PCoMoE: Shifting MoE Inference from Monolithic Expert Selection to Fine-Grained Path Composition

Ziyan Gan[1,*], Fangxin Liu[1,*,†], Chenyang Guan[1], Junjie Wang[1], Ning Yang[1], Haomin Li[1], Xiang Li[2], Siran Yang[2], Jiamang Wang[2], Lin Qu[2], Zongwu Wang[1], Li Jiang[1,†], Haibing Guan[1]

[1]Shanghai Jiao Tong University, China [2]Alibaba Group, China

[*]Equal contribution [†]Corresponding authors

{Aquarius.o, liufangxin, ljiang_cs, hbguan}@sjtu.edu.cn

Abstract

Mixture-of-Experts (MoE) architectures scale Large Language Model (LLM) capacity efficiently by activating a sparse subset of experts per token. However, modern MoE inference remains heavily constrained by the rigid, whole-expert abstraction. Existing frameworks manage, schedule, or prune experts as atomic execution units, which fixes the optimization boundary too early and leaves fine-grained intra-expert computational redundancy under-explored. In this work, we present **PCoMoE**, a path-compositional execution framework that shifts MoE inference from coarse-grained expert selection to fine-grained path composition. PCoMoE incorporates a path-level formulation of expert computation, a compatibility-aware layer-wise pruning strategy to suppress low-value path combinations, and a hardware-friendly execution engine to exploit reusable sub-expert structures under strictly bounded overheads. Experimental results demonstrate that PCoMoE achieves up to a 1.31× end-to-end inference speedup while enhancing model accuracy by 10%. The code is available at https://github.com/gzyyy0/PCoMoE

1 Introduction

Recent advances in Large Language Models (LLMs) demonstrate that scaling parameter capacity directly drives model capability (Kaplan et al., 2020; Brown et al., 2020; Hoffmann et al., 2022). Among existing scaling paradigms, the Mixture-of-Experts (MoE) architecture has emerged as a dominant design by decoupling total parameter count from per-token activated computation. By executing only a sparse subset of experts for each input token, MoE models scale model capacity without proportionally increasing training and inference computational costs (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Du et al., 2022). Consequently, MoE has become a cornerstone for modern high-throughput LLM serving.

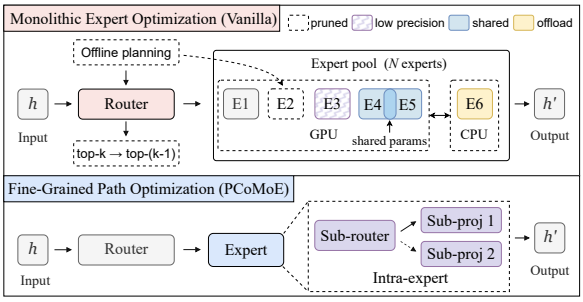

Figure 1: Inter-expert vs. intra-expert optimization in MoE inference. Inter-expert approaches operate on routing decisions and complete experts, whereas intra-expert optimization exposes sub-structures within each expert.

However, theoretical computational sparsity does not automatically translate into end-to-end system efficiency (Liu et al., 2021). MoE inference introduces severe dynamic execution overheads, where irregular token-to-expert routing patterns vary drastically across different tokens and layers (Xue et al., 2024; Tang et al., 2024). Existing optimization frameworks primarily adopt an *inter-expert* perspective to mitigate these overheads. Complementary model-level acceleration techniques further explore adaptive layer skipping (Liu et al., 2025) and precision-aware computation (Liu et al., 2024). As illustrated in Figure 1, these approaches focus on optimizing scheduling, expert caching, offloading, or expert-level skipping and reuse. Specifically, prior work has explored multiple directions, including simplifying expert selection (Zhong et al., 2024; Chitty-Venkata et al., 2025), caching and offloading experts (Xue et al., 2024; Tang et al., 2024), improving expert placement and prefetching (Fang et al., 2025; Liu et al., 2026), and enabling expert-level sharing and reuse (Vankov et al., 2026; Tan et al., 2025). While effective at the macro level, these methods treat each expert as an atomic, indivisible execution unit. This coarse-grained, expert-centric abstraction implicitly assumes that expert computation can only be managed or pruned as a whole,

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 46662–46672
October 24-29, 2026 ©2026 Association for Computational Linguistics

fixing the optimization boundary too early and overlooking the fine-grained computational redundancy buried inside individual experts.

We argue that treating MoE experts as computationally atomic restricts the optimization space and leads to suboptimal efficiency-quality tradeoffs. An MoE expert typically consists of structured Feed-Forward Network (FFN) transformations with coupled sub-layers that exhibit distinct semantic roles, computation costs, and quality sensitivities. Restricting optimizations to the inter-expert boundary forces a binary choice between executing or skipping an entire expert, thereby introducing severe representation bottlenecks or unnecessary computation. Breaking this monolithic boundary allows systems to expose finer-grained opportunities for intra-expert reuse and reorganization, which is essential to jointly eliminate computational redundancy and maintain model accuracy.

