbf{b}_{b,v}^{(\ell)}, \quad (12)$$

$$\mathbf{g}_{t,b}^{(\ell)} = \text{RMSNorm}\left(W_{b,g}^{(\ell)} \mathbf{z}_{t,b}^{(\ell-1)} + \mathbf{b}_{b,g}^{(\ell)}\right), \quad (13)$$

$$\tilde{\mathbf{z}}_{t,b}^{(\ell)} = \mathbf{v}_{t,b}^{(\ell)} \odot \left(\mathbf{1} + \lambda \text{SiLU}(\mathbf{g}_{t,b}^{(\ell)})\right). \quad (14)$$

LayerNorm is applied between consecutive internal layers:

$$\mathbf{z}_{t,b}^{(\ell)} = \begin{cases} \text{LN}(\tilde{\mathbf{z}}_{t,b}^{(\ell)}), & \ell < L, \\ \tilde{\mathbf{z}}_{t,b}^{(\ell)}, & \ell = L. \end{cases} \quad (15)$$

Thus, each layer processes the token state produced by the preceding layer, and $F_b(\mathbf{x}_t) = \mathbf{z}_{t,b}^{(L)}$ is the final output of block b . The learnable residual scale λ is shared across the internal layers of an IntBMoE module. DPRG increases the expressiveness of each composed block by coupling two compositions of the same expert pool, while adding only a constant factor to its parameter-synthesis and execution costs.

4.6 Output Aggregation and Shared Expert

Selected block outputs are aggregated using routing probabilities:

$$\mathbf{y}_t^{\text{route}} = \sum_{b \in \mathcal{B}_t} \pi_{t,b} F_b(\mathbf{x}_t). \quad (16)$$

We add an always-active shared SwiGLU expert S to model components that need not be differentiated by block routing:

$$\mathbf{y}_t = \mathbf{y}_t^{\text{route}} + S(\mathbf{x}_t). \quad (17)$$

The shared path captures common information, allowing the routed blocks to focus on transformations that benefit from token-dependent selection.

4.7 Complexity and Inference Caching

Consider an IntBMoE module processing T valid tokens. Let each block contain L layers, where layer ℓ maps $d_{\ell-1}$ to d_ℓ , and define $D = \sum_{\ell=1}^L d_{\ell-1} d_\ell$ as the total matrix size of one multi-layer expert basis. Composing all K blocks from E bases costs $O(KED)$, and executing the k selected blocks for all tokens costs $O(TkD)$. Ignoring lower-order

Table 1: Main results on ImageNet-1K. $\uparrow$ indicates that higher values are better. Best results are highlighted with a blue background, and second-best results are shown in blue. Params denotes the total number of trainable parameters in the model; Activated Params denotes the number of trainable parameters activated as a single patch token passes through the entire model. FLOPs denotes the inference computational cost per image, measured with a batch size of 1.

Method	ACC@1 $\uparrow$	ACC@5 $\uparrow$	Params (M)	Activated (M)	FLOPs (G)
Dense	0.6640	0.8769	3.938	3.938	1.453
Switch Transformer	0.7004	0.8937	24.070	3.951	1.458
DeepSeek-V3 MoE	0.7078	0.8969	24.004	6.306	2.382
ReMoE	0.6971	0.8946	24.070	4.574	1.458
V-MoE	0.7167	0.9030	24.070	5.135	1.920
Expert Choice	0.7170	0.9030	24.070	5.135	1.958
DynMoE	0.6975	0.8962	24.070	10.306	4.170
MASS	0.7008	0.8966	24.070	8.827	3.601
Soft MoE	0.7122	0.8996	24.201	23.017	1.656
SMEAR	0.7178	0.9033	24.070	22.887	1.493
Lory	0.6956	0.8892	24.005	22.822	2.003
μ MoE (CP)	0.7012	0.8938	24.070	22.887	9.120
μ MoE (TR)	0.6962	0.8929	24.065	22.882	9.151
IntBMoE	0.7376	0.9148	24.295	23.111	4.063
IntBMoE (cached)	—	—	—	—	3.457

routing operations and the constant factor from the two DPRG paths, the total uncached composition-and-routed-execution cost is $O(KED + TkD)$. The corresponding amortized per-token cost is $O(KED/T + kD)$. The composition term remains linear in E , but it is incurred once for K reusable blocks rather than once per token.

Block composition depends only on the learned block embeddings, hypernetwork, and expert bases. Once the model parameters are fixed for inference, all composed block parameters can be constructed once and cached, removing the $O(KED)$ composition term from request-time computation. The dominant routed-block cost is then $O(TkD)$, in addition to the router, feature filter, and shared expert. Consequently, for fixed K and k , the request-time computation of cached IntBMoE does not increase with the expert-pool size E .

