

PreMoE: Proactive Inference for Efficient Mixture-of-Experts

Zehua Pei^{1,*}, Ying Zhang^{2,*}, Hui-Ling Zhen², Tao Yuan², Xianzhi Yu²,
Zhenhua Dong², Sinno Jialin Pan¹, Mingxuan Yuan², Bei Yu¹

¹The Chinese University of Hong Kong

²Huawei Technologies Co., Ltd

*Equal contribution.

Abstract

Mixture-of-Experts (MoE) models offer dynamic computation, but are typically deployed as static full-capacity models, missing opportunities for deployment-specific specialization. We introduce PreMoE, a training-free framework that proactively compiles sparse MoE variants for targeted deployment scenarios. At its core is Predicted Expert Utility (PEU), a robust metric for estimating expert importance from router logits through high-confidence threshold filtering and logit transformation, which together stabilize utility estimation under aggressive sparsity. Using PEU scores computed on a small calibration set, PreMoE produces domain-aware expert rankings that can be used to compile either domain-specific specialists or high-efficiency multi-domain generalists, without any retraining. Across MoE models ranging from 30B to 718B parameters, PreMoE preserves near-full accuracy at model-dependent sparsity levels of up to 50%; on DeepSeek-R1, 50% sparsity halves the deployment NPU count and increases throughput by 23%. Controlled studies across calibration sources, formats, random subsets, and domain mixtures confirm that these gains are stable rather than tied to a particular calibration set. PreMoE further exposes a practical deployment trade-off: specialists maximize in-domain efficiency, while synthesized generalists retain broader cross-domain capability at the same sparsity budget. ¹

1 Introduction

Mixture of Experts (MoE) architectures (Shazeer et al., 2017; Lepikhin et al., 2020; Zhou et al., 2022) have enabled unprecedented scaling of language models (Jiang et al., 2024; Dai et al., 2024; Guo et al., 2025; Tang et al., 2025), yet despite their dynamic design, they face a practical contradiction: they are deployed statically. This approach requires the entire set of experts to reside in memory, failing to capitalize on the model’s potential for specialization and leading to significant inefficiency. Prior work has explored expert pruning to address this challenge (Chen et al., 2022; Muzio et al., 2024; Lu et al., 2024), but these methods rely on coarse statistics like activation frequency that fail to capture activation quality, often requiring costly finetuning to recover performance, or employ search procedures that become intractable at scale. We propose a proactive compilation approach that uses a refined signal to compile specialized MoE instances before deployment, without finetuning or expensive search.

We introduce **PreMoE**, a training-free framework that actualizes this approach (Figure 1). At its heart is our novel Predicted Expert Utility (PEU) metric, which robustly measures expert importance by analyzing the model’s native router logits through high-confidence filtering (TopK filtering and adaptive threshold filtering) and logit transformation. By computing PEU scores for all experts across all layers on a domain-specific calibration dataset, PreMoE yields a layer-wise PEU ranking that quantifies which experts are critical for that domain.

¹Code: <https://github.com/JarvisPei/PreMoE>

We refer to this ranking as the domain’s “computational pattern” and use it as a blueprint to identify minimal, high-performance expert subsets and compile specialized model instances.

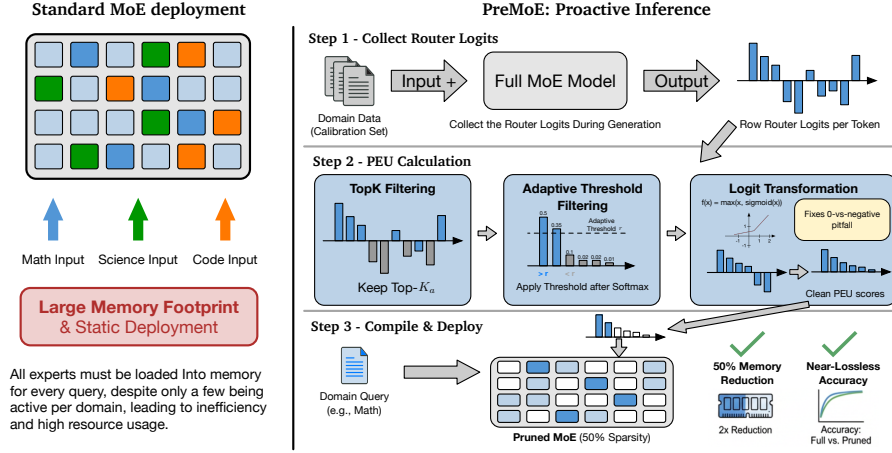


Figure 1: Overview of PreMoE. **Left:** Standard MoE deployment requires all experts in memory despite only a few being active per domain. **Right:** PreMoE’s proactive inference pipeline: (1) collect router logits during generation on calibration data, (2) compute PEU scores via a processing pipeline (TopK filtering → Adaptive threshold filtering → Logit transformation), and (3) compile a pruned MoE with 50% sparsity achieving near-lossless accuracy and 2× memory reduction.

Our contributions are:

- We find that expert utility is highly domain-specific and predictable when measured with a principled signal (Figure 2), enabling effective pre-deployment expert selection.
- We propose PreMoE, a training-free framework built on our Predicted Expert Utility (PEU) metric, which refines router logits via high-confidence filtering and logit transformation to compile specialized MoE instances.
- We demonstrate training-free expert pruning across models from 30B to 718B parameters, preserving near-full accuracy at model-dependent sparsity levels of up to 50%. On DeepSeek-R1, 50% sparsity halves deployment NPUs and parameters while increasing throughput by 23%; controlled analyses confirm stability across calibration sources, formats, sample sizes, random seeds, and domain mixtures.

2 Related Work

Expert pruning methods target the unique structure of Mixture-of-Experts (MoE) models, where expert utility is often imbalanced across tasks. Prior approaches can be broadly categorized into two paradigms, each with significant limitations. We provide broader context on MoE architectures and general LLM efficiency techniques in Appendix F.

The first paradigm relies on *observation-based statistics*. These methods analyze activation patterns during calibration, using metrics such as activation frequency or gating scores to identify and prune experts (Chen et al., 2022; Muzio et al., 2024). While straightforward, coarse statistics can fail to distinguish broadly useful generalists from infrequently used but critical specialists. Activation-aware methods provide richer signals: REAP (Lasby et al., 2025), for example, combines router gates with expert-output norms for one-shot generative MoE compression. Such signals can improve pruning quality, but require instrumentation and reductions on the expert-output path. PreMoE instead asks how far the router’s learned preference alone can go: it profiles scalar router logits before expert dispatch, then filters low-confidence routing noise.

A second paradigm attempts to find optimal expert subsets through computationally intensive *search* procedures. For instance, EEP (Liu et al., 2024) employs an evolutionary search, while NAE (Lu et al., 2024) uses enumeration to evaluate expert combinations based

on minimizing reconstruction loss. Although these methods can yield better results than simple heuristics, their computational cost makes them intractable for modern, large-scale MoE models. These search-based techniques are typically demonstrated on models with a small number of experts (e.g., 8 in Mixtral-7B) and would be infeasible for models like DeepSeek-R1 with 256 experts per layer.

PreMoE departs from these paradigms by using refined router logits with confidence filtering to predict expert utility, enabling training-free compilation that scales to large MoE models.

3 The PreMoE Framework

3.1 Motivation: The Predictability of Expert Activations

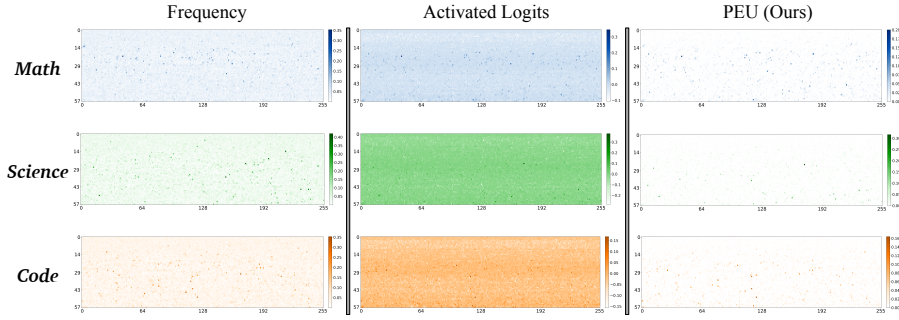


Figure 2: Comparison of expert utility estimation methods across three domains (**Math**, **Science**, **Code**) on DeepSeek-R1. Each heatmap shows 58 MoE layers (y-axis) by 256 experts (x-axis). **Left column (Frequency):** Simple activation counting produces diffuse, noisy patterns with minimal domain-specific structure. **Middle column (Activated Logits):** Aggregating logits only from activated experts yields denser but still broadly distributed signals. **Right column (Ours):** Our PEU metric, combining high-confidence filtering and logit transformation, reveals sharp, sparse, and highly structured patterns unique to each domain. The contrast demonstrates that expert utility is predictable and domain-specific when measured with a principled signal, motivating our proactive approach.

The foundation of PreMoE rests on a key insight: expert utility in large MoE models is not random, but highly structured and therefore predictable when measured with the right signal. Figure 2 compares three approaches to estimating expert utility across Math, Science, and Code domains. In contrast to simple frequency or activated-logits aggregation, our PEU metric reveals sharp, sparse, and highly domain-specific patterns. These patterns are not only visually distinct across domains but also stable within each domain, indicating that expert utility is predictable when properly measured. This predictability enables a paradigm shift: from reactive, static deployment to proactive compilation, where we build specialized, efficient model instances tailored to specific deployment scenarios.

Notation. We briefly recap the MoE forward pass to establish notation. For an input token $\mathbf{x}$, a router network produces logits $\mathbf{s}(\mathbf{x}) = \{s_1(\mathbf{x}), \dots, s_{N_r}(\mathbf{x})\}$, where $s_i(\mathbf{x})$ represents the unnormalized preference for the i -th routed expert. The top K experts, forming set $\mathcal{K}$, are activated with gating scores g_i . The layer output is $F_{\text{MoE}}(\mathbf{x}) = \mathbf{x} + \sum_{i \in \mathcal{K}} g_i E_i^r(\mathbf{x})$.

3.2 Forecasting Expert Utility

To build a proactive system, we need a reliable, low-overhead signal to forecast expert utility. The most natural choice is the router’s own logits, as they represent the model’s internal, token-by-token preference for each expert. Crucially, unlike simple activation frequency, logits capture the *strength* of the router’s preference. However, raw logits can be noisy. Our goal is to refine this signal to robustly measure the router’s *decisive preference*.

To achieve this, we introduce a two-stage process. First, we **filter for high-confidence activations** using a top- K_a candidate pool and a confidence threshold r . Second, for only the logits that pass this filter, we apply a **logit transformation** $f(s)$ to make them suitable for robust aggregation. This yields a token-level utility score $\tilde{s}_i(\mathbf{x})$ for each expert, formalized as:

$$\mathcal{E}_{K_a} = \text{TopK}(\{s_j(\mathbf{x})\}_{j=1}^{N_r}, K_a) \quad (1)$$

$$p_i(\mathbf{x}) = \frac{\exp(s_i(\mathbf{x}))}{\sum_{k \in \mathcal{E}_{K_a}} \exp(s_k(\mathbf{x}))}, \quad i \in \mathcal{E}_{K_a} \quad (2)$$

$$\tilde{s}_i(\mathbf{x}) = \begin{cases} f(s_i(\mathbf{x})), & i \in \mathcal{E}_{K_a} \wedge p_i(\mathbf{x}) \geq r, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

Why high-confidence filtering? Router preference signals are heavy-tailed across tokens; without filtering, aggregations are contaminated by many low-confidence activations that dilute expert utility estimates. Our high-confidence filtering comprises two stages: (1) **TopK filtering**, which narrows the candidate pool to the top- K_a experts by logit magnitude, and (2) **adaptive threshold filtering**, which applies a confidence threshold r to retain only decisive activations. Together, these act as a principled denoiser. We set r adaptively for each MoE layer l as the average probability of the top-ranked expert within the top- K_a pool:

$$r_l = \mathbb{E}_{\mathbf{x} \in \mathcal{X}_T} \left[\max_{i \in \mathcal{E}_{K_a}^l(\mathbf{x})} p_i^l(\mathbf{x}) \right] \quad (4)$$

This layer-wise adaptive rule is computed on the calibration set and makes our method robust across different models and domains with minimal tuning.