To exploit these fine-grained opportunities, we present **PCoMoE**, a path-compositional execution framework that optimizes MoE inference via intra-expert sub-transformation composition. PCoMoE decomposes monolithic experts into composable internal sub-structures, transforming standard expert-level selection into path-level composition over an expanded path space. Instead of relying on rigid expert boundaries, PCoMoE adaptively reorganizes internal expert computations to maximize inference efficiency. To regulate the expanded path space without introducing execution chaos, PCoMoE incorporates compatibility-aware path routing and layer-wise pruning to suppress low-value combinations based on layer-specific sensitivity. Furthermore, we co-design a hardware-friendly path execution engine that groups dispatches by source and shares expansion-side operations, converting path-level flexibility into real end-to-end speedups under strictly bounded system overheads. Our main contributions are threefold. **(1) Path-Level Composition:** We reformulate MoE inference from vanilla monolithic selection to fine-grained sub-transformation composition, unlocking latent intra-expert execution trajectories while preserving baseline vanilla pathways. **(2) Compatibility-Aware Gating:** We introduce structured compatibility gating with iterative structural pruning to suppress low-value off-diagonal routes, minimizing combinatorial scheduling overheads while guaranteeing representation fidelity. **(3) Hardware-Efficient Runtime:** We develop an execution pipeline mapping active compositional paths directly to source-grouped compute clusters, translating algorithmic flexibility into deterministic acceleration. Comprehensive evaluations show that PCoMoE achieves up to a 1.31× serving speedup while consistently outperforming vanilla MoE baselines across downstream tasks by up to 10% in accuracy.

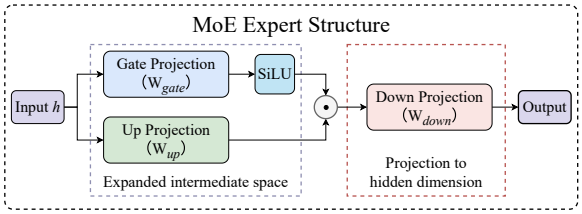

Figure 2: Representative internal structure of an MoE expert, illustrated with a SwiGLU-style gated FFN.

2 Background

A vanilla MoE layer consists of a router and a set of sparse feed-forward experts (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022). Given the hidden representation h of an input token, the router computes routing scores to select the top-k experts for execution. Denoting the selected expert set by $\mathcal{E}(h)$, the output of the MoE layer is formulated as

$$y = \sum_{e \in \mathcal{E}(h)} \alpha_e \, F_e(h),$$

where α_e is the routing weight assigned to expert e, and $F_e(\cdot)$ denotes the non-linear transformation implemented by expert e. At runtime, tokens are dispatched according to routing decisions, grouped by expert for batched execution, and scattered back before weighted aggregation. This mechanism dictates that a vanilla MoE layer operates on a rigid dispatch-execute-combine pipeline (Lepikhin et al., 2020; Xue et al., 2024). Consequently, MoE inference introduces highly input-dependent execution footprints, where activated expert profiles vary dynamically across tokens and layers.

Although vanilla MoE is traditionally managed at the expert selection level, each individual expert is a structured feed-forward module rather than an atomic computational primitive. Modern MoE models predominantly employ the SwiGLU-style gated FFN architecture (Shazeer, 2020) as their core expert instantiation, a design validated in representative open-source models including Mixtral and Qwen-MoE (Jiang et al., 2024; Qwen Team, 2024). For an input hidden state h, the computation

of expert e is defined as

$$F_e(h) = W_{\text{down}}^{(e)}\Big(\text{SiLU}\Big(W_{\text{gate}}^{(e)}h\Big) \odot \Big(W_{\text{up}}^{(e)}h\Big)\Big),$$

where $W_{\text{gate}}^{(e)}$ and $W_{\text{up}}^{(e)}$ project the input tensor into an expanded intermediate space, $\odot$ denotes element-wise multiplication, and $W_{\text{down}}^{(e)}$ maps the fused activations back to the hidden dimension.

Figure 2 illustrates this intra-expert computational layout. The gate and up branches execute parallel matrix multiplications on the expansion side, followed by element-wise activation fusion and a final down projection. Mechanistically, these parallel branches serve distinct algorithmic roles: the gate branch modulates activation thresholds, the up branch extracts expanded feature representations, and the down branch aggregates these features back into the model dimension. This inner layout implies that the computational workload within an MoE expert can be reorganized at a finer granularity than the atomic whole-expert interface enforced by vanilla routing.

In actual hardware deployments, end-to-end MoE execution latency is governed by system-level orchestrations beyond raw arithmetic FLOPs. It encompasses routing overhead, token dispatch, expert-wise packing, batched kernel launches, result scattering, and weighted tensor aggregation (Xue et al., 2024; Tang et al., 2024). Existing hardware-aware optimizations generally target two distinct granularities: minimizing the expert computation exposed by the router through pruning or skipping, or maximizing the execution efficiency of selected experts via batched routing, sharing, caching, or prefetching. Despite mitigating different system bottlenecks, these frameworks consistently preserve the whole-expert abstraction as their basic execution unit. This rigid boundary artificially restricts the optimization space, concealing critical opportunities for structural reorganization and cross-expert computation reuse within individual sub-expert layers.

3 Motivation

Vanilla MoE inference restricts routing to monolithic expert boundaries. While computationally convenient, this coarse abstraction overlooks the structural misalignment between sparsity-induced redundancy and indivisible expert modules.

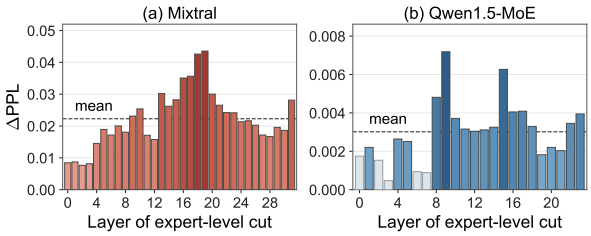

Figure 3: Quality impact of single-layer whole-expert reduction. Each bar reports the ΔPPL after changing top-k to top-$(k-1)$ routing at one MoE layer; darker bars indicate larger values.