5 Visual Experiments

5.1 Experimental Setup

ImageNet-1K [Russakovsky et al. \(2015\)](#) is our primary benchmark. It contains 1.28 million training images and 50,000 validation images from 1,000 classes. All methods use an eight-layer DeiT-Tiny-style backbone [Touvron et al. \(2021\)](#) and are trained from scratch on the official training split. We report the mean Top-1 and Top-5 validation accuracy over three random seeds.

We compare IntBMoE with the dense backbone and two families of MoE methods. Sparse-routing baselines include Switch Transformer [Fedus et al. \(2022\)](#), DeepSeek-V3 MoE [Liu et al. \(2024\)](#), ReMoE [Wang et al. \(2024\)](#), V-MoE [Riquelme et al. \(2021\)](#), Expert Choice [Zhou et al. \(2022\)](#), DynMoE [Guo et al. \(2025\)](#), and MASS [Park & Park \(2026\)](#). Dense-participation baselines include Soft MoE [Puigcerver et al. \(2024\)](#), SMEAR [Muqeeth et al. \(2023\)](#), Lory [Zhong et al. \(2024\)](#), and μ MoE [Oldfield et al. \(2024\)](#). Complete experimental configurations and implementation details are provided in Appendix A.

Table 2: Component ablations on ImageNet-1K.

Metric	Full	1-Layer	Fixed λ	w/o Gate	w/o Shared	w/o Filter	Softmax Coeff.
Top-1 $\uparrow$	0.7376	0.6878	0.7244	0.6804	0.7332	0.7372	0.7263
Top-5 $\uparrow$	0.9148	0.8910	0.9078	0.8831	0.9108	0.9132	0.9072

5.2 Experimental Results

Main results. Table 1 presents the primary comparison. IntBMoE achieves 73.76% Top-1 and 91.48% Top-5 accuracy. Relative to the dense backbone, these scores represent gains of 7.36 and 3.79 percentage points, respectively. IntBMoE also outperforms all sparse-routing and dense-participation MoE baselines, exceeding the strongest competitor, SMEAR, by 1.98 Top-1 percentage points and 1.15 Top-5 percentage points.

All MoE methods use comparable total parameter budgets of approximately 24M. IntBMoE activates 23.111M parameters because the full expert pool participates in constructing its reusable blocks. Despite full expert participation, each token executes only its selected blocks. The total inference cost is 4.063 GFLOPs per image without caching and 3.457 GFLOPs per image with caching, measured with a batch size of 1.

Ablation studies. To assess the contribution of each component, Table 2 reports ablations on ImageNet-1K. The *1-Layer* variant replaces each two-layer block with a single layer and enlarges its expert pool to preserve the total number of expert parameters. *Fixed λ* makes λ non-learnable and sets it to 1. *w/o Gate* removes the composed gate path and multiplicative modulation, reducing equation 14 to $\tilde{\mathbf{z}}_{t,b}^{(\ell)} = \mathbf{v}_{t,b}^{(\ell)}$. *w/o Shared* removes the shared SwiGLU expert, while *w/o Filter* bypasses block-conditioned feature filtering by setting $\mathbf{z}_{t,b}^{(0)} = \mathbf{x}_t$. Finally, *Softmax Coeff.* replaces our unconstrained, variance-scaled coefficient formulation with separately softmax-normalized value and gate coefficients.

Removing the gate path causes the largest performance drop, followed by collapsing each two-layer block into a parameter-matched single layer. The shared expert and feature filter provide smaller but consistent gains. Both fixing λ and softmax-normalizing the composition coefficients underperform the full formulation.

5.3 Sensitivity to Architectural Hyperparameters

We vary the codebook size K , the number of selected blocks k , the expert-pool size E at each internal layer, and the block depth L on ImageNet-1K. We change one hyperparameter at a time from the default configuration $(K, E, k, L) = (8, 16, 2, 2)$. For the $K = 1$ setting, we set $k = 1$. Figure 3 reports the resulting Top-1 accuracies.

Increasing either K or E improves accuracy, but further gains are small beyond their default values. Expanding K from 8 to 32 increases Top-1 accuracy by only 0.07 percentage points, and expanding E from 16 to 64 increases it by only 0.15 percentage points. Increasing k consistently improves Top-1 accuracy. We use $k = 2$ to obtain a substantial gain over $k = 1$ while limiting per-token computation. For block depth, $L = 2$ achieves the highest Top-1 accuracy.

