Less Prune, *MoRE* Experts: Recognizing and Restructuring Latent Experts for Model Compression

Jiamu Zhang[1], Alessandro Mason[2], Ning Xie[3], Aarav Swami[2], Ashley Chen[2], Shuai Xu[2], Vipin Chaudhary[2], Hanjie Chen[1]

[1]Department of Computer Science, Rice University, USA
[2]Department of Computer and Data Science, Case Western Reserve University, USA
[3]Department of Computer Science, Florida International University, USA

Correspondence: mz81@rice.edu, hc86@rice.edu

Abstract

This work is ongoing and currently under review.

Large language models increasingly rely on conditional computation to balance reasoning ability and deployment efficiency. However, most existing compression methods for LLMs rely on static pruning, which permanently removes parameters and often degrades performance under distribution shift and knowledge-intensive NLP tasks.

In this work, we propose MoRE (Mixture-of-Recognized-Experts), a training-free framework that converts pretrained Transformers into input-dependent conditional computation without modifying model weights.

Our analysis reveals that pretrained MLP layers exhibit a hybrid structure consisting of a shared backbone that is consistently activated across inputs and input-dependent residual components that enable specialization. MoRE explicitly exploits this structure through a training-free routing mechanism, enabling effective conditional computation. As a result, MoRE selectively activates different subsets of pretrained parameters for different inputs at inference time, reducing computation without modifying model weights.

Experiments across multiple model families and benchmarks demonstrate that MoRE achieves competitive, and in many cases superior, accuracy–efficiency trade-offs compared to structured pruning methods, while remaining robust under distribution shift and knowledge-intensive question answering.

1 Introduction

Large language models (LLMs) have continued to scale in both parameter count and architectural complexity, leading to substantial challenges for deployment under realistic latency, memory, and energy constraints (Bai et al., 2024; Zhou et al., 2024; Sui et al., 2025). As a result, improving deployment time efficiency has become a central problem in large model deployment. Among existing approaches, pruning (Frantar and Alistarh, 2023a; Ma et al., 2023a; Sun et al., 2024) constitutes one of the major paradigms for reducing inference cost.

While pruning is often studied from a systems perspective, its impact on knowledge-intensive NLP tasks remains underexplored. Existing pruning methods can be broadly categorized into unstructured and structured approaches, with intermediate variants such as semi-structured sparsity bridging the two extremes (Zhou et al., 2021). Unstructured pruning (Frantar and Alistarh, 2023b; Jaiswal et al., 2023; Xia et al., 2023; Bhuiyan et al., 2025) removes individual weights, resulting in fine-grained sparsity patterns that maximize flexibility and compression ratio, but the resulting irregular sparsity leads to non-contiguous memory access and limited parallelism. As a result, achieving practical inference speedups typically requires specialized sparse kernels or custom CUDA implementations. In contrast, structured pruning (Xia et al., 2024; Ma et al., 2023b; Li et al., 2025; Ashkboos et al., 2024) removes entire architectural components, such as channels in feed-forward layers, attention heads in multi-head self-attention, or predefined blocks of parameters, thereby preserving regular computation patterns compatible with dense kernels (Men et al., 2024; Shopkhoev et al., 2025; Gromov et al., 2025). While this form of sparsity is more hardware-friendly, it imposes coarse-grained parameter removal, often reducing model expressive capacity and leading to noticeable accuracy degradation, especially under aggressive compression.

Beyond these trade-offs, a more fundamental limitation underlies static pruning: once removed, parameters are implicitly treated as permanently unimportant across all future inputs, regardless of input distribution or downstream task. In practice,

however, parameter importance is highly input-dependent: we observe substantial variation in activation patterns across inputs, tasks, and model families, indicating that capacity rarely used in one setting may be essential in another (see Appendix E.2 for a comprehensive empirical analysis).

Importantly, estimating parameter importance via task-specific calibration does not resolve this limitation: even when pruning decisions are informed by a calibration set, the resulting sparse model remains static and fixed prior to inference, without any mechanism to adapt its computation based on individual inputs at run-time. In contrast, dynamic or run-time pruning methods explicitly adapt sparsity patterns during inference to input characteristics, whereas static pruning determines a single fixed sparse structure offline prior to deployment (Liang et al., 2021).

This introduces nontrivial assumptions about data availability and deployment budget, in that redundancy identified under a limited calibration distribution must generalize to all future inputs. When such generalization assumptions fail, permanently removing capacity becomes brittle. **Our empirical analysis** shows that activation patterns differ significantly across domains, suggesting that neuron importance is strongly data-dependent(see Appendix E.2). This suggests a different deployment-time objective: instead of permanently removing capacity, can we retain it and *selectively* use it only when needed, i.e., conditionally activate different subsets of parameters for different inputs at inference time?

Motivated by the need for *training-free*[1] deployment-time conditional computation, we propose **MoRE** (**M**ixture-**o**f-**R**ecognized-**E**xperts), a training-free conditional compression framework that transforms static redundancy in pretrained LLMs into input-dependent sparse computation at inference time. Here, "experts" refer to *input-specialized parameter subsets induced by activation patterns in pretrained networks*. Unlike conventional MoE approaches that rely on additional training to create experts and optimize routing (Wei et al., 2024; Lepikhin et al., 2020), MoRE introduces no new parameters and requires no finetuning; instead, it leverages latent expert-like structures already present in pretrained networks and reorganizes them into conditionally activated sub-networks without modifying model weights. This paper makes the following contributions:

- We identify a fundamental limitation of static pruning for pretrained LLMs: parameter importance is highly input-dependent, making permanent parameter removal brittle under distribution shift and diverse deployment scenarios.
- We propose **MoRE**, a training-free compression framework that transforms static redundancy in pretrained LLMs into input-dependent sparse computation at inference time, without modifying model weights or introducing additional training.
- We empirically demonstrate that MoRE achieves favorable accuracy–efficiency tradeoffs across multiple model families and benchmarks, while remaining robust under distribution shift and knowledge-intensive question answering compared to structured pruning baselines.

2 Method

2.1 Mixture of Recognized Experts

We first introduce the notation used throughout this section. Consider a pretrained dense Transformer and let $\mathcal{L}$ denote the set of layers (typically MLP blocks) to be MoE-ified. Fix any $\ell \in \mathcal{L}$. Let $\mathbf{x}^{(\ell,t)} \in \mathbb{R}^d$ be a token hidden state of the t-th token in the calibration set at layer ℓ. When constructing recognized experts, t indexes tokens from a calibration set. Then the MLP at layer ℓ maps $\mathbf{x}^{(\ell,t)}$ to an output hidden state $\mathbf{h}^{(\ell,t)} \in \mathbb{R}^m$ via an intermediate activation function ϕ_ℓ, where m is the MLP expansion width (MLP intermediate size). As the layer ℓ will be fixed throughout this section, to simplify notation, we will omit the index ℓ from functions, vectors and matrices whenever there is no risk of confusion. Specifically, we consider the 2-layers MLP formulation without gating,

$$\mathbf{h}^{(t)} = \phi\Big(\mathbf{W}_{\text{up}}\mathbf{x}^{(t)}\Big), \quad \mathbf{y}^{(t)} = \mathbf{W}_{\text{down}}\mathbf{h}^{(t)}. \quad (1)$$

Here $\mathbf{W}_{\text{up}} \in \mathbb{R}^{m\times d}$, $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d\times m}$, and $\phi(\cdot)$ denotes an element-wise nonlinearity (such as GELU or SiLU). We will not consider gate projections but focus only on the $\mathbf{W}_{\text{up}} \to \phi(\cdot) \to \mathbf{W}_{\text{down}}$ pathway in this paper.

The goal of recognized experts is to decompose the m intermediate channels into (i) a shared backbone that is consistently important across inputs,

[1] Here, "training-free" means no gradient-based optimization or parameter updates; lightweight calibration is used only to collect activation statistics.

and (ii) multiple specialized subspaces that exhibit input-dependent patterns. To formalize this, we introduce co-activation statistics and importance scoring below.

2.2 Activation frequency vector and co-activation matrix

Let T be a calibration dataset containing N tokens in total. To compress the MLP at layer ℓ, we collect token-level activation pairs from T

$$\mathcal{T} = \left\{ \left(\mathbf{x}^{(t)}, \mathbf{h}^{(t)} \right) \right\}_{t=1}^{N}. \tag{2}$$

Our goal is to construct r experts $\{E_1, \ldots, E_r\}$ for layer ℓ where an expert E_i is defined by a subset of channel indices $E_i \subset [m]$.

Activation channel set of a token hidden state. Let $\mathbf{h}^{(t)}$ be the intermediate activation vector of a token hidden state $\mathbf{x}^{(t)}$, define its element-wise magnitude vector $\mathbf{a}^{(t)} \in \mathbb{R}^m_{\geq 0}$ as

$$a_i^{(t)} = |h_i^{(t)}| \quad \text{for every } 1 \leq i \leq m. \tag{3}$$

Let $\rho_{\text{act}} \in (0, 1)$ be a hyperparameter called *activation ratio* and define $k_{\text{act}} = \max(1, \lfloor \rho_{\text{act}} \cdot m \rfloor)$ to be the *activation channel number*. That is, we treat only the top-k_{act} entries of the intermediate activation $\mathbf{h}^{(t)}$ as activated channels and the remaining $m - k_{\text{act}}$ as silent channels when the input vector is $\mathbf{x}^{(t)}$. This choice provides a simple and robust proxy for identifying dominant channels in each token activation.