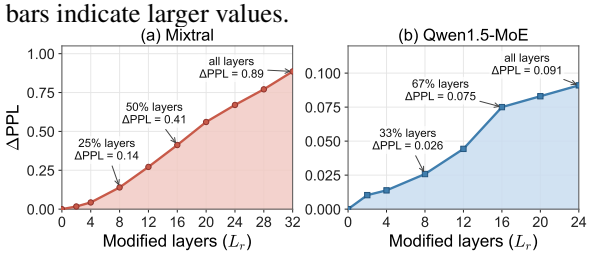

Figure 4: Cumulative quality impact of reducing one activated expert across multiple MoE layers.

3.1 Inefficiency of Coarse-Grained Routing

Traditional MoE acceleration relies on eliminating entire expert executions. Since the vanilla MoE interface exposes computation via top-k routing, a baseline replaces top-k routing with top-$(k-1)$ routing to decrease the number of activated experts per token. We sweep individual layers to profile the perplexity impact of this single-layer pruning.

As shown in Figure 3, imposing the same expert-level pruning across different layers yields highly divergent quality degradation. Certain layers exhibit minimal perplexity increases, whereas others incur substantial degradation that exceeds the average trend, proving that whole-expert reduction is neither uniformly safe nor uniformly detrimental. This variation demonstrates that inter-expert redundancy is highly non-homogeneous and layer-dependent. While specific layers tolerate complete expert deletion, this local redundancy cannot be generalized into a fixed global pruning rule. Multi-layer reduction compounds these errors.

As illustrated in Figure 4, local errors compound catastrophically across deep networks when expert reduction is extended globally. Although the absolute degradation scales differently across models, the underlying trend remains consistent: locally tolerable approximations aggregate into severe end-to-end quality drops. Consequently, repeated whole-expert reduction rapidly depletes safe redundancy, proving that monolithic expert boundaries are too coarse for optimal efficiency-quality trade-offs.

The central challenge is therefore not merely

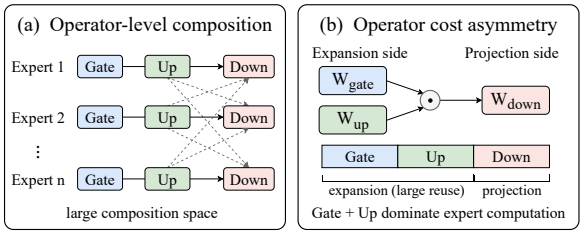

Figure 5: Intra-expert composition and cost asymmetry. (a) Separately composing Gate, Up, and Down expands the routing decisions. (b) Grouping Gate and Up as the expansion side, and Down as the projection side, exposes a structured reuse boundary with asymmetric computational costs.

adjusting the number of reduced experts, but breaking the whole-expert abstraction to exploit finer computational granularities.

3.2 Intra-Expert Operator Asymmetry

To bypass these bottlenecks, we analyze the internal computational structures of individual experts. A vanilla SwiGLU expert maps inputs through parallel gate and up projections on the expansion side (comprising two-thirds of total expert FLOPs) before a final down projection back to the hidden dimension. Directly treating these operators as independent, unconstrained composition units (Figure 5(a)) introduces prohibitive control-flow overheads. Instead, the intrinsic expansion-projection boundary (Figure 5(b)) defines a structured middle ground. In a top-2 Mixtral architecture, reusing the expansion side eliminates approximately one-third of total expert computation, whereas reusing the projection side saves only one-sixth, concentrating potential efficiency gains on the expansion side.

Profiling operator substitution on Mixtral confirms that this compute-dense expansion side also exhibits superior structural tolerance for representation sharing. Substituting the gate and up projections with an alternative source expert yields lower representation error in 23 out of 32 layers under optimal independent pairing. Under a stricter constraint where all experts within a layer share a single replacement source, 25 out of 32 layers still favor expansion-side substitution. This dual computational and structural asymmetry demonstrates that the expansion-projection boundary serves as an optimal abstraction, avoiding the accuracy drops of whole-expert dropping while circumscribing the irregular execution overheads of arbitrary operator splitting.

However, translating this theoretical boundary into serving gains introduces three critical co-design challenges: (1) the path search space scales combinatorially compared to vanilla expert selection, (2) additional control logic or dynamic indexing can easily offset computational savings under strict latency constraints, and (3) compositional stability varies heavily across layers, threatening feature fidelity. Addressing these hardware-software trade-offs requires an execution framework that simultaneously regulates the path composition space, minimizes runtime dispatch overheads, and guarantees model quality.

4 Design

Figure 6 illustrates the PCoMoE overview, which bypasses the monolithic constraints of vanilla MoE inference via fine-grained sub-transformation composition. While preserving the vanilla routing-compatible interface at the layer boundary, PCoMoE decouples each internal expert into standalone expansion-side and projection-side operators. For an MoE layer with n experts, cross-composing these operators expands the optimization landscape into an $n \times n$ compositional path space. Within this matrix, diagonal trajectories maintain the original expert computations to anchor baseline stability, whereas off-diagonal paths represent newly synthesized execution routes that introduce zero parameter or storage overheads.