Why logit transformation? Raw router logits are signed, and naive averaging leads to a 0-vs-negative pitfall where unselected experts (scored as 0) can spuriously outrank selected experts with negative logits. A transformation that maps negatives away from the positive range is therefore necessary. Our default choice, $f(s) = \max(s, \text{sigmoid}(s))$, retains large positive evidence while rectifying negatives via $\text{sigmoid}(s)$, avoiding compression of strong signals. In practice, threshold filtering stabilizes utility estimation and largely removes sensitivity to the particular transformation choice, enabling recovery of full-model accuracy at high sparsity. The final **Predicted Expert Utility (PEU)** for an expert is the average of these token-level scores over a calibration dataset, $\text{PEU}_i^T = \mathbb{E}_{\mathbf{x} \in \mathcal{X}_T} [\tilde{s}_i(\mathbf{x})]$. This PEU score provides a robust forecast of expert importance for a given domain T . Full derivations are in Appendix A.1.

From PEU to Computational Patterns. By computing PEU scores for all N_r experts across all MoE layers in the model, we obtain a complete ranking of expert importance for domain T . We define this layer-wise collection of PEU scores as the **computational pattern** for domain T :

$$\text{Pattern}^T = \left\{ \left\{ \text{PEU}_i^{T,l} \right\}_{i=1}^{N_r} \right\}_{l=1}^L \quad (5)$$

where L is the number of MoE layers in the model. This pattern is a compact, interpretable representation of which experts are critical for a domain, serving as a blueprint for constructing specialized model instances. Crucially, extracting a pattern requires only one offline forward pass over a small calibration set, with no gradient computation or parameter updates. Five to ten samples already recover most of the final accuracy, while our larger default sets mainly reduce variance in the PEU ranking (Appendix C.7). The pattern is computed once before deployment and its cost is amortized across inference requests.

3.3 Compiling Instances from Patterns

Given a computational pattern, we compile specialized MoE instances by populating a lightweight model skeleton *only* with the expert weights identified by the pattern—the full model is never loaded into memory. The detailed algorithm is provided in Appendix A.2.

Table 1: Performance of Domain-Specific Specialists. For each base model we prune routed experts to the highest sparsity that preserves near loss-less accuracy: DeepSeek-R1 at 50% (keep 128/256 per layer), openPangu-Ultra at 31.25% (keep 176/256), and Qwen3-30B-A3B at 50% (keep 64/128). Entries are accuracy (%); **Avg** is the macro average over the listed benchmarks; Δ is the average accuracy change compared to the full model. Baselines: Random, Frequency (activation counts), Act-Logits (aggregate logits of activated experts without threshold filtering), All-Logits (aggregate logits of all experts), SEER (L) and SEER (G) from Muzio et al. (2024), and EASY-EP.

Method	MATH-500	AIME 2024	AIME 2025	CNMO 2024	GPQA	LiveCodeBench	Avg	Δ
DeepSeek-R1 (50% sparsity)								
Full	96.60	77.08	65.83	71.18	73.23	69.12	75.51	–
Random	54.00	6.67	1.25	30.88	39.90	27.43	26.69	-48.82
Frequency	88.80	60.83	46.19	64.58	33.33	5.88	49.93	-25.58
All-Logits	3.60	0.00	0.00	0.35	28.79	0.00	5.46	-70.05
SEER (L)	54.80	2.91	2.08	36.63	36.87	8.09	23.56	-51.95
SEER (G)	53.00	3.33	1.67	36.63	36.87	14.34	24.31	-51.20
Act-Logits	88.20	70.00	55.00	62.62	48.48	52.94	62.87	-12.64
EASY-EP	97.20	79.17	68.33	72.18	70.12	61.11	74.69	-0.82
PreMoE	97.60	79.58	68.33	75.00	72.22	66.36	76.52	+1.01
openPangu-Ultra (31.25% sparsity)								
Full	97.40	80.83	75.42	77.43	76.77	67.65	79.25	–
Random	85.00	41.25	34.17	54.51	60.10	25.74	50.13	-29.12
Frequency	96.40	76.25	65.83	75.69	52.02	51.84	69.67	-9.58
All-Logits	12.00	0.00	0.00	2.95	16.67	0.74	5.39	-73.86
SEER (L)	95.20	75.41	62.08	73.61	62.63	58.82	71.29	-7.96
SEER (G)	96.00	71.67	69.17	72.40	57.58	58.82	70.93	-8.32
Act-Logits	97.20	77.50	73.33	72.40	79.29	61.76	76.91	-2.34
EASY-EP	96.82	77.20	72.17	73.43	76.74	62.36	76.45	-2.80
PreMoE	96.80	80.41	71.67	79.17	75.76	66.91	78.45	-0.80
Qwen3-30B-A3B (50% sparsity)								
Full	97.20	91.25	82.92	78.65	68.69	65.44	80.69	–
Random	44.80	5.00	2.92	7.99	17.68	0.00	13.07	-67.62
Frequency	90.40	60.42	46.67	54.69	47.98	25.37	54.26	-32.76
All-Logits	1.60	0.00	0.00	0.00	0.51	0.00	0.35	-80.34
SEER (L)	1.60	0.00	0.00	0.00	0.00	2.21	0.64	-80.05
SEER (G)	0.20	0.00	0.00	0.00	0.00	1.84	0.34	-80.35
Act-Logits	1.40	0.00	0.00	0.00	3.54	0.00	0.82	-79.87
EASY-EP	58.62	62.46	44.92	52.65	50.48	56.44	54.26	-26.43
PreMoE	96.40	88.33	79.58	81.94	68.18	65.07	79.92	-0.77

We propose two primary compilation strategies based on computational patterns:

1. Compiling Domain-Specific Specialists. For an application with a narrow focus, we identify the computational pattern for that domain. The top- M experts with the highest PEU scores are selected, where M is the desired budget. These selected experts form the new, pruned set of routed experts $\{E_i^r(\mathbf{x})\}_{i=1}^M$ for the compiled specialist instance.

2. Compiling High-Efficiency Generalists. For general-purpose applications, we create a single, multi-domain model by synthesizing a unified computational pattern. Instead of taking a simple union of top experts, we first average token-level utility within each domain and then combine the domain utilities with weights w_d . Our deployment-agnostic default is $w_d = 1/D$, so domains contribute equally regardless of calibration-set size; application-specific priors can instead emphasize a target workload. This blended ranking identifies experts useful across the targeted domains, and its top- M experts form a sparse generalist that preserves broad capability under a fixed expert budget. Appendix D.2 shows that changing the weights produces predictable domain trade-offs.

4 Experiments

4.1 Experimental Setup

Models and Datasets. Our experiments are conducted on three state-of-the-art MoE models: **DeepSeek-R1-671B** (Guo et al., 2025) (256 experts per layer), **openPangu-Ultra-MoE-718B** (Tang et al., 2025) (256 experts per layer), and **Qwen3-30B-A3B** (Qwen3-235B-A22B-

Table 2: Performance of the High-Efficiency Generalist. The PreMoE Generalist is compiled by synthesizing PEU scores at the same sparsity as specialists. The “Trivial Union” baseline unions smaller fixed-size per-domain expert sets, yielding lower effective sparsity than PreMoE. Delta (Δ): average accuracy change compared to the full model.

Setting	Sparsity	MATH-500	AIME 2024	AIME 2025	CNMO 2024	GPQA	LiveCodeBench	Avg	Δ
DeepSeek-R1									
Full Model	0%	96.60	77.08	65.83	71.18	73.23	69.12	75.51	–
Trivial Union	39.87%	96.60	78.75	70.42	75.17	72.22	66.18	76.56	+1.05
PreMoE Generalist	50%	96.40	78.33	70.42	75.17	70.71	65.07	76.02	+0.51
openPangu-Ultra									
Full Model	0%	97.40	80.83	75.42	77.43	76.77	67.65	79.25	–
Trivial Union	26.24%	97.00	80.00	70.42	80.38	74.24	63.60	77.61	-1.64
PreMoE Generalist	31.25%	97.40	80.00	73.34	78.82	78.79	65.81	79.03	-0.22
Qwen3-30B-A3B									
Full Model	0%	97.20	91.25	82.92	78.65	68.69	65.44	80.69	–
Trivial Union	33.76%	96.40	87.92	84.17	83.33	71.72	61.76	80.88	+0.19
PreMoE Generalist	50%	96.60	86.25	79.17	79.34	68.69	54.04	77.35	-3.34

Thinking-2507) (Team, 2025) (128 experts per layer). We use a diverse set of open-source training datasets to create calibration sets for three primary domains: **Math** (800 samples from Nemotron-Post-Training-Dataset-v1 (Nathawani et al., 2025) “math” split), **Science** (200 samples from Nemotron-Post-Training-Dataset-v1 (Nathawani et al., 2025) “stem” split), and **Code** (600 samples from OpenCoder (Huang et al., 2024), with 300 samples under opcsft-stage1 and 300 samples under opcsft-stage2). For the High-Efficiency Generalist, we augment this data with 600 samples from LMSYS-Chat-1M (Zheng et al., 2023) to broaden its capabilities. Samples are drawn randomly without benchmark-specific selection. We collect router statistics on each question together with its full-model reasoning trace under fixed per-model generation settings, covering both prefill and decoding-time routing. Calibration updates no weights and uses no evaluation answers; held-out GSM8K/MBPP validation and prompt-level decontamination checks are reported in Appendix D.3. Notably, just 5–10 samples recover most of the final performance, while larger default sets mainly reduce ranking variance (Appendix C.7).

Evaluation Benchmarks. To evaluate the performance of the compiled models, we use a challenging set of reasoning benchmarks across our three target domains. For **Math**, we use MATH-500 (Hendrycks et al., 2020), AIME 2024, AIME 2025, and CNMO 2024. For **Science**, we use GPQA (Rein et al., 2024), a graduate-level question-answering benchmark. For **Code**, we use LiveCodeBench (Jain et al., 2024) (272 problems in the window from 8/1/2024 to 1/1/2025) for evaluating code generation capabilities.

Implementation Details Pattern collection is performed offline on servers equipped with 64 Ascend 910B2-64GB NPUs. For hyperparameters, we set K_a to match the model’s default number of activated experts (e.g., 8 for DeepSeek-R1-671B), and we use $f(s) = \max(s, \text{sigmoid}(s))$ as our default logit transformation. The confidence threshold r is set adaptively for each domain and for each MoE layer l following the definition in Eq. 4 (see Methodology). This layer-wise adaptive approach makes our method robust across different models and domains with minimal tuning.

Baselines. We compare PreMoE against a suite of baselines: **Random** selection, expert ranking by activation **Frequency**, and several variants of logit collection based on **all-logits** and **activated-logits** (denoted ‘All-Logits’ and ‘Act-Logits’ in tables). We also compare with local and global **SEER-MoE** (Muzio et al., 2024), **EASY-EP** (Dong et al., 2025), which uses few-shot demonstrations to identify relevant experts, and **REAP** (Lasby et al., 2025), which scores experts by router-weighted activation norms. REAP is evaluated separately under a matched protocol because its experiment reports normalized retention rather than the absolute scores in Table 1.