Correspondingly, we define the token-wise activation channel set of $\mathbf{x}^{(t)}$ to be

$$S^{(t)} = \text{TopK}\left(\mathbf{a}^{(t)}, k_{\text{act}} \right) \subseteq [m], \tag{4}$$

where the function $\text{TopK}(\mathbf{x}, \ell)$ returns the indices of the ℓ largest entries of vector $\mathbf{x}$.

Channel frequency. Next we aggregate the sets of activation channels of all token hidden states $\mathbf{h}^{(t)}$ in $\mathcal{T}$ to get the (normalized) activation frequency vector $\mathbf{f} \in \mathbb{R}^m_{\geq 0}$ and co-activation matrix $\mathbf{C} \in \mathbb{R}^{m \times m}_{\geq 0}$ of the $\mathcal{T}$. Specifically, we define the i^{th} entry of $\mathbf{f}$ to be the (relative) frequency of channel i (i.e. the i^{th} neuron or the i^{th} column of $\mathbf{W}_{\text{down}}$) appears in the activation channel sets among all token hidden states in $\mathcal{T}$

$$f_i = \frac{1}{N} \sum_{t=1}^{N} \mathbb{1}_{i \in S^{(t)}}, \tag{5}$$

for every $1 \leq i \leq m$. Note that $\sum_{i=1}^{m} f_i = k_{\text{act}}$.

Co-activation matrix. Similarly, we define the (i, j)-th entry of the co-activation matrix $\mathbf{C}$ to be the (relative) frequency that both channel i and channel j appear in the activation channel set of the same token hidden state $\mathbf{C} \in \mathbb{R}^{m \times m}_{\geq 0}$ as

$$C_{ij} = \frac{1}{N} \sum_{t=1}^{N} \mathbb{1}_{i \in S^{(t)}} \cdot \mathbb{1}_{j \in S^{(t)}}, \tag{6}$$

for every $1 \leq i, j \leq m$.

Identifying Globally Important Backbone Channels. To identify backbone channels that are consistently important across inputs, we assign an importance score to each intermediate channel by jointly considering its activation statistics and pretrained projection weights. Intuitively, backbone channels correspond to directions in the intermediate space that contribute broadly to the feed-forward computation across many inputs, rather than being selectively activated in specific contexts.

Formally, for each intermediate channel j, we define its importance score s_j as

$$\begin{aligned} s_j \;=\; & \sum_i \left| W_{\text{up}}[j, i] \right| \cdot \mathbb{E}_{t \sim \mathcal{T}} \left[x_i^{(t)\,2} \right] \\ & + \sum_k \left| W_{\text{down}}[k, j] \right| \cdot \mathbb{E}_{t \sim \mathcal{T}} \left[h_j^{(t)\,2} \right]. \end{aligned} \tag{7}$$

Intuitively, this score measures how strongly a channel participates in the feed-forward computation by combining its typical activation magnitude with the strength of its incoming and outgoing projections, so channels with large s_j contribute broadly across many inputs.

Let $\rho_{\text{share}} \in (0, 1)$ be a hyperparameter which denotes the shared-channel ratio and define $k_{\text{share}} = \max(1, \lfloor \rho_{\text{share}} \cdot m \rfloor)$. We then select the top-k_{share} channels with the largest importance scores to form the backbone set C_{share}, which is shared across all recognized experts. The corresponding backbone weights are defined as

$$\begin{aligned} W_u^{\mathcal{B}} &= \left[W_u[:, j] \right]_{j \in C_{\text{share}}}, \\ W_d^{\mathcal{B}} &= \left[W_d[j, :] \right]_{j \in C_{\text{share}}}. \end{aligned} \tag{8}$$

This formulation follows prior work on activation-aware importance scoring (Sun et al., 2024), where input variance and activation variance are used to approximate channel contribution in the up- and down-projection, respectively, but is used here solely to identify globally shared channels rather than to remove parameters.

Clustering non-shared channels. Let $R = [m] \setminus C_{\text{share}}$ denote the set of non-shared channels. To partition these non-shared channels into similar channel groups, we consider the co-activation matrix $\mathbf{C}$ restricted to non-shared channels, i.e. the principal submatrix $\mathbf{C}_R$. As $\mathbf{C}_R$ is symmetric with all off-diagonal entries non-negative, we may naturally view the off-diagonal entries of $\mathbf{C}_R$ as the entries of an adjacency matrix of a weighted undirected graph G over the vertex set R (that is, the adjacency matrix A_G is obtained from $\mathbf{C}_R$ by zeroing out its diagonal entries), and apply spectral clustering algorithm to partition R into r disjoint clusters:

$$\{S_1, \ldots, S_r\} = \text{SpectralCluster}(\mathbf{C}_R, r)\,,$$

such that $\uplus_{i=1}^{r} S_i = R$. Here r is another hyperparameter of the compression scheme.

Finally we define r experts as r expert channel sets:

$$E_i = S_i \cup C_{\text{share}} \quad \text{for } i = 1, \ldots, r.$$

where $|E_i|$ denotes the number of channels assigned to expert E_i. Note that $\mathcal{E} = \{E_1, \ldots, E_r\}$ form a sunflower over $[m]$.

Crucially, this process does not introduce new parameters or optimization objectives. The expert decomposition is *recognized* from the pretrained structure: weight and activation statistics reveal latent modularity, and clustering partitions the residual space into specialized components. This distinguishes **MoRE** from pruning variants that directly remove weights based on magnitude or importance scores.

2.3 Training-Free Routing

After constructing recognized experts, the remaining challenge is to determine, for each input token, which expert-specific subspace should be activated.

Expert Input Spaces & Expert Prototypes. Unlike learned MoE routers that optimize routing via gradient-based training, MoRE performs routing by matching token representations to expert prototypes derived from calibration statistics. Under the training-free constraint, one cannot learn a routing function to assign inputs to experts. Instead, we approach routing by *characterizing the regions of representation space* to which each expert is naturally responsive.

Intuitively, if we can describe what kinds of hidden representations an expert tends to specialize in, then routing reduces to matching an input token to the expert whose specialization it best aligns with, so in our design, calibration statistics are used to form the representations of each expert's input space, and each input token is routed to the expert with the closest representation. Concretely, expert prototypes are constructed once using calibration tokens, and are then fixed and reused for routing at inference time.

Token-to-expert assignment. For each token hidden state $\mathbf{x}^{(t)}$ (with intermediate activation $\mathbf{h}^{(t)}$) and expert E_i, we define the expert-specific activation energy to be

$$\mathcal{A}(\mathbf{h}^{(t)}, E_i) = \frac{\|\mathbf{h}_{E_i}^{(t)}\|_2}{|E_i|^{\alpha}}, \tag{9}$$

i.e., the ℓ_2 norm of $\mathbf{h}^{(t)}$ restricted to entries in E_i, with $\alpha \in [0, 1]$ being a debiasing factor that compensates for differences in expert subspace dimensionality.

Then each token is assigned to the expert with whom the activation energy is maximized:

$$\text{Expert}(\mathbf{x}^{(t)}) = \arg\max_{i \in [r]} \mathcal{A}(\mathbf{h}^{(t)}, E_i), \tag{10}$$

and denote by $T_i = \{\mathbf{x}^{(t)} \mid \text{Expert}(\mathbf{x}^{(t)}) = i\}$ the set of tokens assigned to expert E_i. Note that $\{T_1, \ldots, T_r\}$ forms a partition of the calibration set T.

Characteristic vectors of experts. We define the *characteristic vector* $\boldsymbol{\mu}_i$ of an expert E_i to be the centroid of vectors in T_i:

$$\boldsymbol{\mu}_i = \frac{1}{|T_i|} \sum_{\mathbf{x}^{(t)} \in T_i} \mathbf{x}^{(t)}, \tag{11}$$

which are the representations of experts' *input space*[2] .

Similarity-based routing at inference. Given an inference-time token representation $\mathbf{x}$, routing is performed by similarity comparisons with the characteristic vectors of experts:

$$\text{Expert}(\mathbf{x}) = \arg\max_{i \in [r]} \text{sim}(\mathbf{x}, \boldsymbol{\mu}_i)\,, \tag{12}$$

A standard choice is the cosine similarity,

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\|_2 \, \|\mathbf{v}\|_2 + \epsilon}, \tag{13}$$

where a tiny stability parameter ϵ is added to prevent the "division by zero" error.

[2]This routing mechanism is conceptually related to nearest-prototype assignment in representation space, but differs from learned MoE routers in that no parameters are trained.

3 Experiments

3.1 Experimental Settings

Models and Tasks. We evaluate MoRE across multiple model families and deployment settings. Our evaluation covers decoder-only language models ranging from 3B to 70B parameters, including LLaMA-2 (7B, 13B) (Touvron et al., 2023), LLaMA-3 series (3.2-3B, 3.1-8B, 3.1-8B-Instruct) (Grattafiori et al., 2024) and Qwen-2.5-7B-Instruct (Qwen et al., 2025).

We consider two complementary evaluation settings: **(i) Language Modeling and Commonsense Tasks** measure next-token prediction quality and short-form reasoning using perplexity on WikiText-2 (Merity et al., 2016) and PTB (Marcus et al., 1993), as well as zero-shot accuracy on WinoGrande (Sakaguchi et al., 2019), ARC-Easy/ARC-Challenge (Clark et al., 2018), and SciQ (Welbl et al., 2017). These benchmarks align with the standard evaluation protocol adopted by most prior pruning and compression work.