To prevent this expanded space from inducing combinatorial scheduling overheads, PCoMoE actively filters candidate trajectories via a compatibility-guided pruning mechanism. The framework assigns each path a composite score that balances vanilla routing priors with learned source-to-target compatibility. This scoring function actively suppresses low-value off-diagonal trajectories, confining the search space to a compact, high-fidelity set of active execution paths.

The remaining active paths map directly onto a hardware-efficient pipeline that eliminates redundant kernel launches through source-grouped computation reuse. Instead of executing each path in isolation, PCoMoE aggregates tokens sharing identical expansion-side operators, allowing them to reuse a single compute footprint before dynamic dispatching to their respective projection-side destinations. This hardware-software co-design seamlessly translates algorithmic path-level flexibility into end-to-end serving acceleration under strictly bounded runtime overheads.

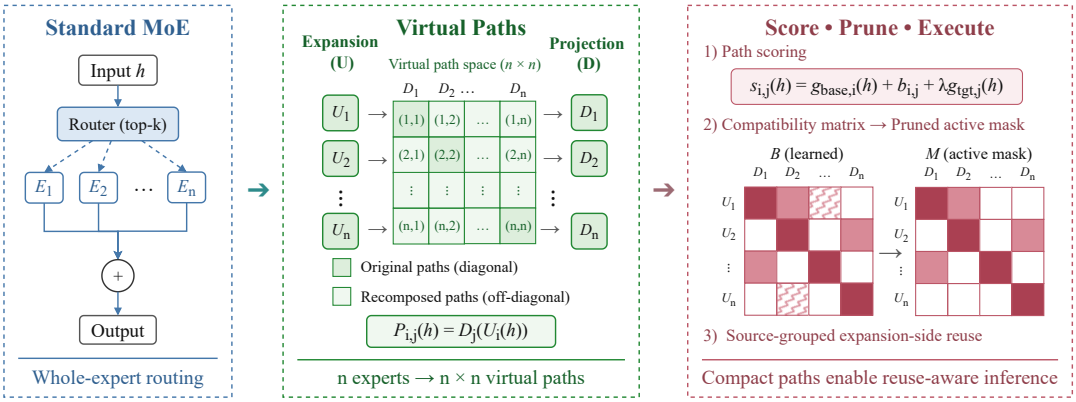

Figure 6: Overview of PCoMoE. PCoMoE decomposes each expert into expansion-side and projection-side operators, forming an expanded path space over their cross-compositions. To make this space practical, PCoMoE scores compositional paths via learned compatibility, prunes low-value paths into a compact active set, and executes the remaining paths through source-grouped expansion-side reuse.

4.1 Compositional Path Formulation

PCoMoE redefines the computational granularity of the MoE layer by decoupling SwiGLU experts along their structural asymmetry boundaries. The expansion-side operator aggregates the compute-dominant gate and up projections, while the projection-side operator maps intermediate activations back to the hidden dimension. Formulating their ordered combination as a compositional path concentrates optimization capabilities precisely where computational reuse potential is maximized, shifting the execution paradigm from monolithic expert selection to flexible path composition.

For an MoE layer containing n physical experts, the computation of an individual expert e is decoupled into an expansion-side function $U_e(h)$ and a projection-side function $D_e(z)$. We express the expansion side as

$$U_e(h) = \mathrm{SiLU}\left(W_{\text{gate}}^{(e)} h\right) \odot \left(W_{\text{up}}^{(e)} h\right),$$

and the projection side as

$$D_e(z) = W_{\text{down}}^{(e)} z,$$

which reconstructs the original vanilla expert transformation via

$$F_e(h) = D_e(U_e(h)).$$

We define the pools of reusable expansion and projection operators across the layer as $\mathcal{U} = \{U_1, \ldots, U_n\}$ and $\mathcal{D} = \{D_1, \ldots, D_n\}$, respectively. A compositional path $P_{i,j}(h)$ represents the ordered composition of an expansion source i and a projection target j:

$$P_{i,j}(h) = D_j(U_i(h)), \qquad i, j \in \{1, \ldots, n\}.$$

This formulation expands the available computational trajectories from n physical modules to an n^2 compositional space. The diagonal path $P_{i,i}$ represents the original vanilla expert F_i, while an off-diagonal path $P_{i,j}$ $(i \neq j)$ synthesizes a new computational trajectory combining the expansion side of expert i with the projection side of expert j.

This optimization landscape is purely architectural and introduces zero parameter or storage overheads. All routes are dynamically synthesized by recombining existing physical operators without creating new weights. Restricting execution to diagonal paths naturally reduces PCoMoE back to the vanilla MoE layer, whereas off-diagonal paths serve as selectively activated candidates to maximize efficiency-quality trade-offs.

Consequently, the MoE layer output generalizes from expert-level routing to path-level aggregation. Let $\Pi(h) \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$ denote the active compositional paths selected for a token hidden state h, and let $\beta_{i,j}(h)$ represent the corresponding path weight. The PCoMoE layer output is formulated as

$$y = \sum_{(i,j) \in \Pi(h)} \beta_{i,j}(h) P_{i,j}(h).$$

The vanilla MoE layer remains a strict subset of this generalized formulation, where setting $\Pi(h) = \{(e, e) \mid e \in \mathcal{E}(h)\}$ and $\beta_{e,e}(h) = \alpha_e(h)$ exactly recovers the original weighted expert summation. Although this formulation unlocks a fine-grained optimization landscape, evaluating the entire n^2 space introduces prohibitive computational overheads. PCoMoE treats this matrix strictly as a candidate space, using a compatibility-driven pruning mechanism to isolate a compact, high-value active path set for execution.