4.2 Main Results: Efficacy of PreMoE

4.2.1 Compiling Domain-Specific Specialists

We first evaluate PreMoE’s ability to create sparse, high-performance models specialized for a single domain. For each base model, we first determine a target sparsity using a small

Table 3: Quality–profiling-cost comparison on DeepSeek-R1 at 50% sparsity. Accuracy is normalized to the same dense model and averaged over MATH-500, GPQA, and Live-CodeBench. Profiling overhead is measured beyond the shared sparse-MoE calibration forward pass on approximately 600 samples.

Method	Profiled signal	Norm. avg.	Overhead
REAP	router + activation norm	92.0	+75%
PreMoE	scalar router logits	91.2	+12%

sweep with PreMoE, selecting the highest sparsity at which the compiled specialist remains within a 1-point average drop from the full model. We then evaluate all baselines under the same sparsity budget for a fair comparison: 50% for DeepSeek-R1 (keeping 128 of 256 experts per MoE layer), 31.25% for openPangu-Ultra-MoE (keeping 176 of 256), and 50% for Qwen3-30B-A3B (keeping 64 of 128). Table 1 shows the results.

Across all three models, PreMoE is the only method that consistently matches the full baseline, achieving $\Delta=+1.01$ on DeepSeek-R1, $\Delta=-0.80$ on openPangu-Ultra, and $\Delta=-0.77$ on Qwen3-30B-A3B. The ablation variant Act-Logits—which uses only TopK filtering, without threshold filtering or logit transformation—is substantially less stable, with drops of 12.64 points on DeepSeek-R1, 2.34 points on openPangu-Ultra, and 79.87 points on Qwen3-30B-A3B. This shows that our threshold filtering and logit transformation are not merely incremental refinements, but are crucial for robust expert selection under aggressive sparsity.

Compared with EASY-EP, PreMoE achieves higher average accuracy on all three base models: 76.52 vs. 74.69 on DeepSeek-R1, 78.45 vs. 76.45 on openPangu-Ultra, and 79.92 vs. 54.26 on Qwen3-30B-A3B. The contrast is especially striking on Qwen3-30B-A3B, where EASY-EP suffers a 26.43-point average drop while PreMoE remains near loss-less. In this challenging regime, nearly all alternative baselines fail: on Qwen3-30B-A3B, Random, All-Logits, SEER, and Act-Logits collapse almost entirely, while Frequency still incurs a 32.76-point drop. Overall, these results show that PreMoE can reliably compile domain-specific specialists across a wide range of MoE scales, with the largest advantage appearing precisely where expert sharing makes pruning hardest.

4.2.2 Compiling High-Efficiency Generalists

A key advantage of PreMoE is its ability to create a single, sparse model that retains broad, multi-domain capabilities. We compile a generalist model by synthesizing patterns from the Math, Science, and Code domains, rather than simply concatenating per-domain specialist masks. As shown in Table 2, the compiled generalist achieves minimal performance loss on the larger models ($\Delta=+0.51$ on DeepSeek-R1; $\Delta=-0.22$ on openPangu-Ultra) at the same sparsity as specialists. On Qwen3-30B-A3B, the generalist shows a larger drop ($\Delta=-3.34$), consistent with the specialist results and suggesting lower expert redundancy at this scale.

We compare to a Trivial Union baseline that unions fixed-size per-domain expert sets. While Trivial Union maintains similar accuracy, it achieves lower effective sparsity, showing that our synthesis approach is more efficient. This indicates that multi-domain compilation is not simply a set-union problem: retaining broad capability requires balancing shared experts with a small number of domain-critical ones. As we show below, this structure also explains why specialists can be highly efficient in-domain while the synthesized generalist offers a better operating point when broader capability retention is required.

Robustness. The same specialist and generalist trends hold across alternative calibration sources, random subsets, token formats, and domain mixtures; full controlled results are in Appendix D.1.

4.2.3 Comparison with Activation-Aware Pruning

REAP combines router gates with expert activation norms. Under the matched 50%-sparsity protocol in Table 3, PreMoE is within 0.8 normalized points of REAP, while its measured

profiling overhead is 12% rather than 75%. This isolates the benefit and cost of activation-side information: PreMoE uses only scalar router scores, whereas REAP additionally reduces expert outputs. Detailed quality, memory, and cost results are in Appendix D.6.

4.3 Where PreMoE Helps Most: Accuracy-Efficiency Trade-offs

We investigate how performance and deployment efficiency scale with sparsity on DeepSeek-R1. Figure 3 visualizes the accuracy-efficiency trade-off, revealing domain-dependent robustness and significant infrastructure savings.

Domain-Dependent Robustness. Mathematical reasoning exhibits remarkable robustness: MATH-500 maintains 96–97% accuracy even at 62.5% sparsity. At 75% sparsity, it retains 93.4%, while GPQA drops to 45.45% and LCB to 49.63%. This suggests that domains differ substantially in expert redundancy: mathematical reasoning appears to rely on a more redundant expert pool, whereas science and code depend on more diverse and specialized capabilities. We therefore do not assume that 50% pruning is universally lossless: the safe budget is selected per model and workload (50% for DeepSeek-R1 and Qwen3-30B-A3B, but 31.25% for openPangu-Ultra). This result is consistent with the overlap analysis above, where only a small subset of top-ranked experts remains strongly domain-specific.

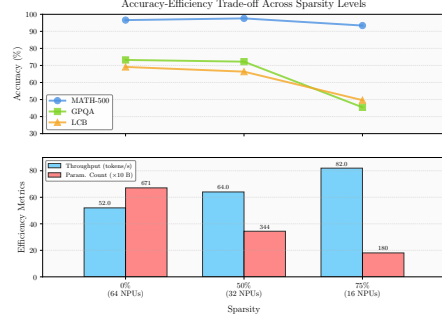


Figure 3: Accuracy-efficiency trade-off: 75% sparsity preserves MATH accuracy; 50% sparsity halves infrastructure with 23% throughput gain.

Practical Deployment Benefits. At 50% sparsity, we achieve near-lossless accuracy with substantial infrastructure savings: 2× fewer NPUs and parameters, plus 23% throughput gain. At 75% sparsity, despite accuracy degradation on some tasks, we gain 4× resource reduction and 58% throughput increase; a potentially acceptable trade-off for latency-critical applications. Complete accuracy and efficiency metrics at various sparsity levels, including comparisons between domain-specific specialists and multi-domain generalists, are provided in Appendix C.8, C.9, and C.10.

4.4 Why PreMoE Works

4.4.1 Logit Transformation and Threshold Filtering

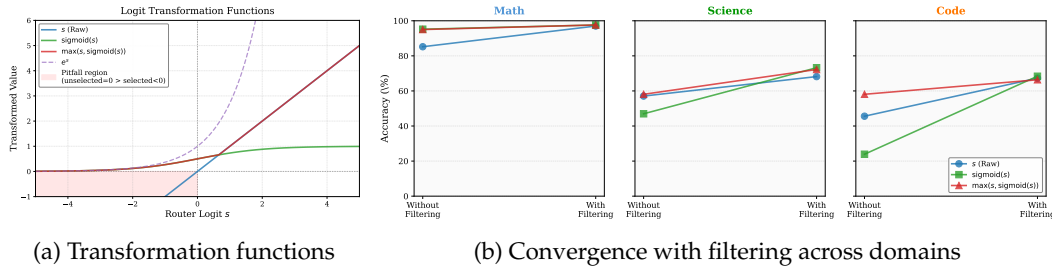


Figure 4: Analysis of logit transformation and threshold filtering. (a) Transformation curves illustrating the 0-vs-negative issue; the shaded region marks cases where unselected experts (score 0) outrank selected experts with negative logits. (b) Convergence across Math, Science, and Code. Without threshold filtering, performance depends strongly on the transformation; with filtering, all methods converge, showing that thresholding is the main factor in stabilizing utility estimation.

All experiments in this subsection use TopK filtering as the baseline, on top of which we study threshold filtering and logit transformation. Raw router logits s can be negative or

Table 4: Expert role diversity. **Left:** Frequently activated generalists with low utility: often selected, but rarely the top choice, so frequency overestimates them while PEU demotes them. **Right:** Infrequent specialists with high utility: rarely selected, but often the top choice when activated, so frequency overlooks them while PEU promotes them.

(a) Frequently-Activated Generalists (Low Utility)

Expert	#Top- K_d	#Rank-1	R-1 Ratio	Freq.	PEU
(0, 161)	31182	7	0.02%	140	256
(57, 205)	42802	0	0.00%	63	252
(2, 54)	48168	3	0.01%	22	256
(6, 158)	61977	92	0.15%	10	254
(57, 24)	47556	0	0.00%	53	253

(b) Infrequent Specialists (High Contextual Utility)

Expert	#Top- K_d	#Rank-1	R-1 Ratio	Freq.	PEU
(3, 243)	8382	4979	59.4%	256	51
(36, 9)	2518	1482	58.9%	255	77
(45, 220)	640	368	57.5%	256	137
(50, 223)	655	352	53.7%	256	148
(57, 95)	8254	4370	53.0%	227	58

positive. When utilities are aggregated across tokens, unselected experts receive a default score of 0, which can incorrectly exceed the scores of selected experts with negative logits. Averaging raw logits is therefore problematic. To address this, we transform logits before aggregation. A natural choice is $\sigma(s)$, which maps logits to $(0, 1)$ and avoids the sign issue, but it also compresses large positive values. Our default rectifier, $f(s) = \max(s, \sigma(s))$ uses $\sigma(s)$ for negative logits while preserving positive logits unchanged. We also evaluate an exponential variant $f_{\text{exp}}(s) = e^s$; additional results are deferred to Appendix B.

Figure 4 shows two main findings. First, without threshold filtering, the choice of logit transformation matters: $f(s) = \max(s, \sigma(s))$ is the most consistent across domains, while $\sigma(s)$ performs well on Math but degrades on Science and Code. Second, once threshold filtering is introduced, all transformations achieve similarly strong performance. This indicates that logit transformation mainly corrects the negative-logit scoring issue, whereas threshold filtering is the primary factor in stabilizing utility estimation. We therefore use $\max(s, \sigma(s))$ as the default.

We further ablate the thresholding strategy. On DeepSeek-R1 at 50% sparsity, the adaptive threshold achieves 98.0/64.71/70.71 on MATH-500/LCB/GPQA, outperforming fixed thresholds of 0.15 (94.2/64.34/68.69) and 0.3 (96.2/65.07/68.18). A full component ablation is provided in Appendix C.1.

4.4.2 Diverse and Critical Expert Roles

Analyzing the PEU rankings on the Code domain reveals distinct expert roles, which in turn clarify why frequency-based pruning fails under aggressive sparsity.

Frequently-Activated Generalists (Low Utility). Some experts are considered by the router thousands of times but are rarely the decisive top choice. Table 4a shows five such examples from the Code domain. Frequency-based methods rank these in the top-63, while PEU correctly demotes them to positions 252 to 256. These are “frequent but weak” generalists that occupy model capacity without providing critical utility for code generation tasks.

Infrequent Specialists (High Contextual Utility). Conversely, some experts are rarely activated but are almost always the top choice when they are. Table 4b shows five such examples. Frequency-based methods rank these at the tail, effectively discarding them, while PEU promotes them to positions 51 to 148. These are rare-but-critical specialists whose removal would eliminate essential capabilities for specific code constructs or programming patterns that occur infrequently in the calibration set but are vital when they appear.