(ii) Knowledge-Intensive Question Answering evaluates transfer behavior beyond language modeling using MMLU (Hendrycks et al., 2021) under a 5-shot setting. Rather than serving as a primary optimization target, MMLU is used as a stress test to examine how compression methods behave under distribution shift and increased knowledge demands. We further use this evaluation to probe whether performance gains observed under language modeling metrics translate to more knowledge-intensive scenarios.

Different model subsets are evaluated under different settings to reflect common deployment practices. Base and smaller instruction-tuned models (LLaMA-2, LLaMA-3.2-3B, LLaMA-3.1-8B, and Qwen-2.5-7B-Instruct) are evaluated on language modeling and commonsense tasks. LLaMA-3.1-8B-Instruct is further evaluated on the MMLU Benchmark.

Baselines. We compare MoRE against representative *structured pruning* baselines, including channel-wise (An et al., 2023), block-wise (Ma et al., 2023b), and layer-wise pruning (Men et al., 2024; Gromov et al., 2025; Shopkhoev et al., 2025). For all baselines, we replicate the original methods and apply compression and evaluation within a unified and controlled pipeline, ensuring consistent model architectures and evaluation protocols. A detailed discussion of these baselines is provided in the Appendix A.

Unlike static pruning, MoRE performs conditional computation by activating different parameter subsets at inference time while preserving all pretrained weights. To enable fair comparison under this dynamic setting, we report the *pruning rate* (PR), defined as $\mathrm{PR} = 1 - \mathrm{APR}$, where APR denotes the expected activated parameter ratio, i.e., the expected fraction of parameters activated per token relative to the original dense model. A detailed definition of this metric, together with analytical justification and empirical validation, is provided in Appendix F.2.

3.2 Results

Table 1: Performance comparison on LLaMA2 models with different scales with PR $\approx 20\%$. We report average perplexity (PPL-Avg) across language modeling benchmarks and average accuracy (QA-Avg) across non-iterative QA tasks.

Method	LLaMA2-7B		LLaMA2-13B	
	PPL-Avg ↓	QA-Avg ↑	PPL-Avg ↓	QA-Avg ↑
Dense	14.29	69.50	16.56	72.28
LLMPruner	26.18	58.77	35.49	64.52
FLAP	20.48	62.15	26.04	64.58
UIDL	31.78	55.77	26.49	63.43
ShortGPT	24.97	56.49	26.49	63.33
Ours	**14.55**	**64.51**	**20.85**	**68.38**

Table 2: We report average perplexity (PPL-Avg) and average QA accuracy (QA-Avg) with PR $\approx 20\%$.

Method	LLaMA3.1-8B		Qwen2.5-7B	
	PPL-Avg ↓	QA-Avg ↑	PPL-Avg ↓	QA-Avg ↑
Dense	8.41	76.13	10.62	70.07
LLMPruner	21.19	60.20	–	–
UIDL	45.61	63.92	19.12	58.58
ShortGPT	45.61	63.92	19.38	57.79
Ours	**10.70**	**71.38**	**14.25**	**62.66**

Overall Performance under Matched Compression Ratios. We evaluate all methods under a matched compute budget across multiple model families. All results are obtained without using domain-specific datasets or task-aware calibration. Across all evaluated backbones, MoRE consistently achieves lower perplexity than existing training-free compression baselines. This trend holds across model scales and families, including both LLaMA-2 and LLaMA-3 variants. On non-iterative commonsense QA benchmarks (WinoGrande, ARC, SciQ), MoRE attains the highest or comparable average accuracy among training-free methods.

References

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2023. Fluctuation-based adaptive structured pruning for large language models. *Preprint*, arXiv:2312.11983.

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *Preprint*, arXiv:2401.15024.

Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, and 1 others. 2024. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*.

Uranik Berisha, Jens Mehnert, and Alexandru Paul Condurache. 2025. Efficient data driven mixture-of-expert extraction from trained networks. *Preprint*, arXiv:2505.15414.

Samiul Basir Bhuiyan, Md. Sazzad Hossain Adib, Mohammed Aman Bhuiyan, Muhammad Rafsan Kabir, Moshiur Farazi, Shafin Rahman, and Nabeel Mohammed. 2025. Z-pruner: Post-training pruning of large language models for efficiency without retraining. *Preprint*, arXiv:2508.15828.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *Preprint*, arXiv:2308.06767.

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. *Preprint*, arXiv:1803.05457.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *Preprint*, arXiv:2401.06066.

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

William Fedus, Barret Zoph, and Noam Shazeer. 2022b. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Preprint*, arXiv:2101.03961.

Yuchen Feng, Bowen Shen, Naibin Gu, Jiaxuan Zhao, Peng Fu, Zheng Lin, and Weiping Wang. 2025. Dive into moe: Diversity-enhanced reconstruction of large language models from dense into mixture-of-experts. *Preprint*, arXiv:2506.09351.

Elias Frantar and Dan Alistarh. 2023a. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Elias Frantar and Dan Alistarh. 2023b. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

Trevor Gale, Erich Elsen, and Sara Hooker. 2019. The state of sparsity in deep neural networks. *Preprint*, arXiv:1902.09574.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. 2025. The unreasonable ineffectiveness of the deeper layers. *Preprint*, arXiv:2403.17887.

Hongcan Guo, Haolang Lu, Guoshun Nan, Bolun Chu, Jialin Zhuang, Yuan Yang, Wenhao Che, Sicong Leng, Qimei Cui, and Xudong Jiang. 2025a. Advancing expert specialization for better moe. *Preprint*, arXiv:2505.22323.

Wentao Guo, Mayank Mishra, Xinle Cheng, Ion Stoica, and Tri Dao. 2025b. Sonicmoe: Accelerating moe with io and tile-aware optimizations. *Preprint*, arXiv:2512.14080.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*.

Ethan He, Abhinav Khattar, Ryan Prenger, Vijay Korthikanti, Zijie Yan, Tong Liu, Shiqing Fan, Ashwath Aithal, Mohammad Shoeybi, and Bryan Catanzaro. 2025. Upcycling large language models into mixture of experts. *Preprint*, arXiv:2410.07524.

Jiujun He and Huazhen Lin. 2025. Olica: Efficient structured pruning of large language models without retraining. *Preprint*, arXiv:2506.08436.

Xu Owen He. 2024. Mixture of a million experts. *Preprint*, arXiv:2407.04153.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Yongqi Huang, Peng Ye, Chenyu Huang, Jianjian Cao, Lin Zhang, Baopu Li, Gang Yu, and Tao Chen. 2025. Ders: Towards extremely efficient upcycled mixture-of-experts models. *Preprint*, arXiv:2503.01359.

Ajay Jaiswal, Shiwei Liu, Tianlong Chen, and Zhangyang Wang. 2023. The emergence of essential sparsity in large pre-trained models: The weights that matter. *Preprint*, arXiv:2306.03805.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. *Preprint*, arXiv:1905.00414.

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. *Preprint*, arXiv:2006.16668.

Guanchen Li, Yixing Xu, Zeping Li, Ji Liu, Xuanwu Yin, Dong Li, and Emad Barsoum. 2025. Týr-the-pruner: Structural pruning llms via global sparsity distribution optimization. *Preprint*, arXiv:2503.09657.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Preprint*, arXiv:2101.09671.

Seng Pei Liew, Takuya Kato, and Sho Takase. 2025. Scaling laws for upcycling mixture-of-experts language models. *Preprint*, arXiv:2502.03009.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023a. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023b. Llm-pruner: On the structural pruning of large language models. *Preprint*, arXiv:2305.11627.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *Preprint*, arXiv:2403.03853.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, and Chong Yu. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

J Pablo Muñoz, Jinjie Yuan, and Nilesh Jain. 2025. Multipruner: Balanced structure removal in foundation models. *arXiv preprint arXiv:2501.09949*.

Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. 2025a. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. In *The Thirteenth International Conference on Learning Representations*.

Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. 2025b. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. *Preprint*, arXiv:2502.19261.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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

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

Dmitriy Shopkhoev, Ammar Ali, Magauiya Zhussip, Valentin Malykh, Stamatios Lefkimmiatis, Nikos Komodakis, and Sergey Zagoruyko. 2025. Replaceme: Network simplification via depth pruning and transformer block linearization. *Preprint*, arXiv:2505.02819.

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. 2025. Evopress: Accurate dynamic model compression via evolutionary search. *Preprint*, arXiv:2410.14649.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *Preprint*, arXiv:2503.16419.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. 2024. A simple and effective pruning approach for large language models. *Preprint*, arXiv:2306.11695.

Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. 2024. Scattered mixture-of-experts implementation. *Preprint*, arXiv:2403.08245.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Tianwen Wei, Bo Zhu, Liang Zhao, Cheng Cheng, Biye Li, Weiwei Lü, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Liang Zeng, Xiaokun Wang, Yutuan Ma, Rui Hu, Shuicheng Yan, Han Fang, and Yahui Zhou. 2024. Skywork-moe: A deep dive into training techniques for mixture-of-experts language models. *Preprint*, arXiv:2406.06563.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *Preprint*, arXiv:1707.06209.

Miles Williams and Nikolaos Aletras. 2024. On the impact of calibration data in post-training quantization and pruning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 10100–10118. Association for Computational Linguistics.

Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. 2023. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Preprint*, arXiv:2309.10285.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning. *Preprint*, arXiv:2310.06694.

Kang Zhao, Tao Yuan, Han Bao, Zhenfeng Su, Chang Gao, Zhaofeng Sun, Zichen Liang, Liping Jing, and Jianfei Chen. 2025. Beyond 2:4: exploring v:n:m sparsity for efficient transformer inference on gpus. *Preprint*, arXiv:2410.16135.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2021. Learning n:m fine-grained structured sparse neural networks from scratch. *Preprint*, arXiv:2102.04010.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, and 1 others. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2024. A survey on model compression for large language models. *Preprint*, arXiv:2308.07633.