5.4 Effective Full-Pool Expert Participation

Although every block is constructed from all expert bases, this design alone does not show that every expert makes a meaningful contribution to model performance. We therefore evaluate the contribution of each expert basis on ImageNet-1K by removing it from all block compositions in one MoE layer at a time. We examine all 16 expert bases in Layers 0, 2, 4, and 6, producing 64 removal settings in total, and evaluate the original checkpoint without retraining. Removing an expert may reduce the magnitude of the composed weights

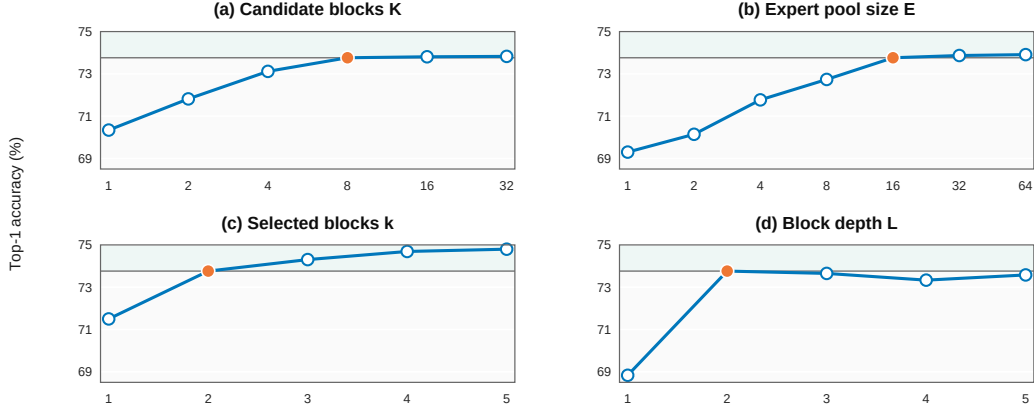


Figure 3: Sensitivity of IntBMoE to architectural hyperparameters on ImageNet-1K. Each panel reports Top-1 accuracy while varying one hyperparameter; higher values are better. All panels use the same vertical scale. Orange points and gray horizontal lines mark the default configuration $(K, E, k, L) = (8, 16, 2, 2)$ and its accuracy, respectively.

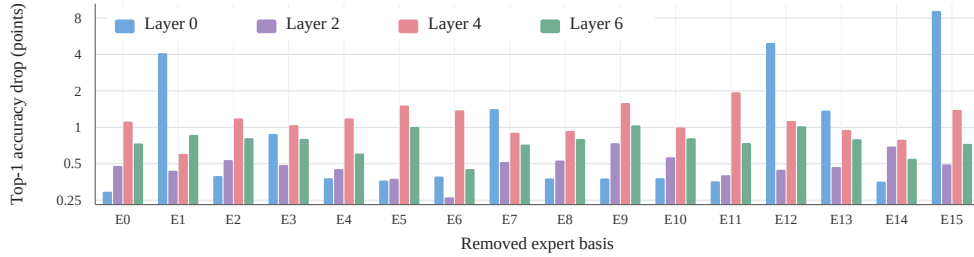


Figure 4: Top-1 accuracy drop on ImageNet-1K after removing individual expert bases from one MoE layer at a time. The vertical axis uses a logarithmic scale.

simply because fewer expert bases remain. To isolate the contribution of the removed expert, we compensate for this scale change by replacing $1/\sqrt{E}$ with $1/\sqrt{E} - 1$.

Figure 4 shows that removing any of the examined expert bases reduces Top-1 accuracy. The mean drops at Layers 0, 2, 4, and 6 are 1.60, 0.49, 1.17, and 0.78 percentage points, respectively. Contributions are highly uneven in Layer 0, where removing E1, E12, or E15 lowers accuracy by 4.10, 4.99, and 9.15 percentage points, while the effects of most other bases are much smaller. The removal effects become more balanced in later layers, ranging from 0.26 to 0.74 percentage points in Layer 2, 0.60 to 1.95 percentage points in Layer 4, and 0.45 to 1.04 percentage points in Layer 6. This suggests that the earliest MoE layer relies strongly on a few expert bases, whereas later layers distribute useful contributions more evenly across their pools. Even the smallest decrease is 0.26 percentage points, indicating that every expert basis contributes to the model. Together, the results support the conclusion that the model makes effective use of the full expert pool throughout its MoE layers.

5.5 Block Routing and Block-Specific Composition Coefficients

Class-conditioned block allocation. On ImageNet-1K, we compute the proportion of patch-token routing assignments received by each block within each ground-truth class. Figure 5 shows deviations from the uniform 12.5% allocation for the same eight representative classes at layers 4 and 6.