These contrasting cases explain why PreMoE maintains performance at high sparsity while frequency-based methods fail: PEU preserves mission-critical specialists while pruning low-utility generalists, whereas frequency does the opposite. This distinction is critical: when we prune to 50% sparsity on DeepSeek-R1 (keeping 128/256 experts), PEU retains the high-utility specialists while Frequency-based methods lose them, explaining the 60.48 percentage point accuracy gap on Code (66.36% vs. 5.88% in Table 1).

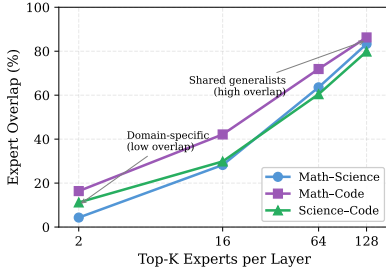


Figure 5: Top-2 specialists exhibit <17% overlap, while top-128 sets share 80-86% experts, revealing strong specialization.

Cross-Domain Expert Overlap.

To quantify expert domain-specificity across utility levels, we measured overlap between domain specialists’ top-ranked experts on DeepSeek-R1. Figure 5 shows overlap declines sharply with expert criticality: at top-128 (50% sparsity), domains share 80–86% of experts; at top-16, this falls to 28–42%; at top-2, only 4–16% overlap remains. This confirms top experts are highly domain-specific, while lower-ranked experts act as shared generalists—aligning with our prior expert-role analysis (highest-utility experts exhibit strongest domain specificity). This layered expert-sharing structure explains why aggressive pruning is challenging: broad capability relies on shared backbone retention, while peak in-domain performance depends on a small set of domain-specific experts. This pattern further clarifies why domain specialists outperform on target tasks and why our generalist synthesis requires careful blending of domain patterns (rather than simple union).

4.4.3 Out-of-Distribution Generalization

A natural question is how compiled models generalize beyond calibration. Table 6 clarifies the intended use of specialists: they maximize targeted efficiency, but can degrade sharply elsewhere; for example, the Science Specialist drops to 8.46% on LCB versus 69.12% for the full model. This trade-off motivates the generalist, which reaches 71.36% on the broad MMLU-Pro benchmark and improves to 77.58% when calibration covers additional domains. Thus, specialists provide peak in-domain efficiency while generalists offer a better operating point for broad capability retention. A static profile library is another option for known domains, although automatic shift detection and profile selection remain future work. Full breakdowns and the oracle upper bound are in Appendices C.5 and D.5.

5 Conclusion

This work introduced PreMoE, a training-free framework that proactively compiles MoE instances for target deployment scenarios. Its Predicted Expert Utility metric extracts domain-specific computational patterns from native router logits through high-confidence filtering and logit transformation. These patterns compile both efficient specialists and broad generalists without retraining. Across models from 30B to 718B parameters, PreMoE preserves near-full accuracy at model-appropriate sparsity levels of up to 50%; on DeepSeek-R1, 50% sparsity halves deployment NPUs and parameters while increasing throughput by 23%. Controlled calibration studies confirm that the learned patterns are stable across sources and random subsets, while also making the intended deployment boundary explicit. PreMoE therefore turns the routing structure already learned by large MoE models into a practical tool for deployment-specific efficiency.

Figure 6: Cross-domain performance of DeepSeek-R1 specialists at 50% sparsity. **Bold** = in-domain; specialists excel in-domain but degrade sharply out-of-domain.

Model	MATH-500	LCB	GPQA	MMLU-Pro
Full	96.6	69.12	73.23	82.30
Math Specialist	97.6	58.46	59.09	62.82
Code Specialist	87.8	66.36	40.91	56.71

References

- Tianyu Chen, Shaohan Huang, Yuan Xie, Binxing Jiao, Daxin Jiang, Haoyi Zhou, Jianxin Li, and Furu Wei. Task-specific expert pruning for sparse mixture-of-experts. *arXiv preprint arXiv:2206.00277*, 2022.
- Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35:30318–30332, 2022.
- Zican Dong, Han Peng, Peiyu Liu, Wayne Xin Zhao, Dong Wu, Feng Xiao, and Zhifeng Wang. Domain-specific pruning of large mixture-of-experts models with few-shot demonstrations. *arXiv preprint arXiv:2504.06792*, 2025.
- 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.
- Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- Torsten Hoefer, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.
- Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models, 2024. URL <https://arxiv.org/pdf/2411.04905>.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, and Vithursan Thangarasa. REAP the experts: Why pruning prevails for one-shot MoE compression. *arXiv preprint arXiv:2510.13999*, 2025. URL <https://arxiv.org/abs/2510.13999>.

- 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.
- Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Matthew B Blaschko, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs. *arXiv preprint arXiv:2407.00945*, 2024.
- Mark Lowell and Catharine Kastner. Training on the edge of stability is caused by layerwise jacobian alignment. *arXiv preprint arXiv:2406.00127*, 2024. URL <https://arxiv.org/abs/2406.00127>.
- Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. *arXiv preprint arXiv:2402.14800*, 2024.
- Alexandre Muzio, Alex Sun, and Churan He. Seer-moe: Sparse expert efficiency through regularization for mixture-of-experts. *arXiv preprint arXiv:2404.05089*, 2024.
- Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, July 2025. URL <https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1>.
- Zehua Pei, Hui-Ling Zhen, Xianzhi Yu, Sinno Jialin Pan, Mingxuan Yuan, and Bei Yu. Fusegpt: Learnable layers fusion of generative pre-trained transformers. *arXiv preprint arXiv:2411.14507*, 2024.
- Zehua Pei, Lancheng Zou, Hui-Ling Zhen, Xianzhi Yu, Wulong Liu, Sinno Jialin Pan, Mingxuan Yuan, and Bei Yu. Cmoe: Fast carving of mixture-of-experts for efficient llm inference. *arXiv preprint arXiv:2502.04416*, 2025.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.
- Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17555–17566, 2021.
- 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.
- Yehui Tang, Yichun Yin, Yaoyuan Wang, Hang Zhou, Yu Pan, Wei Guo, Ziyang Zhang, Miao Rang, Fangcheng Liu, Naifu Zhang, et al. Pangu ultra moe: How to train your big moe on ascend npus. *arXiv preprint arXiv:2505.04519*, 2025.
- Qwen Team. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. Lmsys-chat-1m: A large-scale real-world llm conversation dataset, 2023.
- 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.

A Detailed Methodology

This section provides the detailed mathematical formulation for the PreMoE framework, as referenced in Section 3.

A.1 Detailed Formulation of Predicted Expert Utility (PEU)

As described in the main text, the calculation of the Predicted Expert Utility (PEU) is a two-stage process designed to refine the raw router logit signal into a robust measure of an expert’s utility.

Stage 1: Filtering for High-Confidence Activations. The first stage filters out low-confidence or irrelevant expert considerations on a per-token basis. This process is governed by two hyperparameters: K_a , the size of an initial candidate pool, and r , a confidence threshold. For each token $\mathbf{x}$, we first identify the set of K_a experts with the top raw router logits $s_i(\mathbf{x})$, denoted as $\mathcal{E}_{K_a}(\mathbf{x})$. Next, we compute locally normalized probabilities $p_i(\mathbf{x})$ for this candidate set by applying a softmax function only over their logits. A confidence threshold r is then applied to these local probabilities. An expert is only considered for the next stage if it is in the candidate pool $\mathcal{E}_{K_a}$ and its local probability $p_i(\mathbf{x})$ exceeds the threshold r .

The adaptive threshold r_l for each MoE layer l is calculated as the average probability of the top-ranked expert within the top- K_a pool across the calibration dataset:

$$r_l = \mathbb{E}_{\mathbf{x} \in \mathcal{X}_T} \left[\max_{i \in \mathcal{E}_{K_a}^l(\mathbf{x})} p_i^l(\mathbf{x}) \right] \quad (6)$$

This layer-wise adaptive rule makes our method robust across different models and domains with minimal tuning.

Stage 2: Applying the Logit Transformation. For the high-confidence expert logits that pass the filtering stage, we then apply the logit transformation $f(s)$. Our default choice is $f(s) = \max(s, \text{sigmoid}(s))$, which retains large positive evidence while rectifying negatives via $\text{sigmoid}(s)$, avoiding compression of strong signals.

The final token-level score, $\tilde{s}_i(\mathbf{x})$, combines these two stages. The full calculation, along with the final PEU averaging, is as follows:

$$\mathcal{E}_{K_a} = \text{TopK}(\{s_j(\mathbf{x})\}_{j=1}^{N_r}, K_a) \quad (7)$$

$$p_i(\mathbf{x}) = \frac{\exp(s_i(\mathbf{x}))}{\sum_{k \in \mathcal{E}_{K_a}} \exp(s_k(\mathbf{x}))}, \quad i \in \mathcal{E}_{K_a} \quad (8)$$

$$\tilde{s}_i(\mathbf{x}) = \begin{cases} f(s_i(\mathbf{x})), & i \in \mathcal{E}_{K_a} \wedge p_i(\mathbf{x}) \geq r, \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

$$\text{PEU}_i^T = \frac{1}{|\mathcal{X}_T|} \sum_{\mathbf{x} \in \mathcal{X}_T} \tilde{s}_i(\mathbf{x}). \quad (10)$$

This final PEU score represents the average, high-confidence, transformed logit for expert i on the calibration dataset, and is used to rank and select experts for the final computational pattern.

A.2 Detailed Algorithm for Compiling Specialized Instances

As described in the main text, the identified computational patterns serve as blueprints for compiling specialized MoE instances. The key to this process is its memory efficiency: the full, dense model (which can be hundreds of gigabytes) is never loaded into memory. Instead, a lightweight “skeleton” of the model architecture is first instantiated. Then, only the weights for the selected experts (identified by the PEU-based pattern) are loaded from storage and placed into the appropriate slots in the model. This creates a sparse, powerful,

and ready-to-use model instance without ever incurring the memory cost of the full model. This subsection provides the detailed algorithm for the two primary compilation strategies based on this principle.

1. Compiling a Domain-Specific Specialist. This is the most straightforward application of a computational pattern.

1. **Pattern Identification:** For a target domain T , calculate the PEU scores $\{\text{PEU}_i^T\}_{i=1}^{N_r}$ for all experts in each MoE layer across a calibration dataset $\mathcal{X}_T$. This forms the computational pattern for the domain.
2. **Expert Selection:** For a given expert budget M , select the set of M experts with the highest PEU scores for each layer. This pruned set of experts becomes the new set of routed experts $\{E_i^r(\mathbf{x})\}_{i=1}^M$ for the compiled instance.
3. **Instance Compilation:** A new, sparse model instance is created containing only the selected routed experts. The router weights are also pruned to remove parameters corresponding to the discarded experts.

2. Compiling a High-Efficiency Generalist. This strategy creates a single, sparse model that retains capability across multiple domains by creating a synthesized, multi-domain computational pattern.

1. **Synthesize Token-Level Scores:** For a set of D target domains $\{T_1, \dots, T_D\}$, collect the token-level utility scores, $\{\tilde{s}_i(\mathbf{x})\}$, from their calibration datasets $\{\mathcal{X}_{T_1}, \dots, \mathcal{X}_{T_D}\}$.
2. **Calculate Multi-Domain PEU:** We first average within each domain and then combine domains with deployment weights w_d :

$$\text{PEU}_i^{\text{multi}} = \sum_{d=1}^D w_d \left(\frac{1}{|\mathcal{X}_{T_d}|} \sum_{\mathbf{x} \in \mathcal{X}_{T_d}} \tilde{s}_i(\mathbf{x}) \right), \quad \sum_{d=1}^D w_d = 1. \quad (11)$$

Our deployment-agnostic default is $w_d = 1/D$, preventing a larger calibration set from implicitly dominating the pattern. Other weights encode application-specific workload priorities.