A Related Works.

A.1 Unstructured Pruning

Unstructured pruning removes individual weights without enforcing regular sparsity patterns, often achieving high compression ratios while preserving model accuracy (Frantar and Alistarh, 2023a; Sun et al., 2024). However, the resulting irregular sparsity is difficult to exploit efficiently on modern hardware, limiting practical inference speedups (Han et al., 2016). From a systems perspective, unstructured sparsity introduces irregular memory access and control flow, as surviving nonzero weights are scattered without predictable layout. This irregularity prevents efficient vectorization and reduces arithmetic intensity, leading to poor utilization of modern hardware such as GPUs and TPUs, which are optimized for dense or regular block-based computation. As a result, realizing inference speedups from unstructured pruning typically requires specialized sparse kernels, custom hardware support, or a significant engineering effort to overcome memory bandwidth and scheduling bottlenecks (Gale et al., 2019; Mishra et al., 2021). In particular, sparse matrix operations often become memory-bound rather than compute-bound, negating the theoretical reduction in floating-point operations.

Despite these challenges, unstructured pruning remains attractive due to its flexibility and strong accuracy retention at high sparsity levels, and is widely used in scenarios where model size reduction or storage efficiency is prioritized over raw inference throughput.

A.2 Semi-Structured Pruning

Semi-structured sparsity, such as $N:M$ patterns, introduces additional regularity by enforcing a fixed number of nonzero weights within small contiguous blocks. Compared to fully unstructured sparsity, these constraints enable more predictable memory access and control flow, allowing better utilization of vector units and partial support from modern accelerators. As a result, $N:M$ sparsity represents a practical compromise between fine-grained flexibility and hardware efficiency (Zhou et al., 2021; Zhao et al., 2025).

Nevertheless, semi-structured pruning still yields statically compressed models with fixed sparsity patterns that execute identical computation for all inputs. Realizing inference speedups typically requires specialized kernels or dedicated hardware support to match the imposed sparsity structure. Consequently, unstructured and semi-structured pruning methods are complementary to, but fundamentally different from, structured pruning approaches that operate on explicit architectural units.

A.3 Structured Pruning by Granularity

In contrast to fine-grained sparsification, structured pruning explicitly targets semantically meaningful architectural units, and the notion of "structure" can manifest at different levels of granularity within the Transformer architecture. Existing pruning methods for large language models can be broadly categorized by the granularity at which parameters or structures are removed. Unlike unstructured sparsification, structured pruning removes predefined architectural units, leading to hardware-friendly reductions in inference cost. Below we review structured pruning methods at different granularities.

Channel-, Dimension- and Block-Level Pruning. A number of methods prune model parameters at the level of MLP channels, attention heads, or embedding dimensions. LLM-Pruner (Ma et al., 2023b) applies gradient-based importance criteria to prune coupled transformer components, including attention heads and MLP channels, followed by limited fine-tuning. T'ry-the-Pruner (Li et al., 2025) and EvoPress (Sieberling et al., 2025) focus on channel- or head-level pruning using one-shot or search-based strategies to reduce calibration and retraining costs. FLAP (An et al., 2023) estimates channel importance using activation fluctuation statistics and compensates pruned structures with bias injection.

SliceGPT (Ashkboos et al., 2024) performs fine-grained structured compression by rotating the embedding space and slicing dimensions, enabling retraining-free parameter reduction. Although operating at a finer granularity than explicit channel pruning, such dimension-level methods similarly preserve the overall transformer structure. Olica (He and Lin, 2025) combines channel-level pruning with low-rank factorization, representing a hybrid approach that modifies both structural connectivity and internal parameterization.

Block-level methods remove or simplify entire transformer modules, such as MLP blocks or attention blocks. MultiPruner (Muñoz et al., 2025) prunes blocks, channels, and attention heads jointly, with block removal serving as a primary source of computational savings.

Layer-Level Pruning. Layer pruning removes complete transformer layers, resulting in the most aggressive form of structured compression. ShortGPT (Men et al., 2024) prunes layers based on block influence scores that measure input–output similarity. ReplaceMe (Shopkhoev et al., 2025) simplifies deep transformer models by replacing removed layers with lightweight linear approximations. UIDL (Gromov et al., 2025) examines the ineffectiveness of deep layers and designs a method to prune deeper layers based on representational redundancy.

A.4 Mixture-of-Experts: Training-Based Upcycling and Efficient Inference

MoE from Scratch. Mixture-of-Experts (MoE) models introduce conditional computation by activating only a subset of parameters for each input token (Fedus et al., 2022a; Shazeer et al., 2017; Dai et al., 2024). In their canonical form, MoE architectures are trained from scratch, with expert specialization and routing behavior jointly learned through optimization (Wei et al., 2024; Lepikhin et al., 2020). This design enables scaling model capacity while controlling per-token computation, and has been adopted in large-scale language models to achieve favorable accuracy–efficiency trade-offs.

MoE Upcycling from Dense Checkpoints. Beyond training from scratch, several works explore converting pretrained dense models into MoE architectures through dense-to-sparse upcycling (He et al., 2025). These approaches aim to reuse pretrained weights while introducing expert structure and routing mechanisms via additional training (Huang et al., 2025; Liew et al., 2025; Nakamura et al., 2025a). Typically, expert specialization is induced by retraining selected modules or reconstructing experts from dense representations, often requiring nontrivial optimization to achieve stable routing behavior (Guo et al., 2025a). While upcycling reduces the cost of training large MoE models from scratch, it nevertheless relies on training to create and align conditional structure, distinguishing it from fully training-free approaches.

Dense-to-MoE Conversion and Expert Extraction. Several recent works explore transforming pretrained dense models into MoE-like architectures or extracting expert subnetworks post-training. (Berisha et al., 2025) propose an efficient data-driven approach to extract experts from trained networks by clustering activation patterns in MLP layers and constructing an MoE variant based on these clusters. (Feng et al., 2025) reconstructs a sparse MoE from a dense model by enhancing expert diversity through domain affinity mining and pruning-based reconstruction, often involving further retraining of routers and experts to achieve strong performance.

Efficient Inference for Conditional Computation. A complementary line of work focuses on accelerating inference for models with conditional computation, including MoE architectures with learned routing. These methods optimize expert dispatch, batching, communication, and kernel execution to improve throughput and latency (Guo et al., 2025b; Tan et al., 2024), assuming that the routing structure is already available. Such systems-level advances are orthogonal to our focus, as they address how to efficiently execute conditional computation, rather than how conditional structure arises in pretrained dense models.

B Rethinking Static Compression in Pretrained LLMs

B.1 Permanent Parameter Removal in Pruning

Having discussed why static parameter removal is brittle for pretrained LLMs, we next revisit the evidence for input-dependent parameter usage and clarify the deployment constraints that arise under *static compression*.[3]

Empirically, we observe that activation patterns in pretrained LLMs vary substantially across input distributions and downstream tasks, indicating that few parameters are consistently active or inactive

[3]By *static compression*, we refer to post-training methods that permanently remove model parameters based on fixed importance estimates.

across all settings (see Appendix E.2 for a comprehensive study on the natural activation patterns and their dependence on model and task). Consequently, most high-performing pruning methods for LLMs rely heavily on downstream calibration data to estimate parameter importance under a specific task or distribution and to ensure that removed weights do not harm performance on that setting (Cheng et al., 2024).

B.2 Input-Dependent Parameter Importance in LLMs

To address the limitations of static parameter removal in pruning, an alternative paradigm has gained increasing attention: conditional computation, in which the layer in the model dynamically selects a subset of model parameters to execute based on the input (Shazeer et al., 2017). Unlike pruning, which permanently removes parameters after calibration, conditional computation preserves all parameters and adapts computation at inference time, allowing different inputs to engage different parts of the model.

Mixture-of-Experts (MoE) (Fedus et al., 2022a; Shazeer et al., 2017) architectures represent a prominent instantiation of conditional computation and provide a useful reference point for understanding the potential and limitations of input-dependent activation. In MoE models, a lightweight routing mechanism activates only a small subset of experts for each input token, thereby reducing per-token computation while maintaining—or even increasing—overall model capacity (Fedus et al., 2022b; He, 2024). This design has been adopted in many recent large-scale language models due to its favorable efficiency–capacity trade-off. A key insight underlying MoE models is that input-dependent activation enables higher expressive capacity under a fixed computation budget by allocating capacity selectively across inputs. For the same number of activated parameters, conditionally computed models can represent a richer family of functions than dense models, as different inputs can leverage different subnetworks.

B.3 Calibration Dependency and Deployment Constraints

While conditional computation has demonstrated strong efficiency–capacity trade-offs, existing MoE models typically incur additional costs. Whether constructed via dense-to-sparse up-cycling or trained from scratch, MoE architectures fundamentally rely on additional training to induce expert specialization and to learn routing mechanisms that assign input tokens to experts (Liew et al., 2025; He et al., 2025; Nakamura et al., 2025b; Guo et al., 2025a).

From a post-training compression and deployment perspective, this reliance on training is often impractical. Many real-world scenarios provide no access to downstream data distributions, no budget for finetuning, and only permit deploy-time modification of pretrained models (Williams and Aletras, 2024; Zhu et al., 2024). This constraint is especially restrictive for highly expressive pretrained LLMs, where few parameters are truly globally unimportant; instead, most weights contribute in an input- and context-dependent manner, making static removal or retraining-based specialization difficult to justify. These observations suggest that effective post-training compression requires a mechanism that preserves model capacity while adapting computation to input variation, without relying on additional training.