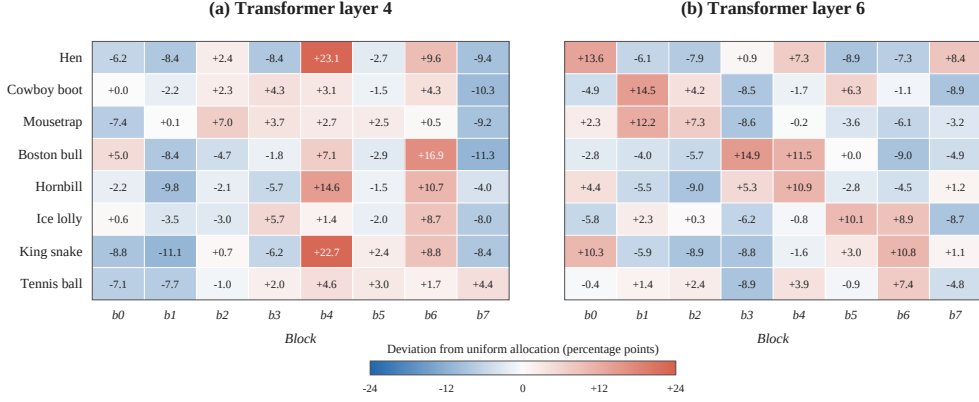


Figure 5: Class-conditioned block routing for the same eight representative ImageNet-1K classes at Transformer layers 4 and 6. Each cell shows the token-assignment proportion for a class and block minus the uniform allocation of 12.5%, in percentage points. Positive (red) and negative (blue) values indicate above- and below-uniform use of a block, respectively. Each Top- k selection is counted as one token-block assignment.

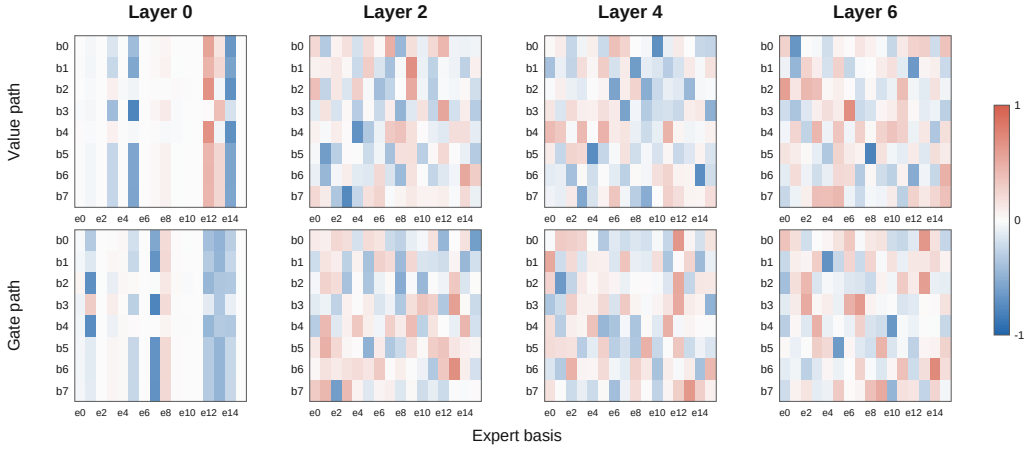


Figure 6: Expert-composition coefficients learned by the eight blocks in Transformer layers 0, 2, 4, and 6 on ImageNet-1K, ordered from shallow to deep along the backbone. Rows represent blocks and columns represent expert bases. Each block’s coefficient vector is ℓ_2 -normalized only for visualization. Red and blue denote positive and negative coefficients, respectively.

Different image classes favor different blocks within the same layer, and these preferences often change with depth. For example, at layer 4, hen favors b_4 , whereas Boston bull favors b_6 . At layer 6, their preferred blocks shift to b_0 and b_3 , respectively. These patterns show that routing is both class-conditioned and depth-dependent.

Block-specific composition coefficients. We visualize the expert-composition coefficients generated for every block on ImageNet-1K, separately for the value and gate paths. As shown in Figure 6, blocks within the same layer assign different positive and negative weights to the same expert bases. The patterns also vary across layers and between the two DPRG paths. Because all blocks in a layer compose the same ordered expert pools, these differences show that the codebook entries learn block-specific composition recipes rather than collapsing to identical fusion coefficients.

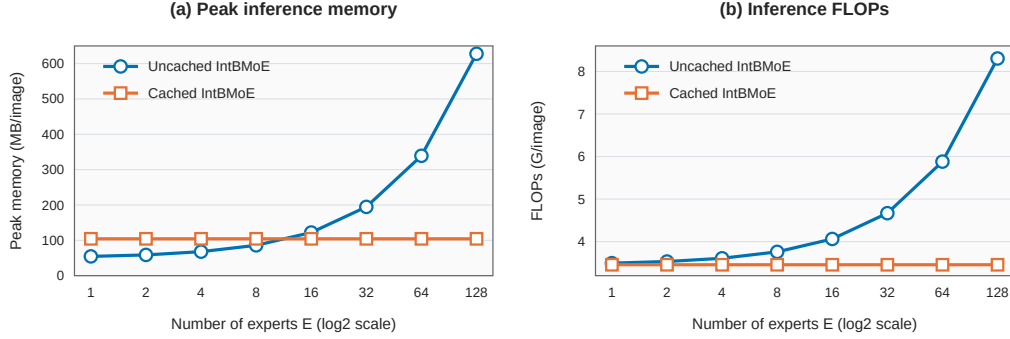


Figure 7: Peak inference memory and inference FLOPs of cached and uncached IntBMoE as the number of experts E increases. Both quantities are measured for inference on a single image with a batch size of 1 using the ImageNet-1K model configuration. The horizontal axis uses a log₂ scale.