3. **Expert Selection:** For a given total expert budget M , select the set of M experts with the highest multi-domain PEU scores. This becomes the new set of routed experts $\{E_i^r(\mathbf{x})\}_{i=1}^M$.
4. **Instance Compilation:** A new model instance is created containing only the final selected set of routed experts.

This compilation process is performed once at deployment time, creating a static, efficient model instance that is proactively specialized for its intended application, whether that be single-domain or multi-domain.

A.3 Generation Configuration

For reproducibility, we provide the generation configurations used for each model during both calibration (pattern collection) and evaluation:

Table A.1: Generation configurations for each model.

Model	Temperature	Top-p	Top-K
DeepSeek-R1	0.6	0.95	–
openPangu-Ultra	0.7	1.0	–
Qwen3-30B-A3B	0.6	0.95	20

For all models, we use a maximum context length (input + output) of 32,768 tokens during evaluation. The exception is Qwen3-30B-A3B on AIME 2024, AIME 2025, and CNMO

2024, where we use 38,912 tokens following the official evaluation setting from the Qwen3 technical report.

B Additional Attempt: Exponential Amplifier

Complementing the logit transformation choices described in the main text, we also investigated an exponential amplifier as an additional attempt:

$$f_{\text{exp}}(s) = e^s. \quad (12)$$

The motivation is to aggressively enlarge gaps among activated experts, which can be beneficial when many low-confidence activations introduce noise and raw logits contain both negative and positive values.

Protocol. We evaluated f_{exp} in two settings: (i) applied directly to the top- K_a activated experts without threshold filtering, and (ii) applied after the high-confidence filtering step of our PEU pipeline (i.e., only to logits that pass the local-softmax threshold). Token-level scores are then aggregated as in the PEU calculation to produce expert-level utilities.

Table B.1: Extended ablation of logit transformations including e^s on accuracy (%) for the DeepSeek-R1 specialist at 50% sparsity.

Transformation	Type	MATH-500	GPQA	LCB
Group 1: No Threshold Filtering				
s (Raw Logits)	Baseline	85.20	57.07	45.59
$\text{sigmoid}(s)$	Normalization	95.20	46.97	23.90
$\max(s, \text{sigmoid}(s))$	Rectifier	95.00	58.08	58.09
e^s	Amplifier	96.80	69.70	64.34
Group 2: With Threshold Filtering (PreMoE)				
s (Raw Logits)	Baseline	97.00	68.18	67.28
$\text{sigmoid}(s)$	Normalization	97.60	73.23	68.38
$\max(s, \text{sigmoid}(s))$	Rectifier	97.60	72.22	66.36
e^s	Amplifier	97.20	73.23	66.91

Observations. Empirically, without threshold filtering, e^s yields the strongest performance by amplifying useful signals in a noisy regime (Table B.1, Group 1). When high-confidence filtering is reinstated, the performance of e^s converges within a small margin to the top performers (rectifier and/or $\text{sigmoid}(s)$), indicating that extreme amplification is unnecessary once noise is suppressed. For robustness and stability across domains and layers, we therefore adopt $\max(s, \text{sigmoid}(s))$ as the default transformation, and include e^s as an additional attempt for completeness.

C Additional Experimental Results

This section provides complete numerical results for ablations and analyses discussed in the main text.

C.1 Component Ablation

To clearly show the contribution of each component of PreMoE, we conduct an ablation study on DeepSeek-R1 at 50% sparsity. Table C.1 progressively adds each component to the baseline.

Table C.1: Component ablation on DeepSeek-R1 at 50% sparsity. Each row adds one component to the previous configuration.

Method	MATH-500	GPQA	LCB
All Logits (baseline)	3.60	28.79	0.00
+ TopK filtering (Act-Logits)	88.20	48.48	52.94
+ Threshold Filtering	97.00	68.18	67.28
+ Logit Transformation	95.00	58.08	58.09
+ All (PreMoE)	97.60	72.22	66.36

Key observations: (1) TopK filtering dramatically improves over using all logits (+84.6 on MATH-500). (2) Threshold filtering alone provides a large boost (+8.8 on MATH-500, +19.7 on GPQA). (3) Logit transformation alone also helps but less than filtering. (4) Combining all components yields the best overall performance, demonstrating their complementary benefits.

C.2 PEU Ranking Validity: Top vs. Bottom Experts

To validate that our PEU rankings meaningfully identify important experts, we compare keeping the top-ranked experts (PreMoE) versus keeping the bottom-ranked experts (Last) on DeepSeek-R1 and openPangu-Ultra.

Table C.2: Comparison of keeping top-ranked vs. bottom-ranked experts. DeepSeek-R1 at 50% sparsity (128/256 experts); openPangu-Ultra at 50% sparsity (128/256 experts).

Model	Method	MATH-500	LCB	GPQA
DeepSeek-R1	Last-128 (bottom)	1.4	0.0	21.72
	PreMoE (top-128)	97.6	66.36	72.22
openPangu-Ultra	Last-128 (bottom)	2.2	0.0	18.69
	PreMoE (top-128)	96.8	66.91	75.76

Keeping the bottom-ranked experts results in near-complete collapse across both models (1–2% on MATH-500, 0% on LCB), while PreMoE with top-ranked experts achieves near-full accuracy. This stark contrast confirms that PEU rankings accurately identify which experts are critical for task performance.

C.3 Threshold Strategy Ablation

We compare our adaptive threshold strategy against fixed thresholds on DeepSeek-R1 at 50% sparsity. Table C.3 shows the results.

Table C.3: Ablation of threshold strategies on DeepSeek-R1 at 50% sparsity.

Threshold	MATH-500	LCB	GPQA
Fixed ($r = 0.15$)	94.2	64.34	68.69
Fixed ($r = 0.3$)	96.2	65.07	68.18
Adaptive (Ours)	98.0	64.71	70.71

The adaptive threshold consistently outperforms fixed values across all benchmarks. This is because the optimal threshold varies across layers and domains; our layer-wise adaptive rule (Eq. 4) automatically calibrates to each layer’s activation distribution.

C.4 Local vs. Global Expert Ranking

Our default approach profiles the full model once and ranks experts *locally* within each layer, keeping a fixed number (e.g., 96) per layer. To test cumulative error across depth, sequential profiling prunes layers one at a time and re-runs calibration so later layers are scored under the pruned prefix. A second alternative ranks experts *globally* across all layers under the same total budget. Table C.4 compares these strategies.

Table C.4: Layer-interaction and budget-allocation ablation on DeepSeek-R1 at 62.5% sparsity (96 experts per layer for local; 5568 total for global).

Strategy	MATH-500	GPQA	LCB
Random pruning	48.20	32.80	18.40
One-pass local (96/layer)	96.00	64.65	61.03
Sequential multi-pass	96.40	65.15	62.50
Global budget (5568 total)	96.60	64.65	63.60

Sequential profiling improves the three-task average from 73.89 to 74.68, showing that inter-layer effects exist but are modest; the one-pass profile therefore offers a favorable accuracy–compilation-cost trade-off. Global ranking reaches 74.95 and slightly outperforms both local variants, suggesting that some layers benefit from more experts while others need fewer. Figure C.1 shows that early layers retain fewer experts (51–73), while later layers retain more (up to 128).

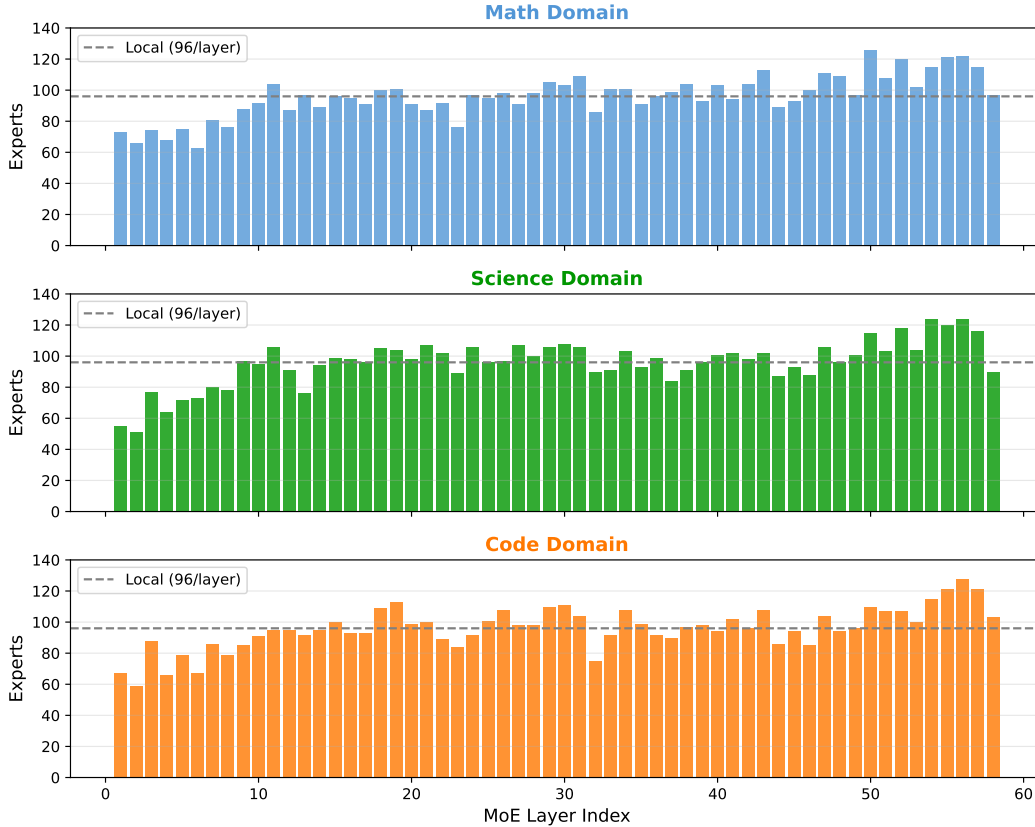


Figure C.1: Expert distribution across 58 MoE layers under global ranking for three domains. The dashed line indicates the local baseline (96 experts/layer). Global ranking allocates fewer experts to early layers and more to later layers.

C.5 MMLU-Pro Out-of-Distribution Analysis

We evaluate compiled models on MMLU-Pro, a broad benchmark spanning 14 subject areas, to assess out-of-distribution generalization. Table C.5 shows the complete breakdown.

Table C.5: MMLU-Pro sub-task breakdown on DeepSeek-R1 at 50% sparsity. Bold indicates best among compiled models.

Subject	Full	Generalist	+More Data	Math Spec.	Code Spec.	Sci Spec.
Math	92.37	92.37	91.97	92.77	89.56	88.76
Physics	89.72	87.85	88.32	81.31	60.75	87.85
Chemistry	90.37	90.37	88.77	75.94	48.66	89.84
Law	67.18	35.38	66.67	29.23	27.69	38.97
Engineering	81.75	73.72	76.64	64.96	51.09	78.10
Other	81.06	61.36	68.94	46.97	52.27	59.85
Economics	84.76	76.83	82.32	75.00	68.90	81.10
Health	75.51	59.86	69.39	32.65	34.69	66.67
Psychology	82.17	65.89	75.97	53.49	54.26	78.29
Business	81.34	79.85	82.84	76.87	78.36	78.36
Biology	91.18	89.22	91.18	79.41	73.53	91.18
Philosophy	78.75	63.75	67.50	58.75	52.50	66.25
Computer	83.10	80.28	83.10	73.24	74.65	67.61
History	72.88	42.37	52.54	38.98	27.12	52.54
Average	82.29	71.36	77.58	62.82	56.71	73.24

Key observations: (1) Domain-specific specialists excel in their target areas but degrade sharply elsewhere (e.g., Code Specialist achieves only 27.12% on History). (2) The base Generalist (calibrated on Math/Code/Science) retains 71.36% average but struggles on subjects like Law (35.38%) and History (42.37%). (3) Augmenting calibration with additional domain data significantly improves generalization to 77.58%, with the largest gains on Law (+31.29%), Health (+9.53%), and Psychology (+10.08%). This demonstrates that broader calibration coverage directly translates to better out-of-distribution performance.