C Extended Method.

C.1 Motivation: Non-Uniform Channel Contributions in Dense Feed-Forward Layers

To enable a *training-free* transformation from dense feed-forward layers to input-conditioned activated MoE style layer, a key prerequisite is that the original MLP layers exhibit some form of *latent separability*. We need to identify exploitable structure in dense feed-forward layers without modifying model weights, so that the model will achieve input-dependent sparse computation by selectively activating different subsets of parameters for different inputs at inference time.

One might conjecture that latent separability could be trivially exploited by partitioning MLP channels (i.e., individual hidden dimensions in the feed-forward layers) into disjoint groups and activating different groups for different inputs. Under such a formulation, the dense feed-forward layer would admit a clean decomposition into independent subnetworks, each responsible for a distinct subset of inputs. However, this idealized form of separability rarely holds in practice. Empirically, we find that naive expert partitioning—where channels are rigidly split into disjoint experts—fails to preserve model performance, as it disrupts shared computation that is consistently required across inputs. We provide both activation

visualizations and quantitative evidence showing that such naive splits lead to substantial accuracy degradation (Appendix E.3).

At the other extreme, if all channels in a feed-forward network contributed uniformly and interchangeably across inputs—that is, each channel exhibited similar output magnitude for any input token—then any rule-based conditional activation would inevitably disrupt the original computation. This motivates us to ask a basic but underexplored question:

Do all channels in pretrained FFNs contribute similarly to computation, or is there an inherent imbalance that can be exploited for conditional execution?

Non-Uniform Channel Contributions To answer this question, we analyze the activation contribution spectrum of channels in pretrained FFN layers. For each channel, we aggregate its activation magnitude across a large set of input tokens and normalize by the total activation mass. Figure 1 visualizes the resulting channel activation spectrum, with channels ranked by their total contribution.

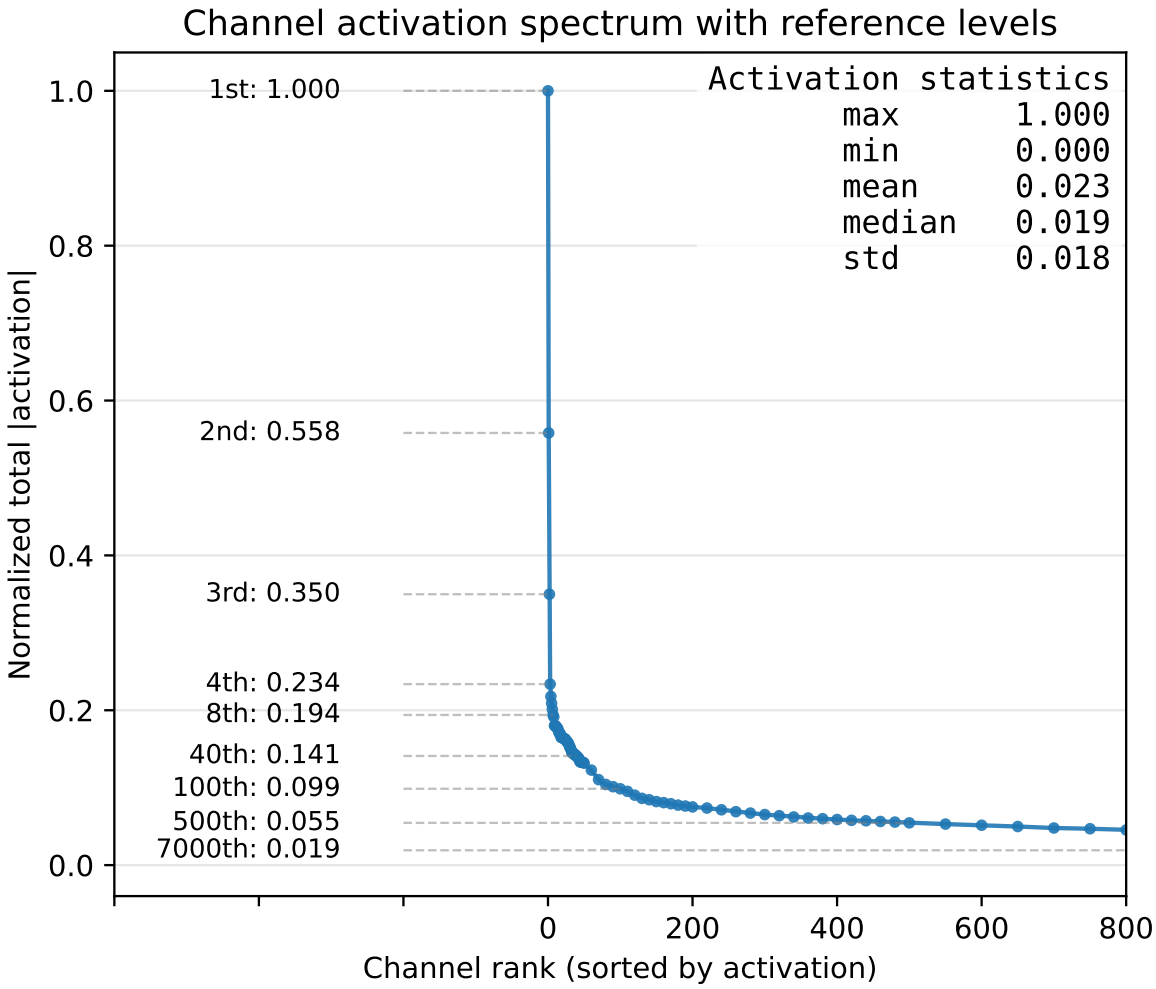

Figure 1: Channel activation spectrum of a pretrained MLP layer. A pronounced long-tail pattern emerges, where a small subset of channels dominates total activation, while most channels are weakly activated, indicating a hybrid structure of shared backbone and input-dependent channels.

We observe a highly skewed distribution: a small fraction of channels consistently accounts for a disproportionate share of total activation, while the majority of channels contribute only weakly on average. For example, the top-ranked channels alone capture a substantial portion of the total activation mass, whereas hundreds or thousands of remaining channels exhibit long-tail, low-frequency contributions.

This phenomenon suggests that dense FFNs implicitly encode non-uniform channel importance, even though all channels are executed for every input during inference.

Backbone Channels vs. Input-Dependent Channels. The observed activation imbalance motivates a *structural hypothesis*: **a subset of channels may consistently contribute to core computations across diverse inputs, while the remaining channels participate in a more input-dependent manner, with different tokens engaging different subsets of these channels.**

Conceptually, this hybrid activation pattern can be viewed as being well-approximated by a *sunflower-like*[4] structure over the set of token-wise activation channels: a shared core of frequently activated channels corresponds to the core of the sunflower, while input-dependent subsets of channels form its petals. We do not assume that token-wise activation patterns form an exact sunflower. Rather, we use the sunflower abstraction as a low-complexity structural proxy to capture the coexistence of a shared core and input-dependent residuals, which significantly simplifies both reasoning and algorithm design.

Under this abstraction, channels with high activation frequency can be interpreted as forming a **shared backbone** that supports general model functionality, whereas sparsely activated channels may enable **contextual specialization** by contributing selectively depending on the input. We emphasize that this interpretation is conjectural at this stage and serves as a guiding intuition, rather than a definitive empirical conclusion.

This hypothesis naturally raises a research question: ***If dense feed-forward layers indeed exhibit a backbone–specialization structure, how can conditional computation be constructed to exploit this organization without retraining or modifying model weights?***

Instead, a more flexible strategy is to preserve a shared set of backbone channels that are always executed, while conditionally engaging subsets of sparsely activated channels based on input characteristics. Such a design would align with the intrinsic activation structure of pretrained models, while avoiding the brittleness of static channel removal.

Motivated by this perspective, we design **MoRE** to explicitly exploit the non-uniform contribution structure of dense FFNs. Rather than forcing a hard partition of channels into disjoint experts, MoRE maintains a shared backbone and conditionally activates specialized channel subsets based on the properties of input tokens. From a combinatorial perspective, modeling activation patterns as a *sunflower*-like set system implies that the effective diversity of token-wise activation supports is far smaller than that of a general k-subset family. At a high level, MoRE proceeds in three stages: (1) identify globally shared backbone channels that are consistently important across inputs; (2) partition the remaining channels into expert-specific subspaces based on co-activation patterns; and (3) route each input token to the expert whose specialization best matches its representation, using a training-free similarity-based criterion. We next formalize each step.

C.2 Notation Reference.

For any natural number $m \geq 1$, we use $[m]$ to denote the set $\{1, \ldots, m\}$. We use bold lower case letters such as $\mathbf{x}, \mathbf{y}, \ldots$ to denote vectors and upper case letters such as $\mathbf{A}, \mathbf{B}, \ldots$ to denote matrices. For a vector $\mathbf{x} \in \mathbb{R}^m$, its i^{th} entry is denoted by x_i, and its norm (i.e. ℓ_2-norm) is $\|\mathbf{x}\|_2 = \left(\sum_{i=1}^m |x_i|^2\right)^{1/2}$. Similarly, for a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, its (i, j)-th entry is denoted by $M_{i,j}$ and its Frobenius norm is defined by $\|\mathbf{M}\|_F = \left(\sum_{i=1}^m \sum_{j=1}^n |M_{ij}|^2\right)^{1/2}$. If $\mathbf{x} \in \mathbb{R}^m$ and $S \subset [m]$, then $\mathbf{x}_S$ is the sub-vector of $\mathbf{x}$ indexed by S. Similarly, if $\mathbf{M} \in \mathbb{R}^{m \times n}$, $S \subset [m]$ and $T \subset [n]$, then $\mathbf{M}_{S,T}$ is the sub-matrix of $\mathbf{M}$ whose rows are indexed by S and columns indexed by T. When $S = T$, we simply write $\mathbf{M}_S$.