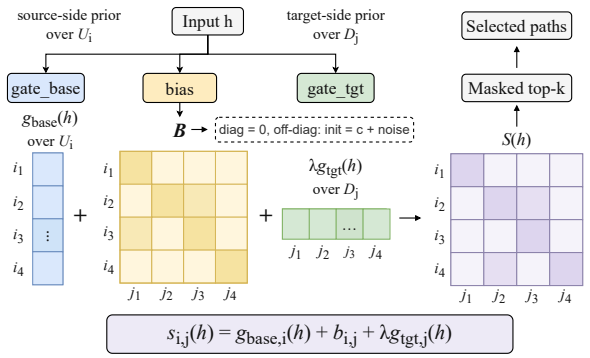

Figure 7: Compatibility-aware path gate for scoring source-to-target compositional paths.

4.2 Compatibility-Aware Path Routing

The compositional path formulation expands the architectural search landscape into an $n \times n$ routing matrix. To suppress noisy off-diagonal recompositions and bound runtime dispatch complexity, PCoMoE introduces a compatibility-aware path gating mechanism (Figure 7) that factors each trajectory score into a source-side routing prior, a target-side routing prior, and a learned pairwise compatibility bias. For a compositional path $P_{i,j}^{(\ell)}$ in layer ℓ, the gating function computes the composite routing score as

$$s_{i,j}^{(\ell)}(h) = g_{\text{base},i}^{(\ell)}(h) + b_{i,j}^{(\ell)} + \lambda g_{\text{tgt},j}^{(\ell)}(h),$$

where $g_{\text{base}}^{(\ell)}(h) \in \mathbb{R}^n$ denotes the base expert-routing signal, $B^{(\ell)} = [b_{i,j}^{(\ell)}] \in \mathbb{R}^{n \times n}$ represents the learned compatibility bias, $g_{\text{tgt}}^{(\ell)}(h) \in \mathbb{R}^n$ provides a projection-side prior, and λ scales this target-side preference. The base routing signal anchors the optimization to the pretrained vanilla checkpoints by broadcasting across source indices, while the target prior regularizes projection choices. Crucially, the compatibility matrix $B^{(\ell)}$ parameterizes the structural alignment between expansion source U_i and projection target D_j, transforming the raw Cartesian product into a filtered execution topology.

PCoMoE initializes this compatibility matrix to strictly prioritize vanilla computational trajectories. Diagonal entries are zero-initialized via $b_{i,i}^{(\ell)} = 0$ to safeguard baseline expert behavior, whereas off-diagonal elements receive a negative offset with bounded random noise:

$$b_{i,j}^{(\ell)} \sim c + \epsilon, \qquad i \neq j,\ c < 0.$$

This negative prior prevents unoptimized off-diagonal trajectories from disrupting early training phases while breaking structural symmetry. During fine-tuning, this bias is dynamically converted into a layer-specific active mask $M^{(\ell)} \in \{0,1\}^{n \times n}$. Rather than applying a post-hoc, one-shot pruning step, PCoMoE iteratively tightens a structural sparsity criterion at progressive training intervals to filter low-value off-diagonal routes while hard-coding $m_{i,i}^{(\ell)} = 1$. This continuous regularization allows the remaining routing weights to co-adapt with the contracting active path set, preserving model convergence stability.

The active mask operates independently across layers on routing eligibility rather than on physical parameters, enabling each layer to autonomously converge to its optimal path density based on local feature sensitivity. Because no physical weights are deleted, suppressed paths are merely excluded from the active runtime dispatch graph. During inference, inactive paths are masked out prior to gating, ensuring that the online router evaluates only the optimized, compact set of active trajectories induced by $M^{(\ell)}$, thereby successfully minimizing control-flow overheads.

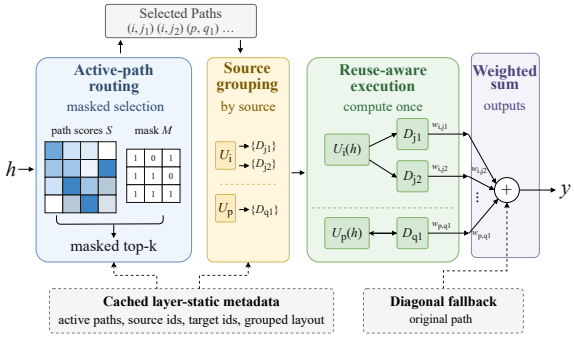

Figure 8: Hardware-efficient inference execution of active compositional paths in PCoMoE.

4.3 Hardware-Efficient Path Execution

To translate the pruned path space into physical serving gains, PCoMoE decouples its runtime execution into a routing phase and a source-grouped reuse phase (Figure 8).

Compositional Path Routing. During inference, the framework isolates active trajectories by masking the $n \times n$ routing matrix with the pruned path set $\mathcal{A}^{(\ell)} = \{(i,j) \mid m_{i,j}^{(\ell)} = 1\}$. The router selects the active path subset via

$$\Pi^{(\ell)}(h) = \text{TopK}_{(i,j) \in \mathcal{A}^{(\ell)}} s_{i,j}^{(\ell)}(h),$$

collapsing dense evaluation into a sparse search. PCoMoE materializes path indices, source-to-target layouts, and grouped dispatch metadata of-

Table 1: Zero-shot accuracy across MoE backbones (higher is better).