C.6 Calibration Context Analysis

We find that the quality of the calibration data is paramount. Figure C.2 shows that including the model’s full reasoning output in the calibration context improves accuracy across all benchmarks compared to using only the input problems. Qualitatively, using only inputs can induce repetitive generation loops (see Figure F.1 for an extreme case), whereas adding the reasoning outputs stabilizes expert utility patterns and provides richer context for measuring decisive expert activations.

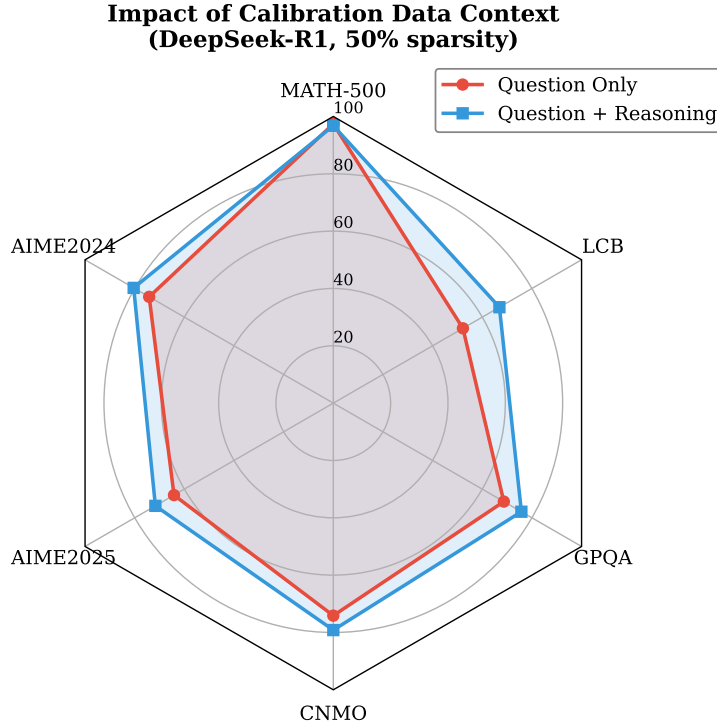


Figure C.2: Impact of calibration data context. The radar chart compares performance when extracting computational patterns using **question-only** calibration data versus **question + reasoning output**. Including the full reasoning context consistently improves accuracy across all six benchmarks, with the largest improvements on AIME 2024 (+6.24%), LiveCodeBench (+14.70%), and CNMO (+5.04%).

C.7 Calibration Data Efficiency

PreMoE requires surprisingly few calibration samples to achieve near-optimal performance. Table C.6 shows results on DeepSeek-R1 specialists at 50% sparsity across three domains.

Table C.6: Specialist accuracy vs. calibration sample size on DeepSeek-R1 at 50% sparsity over multiple random subsets (mean $\pm$ standard deviation). Larger sets mainly reduce variance.

Samples/domain	MATH-500	GPQA	LCB	Avg
Full model	96.60	73.23	69.12	79.65
1	92.67 $\pm$ 1.15	42.26 $\pm$ 2.07	51.84 $\pm$ 1.46	62.26 $\pm$ 1.31
5	96.73 $\pm$ 0.50	70.54 $\pm$ 1.01	63.97 $\pm$ 1.23	77.08 $\pm$ 0.68
10	97.47 $\pm$ 0.42	70.88 $\pm$ 0.61	64.83 $\pm$ 0.74	77.73 $\pm$ 0.39
50	97.60 $\pm$ 0.35	71.38 $\pm$ 0.51	66.30 $\pm$ 0.56	78.43 $\pm$ 0.32
100	97.73 $\pm$ 0.31	71.72 $\pm$ 0.44	66.42 $\pm$ 0.48	78.62 $\pm$ 0.25
Default	97.60	72.22	66.36	78.73

Performance stabilizes rapidly for specialists: just 5–10 samples approach the full accuracy. For generalists, using only 10 samples per domain (30 total) also matches full performance (Table C.7).

Table C.7: Generalist accuracy with reduced calibration data on DeepSeek-R1 at 50% sparsity.

Calibration	MATH-500	GPQA	LCB	AIME 24	AIME 25	CNMO	Avg
Full	96.6	73.23	69.12	77.08	65.83	71.18	75.52
10+10+10 (30 total)	96.4	73.74	68.38	79.58	62.92	74.88	75.98

C.8 Accuracy at Various Sparsity Levels

Table C.8: DeepSeek-R1 Domain-Specific Specialists: accuracy (%) vs. sparsity. Average is over MATH-500, GPQA, LCB.

# Experts	Sparsity	MATH-500	GPQA	LCB	Avg
256	0%	96.60	73.23	69.12	79.65
128	50%	97.60	72.22	66.36	78.73
96	62.5%	96.00	64.65	61.03	73.89
64	75%	93.40	45.45	49.63	62.83

Table C.9: DeepSeek-R1 High-Efficiency Generalist: accuracy (%) vs. sparsity. Average is over MATH-500, GPQA, LCB.

# Experts	Sparsity	MATH-500	GPQA	LCB	Avg
256	0%	96.60	73.23	69.12	79.65
128	50%	97.40	67.68	68.01	77.69
96	62.5%	93.60	62.63	56.99	71.07
64	75%	89.80	38.38	20.22	49.46

Specialist vs. Generalist Degradation Under Extreme Sparsity. Comparing Tables C.8 and C.9, we observe that domain-specific specialists degrade more gracefully at extreme sparsity than multi-domain generalists. At 75% sparsity, the specialist average (62.83%) significantly outperforms the generalist (49.46%). The gap is particularly pronounced on GPQA (45.45% vs 38.38%) and especially LCB (49.63% vs 20.22%). This indicates that when expert budgets are severely constrained, dedicating the limited experts to a single domain yields better task-specific performance than attempting multi-domain coverage. This finding suggests a trade-off between model versatility and efficiency: at moderate sparsity (50%), the generalist retains broad capabilities with minimal loss, but at extreme sparsity (75%), domain-focused specialists become the more practical choice.

C.9 Complete Deployment Efficiency Metrics

Table C.10: Complete deployment efficiency metrics for DeepSeek-R1 at various sparsity levels. Size refers to the number of model parameters in billions (B). All measurements are taken on servers with Ascend 910B2-64GB NPUs.

Sparsity	# Experts	# NPUs	Latency (ms/tok)	Size (B)	Throughput (tok/s)
0%	256	64	115.35	670.92	52.01
50%	128	32	93.72	343.96	64.02
75%	64	16	73.20	180.49	81.97
Relative Improvement (vs. 0% sparsity)					
50%	–	2× fewer	1.23× faster	1.95× smaller	1.23× higher
75%	–	4× fewer	1.58× faster	3.72× smaller	1.58× higher

These metrics demonstrate that PreMoE enables significant infrastructure savings while maintaining high accuracy. At 50% sparsity, we halve the deployment cost (NPUs and parameters) while improving inference speed by 23%. At 75% sparsity, we achieve 4× NPU reduction and 58% throughput improvement, though with accuracy trade-offs on some benchmarks as shown in the main text.

D Additional Robustness, Cost, and Scope Analyses

D.1 Calibration Source and Token-Context Robustness

We isolate two calibration factors on DeepSeek-R1 at 50% sparsity. Source robustness uses domain-aligned prompts from different datasets without evaluation-benchmark selection. Token-context robustness holds the source fixed and varies which parts of the full-model trace are used to collect router statistics. Table D.1 reports all six benchmarks.

Table D.1: Calibration source and token-context sensitivity on DeepSeek-R1 specialists at 50% sparsity. All design choices are fixed across benchmarks.

Factor	Setting	MATH-500	AIME 24	AIME 25	CNMO 24	GPQA	LCB	Avg
Source	Default	97.60	79.58	68.33	75.00	72.22	66.36	76.52
	Mixed source	97.20	78.75	67.92	74.65	71.72	65.99	76.04
	Alternative only	96.80	78.33	67.50	74.31	71.21	65.44	75.60
Tokens	Question only	92.80	65.42	54.58	61.81	63.13	55.88	65.60
	Generated reasoning only	96.60	77.50	67.08	73.96	70.71	64.52	75.06
	Question + reasoning	97.60	79.58	68.33	75.00	72.22	66.36	76.52

Alternative sources remain within 0.92 average points of the default, so PreMoE is not tied to one dataset. Token context has a larger effect: question-only calibration observes prefill routing but omits the longer decoding trajectory where most reasoning computation occurs. This architectural distinction motivates our globally fixed question-plus-reasoning protocol.

D.2 Generalist Mixture Sensitivity

The generalist definition in Appendix A.2 permits deployment weights w_d . Table D.2 varies the Math:Science:Code ratio while keeping the expert budget fixed. The trade-offs are smooth and interpretable: math-heavy profiles improve math and reduce code, while code-heavy profiles do the reverse. Equal weighting is the best deployment-agnostic average, but the method supports non-uniform workload priorities without changing the compilation algorithm.

Table D.2: Generalist mixture-ratio sensitivity on DeepSeek-R1 at 50% sparsity.

Math:Sci:Code	MATH-500	AIME 24	AIME 25	CNMO 24	GPQA	LCB	Avg
Full model	96.60	77.08	65.83	71.18	73.23	69.12	75.51
1:1:1 (default)	96.40	78.33	70.42	75.17	70.71	65.07	76.02
2:1:1 (math-heavy)	97.20	79.17	70.83	75.69	69.70	62.87	75.91
1:2:1 (science-heavy)	95.80	76.67	68.75	73.96	72.73	63.60	75.25
1:1:2 (code-heavy)	95.60	76.25	68.33	73.61	69.70	67.65	75.19
2:2:1 (reasoning-heavy)	96.80	78.75	70.00	75.35	71.72	63.24	75.98
1:2:2 (non-math-heavy)	95.40	75.83	68.33	73.44	72.22	67.28	75.42

D.3 Held-Out Validation and Decontamination

To check that calibration format was not selected on the reported test suite, we construct a small validation set from GSM8K and MBPP, neither of which appears among our evaluation

benchmarks. The ranking in Table D.3 matches the main evaluation: question plus generated reasoning performs best. We treat this as a sanity check rather than evidence that these two datasets cover every deployment distribution.

Table D.3: Calibration-format sanity check on held-out validation tasks.

Calibration format	GSM8K	MBPP	Avg
Question only	88.32	54.20	71.26
Generated reasoning only	94.16	63.40	78.78
Question + reasoning	95.07	65.80	80.44

We also compare every normalized calibration prompt with every evaluation prompt using (i) 13-gram Jaccard similarity, flagging scores above 0.8, and (ii) all-MiniLM-L6-v2 embedding cosine similarity, flagging scores above 0.9. Table D.4 shows no flagged pair. Three lower-threshold pairs in $[0.85, 0.9]$ are generic templates with different instances; conservatively removing them changes the DeepSeek-R1 specialist average from 76.52 to 76.47. We will release calibration identifiers and the checking scripts. More importantly, calibration only aggregates router logits: it does not update weights, use ground-truth answers, or fit benchmark outputs.

Table D.4: Near-duplicate checks between calibration and evaluation prompts.