Sunflower. Let $\mathcal{E}$ be a set system over the universe $[m]$ (i.e. a collection of subsets $\mathcal{E} = \{E_1, \ldots, E_r\}$ with each $E_i \subseteq [m]$). $\mathcal{E}$ is called is a *sunflower* if there is a subset $C \subset [m]$ (called the *core* or *kernel* of the sunflower) such that, for each pair of distinct subsets $E_i, E_j \in \mathcal{E}$, we have $E_i \cap E_j = C$. That is, a set system $\mathcal{E}$ is a sunflower if all sets (called *petals* of the sunflower) share the same common subset of elements (namely the core C). We emphasize that MoRE does not assume the activation supports strictly form a sunflower; the definition is used only to describe the structure of recognized experts.

[4] In mathematics, a *sunflower* refers to a collection of sets that **share a common core while differing in their non-shared elements**. We use the term as a structural abstraction rather than a strict combinatorial assumption, to capture the coexistence of a shared backbone and input-dependent residuals (see Appendix C.2).

D Calibration Robustness and Effective Sample Size.

D.1 Empirical Effects of Calibration Set Size

This section examines the robustness of MoRE to calibration set size, focusing on whether reliable conditional structures can be extracted from a limited number of samples. All results are reported on LLaMA-3.1-8B under a fixed compression budget of approximately 20%, using the same general-domain calibration source.

Table 3: Effect of calibration size on MoRE under a fixed compression budget ($\sim$20%) on LLaMA-3.1-8B. All results use general-domain calibration without access to downstream tasks. Increasing calibration size beyond 100 samples does not yield consistent performance gains.

Method	**PPL-Avg** $\downarrow$	**Task-Avg** $\uparrow$
50 samples	10.78	70.71
100 samples	**10.70**	**71.38**
500 samples	12.33	68.70

We evaluate MoRE with calibration sizes of 50, 100, and 500 samples (with 256 tokens/sample). As shown in Table 3, performance improves substantially when increasing the calibration size from 50 to 100 samples, while further increasing the calibration size to 500 does not yield consistent gains. In fact, both perplexity and downstream task accuracy slightly degrade in several cases at 500 samples.

This empirical observation shows that MoRE does not benefit monotonically from larger calibration sets. But instead, MoRE relies on estimating coarse but stable structural signals—such as globally important channels and dominant co-activation patterns—rather than fitting fine-grained, task-specific statistics.

D.2 Effective Sample Size

Empirically, activation patterns concentrate on a much smaller subset of the combinatorial space, indicating that the number of *effectively* populated patterns is far smaller than $\binom{m}{k_{\text{act}}}$. We refer to this phenomenon as *channel concentration* in the following text.

To verify whether such channel concentration occurs in practice, we empirically analyze the distribution of activation patterns and measure the fraction of channels that repeatedly appear among the top-k_{act} activations across calibration samples, where $k_{\text{act}} = \alpha \cdot m$ for small ratios α.

Table 4: Empirical channel coverage under different effective top-K_{act} activation ratios. Results are reported on LLaMA-3.1-8B using a middle-layer MLP. Despite observing over 25k tokens, many intermediate channels are never selected among the top-k_{act} activations.

α	**Top-**K_{act}	**Total Channels** (m)	**Never Activated**	**Never-Activated Ratio**
1×10^{-3}	14	14,336	7,478	52.16%
5×10^{-4}	7	14,336	10,434	72.78%

Table 4 provides empirical evidence of strong channel concentration in practice. Even with more than 25k observed tokens, a substantial fraction of intermediate channels are never selected among the top-k_{act} activations, and the set of channels that are repeatedly activated remains small under different top-K_{act} ratios. This indicates that activation patterns occupy only a limited subset of the combinatorial space, and that the effective support of channel usage is far smaller than the nominal intermediate dimension m.

Consequently, the effective number of activation bins M_{eff} is much smaller than the worst-case upper bound $\binom{m}{k_{\text{act}}}$ assumed in the theoretical analysis. This observation provides an empirical case study suggesting that the effective sample size in practice can be substantially smaller than the worst-case bound considered in the previous robustness analysis.

E Extended Ablation Study & Analysis

E.1 Analysis and Ablation Studies

E.1.1 Transfer Behavior on Knowledge-Intensive QA

We evaluate transfer robustness on the MMLU benchmark under a 5-shot setting, using LLaMA-3.1-8B-Instruct under the same compute budget. This experiment probes whether efficiency gains obtained on language modeling tasks translate to knowledge-intensive and distribution-shifted scenarios.

Table 5: Transfer performance on MMLU under a fixed compute reduction (PR $\approx 15\%$) using LLaMA-3.1-8B-Instruct.

Method	STEM	Medical	Humanities	Subject Avg.
Dense	46.89	68.70	69.97	61.85
ShortGPT	20.94	46.06	54.32	40.44
Ours	**46.05**	**66.86**	**68.33**	**60.41**

As shown in Table 14, MoRE maintains stable performance across STEM, medical, and humanities subjects, with only a small drop relative to the dense model. In contrast, ShortGPT exhibits consistent degradation across all subject groups, with particularly pronounced losses on STEM categories. The gap between MoRE and ShortGPT is uniform across subjects, indicating a systematic difference in transfer robustness rather than task-specific effects.

E.1.2 Understanding Recognized Experts: Backbone vs. Residual Specialization.

We ablate MoRE to isolate the contributions of backbone channels, residual capacity, and routing alignment to conditional computation in pretrained MLP layers. All variants are evaluated on LLaMA-3.1-8B under a fixed compression budget of approximately 20% (same setting as Section 3.2) and share identical expert partitions and parameter budgets; they differ only in channel activation and routing.

Backbone-Only Activation. The *MoRE-Backbone* variant activates only globally important channels while disabling all residual channels. As shown in Table 6, the model remains functional but exhibits substantial degradation in both perplexity and downstream accuracy, suggesting that shared backbone channels alone cannot support input-specific variation.

Table 6: Abbreviated result of ablation study on backbone and residual specialization under identical compression budgets. All MoRE variants share the same expert structure and parameter budget, and differ only in which channel subsets are activated and how inputs are routed.

Method	PPL-Avg ↓	Task-Avg ↑
MoRE-Backbone	17.16	61.68
MoRE-Random	16.02	63.04
MoRE-Full	**10.70**	**71.38**

Residual Capacity without Alignment. The *MoRE-Random* variant activates residual channels but assigns tokens to experts using random routing. Compared to Backbone-only activation, performance improves across all metrics, indicating that residual channels provide additional capacity; however, the gains remain limited and fall short of *MoRE-Full*.

Aligned Conditional Specialization. The *MoRE-Full* variant activates both backbone and residual channels and routes tokens using training-free, representation-derived routing. MoRE-Full substantially outperforms both ablations, recovering a substantial fraction of the dense model's performance under the same compute budget without modifying pretrained weights or introducing finetuning. Taken together, these results isolate routing alignment as the key factor that enables residual capacity to translate into effective input-dependent specialization.

E.2 Data- & Model-Dependent Activation Pattern.

To quantify differences in activation patterns across layers and models, we measure representation similarity using Centered Kernel Alignment (CKA) (Kornblith et al., 2019). CKA is a normalized

similarity metric derived from the Hilbert–Schmidt Independence Criterion (HSIC) and is invariant to isotropic scaling and orthogonal transformations of representations. In this work, we adopt the RBF-kernel variant of CKA to capture nonlinear similarities between activation patterns, where lower CKA values indicate larger representational shifts.

Formally, given two activation matrices X and Y, CKA is defined as

$$\mathrm{CKA}(X, Y) = \frac{\left\langle \tilde{K}_X,\ \tilde{K}_Y \right\rangle_F}{\sqrt{\left\langle \tilde{K}_X,\ \tilde{K}_X \right\rangle_F \left\langle \tilde{K}_Y,\ \tilde{K}_Y \right\rangle_F}}, \tag{14}$$

where $\tilde{K}_X = HK_XH$ and $\tilde{K}_Y = HK_YH$ are the centered kernel matrices, with $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$. The RBF kernel matrices are given by

$$(K_X)_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \quad (K_Y)_{ij} = \exp\left(-\frac{\|y_i - y_j\|^2}{2\sigma^2}\right). \tag{15}$$

To better understand how activation patterns vary across depth, model families, and data distributions, we conduct a series of representation similarity analyses using CKA. We progressively examine (i) depth-wise differences within the same model, (ii) cross-model differences at aligned depths, and (iii) data-induced shifts under different input distributions.

We first analyze depth-wise activation pattern differences within each model. Table 7 reports the CKA similarity between shallow, middle, and deep layers for three representative models.

Table 7: Activation pattern differences across depths for **LLaMA-3.1-8B**, **LLaMA-3.1-8B-Instruct**, and **Qwen-2.5-7B-Instruct**. We compare the activation pattern of shallow, middle, and deep layers using representation similarity (CKA). "Depth A – Depth B" denotes the similarity between the corresponding layers of the two depths within a model. Lower CKA indicates a larger activation pattern shift.