Model	Method	BoolQ	ARC-E	ARC-C	HellaSwag	WinoGrande	Avg.
Mixtral-8x7B	Vanilla	85.93	82.62	59.56	84.10	77.11	77.86
	Vanilla-FT	86.45	83.78	61.68	83.34	76.56	78.36
	MoE-I^2	82.60	78.20	52.20	61.10	71.50	69.10
	MoE-Pruner	86.00	81.90	53.30	62.30	75.50	71.80
	MoEITS	87.33	83.08	55.89	82.83	**80.98**	78.02
	PCoMoE	**88.38**	**85.06**	**62.71**	**85.46**	78.22	**79.97**
	Δ *vs Vanilla-FT*	+1.93	+1.28	+1.03	+2.12	+1.66	+1.61
Qwen1.5-MoE	Vanilla	79.72	69.28	44.20	77.25	69.06	67.90
	Vanilla-FT	82.02	77.01	51.10	74.96	69.06	70.83
	MoE-I^2	75.08	71.68	41.13	53.08	66.54	61.50
	MoE-Pruner	69.14	52.02	29.10	42.99	59.12	50.47
	MoEITS	75.20	72.30	36.01	61.27	67.46	62.45
	PCoMoE	**86.18**	**80.43**	**52.30**	**77.67**	**71.90**	**73.70**
	Δ *vs Vanilla-FT*	+4.16	+3.42	+1.20	+2.71	+2.84	+2.87
DeepSeek-V2-Lite	Vanilla	80.61	74.16	46.33	**77.74**	**71.27**	70.02
	Vanilla-FT	81.83	75.32	47.94	76.23	70.40	70.34
	MoE-I^2	76.79	71.80	42.58	55.16	67.64	62.79
	MoE-Pruner	76.61	71.89	40.02	50.94	67.64	61.42
	MoEITS	80.03	77.61	43.77	70.80	67.72	67.99
	PCoMoE	**83.94**	**79.59**	**51.02**	76.66	71.19	**72.48**
	Δ *vs Vanilla-FT*	+2.11	+4.27	+3.08	+0.43	+0.79	+2.14

fline. Based on this cached layout, the fusion optimization combines online filtering, top-k selection, and dispatch preparation into a single lightweight indexing operation, avoiding reconstruction of the full $n \times n$ path space. Because all diagonal paths remain active, the routing graph automatically falls back to vanilla expert execution under low-confidence scenarios, preserving the native MoE interface.

Source-Grouped Compute Reuse. Following routing, PCoMoE aggregates active trajectories by their expansion-side sources to eliminate redundant tensor evaluations. This source-grouped dispatch changes the dispatch unit from individual paths (i, j) to source groups, allowing paths that share the same expansion-side operator to be scheduled together. Denoting the unique selected sources as $\mathcal{S}^{(\ell)}(h) = \{i \mid \exists j, (i, j) \in \Pi^{(\ell)}(h)\}$ and their associated projection targets as $\mathcal{T}_i^{(\ell)}(h) = \{j \mid (i, j) \in \Pi^{(\ell)}(h)\}$, the final layer output is formulated as:

$$y = \sum_{i \in \mathcal{S}^{(\ell)}(h)} \sum_{j \in \mathcal{T}_i^{(\ell)}(h)} \beta_{i,j}^{(\ell)}(h)\, D_j\left(U_i(h)\right).$$

By executing each compute-heavy expansion operator $U_i(h)$ exactly once per unique source, expansion evaluations scale with the active source count $|\mathcal{S}^{(\ell)}(h)|$ instead of the total path budget $|\Pi^{(\ell)}(h)|$. This hardware-software co-design converts structural sharing into deterministic latency reductions.

5 Evaluation

5.1 Experimental Setup

Implementation. Performance profiling is conducted on a server equipped with NVIDIA H20 GPUs and Intel Xeon 6759P-C CPUs. All evaluations are performed on a single GPU under a unified precision and software environment.

Models. We validate PCoMoE across three MoE models spanning diverse routing granularities: Qwen1.5-MoE-A2.7B (60 routed experts, top-4 activation) (Qwen Team, 2024), Mixtral-8x7B-v0.1 (8 routed experts, top-2 activation) (Jiang et al., 2024), and DeepSeek-V2-Lite (64 routed experts, top-6 activation) (DeepSeek-AI et al., 2024).

Fine-tuning. We use a 25K-example mixture of Alpaca and SQuAD with AdamW. Vanilla-FT and PCoMoE use the same LoRA configuration for gating adaptation, with rank 16 and a batch size of 32, while all expert weights remain frozen. PCoMoE additionally learns path compatibility biases for compositional routing and progressive pruning.

Datasets. Downstream model quality is verified on five benchmarks: BoolQ (Clark et al., 2019), ARC-Easy, ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), and WinoGrande (Sakaguchi et al., 2019). We report standard accuracy for BoolQ and WinoGrande, normalized accuracy for ARC and HellaSwag, and the

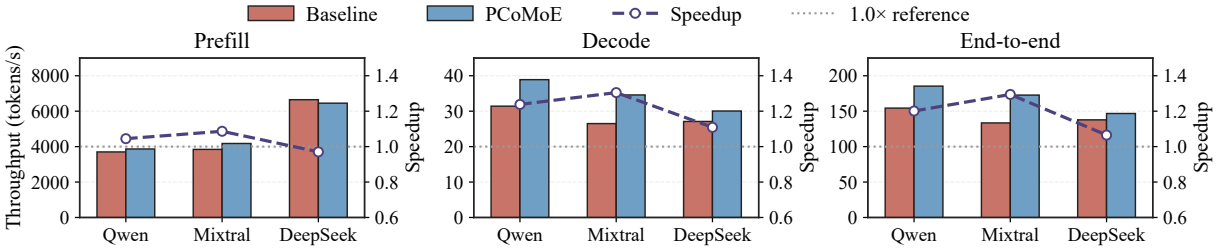

Figure 9: Token throughput and speedup of PCoMoE against vanilla MoE baselines.

macro average across all five tasks as the aggregate representation fidelity metric.