The degree of recipe differentiation increases with network depth. For the value path, the mean pairwise cosine similarity between block recipes decreases monotonically from 0.796 at layer 0 to 0.079, 0.019, and 0.010 at layers 2, 4, and 6, respectively. Gate-path similarity follows the same trend, decreasing from 0.726 to 0.126, 0.083, and 0.043. Thus, the shallowest layer largely shares an expert-composition pattern across blocks, whereas deeper layers construct substantially more diverse recipes. This trend is consistent with shallow visual features being more general and deeper representations benefiting from finer block-specific expert composition.

5.6 Inference Cost and Caching

Figure 7 shows how the inference memory and FLOPs of cached and uncached IntBMoE change as the number of experts increases in the ImageNet-1K setup. Without caching, peak inference memory increases from 54.57 to 627.83 MB and inference cost increases from 3.495 to 8.305 GFLOPs as the expert-pool size E grows from 1 to 128. With caching, they remain constant at 104.33 MB and 3.457 GFLOPs, respectively. Caching introduces a fixed memory cost for the precomposed blocks, so it uses more memory for small expert pools, but becomes more memory-efficient from 16 experts onward. These results show that caching decouples request-time memory and computation from the size of the expert pool. During training, all images in a minibatch share the same composed blocks. The composition is therefore performed once per batch, and its cost is amortized across the images in that batch.

6 Generalization to Language Modeling and Recommendation

Tasks and datasets. We use MiniPile Kaddour (2023) and IntTravel Yan et al. (2026) to test whether the same architecture generalizes to language modeling and sequential recommendation. MiniPile uses an 18-layer Llama-style causal Transformer Touvron et al. (2023), while IntTravel uses a causal Transformer. MiniPile is a 6-GB subset of the deduplicated Pile. We evaluate causal language modeling on MiniPile using token-level cross-entropy and perplexity. IntTravel contains 162.8 million users, 7.3 million POIs, and 4.13 billion interactions. We use its “Where” task to predict the destination POI of the next journey and report HR@1, HR@5, and NDCG@5.

Experimental results. Table 3 compares IntBMoE with the dense backbone, sparse-routing methods, and dense-participation methods on MiniPile and IntTravel. For these causal prediction tasks, we make Soft MoE causal by constructing each slot using only the current and preceding tokens. SMEAR uses token-level composition. Complete model, training, routing, and baseline configurations are provided in Appendix A.

Table 3: Generalization results on MiniPile and IntTravel. $\uparrow$ indicates that higher values are better, and $\downarrow$ indicates that lower values are better. Best results per column are highlighted with a blue background, and second-best results are shown in blue.

Method	MiniPile		IntTravel		
	Loss $\downarrow$	PPL $\downarrow$	HR@1 $\uparrow$	HR@5 $\uparrow$	NDCG@5 $\uparrow$
Dense	2.8131	16.6621	0.6748	0.8659	0.7785
Switch Transformer	2.7156	15.1135	0.6798	0.8667	0.7812
DeepSeek-V3 MoE	2.7119	15.0580	0.6805	0.8674	0.7818
ReMoE	2.7346	15.4034	0.6833	0.8685	0.7836
DynMoE	2.7108	15.0406	0.6818	0.8677	0.7826
MASS	2.7608	15.8128	0.6822	0.8679	0.7829
Soft MoE (causal)	2.7112	15.0466	0.6832	0.8683	0.7835
SMEAR (token-level)	2.7165	15.1277	0.6825	0.8676	0.7828
Lory	2.7519	15.6721	0.6811	0.8677	0.7823
μ MoE (CP)	2.7101	15.0306	0.6835	0.8682	0.7837
μ MoE (TR)	2.7171	15.1359	0.6837	0.8685	0.7835
IntBMoE	2.6802	14.5878	0.6852	0.8692	0.7850

Table 4: Component ablations on MiniPile and IntTravel.