Model	Shallow – Middle	Middle – Deep	Shallow – Deep
LLaMA-3.1-8B	0.8022	0.8223	**0.6523**
LLaMA-3.1-8B-Instruct	0.8071	0.8120	**0.6455**
Qwen-2.5-7B-Instruct	**0.6538**	0.8076	0.6948

Across the three tables, we observe structured and non-uniform shifts in activation patterns. Within the LLaMA family (Table 7), representations diverge more as layers are farther apart, with the largest shift occurring between shallow and deep layers. In contrast, Qwen exhibits a pronounced transition from shallow to middle depth, while middle and deep layers remain relatively similar, suggesting a depth-wise "phase transition" rather than a gradual drift.

We next compare activation patterns across different model families at aligned depths. This analysis reveals how architectural choices and instruction tuning affect representations independently of depth.

Table 8: Cross-model activation pattern similarity at aligned depths measured by CKA. "Model A – Model B" denotes the similarity between the corresponding layers of the two models. Lower values indicate larger representational divergence.

Depth	Llama Base – Llama Instruct	Llama Base – Qwen Instruct	Llama Instruct – Qwen Instruct
Shallow	0.9741	0.3940	0.3818
Middle	0.9751	0.5933	0.5801
Deep	0.9814	0.6123	0.6064

Cross-model comparisons at aligned depths (Table 8) further show that models from different families (LLaMA vs. Qwen) can differ substantially at the same depth. Interestingly, this divergence becomes smaller at deeper layers, and the largest change again occurs from shallow to middle depth, consistent with the within-model pattern observed for Qwen. One plausible explanation for this depth-dependent convergence is that shallow layers are more tightly coupled to model-specific design choices, such as

tokenization, embedding parameterization, and early feature mixing, which vary substantially across model families.

As a result, representations at shallow depths tend to be highly architecture-dependent. In contrast, middle-to-deep layers may increasingly operate in more abstract, task-conditioned subspaces that support similar functional roles across different models, such as reasoning, aggregation, and decision-oriented transformations. This could partially explain why cross-model representational differences diminish with depth, and why the most pronounced structural transition consistently occurs between shallow and middle layers.

Finally, we study how activation patterns shift under different data distributions. Specifically, we compare representations induced by a general-domain dataset (Pile) and a task-oriented dataset (WinoGrande).

Table 9: Data-induced activation pattern shifts across depths. For each model, we measure the CKA similarity between activations obtained from a general-domain corpus (Pile) and a task-specific dataset (WinoGrande). Lower CKA values indicate stronger data-dependent specialization of activation patterns.

Depth	LLaMA-3.1-8B	LLaMA-3.1-8B-Instruct	Qwen-2.5-7B-Instruct
Shallow	0.1984	0.2126	0.2305
Middle	0.2224	0.2301	0.2671
Deep	0.1583	0.1637	0.2502

Table 9 highlights strong data dependence: even under the same model-layer setting, changing the input distribution (Pile vs. WinoGrande) induces a large representational shift. This case study suggests that activation patterns are highly input- and task-dependent, motivating conditional computation strategies that allocate activated capacity based on the observed calibration statistics rather than relying on a fixed, globally determined partition.

E.3 Naive Expert Partitioning.

We further compare MoRE with a naive clustering-based expert partitioning strategy that enforces disjoint experts.

Table 10: Comparison between **MoRE-Full** and a naive clustering-based expert partitioning strategy. The clustering baseline directly partitions neurons into disjoint experts based on activation similarity, with additional heuristics to merge small experts and constrain oversized ones. Results indicate that naive hard partitioning leads to inferior accuracy–efficiency trade-offs under the same activated parameter budget.

Method	PR (%)	PPL Avg. ↓	Task Avg. ↑
Dense	-	8.41	76.13
MoRE-Full (Ours)	20	10.70	71.38
MoRE-Clustering (Naive)	20	40.30	54.06

Despite using the same activated parameter budget, this baseline suffers from severe performance degradation (Table 10), suggesting that hard neuron partitioning is insufficient for preserving model capacity.

Table 11: Full result of ablation study on backbone and residual specialization under identical compression budgets. All MoRE variants share the same expert structure and parameter budget, and differ only in which channel subsets are activated and how inputs are routed.

Method	PPL-Wiki ↓	PPL-PTB ↓	PPL-Avg ↓	Wino ↑	ARC-E ↑	ARC-C ↑	SciQ ↑	QA-Avg ↑
MoRE-Backbone	14.40	19.92	17.16	58.72	61.95	32.85	93.20	61.68
MoRE-Random	13.34	18.69	16.02	61.01	63.26	34.39	93.50	63.04
MoRE-Full	**8.41**	**12.99**	**10.70**	**67.72**	**76.39**	**45.90**	**95.50**	**71.38**

E.4 Controlling Compression Budgets.

Layer Selection and Budget Allocation. Unlike pruning methods that enforce a fixed sparsity ratio through permanent parameter removal, MoRE realizes compression budgets through conditional activation.

As a result, the "pruning ratio (PR)" is determined by how capacity is dynamically allocated across selected layers and experts, rather than by a single global knob. This behavior is intrinsic to conditional computation and reflects a design trade-off between flexibility and budget realization

In practice, the realized budget depends on which layers are converted into MoRE layers. We primarily apply MoRE to middle-to-deep layers, motivated by evidence (Gromov et al., 2025) that deeper Transformer layers are often more redundant and exhibit higher similarity to neighboring layers, such that removing a substantial fraction of deep layers can cause surprisingly small degradation on common QA benchmarks.

For completeness, we additionally evaluate two alternative strategies: (i) applying MoRE to early-to-middle layers, and (ii) uniformly distributing MoRE layers across depth. These variants serve as ablations to illustrate how layer choice influences budget allocation, rather than as optimized configurations.

Table 12: Layer selection ablation under a fixed compute reduction. We compare different layer selection strategies for applying MoRE, including uniform selection across layers, early–middle layers, and middle–deep layers. Results are reported on language modeling (perplexity) and non-iterative QA benchmarks. **Bold** denotes the best performance in each column, and red denotes the second-best result.

Method	PPL-Wiki ↓	PPL-PTB ↓	PPL-Avg ↓	Wino ↑	ARC-E ↑	ARC-C ↑	SciQ ↑	QA-Avg ↑
Uniform	8.60	13.46	11.03	67.32	75.17	45.31	95.40	70.80
Early–Middle	9.58	14.42	12.00	65.19	74.96	43.00	95.00	69.54
Middle–Deep	**8.41**	**12.99**	**10.70**	**67.72**	**76.39**	**45.90**	**95.50**	**71.38**

E.5 Effect of the Number of Experts.

Expert Counts in MoRE. Table 13 studies the impact of expert count in MoRE under a fixed compression budget, where all other settings are kept unchanged. The default setting of MoRE on LLaMA-3.1-8B-Instruct is $E = 4$. Across all settings, MoRE remains consistently competitive in both perplexity and downstream QA metrics, while different expert granularities lead to modest performance variations.

Table 13: Effect of expert count in **MoRE**: We compare different numbers of experts while keeping other settings unchanged.

Method	PPL-Wiki ↓	PPL-PTB ↓	PPL-Avg ↓	Wino ↑	ARC-E ↑	ARC-C ↑	SciQ ↑	QA-Avg ↑
MoRE (E=2)	7.97	12.43	10.20	69.53	78.20	49.83	95.70	73.32
MoRE (E=4)	8.41	12.99	10.70	67.72	76.39	45.90	95.50	71.38
MoRE (E=8)	8.95	13.71	11.33	66.06	74.45	44.71	95.10	70.08

In MoRE, expert capacity is bounded by a maximum size to satisfy the target compression budget. In this work, we adopt a fixed maximum and minimum expert capacity range across all experiments to ensure fair and consistent budget realization. We leave adaptive or learned expert capacity control to future work.

F Extended Experiments.

F.1 Scope of Evaluation.

Our evaluation focuses on training-free conditional compression at the algorithmic level. In particular, MoRE is applied to feed-forward (FFN) modules in Transformer-based language models (compared to pruning, where both the attention layer and FFN are prunable modules), where conditional computation is most naturally supported. As a result, the achievable compression ratio is inherently constrained by the proportion of FFN parameters in the overall model.

We emphasize that our goal is not to maximize compression at all costs, but to evaluate a perpendicular compression paradigm under compression ratios comparable to structured pruning baselines. This choice ensures fair comparison in terms of granularity and avoids instability introduced by overly aggressive compression.

Overall, we focuses on *algorithm-level efficiency* rather than system-level deployment optimization. We do not incorporate runtime scheduling, kernel fusion, or specialized sparse execution engines. Nevertheless, extensive prior work on sparse MoE acceleration and deployment suggests that reductions in activated

computation can translate into practical speedups under appropriate system support (Tan et al., 2024; Guo et al., 2025b).

F.2 Compression Estimation under Conditional Computation

Estimating Effective Parameter Utilization. We quantify compression in MoRE using the notion of *activated parameter ratio*, which measures the expected fraction of parameters involved in computation at inference time. We consider all linear projections in Transformer MLP and attention modules as prunable parameters for comparison purposes, including the up-, down-, and gate-projection matrices in MLPs, as well as the Q, K, V, and output projections in attention layers. Let P_{attn} and P_{mlp} denote the numbers of parameters in attention and MLP projections, respectively, and let $P_{\text{total}} = P_{\text{attn}} + P_{\text{mlp}}$.