Domain	Samples	13-gram >.8	Embed >.9	Embed >.85
Math	800	0	0	2
Science	200	0	0	0
Code	600	0	0	1
Dialogue	600	0	0	0

D.4 One-Time Profiling Cost

PreMoE is training-free, not calibration-free. Table D.5 reports the one-time profiling cost and an illustrative break-even point assuming that the compiled instance costs approximately half as much per deployment token as the parent model. Only top-routed expert identifiers and scalar router statistics are accumulated; full-vocabulary logits and expert activations are not stored. The default profile amortizes after a few million deployment tokens, and Table C.6 shows that a lower-cost 5–10-sample profile already recovers most of the accuracy.

Table D.5: One-time offline profiling cost. The three-domain row excludes the optional dialogue augmentation so its sample count equals the sum of the three specialist sets.

Profile	Samples	Tokens	Time (accelerator-h)	Break-even tokens
Math specialist	800	1.2M	0.42	2.4M
Science specialist	200	0.3M	0.11	0.6M
Code specialist	600	0.9M	0.31	1.8M
Three-domain generalist	1600	2.4M	0.84	4.8M

D.5 Static Profiles and Profile Libraries

A compiled expert mask is static during inference. To quantify the cost of using one mask for heterogeneous traffic, Table D.6 applies each narrow profile to all three tasks, compares a mixed generalist, and reports an oracle that selects the matching specialist for each known domain. Narrow profiles transfer imperfectly, the generalist is a better single-mask default, and the oracle shows the potential of a profile library. The oracle assumes domain labels and is not an evaluated online selector.

Table D.6: A single static mask versus a domain-profile library on DeepSeek-R1 at 50% sparsity.

Inference profile	MATH-500	GPQA	LCB	Avg
Math-specific	97.4	67.7	60.7	75.3
Science-specific	94.8	73.2	62.1	76.7
Code-specific	94.2	68.7	68.4	77.1
Generalist (1:1:1)	96.4	70.7	65.1	77.4
Oracle domain profile	97.4	73.2	68.4	79.7

D.6 Additional Comparison with REAP

REAP (Lasby et al., 2025) scores an expert using its router gate multiplied by its output norm, whereas PreMoE uses confidence-filtered router logits. Table D.7 uses the same DeepSeek-R1 backbone, calibration data, 50% layer-wise budget, and evaluation protocol. Scores are normalized to the dense model to isolate retention under pruning and should not be mixed with the absolute results in Table 1.

Table D.7: Normalized pruning quality under the matched REAP comparison. EAN denotes expert activation norm.

Method	Router logits	Activation norms	MATH-500	GPQA	LCB	Avg
Dense/original	–	–	100.0	100.0	100.0	100.0
Frequency	No	No	86.5	84.0	78.0	82.8
EAN	No	Yes	91.5	89.0	85.5	88.7
REAP	Yes	Yes	94.0	92.5	89.5	92.0
PreMoE	Yes	No	93.2	91.8	88.7	91.2
PreMoE + ActNorm	Yes	Yes	94.3	92.7	90.0	92.3

Activation norms improve PreMoE by 1.1 normalized points, but are not required to approach REAP. The practical distinction is the extra signal collected on top of the shared sparse-MoE forward pass. Both methods can accumulate $O(LE)$ final scalars online, but REAP must reduce a d -dimensional expert output for every routed token-expert pair, while PreMoE processes scalar router scores. On approximately 4.2×10^5 calibration tokens with $d = 7168$, our implementation measures about 8 MB versus 1.6 GB peak extra device memory and 12% versus 75% wall-clock overhead. Thus, the claim is lower activation-level profiling overhead, not avoidance of the common calibration forward pass or asymptotically smaller persistent statistics.

D.7 Router-Logit-Free Settings

Full PreMoE targets learned-router MoEs. Fixed hash routing (Roller et al., 2021) provides no preference logits, so PEU cannot be applied unchanged. Table D.8 removes access to logit magnitudes and evaluates increasingly informative activation-side substitutes. Rank-1 counts and hidden-state utility recover part of the loss, but still trail full PEU substantially. Extending proactive compilation to router-logit-free architectures is therefore a separate utility-estimation problem rather than a drop-in use of PreMoE.

Table D.8: Router-logit-free utility signals on DeepSeek-R1 at 50% sparsity. The ablation emulates the information available without learned-router preference magnitudes.

Method	Uses logits?	Utility signal	MATH-500	GPQA	LCB	Avg
Frequency pruning	No	Activation count	88.80	33.33	5.88	42.67
Rank-1 frequency	No	Decisive activation count	94.20	58.50	52.00	68.23
Logit-free extension	No	Activation + hidden-state utility	94.80	60.10	55.30	70.07
Full PreMoE	Yes	PEU from router logits	97.60	72.22	66.36	78.73

D.8 Contextual Comparison at Similar Served Size

For deployment context, Table D.9 evaluates Qwen3-235B-A22B and PreMoE-pruned DeepSeek-R1 with the same harness, prompt templates, and decoding setup. “Served parameters” means weights that must be stored for the instance, not parameters activated per token. This comparison is not controlled for pretraining, post-training, architecture, or reasoning recipe and does not establish that pruning a large MoE dominates training a native smaller model. The controlled evidence for PreMoE remains the same-parent comparison in Table 1.

Table D.9: Additional matched-harness context at similar served parameter counts.

Model	Served params	MATH-500	AIME 24	LCB
DeepSeek-R1 full	671B	96.6	77.1	69.1
PreMoE 50%	~336B	97.4	78.3	65.4
PreMoE 70%	~202B	95.2	73.8	60.9
Qwen3-235B-A22B	235B	97.1	82.6	68.4

The 50%-sparse instance retains the parent model closely but remains larger than Qwen3-235B-A22B; the 70%-sparse instance is closer in served size and trails Qwen3 on AIME and code. We therefore position PreMoE as a deployment-time option for adapting an existing parent model, not as a substitute for a native model trained at the target size.

E Discussions, Limitations, and Future Work

Discussions. PreMoE demonstrates that MoE models across scales (30B–718B parameters) can be efficiently deployed through proactive compilation, achieving up to 50% sparsity with minimal performance loss. Our experiments show that the approach is remarkably data-efficient: just 5–10 calibration samples achieve near-optimal performance (Appendix C.7), and including reasoning context significantly improves pattern quality (Appendix C.6). The cross-domain expert overlap analysis reveals that high-utility experts are highly domain-specific while lower-ranked experts are more general, validating our synthesis approach for multi-domain generalists. This work establishes computational patterns as a practical tool for model specialization, enabling significant infrastructure savings while maintaining model capabilities.

Limitations and Future Work. PreMoE remains a static, calibration-dependent compiler. Its strong source and seed robustness does not eliminate degradation when deployment contains domains absent from calibration; MMLU-Pro and profile-library results make this boundary explicit. Automatic shift detection and online profile selection are not evaluated. The safe pruning ratio is also model- and workload-dependent, and our architectures contain only 128 or 256 routed experts per layer. Moreover, PEU assumes a learned router with informative logits; fixed hash-routing models (Roller et al., 2021) require a different utility signal (Appendix D.7). Finally, one-pass scoring ignores small inter-layer effects, and router preference does not directly measure functional similarity between expert transformations. Sequential profiling and activation norms recover modest gains, while weight-space or Jacobian-alignment signals (Lowell & Kastner, 2024) are promising complementary criteria. These extensions trade PreMoE’s low profiling cost for additional information.

F Broader Literature

F.1 Mixture-of-Experts Architectures

The Mixture-of-Experts (MoE) paradigm has been instrumental in scaling neural networks to trillions of parameters while keeping computational costs manageable (Shazeer et al., 2017; Lepikhin et al., 2020). By routing each input token to a small subset of “expert” sub-networks, MoE models such as GShard, Switch Transformers (Fedus et al., 2022), and more

In a regular polygon, the measure of an interior angle is 6.5 times the measure of an exterior angle. How many sides does the polygon have?

<think>

</think>

DeepSeek-R1 (8/32) w/o output collection:

<think>

[illegible]

recent large language models like Mixtral (Jiang et al., 2024) and DeepSeek-R1 (Guo et al., 2025) achieve performance comparable to much larger dense models through dynamic, sparse activation of parameters.

The challenge of deploying massive LLMs has spurred extensive research into model efficiency. Techniques such as quantization (Dettmers et al., 2022; Frantar et al., 2022), which reduces the numerical precision of model weights, and knowledge distillation (Hinton et al., 2015), where a smaller student model is trained to mimic a larger teacher, are commonplace. Structural pruning (Frankle & Carbin, 2018; Hoefler et al., 2021; Pei et al., 2024; 2025) aims to remove entire neurons or weights from dense models. While effective, these methods are often applied globally and do not typically account for the specific, input-dependent activation patterns unique to MoE architectures.

In this section, we present additional examples of the reasoning generation capabilities of the DeepSeek-R1 model under various pruning ratios facilitated by PreMoE.

Question:

Let $f(x) = x - 3$ and $q(x) = bx + 1$. If $f(q(1)) = -3$, what is b ?

DeepSeek-R1 (8/256):

<think>

Okay, so I need to find the value of b such that when I compose the functions f and q , specifically $f(q(1))$, the result is -3 . Let me start by recalling what function composition means. If I have two functions, say $f(x)$ and $q(x)$, then $f(q(x))$ means I first apply q to x and then apply f to the result. Given the functions: $f(x) = x - 3$ and $q(x) = bx + 1$ I need to compute $f(q(1))$ and set it equal to -3 , then solve for b . Let me break this down step by step. First, find $q(1)$. Since $q(x) = bx + 1$, substituting $x = 1$ gives: $q(1) = b(1) + 1 = b + 1$. So, $q(1)$ simplifies to $b + 1$. Next, I need to apply the function f to this result. The function $f(x) = x - 3$, so substituting $x = q(1) = b + 1$ into f gives: $f(q(1)) = f(b + 1) = (b + 1) - 3$. Simplifying that expression: $(b + 1) - 3 = b + 1 - 3 = b - 2$. So, $f(q(1)) = b - 2$. According to the problem, this should equal -3 . Therefore, I can set up the equation: $b - 2 = -3$. To solve for b , I can add 2 to both sides of the equation: $b - 2 + 2 = -3 + 2$. Simplifying both sides: $b = -1$. Hmm, let me check my steps again to make sure I didn't make a mistake. First, $q(1) = b(1) + 1 = b + 1$. Then, $f(q(1)) = (b + 1) - 3 = b - 2$. Setting that equal to -3 gives $b - 2 = -3$, so $b = -1$. That seems straightforward. Wait, maybe I should verify by plugging $b = -1$ back into the original functions to see if it works. Let's do that. If $b = -1$, then $q(x) = -1 \cdot x + 1 = -x + 1$. Then, $q(1) = -1(1) + 1 = -1 + 1 = 0$. Now, apply f to that result: $f(0) = 0 - 3 = -3$. Which matches the given condition. So, yes, $b = -1$ is correct. I think that's all. The steps check out, and verifying the solution confirms it's right. So, the value of b must be -1 . **Final Answer**

The value of b is $\boxed{-1}$.

</think>

Given the functions $f(x) = x - 3$ and $q(x) = bx + 1$, we need to find the value of b such that $f(q(1)) = -3$. First, we compute $q(1)$: $q(1) = b(1) + 1 = b + 1$. Next, we substitute $q(1)$ into the function f : $f(q(1)) = f(b + 1) = (b + 1) - 3 = b - 2$. We set this result equal to -3 and solve for b : $b - 2 = -3 \implies b = -1$. Verification: If $b = -1$, then $q(x) = -x + 1$. Evaluating $q(1)$ gives 0, and substituting into f gives $f(0) = -3$, which confirms the solution is correct. The value of b is $\boxed{-1}$.