Variant	MiniPile		IntTravel		
	Loss $\downarrow$	PPL $\downarrow$	HR@1 $\uparrow$	HR@5 $\uparrow$	NDCG@5 $\uparrow$
IntBMoE	2.6802	14.5878	0.6852	0.8692	0.7850
1-Layer	2.7174	15.1410	0.6824	0.8680	0.7828
Fixed λ	2.6938	14.7871	0.6844	0.8685	0.7827
w/o Gate	2.7660	15.8947	0.6820	0.8676	0.7828
w/o Shared	2.7281	15.3035	0.6846	0.8689	0.7844
w/o Filter	2.6829	14.6277	0.6847	0.8687	0.7846
Softmax Coeff.	2.7071	14.9864	0.6835	0.8686	0.7839

On MiniPile, IntBMoE achieves a test loss of 2.6802 and a PPL of 14.5878. Relative to μ MoE (CP), the strongest baseline, IntBMoE reduces PPL by 2.9%. Relative to the dense backbone, the reduction is 12.4%. On IntTravel, IntBMoE achieves the highest mean HR@1, HR@5, and NDCG@5. These results support generalization to autoregressive language modeling and sequential recommendation.

Table 4 reports component ablations on both MiniPile and IntTravel. The gate path has the largest impact on MiniPile and is among the most influential components on IntTravel. The parameter-matched 1-Layer variant also underperforms the full model, showing that the two-layer block structure provides benefits beyond expert parameter count. The shared expert yields smaller but consistent gains. Fixing λ and removing feature filtering hurt both autoregressive tasks, while the unconstrained, variance-scaled coefficients consistently outperform softmax-normalized composition. These results confirm that the main design choices remain effective beyond vision.

Online evaluation and deployment. We further evaluate cached IntBMoE in AMap’s POI recommendation service for the initial map screen displayed when a user opens the app. The production generative recommender predicts the Top-10 POIs that the user is likely to visit and uses them to determine the map viewport and zoom level. Because this service requires a response within 60 ms, we precompute and cache the input-independent composed block parameters, removing block synthesis from the request-time path.

We conducted a one-week online A/B experiment in which the control group used the existing production model without an MoE module and the treatment group used cached IntBMoE. The experiment handled approximately 5,000 queries per second. Running on

Alibaba T-Head PPUs, cached IntBMoE achieved an average latency of 19ms and a P99 latency of 38ms, remaining within the serving budget while delivering a 2.4% relative improvement in UVCTR. Following the experiment, IntBMoE was fully deployed to serve production traffic.

7 Conclusion

Existing MoE designs couple expert participation, execution, and materialization: sparse routing limits participation, dense output mixing increases execution, and parameter merging increases materialization. We introduced IntBMoE to resolve this coupling, allowing every token to benefit from the full expert pool without dense execution or unbounded parameter materialization. IntBMoE separates reusable block construction from token-level execution. Each candidate block composes the full pool of expert bases, while Top- k routing executes only a few blocks per token, and a finite codebook bounds the number of materialized blocks independently of the number of routing decisions. Within each block, DPRG couples independently composed value and gate paths through residual multiplicative interactions. On the primary ImageNet-1K benchmark, IntBMoE consistently outperforms the evaluated sparse and dense-participation MoE baselines. Analysis on ImageNet-1K shows that different image classes induce distinct block-allocation profiles. It also shows that blocks within each layer learn distinct expert-composition recipes. Auxiliary experiments on MiniPile and IntTravel further demonstrate generalization to language modeling and sequential recommendation. For efficient serving, the composed blocks can be cached to remove block synthesis from the request-time path. The full deployment of IntBMoE in AMap’s latency-critical generative recommendation system further demonstrates its practical utility.

References

- Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024.
- Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.
- Zhiwen Fan, Rishov Sarkar, Ziyu Jiang, Tianlong Chen, Kai Zou, Yu Cheng, Cong Hao, Zhangyang Wang, et al. M³vit: Mixture-of-experts vision transformer for efficient multi-task learning with model-accelerator co-design. *Advances in Neural Information Processing Systems*, 35:28441–28457, 2022.
- 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.
- Hao Gu, Wei Li, Lujun Li, Qiyuan Zhu, Mark Lee, Shengjie Sun, Wei Xue, and Yike Guo. Delta decompression for moe-based llms compression. *arXiv preprint arXiv:2502.17298*, 2025.
- Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, Zhaopeng Tu, and Tao Lin. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. In *International Conference on Learning Representations*, volume 2025, pp. 79643–79672, 2025.
- David Ha, Andrew M Dai, and Quoc V Le. Hypernetworks. In *International conference on learning representations*, 2017.
- 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.