In this work, MoRE sparsifies only MLP modules, while keeping all attention projections unchanged. Under top-1 routing, each input activates all attention parameters and only a subset of MLP parameters. We therefore define the effective activated parameter ratio (APR) of MoRE as

$$\text{APR}_{\text{total}}^{\text{MoRE}} = \frac{P_{\text{attn}} + \mathbb{E}[P_{\text{act,mlp}}]}{P_{\text{total}}}, \tag{16}$$

and thus **PR (pruning rate we used throughout the experiment table)** is defined as:

$$\text{PR} = 1 - \text{APR} \tag{17}$$

where $\mathbb{E}[P_{\text{act,mlp}}]$ denotes the expected number of activated MLP parameters per input. This formulation enables a direct comparison with structured pruning methods under a unified notion of expected computation budget.

Discussion on Uniform Routing Assumption. To estimate $\mathbb{E}[P_{\text{act,mlp}}]$, we assume a uniform routing distribution across experts. We emphasize that this assumption is not intended to model the exact runtime routing behavior of MoRE, which is inherently input-dependent and non-uniform. Instead, it serves as a simple and reproducible proxy for estimating the expected MLP activation ratio under top-1 routing.

Importantly, matching the same total activated parameter ratio as pruning baselines is conservative for MoRE. While structured pruning methods typically reduce parameters in both MLP and attention modules, MoRE keeps all attention parameters fixed and can only reduce computation through MLP sparsification. As a result, achieving the same total APR requires a proportionally stronger reduction in MLP parameters for MoRE. Our comparison, therefore, reflects a conservative setting that favors pruning-based baselines.

We stress that our goal is not to capture exact per-input computation, but to compare methods under the same expected activation budget. This perspective is standard for conditional computation methods and allows us to assess effective model capacity at deployment time in a consistent manner.

A Case Study of MoRE Routing Behavior. To characterize the routing behavior of MoRE, we quantify the deviation between the empirical expert-routing distribution and a uniform reference distribution using the Jensen–Shannon divergence (JSD). Given an empirical routing distribution $p = (p_1, \ldots, p_E)$ over E experts at a layer, and a uniform distribution $u = (\frac{1}{E}, \ldots, \frac{1}{E})$, the JSD is defined as

$$\text{JSD}(p \parallel u) = \tfrac{1}{2}\text{KL}(p \parallel m) + \tfrac{1}{2}\text{KL}(u \parallel m), \quad m = \tfrac{1}{2}(p + u), \tag{18}$$

where $\text{KL}(\cdot \parallel \cdot)$ denotes the Kullback–Leibler divergence. We report JSD using a base-2 logarithm, in which case $\text{JSD} \in [0, 1]$.

Intuitively, JSD measures how far the routing distribution deviates from uniform expert utilization: a value of 0 corresponds to perfectly uniform routing, while larger values indicate increasingly skewed expert selection.

Figure 2 shows the layer-wise JSD of routing distributions in MoRE. The observed variation across layers indicates that routing behavior is depth-dependent rather than uniformly balanced throughout the network.

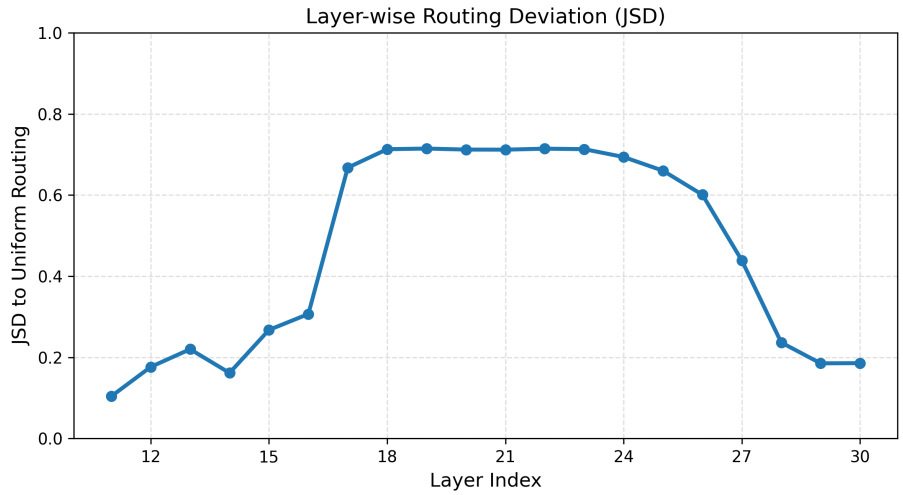

Figure 2: Layer-wise Jensen–Shannon divergence (JSD) between the empirical routing distribution of MoRE and a uniform reference distribution. Lower divergence in early layers and higher divergence in deeper layers indicate structured, layer-dependent expert specialization, rather than global routing collapse or static computation imbalance.

Discussion on Compression Estimation and Evaluation Protocol. A central question raised by conditional compression methods such as MoRE is whether their compression estimation and evaluation protocol are comparable to those of static pruning approaches. Unlike pruning, which permanently removes parameters and induces a fixed computation graph, MoRE preserves all pretrained weights and reduces computation through input-dependent activation. As a result, exact per-input computation is inherently non-uniform and cannot be characterized by a single deterministic parameter count.

To enable a meaningful comparison, we adopt an expected parameter utilization metric under a uniform routing assumption, which serves as a stable proxy for the effective capacity activated at deployment time. This choice does not aim to precisely model runtime routing behavior, but rather to align MoRE with the evaluation paradigm commonly used by static compression methods, where a fixed computation budget is assumed across inputs. Importantly, we treat this estimate as a normalization mechanism for comparison, not as a claim of exact computation equivalence.

Complementary to this estimation, we empirically analyze the actual routing behavior using layer-wise distributional metrics, showing that routing skew varies across depth but remains structured rather than pathological. Taken together, these analyses suggest that while conditional compression fundamentally differs from static pruning in its computation semantics, evaluating methods under a shared expected budget provides a reasonable and reproducible basis for comparing accuracy–efficiency trade-offs.

F.3 Full Experiment Tables.

Table 14: Transfer performance on MMLU under a fixed compute reduction (PR $\approx 15\%$) using LLaMA-3.1-8B-Instruct.

Method	STEM	Medical	Humanities	Subject Avg.
Dense	46.89	68.70	69.97	61.85
ShortGPT	20.94	46.06	54.32	40.44
Ours	**46.05**	**66.86**	**68.33**	**60.41**

Table 15: Comprehensive results across multiple model families under matched compute budgets (PR $\approx$ 20%, and $\approx$ 10% for LLaMA-3.2-3B). We report perplexity on language modeling benchmarks and accuracy on non-iterative QA tasks. Bold numbers indicate the best performance within each column other than the dense model.

Method	PPL-Wiki ↓	PPL-PTB ↓	PPL-Avg ↓	Wino ↑	ARC-E ↑	ARC-C ↑	SciQ ↑	QA-Avg ↑
LLaMA2-7B								
Dense	5.47	23.10	14.29	67.25	73.78	45.05	91.90	69.50
LLMPruner	9.14	43.22	26.18	55.49	66.46	32.42	80.70	58.77
ReplaceMe	10.96	33.25	22.11	58.01	54.55	31.66	83.10	56.83
UIDL	15.47	48.10	31.78	57.46	52.82	32.51	80.30	55.77
ShortGPT	12.18	37.77	24.97	57.54	54.08	31.74	82.60	56.49
FLAP	7.64	33.32	20.48	**61.88**	63.05	32.08	**91.60**	62.15
Ours	**6.27**	**22.87**	**14.55**	60.93	**67.42**	**38.40**	91.30	**64.51**
LLaMA2-13B								
Dense	4.88	28.23	16.56	70.48	76.52	48.81	93.30	72.28
LLMPruner	7.34	63.64	35.49	58.41	**72.47**	39.08	88.10	64.52
ReplaceMe	7.39	45.73	26.56	63.22	63.05	38.23	88.60	63.28
UIDL	8.30	44.69	26.49	64.48	62.33	38.40	88.50	63.43
ShortGPT	8.30	44.69	26.49	64.17	62.50	38.14	88.50	63.33
FLAP	6.52	45.56	26.04	64.40	67.09	35.41	91.40	64.58
Ours	**5.29**	**36.41**	**20.85**	**66.06**	71.97	**42.49**	**93.00**	**68.38**
LLaMA3.1-8B								
Dense	6.24	10.58	8.41	71.43	82.53	54.95	95.60	76.13
LLMPruner	16.12	26.27	21.19	56.99	61.36	28.84	93.60	60.20
ReplaceMe	20.82	32.05	26.43	63.30	66.79	42.75	87.20	65.01
UIDL	45.95	45.27	45.61	62.90	64.14	40.02	88.60	63.92
ShortGPT	45.95	45.27	45.61	63.14	63.97	40.02	88.60	63.93
Ours	**8.41**	**12.99**	**10.70**	**67.72**	**76.39**	**45.90**	**95.50**	**71.38**
LLaMA3.2-3B								
Dense	7.81	12.64	10.23	68.43	72.10	46.33	93.40	70.07
ReplaceMe	13.09	21.06	17.08	63.85	66.16	37.54	89.80	64.34
ShortGPT	13.39	19.52	16.46	62.98	69.07	38.23	93.50	65.95
Ours	**9.38**	**14.73**	**12.06**	**66.14**	**71.04**	**40.44**	**94.70**	**68.08**
Qwen2.5-7B								
Dense	7.81	13.42	10.62	68.43	72.10	46.33	93.40	70.07
UIDL	13.24	25.00	19.12	53.67	63.59	32.94	84.10	58.58
ShortGPT	12.04	25.46	19.38	52.33	62.50	31.74	82.60	57.79
Ours	**10.49**	**18.01**	**14.25**	**59.67**	**64.65**	**40.02**	**86.30**	**62.66**