5.2 Quality and Efficiency

Accuracy. Table 1 lists zero-shot accuracy across five benchmarks. We compare PCoMoE with MoE-I^2 (Yang et al., 2024), MoE-Pruner (Xie et al., 2024), and MoEITS (Balderas et al., 2026). On Qwen1.5-MoE, PCoMoE improves the macro average score from 67.90 to 73.70, corresponding to a +5.80-point gain over Vanilla. Under the matched fine-tuning setting, PCoMoE further outperforms Vanilla-FT by +2.87 points. For Mixtral-8x7B and DeepSeek-V2-Lite, PCoMoE improves over Vanilla by +2.10 and +2.46 points, respectively, while retaining +1.61 and +2.14-point gains over their matched Vanilla-FT baselines. These results indicate that the activated off-diagonal paths introduce viable execution trajectories beyond the gains from standard fine-tuning without disrupting representation capacity.

Inference Acceleration. Figure 9 details stage-specific token throughput. PCoMoE yields up to a 1.305× decode speedup and a 1.294× end-to-end speedup over vanilla baselines. On Mixtral-8x7B, decoding speed increases from 26.50 to 34.58 tokens per second, and end-to-end generation increases from 133.43 to 172.72 tokens per second. PCoMoE delivers over 20% throughput improvements on Qwen1.5-MoE and maintains efficiency gains under the top-6 routing of DeepSeek-V2-Lite. Prefill performance remains identical to vanilla models because the layer-static masking and source-grouped compute reuse loop strictly target the autoregressive decoding phase.

5.3 Ablation and Sensitivity Study

Design Ablation. Table 2 isolates the effects of matched fine-tuning, router adaptation, path composition, and learned compatibility on Qwen1.5-MoE. Vanilla-FT improves the average accuracy from 67.90 to 70.83, showing that standard adaptation contributes part of the quality gain. However,

Table 2: Design ablation on Qwen1.5-MoE. Decode throughput is measured in tokens/s.

Method	Avg. Acc. (%)	Decode	Speedup
Vanilla	67.90	31.42	1.000×
Vanilla-FT	70.83	31.71	1.009×
Router-Only FT	67.51	32.37	1.030×
Frozen-Bias	55.50	40.41	1.286×
PCoMoE	**73.70**	38.90	1.238×

Table 3: Ablation of PCoMoE execution optimizations on Qwen1.5-MoE. Throughput is measured in tokens/s.

Configuration	Throughput	Speedup
Base PCoMoE	110.7	1.00×
+ Routing	127.3	1.15×
+ Fusion	141.7	1.28×
+ Dispatch	160.5	1.45×
+ Routing + Dispatch	171.6	1.55×
+ Fusion + Dispatch	179.4	1.62×
All Optimizations	**191.4**	**1.73×**

Router-Only FT does not reproduce the improvement of PCoMoE, while composition with frozen compatibility biases increases Decode throughput but substantially degrades accuracy to 55.50. Full PCoMoE further improves over Vanilla-FT by 2.87 points while retaining a 1.238× Decode speedup, indicating that learned path compatibility is critical to the final quality–efficiency balance.

Execution Ablation. Table 3 isolates the end-to-end throughput impacts of PCoMoE execution optimizations on Qwen1.5-MoE. Base PCoMoE denotes the path-compositional model after expert decoupling and path construction, but before enabling the execution optimizations in Table 3. Path routing, fusion, and source-grouped dispatch individually yield 1.15×, 1.28×, and 1.45× speedups over this reference. Combining these components compounds efficiency, culminating in a 1.73× cumulative acceleration (191.4 tokens/s). Thus, hardware-aligned scheduling is required to translate path expansion into throughput gains.

Pruning Sensitivity. Figure 10(a) tracks active path density across layers. The initial two MoE

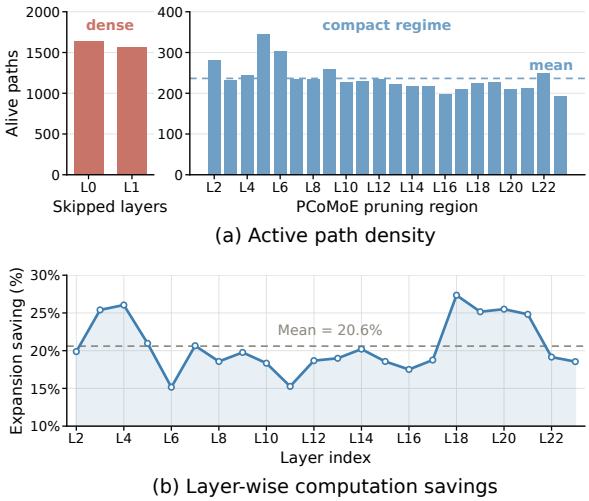

Figure 10: Layer-wise sensitivity analysis.

blocks maintain high density because inseparable compatibility biases render early sparsification lossy. From the third block onward, active paths contract sharply due to the high separability of redundant routes. PCoMoE thus preserves the native configuration in the first two layers and executes structural pruning exclusively from the third block onward.