- Jean Kaddour. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*, 2023.
- Dmitry Lepikhin, Hyoungho 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. *arXiv preprint arXiv:2006.16668*, 2020.
- Jiachen Li, Xinyao Wang, Sijie Zhu, Chia-Wen Kuo, Lu Xu, Fan Chen, Jitesh Jain, Humphrey Shi, and Longyin Wen. Cumo: Scaling multimodal llm with co-upcycled mixture-of-experts. *Advances in Neural Information Processing Systems*, 37:131224–131246, 2024.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1930–1939, 2018.
- Mohammed Muqeeth, Haokun Liu, and Colin Raffel. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745*, 2023.
- Basil Mustafa, Carlos Riquelme, Joan Puigcerver, Rodolphe Jenatton, and Neil Houlsby. Multimodal contrastive learning with limoe: the language-image mixture of experts. *Advances in Neural Information Processing Systems*, 35:9564–9576, 2022.
- James Oldfield, Markos Georgopoulos, Grigorios G Chrysos, Christos Tzelepis, Yannis Panagakis, Mihalis A Nicolaou, Jiankang Deng, and Ioannis Patras. Multilinear mixture of experts: Scalable expert specialization through factorization. *Advances in Neural Information Processing Systems*, 37:53022–53063, 2024.
- Sumin Park and Noseong Park. How many experts are enough? towards optimal semantic specialization for mixture-of-experts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24792–24800, 2026.
- Joan Puigcerver, Carlos Riquelme Ruiz, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *International Conference on Learning Representations*, volume 2024, pp. 28435–28445, 2024.
- Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.
- Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
- Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM conference on recommender systems*, pp. 269–278, 2020.
- Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- 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.
- Ziteng Wang, Jun Zhu, and Jianfei Chen. Remoe: Fully differentiable mixture-of-experts with relu routing. *arXiv preprint arXiv:2412.14711*, 2024.
- Zeyue Xue, Guanglu Song, Qiushan Guo, Boxiao Liu, Zhuofan Zong, Yu Liu, and Ping Luo. Raphael: Text-to-image generation via large mixture of diffusion paths. *Advances in Neural Information Processing Systems*, 36:41693–41706, 2023.
- Huimin Yan, Longfei Xu, Junjie Sun, Zheng Liu, Wei Luo, Kaikui Liu, and Xiangxiang Chu. Intravel: A real-world dataset and generative framework for integrated multi-task travel recommendation. *arXiv preprint arXiv:2602.11664*, 2026.
- Jiahao Yu, Yihai Duan, Longfei Xu, Chao Chen, Shuliang Liu, Kaikui Liu, Fan Yang, Xiangxiang Chu, and Ning Guo. Dsfnet: Learning disentangled scenario factorization for multi-scenario route ranking. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 567–576, 2025.
- Mikołaj Zasada, Łukasz Struski, Jacek Tabor, and Marcin Kurdziel. Softmoe: Soft differentiable routing for mixture-of-experts in llms. *arXiv preprint arXiv:2606.17952*, 2026.
- Hao Zhao, Zihan Qiu, Huijia Wu, Zili Wang, Zhaofeng He, and Jie Fu. Hypermoe: Towards better mixture of experts via transferring among experts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10605–10618, 2024.
- Youwei Zheng, Yuxi Ren, Xin Xia, Xuefeng Xiao, and Xiaohua Xie. Dense2moe: Restructuring diffusion transformer to moe for efficient text-to-image generation. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 18661–18670. IEEE, 2025.
- Zexuan Zhong, Mengzhou Xia, Danqi Chen, and Mike Lewis. Lory: Fully differentiable mixture-of-experts for autoregressive language model pre-training. *arXiv preprint arXiv:2405.03133*, 2024.
- 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.
- Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, et al. Rankmixer: Scaling up ranking models in industrial recommenders. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6309–6316, 2025.

A Detailed Experimental Configurations

A.1 Common Experimental Protocol

All experiments use eight Alibaba T-Head PPUs, each with 96 GB of memory. Within each dataset, all methods share the backbone, input processing, task head, optimization schedule, seed set, and expert initialization; only the FFN or MoE module differs. ImageNet-1K uses seeds 214797, 531770, and 635451, MiniPile uses seeds 1, 145306, and 145849, and IntTravel uses seeds 42, 4981213, and 211231244.

Table 5 summarizes the model dimensions, input sizes, and training lengths. Here, H and d denote the number of Transformer layers and the hidden size. n_{heads} , d_{head} , and d_r denote

Table 5: Dataset-specific model and training configurations.

Dataset	H	d	$n_{\text{heads}} \times d_{\text{head}}$	d_r	E	BS	Input	λ_0	Training
ImageNet-1K	8	192	3×64	64	16	256	224^2 ($p = 16$)	1	200 epochs
MiniPile	18	768	12×64	64	8	8	1024	1	1 epoch
IntTravel	8	96	1×96	64	32	64	126	0	610K steps

the number of attention heads, the head dimension, and the router hidden size. E is the number of expert bases in each IntBMoE internal layer and the routed-expert count for baselines with a shared path. BS, Input, and λ_0 denote the per-device batch size, the image resolution or maximum sequence length, and IntBMoE’s initial residual scale, respectively.