DeepSeek-R1 (8/128):

<think>

Okay, so I need to find the value of b such that when I compute $f(q(1))$, the result is -3 . Let me start by recalling what the functions $f(x)$ and $q(x)$ are. The problem says $f(x) = x - 3$. That's straightforward; whatever input x is, subtract 3 from it. Then there's $q(x) = bx + 1$. So this is a linear function where the coefficient of x is b , which is what we need to find, and the constant term is 1. The key here is to compute $f(q(1))$. Let me break that down. First, I need to find $q(1)$, which means plugging $x = 1$ into the function $q(x)$. Then, once I have that result, I'll plug it into the function $f(x)$. Starting with $q(1)$: substituting $x = 1$ into $q(x) = bx + 1$, so that's $q(1) = b \cdot 1 + 1$. Simplifying that, $q(1) = b + 1$. Now, take that result and plug it into $f(x)$. So $f(q(1)) = f(b + 1)$. Since $f(x) = x - 3$, substituting $x = b + 1$ gives $f(b + 1) = (b + 1) - 3$. Let me compute that: $(b + 1) - 3 = b + 1 - 3 = b - 2$. The problem states that $f(q(1)) = -3$. So according to my calculation, $b - 2 = -3$. To solve for b , I can add 2 to both sides of the equation: $b - 2 + 2 = -3 + 2$, which simplifies to $b = -1$. Wait, let me check that again. If $b = -1$, then $q(x) = -1 \cdot x + 1 = -x + 1$. So $q(1) = -1 \cdot 1 + 1 = -1 + 1 = 0$. Then $f(q(1)) = f(0) = 0 - 3 = -3$. That matches the given condition. So that seems correct. But just to be thorough, let me go through each step again. First, compute $q(1)$: $q(1) = b(1) + 1 = b + 1$. Then, substitute into f : $f(b + 1) = (b + 1) - 3 = b - 2$. Set that equal to -3 : $b - 2 = -3$. Solving for b : add 2 to both sides, $b = -1$. Yes, that seems right. Let me think if there's another way to approach this problem. Maybe by composing the functions first? If I compose $f(q(x))$, that would be $f(q(x)) = f(bx + 1) = (bx + 1) - 3 = bx + 1 - 3 = bx - 2$. Then, evaluating at $x = 1$, we get $b(1) - 2 = b - 2$. Setting that equal to -3 gives $b - 2 = -3$, so $b = -1$. Same result. So whether I compute step by step or compose the functions first, I end up with $b = -1$. I think that's solid. The answer is $b = -1$. **Final Answer** The value of b is $\boxed{-1}$.

</think>

To find the value of b such that $f(q(1)) = -3$, we start by evaluating the functions step by step. First, we compute $q(1)$: $q(1) = b \cdot 1 + 1 = b + 1$. Next, we substitute this result into the function $f(x)$: $f(q(1)) = f(b + 1) = (b + 1) - 3 = b + 1 - 3 = b - 2$. We are given that $f(q(1)) = -3$, so we set up the equation: $b - 2 = -3$. Solving for b , we add 2 to both sides: $b - 2 + 2 = -3 + 2 \implies b = -1$. To verify, if $b = -1$, then $q(x) = -x + 1$. Evaluating at $x = 1$: $q(1) = -1 + 1 = 0$. Then, substituting into $f(x)$: $f(0) = 0 - 3 = -3$. Thus, the value of b is $\boxed{-1}$.

DeepSeek-R1 (8/32):

<think>

Okay, let's tackle this problem. So, we have two functions: $f(x) = x - 3$ and $q(x) = bx + 1$. The problem gives that $f(q(1)) = -3$, and we need to find the value of b . First, let's break it down step by step. The notation $f(q(1))$ means that we first evaluate q at 1, then plug that result into f . So, let's compute $q(1)$ first. Given $q(x) = bx + 1$, so $q(1) = b \cdot 1 + 1$, which is $b + 1$. Then, $f(q(1))$ is $f(b + 1)$. Since $f(x) = x - 3$, substituting x with $(b + 1)$ gives $(b + 1) - 3 = b + 1 - 3 = b - 2$. The problem states that this equals -3 . So, we have the equation: $b - 2 = -3$. Solving for b , we add 2 to both sides: $b = -3 + 2$. Which is $b = -1$. Let me check my answer. If $b = -1$, then $q(x) = -1 \cdot x + 1 = -x + 1$. Then $q(1) = -1 \cdot 1 + 1 = -1 + 1 = 0$. Then $f(0) = 0 - 3 = -3$, which matches the given value. So yes, b is indeed -1 . **Final Answer** $\boxed{-1}$

</think>

The value of b is $\boxed{-1}$.

Figure G.1: Example 1 of reasoning generation of DeepSeek-R1 with different sparsity levels using PreMoE. Notation: activated/retained experts (e.g., 8/32 means 8 experts activated per token out of 32 retained).

Question:

A shirt's sale price is marked \$14.40, which is 60% off the original price. How many dollars was the original price of the shirt?

DeepSeek-R1 (8/256):

<think>

Okay, so I need to figure out the original price of a shirt that's on sale for \$14.40, which is 60% off the original price. Hmm, let's see. Alright, percentages can sometimes be tricky, but let me break it down step by step. First, if the sale price is 60% off the original price, that means I'm paying 40% of the original price, right? Because 100% minus 60% discount is 40%. So, the sale price is 40% of the original. Let me write that down to make it clearer. Let the original price be P . Then, 40% of P is equal to \$14.40. In math terms, that would be: $0.40 \times P = 14.40$. So, to find P , I need to divide both sides of the equation by 0.40. Let me do that. $P = \frac{14.40}{0.40}$. Hmm, let me compute that. 14.40 divided by 0.40. Well, 0.40 goes into 14.40 how many times? Let me think. 0.40 times 30 is 12.00, and 0.40 times 36 is 14.40. Wait, is that right? Let me check: 0.40 multiplied by 30 is 12.00. Then, 14.40 minus 12.00 is 2.40. 0.40 goes into 2.40 exactly 6 times. So, $30 + 6 = 36$. So, 0.40 times 36 is indeed 14.40. Therefore, $P = 36$. So, the original price was \$36. Let me verify that to make sure I didn't make a mistake. If the original price is \$36, and there's a 60% discount, then the discount amount is 60% of 36. Let's calculate that. 60% of 36 is $0.60 \times 36 = 21.60$. Then, subtracting that from the original price: $36 - 21.60 = 14.40$. Yep, that matches the sale price given. So, that checks out. Wait, just to make sure I didn't confuse the percentage. Sometimes people get confused whether the percentage is off the original or the sale price. But in this case, the problem says "60% off the original price," so the discount is calculated based on the original. So, yes, 60% of original is subtracted from original to get the sale price. So, sale price is 40% of original. So, my calculation should be correct. Alternatively, another way to think about it: if 14.40 is 40%, then 10% would be 14.40 divided by 4, which is 3.60. Then, 100% would be 3.60 times 10, which is 36.00. Yep, same result. So, that's another way to confirm. So, both methods give me \$36. So, I think that's the correct original price. Let me just recap to ensure I didn't skip any steps or make any miscalculations. Original price = P . 60% off means you pay 40% of P : $0.40P = 14.40$. Solving for P : $P = 14.40 / 0.40 = 36$. Check: 60% of 36 is 21.60, $36 - 21.60 = 14.40$, which matches the sale price. All steps make sense, calculations check out. Therefore, the original price was \$36. ****Final Answer**** The original price of the shirt was $\boxed{36}$ dollars.

</think>

The sale price of the shirt is \$14.40, which is 60% off the original price. This means the sale price is 40% of the original price. Let P be the original price. We can set up the equation: $0.40 \times P = 14.40$. To find P , we divide both sides by 0.40: $P = \frac{14.40}{0.40} = 36$. We verify this by checking that 60% off \$36 results in the sale price. Calculating 60% of \$36: $0.60 \times 36 = 21.60$. Subtracting this discount from the original price: $36 - 21.60 = 14.40$. This confirms the sale price is correct. Therefore, the original price of the shirt was $\boxed{36}$ dollars.

DeepSeek-R1 (8/128):

<think>

Okay, so I need to find the original price of the shirt before the discount. The sale price is \$14.40, and that's after 60% off the original price. Hmm, let me think. First, if the shirt is 60% off, that means the sale price is 100% - 60% = 40% of the original price. So, \$14.40 is 40% of the original price. To find the original price, I need to figure out what number's 40% is 14.40. Let me write that as an equation. Let's call the original price " x ". So, 40% of x equals 14.40. In math terms, that's $0.40 \times x = 14.40$. To solve for x , I can divide both sides by 0.40. So, $x = 14.40 / 0.40$. Let me calculate that. Hmm, 14.40 divided by 0.40. Maybe I can do this without a calculator. Let's see, 0.40 times 30 is 12.00, right? Because $0.4 \times 30 = 12$. Then, 14.40 is 12.00 + 2.40. So, 0.40 goes into 12.00 thirty times, and then 2.40 divided by 0.40 is 6. So, $30 + 6 = 36$. Therefore, $x = 36$. Wait, let me check that. If the original price is \$36, and it's 60% off, then the discount is 60% of 36. 60% of 36 is $0.60 \times 36 = 21.60$. So, subtracting that from 36 gives $36 - 21.60 = 14.40$, which matches the sale price. So that checks out. Alternatively, if I use the percentage of the original price, 40% of 36 is $0.40 \times 36 = 14.40$. Yep, that's correct. So, the original price was \$36. I think that's the answer. Let me just make sure I didn't make any calculation errors. So, 0.40 times 36. 0.4 times 30 is 12, and 0.4 times 6 is 2.4, so $12 + 2.4 = 14.4$. Perfect. So, the original price is \$36. I don't see any mistakes here. So, the answer should be \$36. ****Final Answer**** The original price of the shirt was $\boxed{36}$ dollars.

</think>

The sale price of the shirt is \$14.40, which is 60% off the original price. This means the sale price is 40% of the original price. Let x be the original price. We have: $0.40x = 14.40$. Solving for x : $x = \frac{14.40}{0.40} = 36$. To verify, 40% of \$36 is: $0.40 \times 36 = 14.40$. Thus, the original price of the shirt was $\boxed{36}$ dollars.

DeepSeek-R1 (8/32):

<think>

Okay, let's tackle this problem. So, the problem states that the sale price is \$14.40, and that's 60% off the original price. Wait, hold on a second. If 60% of the original price is 14.40, then to find the original price, we need to divide 14.40 by 0.60. Let me check that. So, let me write that out. If 60% of the original price is 14.40, then the original price is 14.40 divided by 0.60. Let me compute that. 14.40 divided by 0.60. Hmm, 0.60 times 24 is 14.40, right? Because 0.6 times 20 is 12, and 0.6 times 4 is 2.4, so $12 + 2.4 = 14.4$. So, 0.6 times 24 is 14.4. Therefore, 14.40 divided by 0.60 is 24. So, the original price is 24. Wait, let me check that. If the original price is 24, and 60% off is 24 times 0.6, which is 14.4. So, 14.40. Which matches the sale price. So, that's correct. Therefore, the original price is 24. So, the answer is 24. But wait, the problem says "60% off the original price". So, does that mean that 60% of the original price is the sale price? Wait, let me think. If the shirt's sale price is 60% off the original price, then yes, that's 0.6 times original price. So, original price is sale price divided by 0.6. So, 14.40 divided by 0.6 is 24. So, 24. So, the original price is 24. Yes, that's correct. So, the answer is 24. ****Answer**** $\boxed{24}$

</think>

The original price is 24.

Figure G.2: Example 2 of reasoning generation of DeepSeek-R1 with different sparsity levels using PreMoE. Notation: activated/retained experts.