Reuse Heterogeneity. Figure 10(b) shows the layer-wise arithmetic savings achieved by expansion-side reuse on Qwen1.5-MoE. Reductions average 20.6% but vary from 15% to 27% across blocks, confirming that efficiency gains are non-uniform across the hierarchy. PCoMoE leverages this variance to dynamically allocate layer-wise path budgets based on local compatibility scores, improving overall throughput while avoiding excessive pruning in sensitive regions.

6 Conclusion

This paper introduces **PCoMoE**, an execution framework that shifts MoE inference from monolithic expert selection to fine-grained subtransformation composition. By integrating structural path decomposition, compatibility-guided gating, and source-grouped compute reuse, PCoMoE leverages intra-expert computational redundancy while strictly maintaining native MoE layer interfaces. Evaluations show that PCoMoE improves both inference throughput and downstream representation fidelity over vanilla MoE baselines.

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China (2024YFE0204300), the National Natural Science Foundation of China (Grant No. 62402311), the Natural Science Foundation of Shanghai (Grant No. 24ZR1433700), and the Key Research and Development Program of Shanghai (25LN3201200).

Limitations

First, while we demonstrate PCoMoE using standard SwiGLU-style MoE structures common in prevailing open-source LLMs, the underlying principle of path composition is generic and can be extended to alternative expert internals by adapting the operator boundaries. Second, our current system primarily optimizes autoregressive decoding, the primary latency bottleneck in MoE serving; integrating PCoMoE with complementary distributed or prefill-stage optimizations represents a promising and orthogonal future direction. Finally, although the offline calibration stage incurs zero inference-time overhead, further incorporating this process into automated end-to-end model compilation pipelines will help streamline broader deployments.

References

Luis Balderas, Miguel Lastra, and José M. Benítez. 2026. Moeits: A green ai approach for simplifying moe-llms. *arXiv preprint arXiv:2604.10603*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*.

Krishna Teja Chitty-Venkata, Sandeep Madireddy, Murali Emani, and Venkatram Vishwanath. 2025. LExI: Layer-adaptive active experts for efficient moe model inference. *arXiv preprint arXiv:2509.02753*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, and 138 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, and 8 others. 2022. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, pages 5547–5569.

Zhiyuan Fang, Zicong Hong, Yuegui Huang, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. 2025. Fate: Fast edge inference of mixture-of-experts models via cross-layer gate. *arXiv preprint arXiv:2502.12224*.

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

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*.

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Fangxin Liu, Junjie Wang, Ning Yang, Zongwu Wang, Junping Zhao, Li Jiang, and Haibing Guan. 2025. ASTER: Adaptive dynamic layer-skipping for efficient transformer inference via markov decision process. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 11853–11861.

Fangxin Liu, Ning Yang, Haomin Li, Zongwu Wang, Zhuoran Song, Songwen Pei, and Li Jiang. 2024. SPARK: Scalable and precision-aware acceleration of neural networks via efficient encoding. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1029–1042.

Fangxin Liu, Ning Yang, Jingkui Yang, Zongwu Wang, Chenyang Guan, Yu Feng, Li Jiang, and Haibing Guan. 2026. EARTH: An efficient MoE accelerator with entropy-aware speculative prefetch and result reuse. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 633–646.

Fangxin Liu, Wenbo Zhao, Zhezhi He, Yanzhi Wang, Zongwu Wang, Changzhi Dai, Xiaoyao Liang, and Li Jiang. 2021. Improving neural network efficiency via post-training quantization with adaptive floating-point. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5281–5290.

Qwen Team. 2024. Qwen1.5-moe: Matching 7b model performance with 1/3 activated parameters. Qwen Blog. Qwen1.5-MoE-A2.7B model release.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Zheyue Tan, Zhiyuan Li, Tao Yuan, Dong Zhou, Weilin Liu, Yueqing Zhuang, Yadong Li, Guowei Niu, Cheng Qin, Zhuyu Yao, Congyi Liu, Haiyang Xu, Boxun Li, Guohao Dai, Bo Zhao, and Yu Wang. 2025. ReXMoE: Reusing experts with minimal overhead in mixture-of-experts. *arXiv preprint arXiv:2510.17483*.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. 2024. Hobbit: A mixed precision expert offloading system for fast moe inference. *arXiv preprint arXiv:2411.01433*.

Daniil Vankov, Nikita Ivkin, Kyle Ulrich, Xiang Song, Ashish Khetan, and George Karypis. 2026. Xshare: Collaborative in-batch expert sharing for faster moe inference. *arXiv preprint arXiv:2602.07265*.

Yanyue Xie, Zhi Zhang, Ding Zhou, Cong Xie, Ziang Song, Xin Liu, Yanzhi Wang, Xue Lin, and An Xu. 2024. Moe-pruner: Pruning mixture-of-experts large language model using the hints from its router. *arXiv preprint arXiv:2410.12013*.

Leyang Xue, Yao Fu, Zhan Liu, and Mahesh K. Marina. 2024. Moe-infinity: Activation-aware expert offloading for efficient moe serving. *arXiv preprint arXiv:2401.14361*.

Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. 2024. Moe-i2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. In *Findings of the Association for Computational Linguistics: EMNLP*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. 2024. Adapmoe: Adaptive sensitivity-based expert gating and management for efficient moe inference. *arXiv preprint arXiv:2408.10284*.