A.2 Dataset-Specific Configurations

ImageNet-1K. We use a DeiT-Tiny-style backbone [Touvron et al. \(2021\)](#) shortened to eight Transformer layers. MoE modules replace the FFNs in alternating layers (0, 2, 4, and 6) for patch tokens, while the class token uses a separate dense FFN. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$ and weight decay 0.05. The learning rate warms up from 10^{-6} to 5×10^{-4} over five epochs and then follows cosine decay to 10^{-5} . Standard dropout, attention dropout, patch dropout, and position dropout are disabled, while stochastic depth uses a maximum DropPath rate of 0.1. The augmentation pipeline applies a 224×224 bicubic random resized crop with scale $[0.08, 1.0]$ and aspect ratio $[0.75, 1.33]$, random horizontal flipping with probability 0.5, RandAugment using two operations per image, magnitude 9, and a magnitude standard deviation of 0.5, and pixel-mode random erasing with probability 0.25 and one erased region. Mixup ($\alpha = 0.8$) and CutMix ($\alpha = 1.0$) are applied in batch mode with a combined probability of 1.0 and equal selection probabilities, and label smoothing is 0.1.

MiniPile. We use a Llama-style causal Transformer [Touvron et al. \(2023\)](#) and replace the FFN in every layer with an MoE module. We train for one epoch on 1.523 billion tokens and evaluate on a 15.0-million-token test split. Text is tokenized with a byte-level BPE vocabulary of 32,000. Four-step gradient accumulation gives an effective batch size of 256 sequences across eight devices. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay 0.1. The learning rate warms up to 4×10^{-4} over the first 10% of training and then decays cosinusoidally to 4×10^{-5} .

IntTravel. We use a causal Transformer for sequential recommendation and replace every FFN and the prediction head with MoE modules. We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, weight decay 10^{-6} , and a constant learning rate of 10^{-4} .

A.3 MoE Configurations and Fair Comparison

Expert-parameter matching. We use the parameter count of the backbone’s original Transformer FFN, which is replaced by the MoE module, as the target for a single expert. Each internal IntBMoE layer retains a separate pool of E expert bases, and each basis contains one weight matrix. For parameter accounting, we pair same-index bases from the two internal layers and match their combined parameter count to this target. For the baselines, one complete FFN or SwiGLU expert is matched to the same target by counting all of its weight matrices. This matching sets the input–intermediate–output dimensions of each two-layer IntBMoE block to $d \rightarrow 2844 \rightarrow d$ on MiniPile and $d \rightarrow 4d \rightarrow d$ on ImageNet-1K and IntTravel. We also match the always-active shared SwiGLU expert to the same parameter target by adjusting its intermediate width. To keep the total expert parameter count comparable across methods, a baseline with a shared path uses E routed experts and one shared expert, whereas a baseline without one uses $E + 1$ routed experts. For the CP and Tensor Ring variants of μ MoE, we choose the factorization ranks that make the total trainable parameter count of the full model as close as possible to that of IntBMoE.

Specifically, we solve the CP rank R under this parameter budget; for Tensor Ring, we fix $R_1 = R_2 = 4$ and solve R_3 .

Baseline configurations. Switch Transformer follows its original Top-1 routing rule. Other Top- k token-choice baselines use Top-2 routing when their original configurations do not specify k . Expert Choice retains expert-to-token routing with a capacity factor of 2. μ MoE uses $\text{entmax}_{1.5}$ routing, and MASS uses Top- p routing with $p = 0.5$.

Soft MoE uses one slot per image token on ImageNet-1K. On MiniPile and IntTravel, we make Soft MoE causal by constructing each slot only from the current and preceding tokens. We use one slot per expert to limit computation. SMEAR uses example-level composition for images and token-level composition for autoregressive tasks. Lory uses a segment size of 16.

For DynMoE and MASS, the initial expert-pool sizes on IntTravel, ImageNet-1K, and MiniPile are 16, 8, and 4, respectively. Their maximum pool sizes match the expert counts used by the other methods. We create all expert parameters in advance up to these maxima. DynMoE initializes each expert’s threshold to correspond to a selection probability of 0.5 and uses a routing temperature of 1.0. Every 300 optimizer steps, it removes unused experts and opens a new expert when some tokens activate no expert. MASS detects semantic drift over the previous 200 optimizer steps and expands the pool using significance and similarity thresholds of 0.01 and 0.001. Its redundancy regularizer has a coefficient of 0.01.

IntBMoE configuration. Each module contains $K = 8$ codebook blocks and routes each token to $k = 2$ blocks. The block-embedding dimension, router hidden dimension, and hypernetwork hidden dimension are 32, 64, and 16, respectively. Each block contains two internal DPRG layers. These layers use equally sized, layer-specific expert pools and share the same pair of value and gate composition coefficients. An always-active SwiGLU expert provides the shared path. We initialize the expert matrices with PyTorch’s default `nn.Linear` scheme and set all expert biases to zero. This small initial weight scale limits perturbations from the composed transformations early in training.