EXPERTWEAVER: UNLOCKING THE INHERENT MOE IN DENSE LLMS WITH GLU ACTIVATION PATTERNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) effectively scales model capacity while preserving computational efficiency through sparse expert activation. However, training high-quality MoEs from scratch is prohibitively expensive. A promising alternative is to convert pretrained dense models into sparse MoEs. Existing dense-to-MoE methods fall into two categories: **dynamic structural pruning** that converts dense models into MoE architectures with moderate sparsity to balance performance and inference efficiency, and **downcycling** approaches that use pretrained dense models to initialize highly sparse MoE architectures. However, existing methods break the intrinsic activation patterns within dense models, leading to suboptimal expert construction. In this work, we argue that the Gated Linear Unit (GLU) mechanism provides a natural blueprint for dense-to-MoE conversion. We show that the fine-grained neural-wise activation patterns of GLU reveal a coarse-grained structure, uncovering an inherent MoE architecture composed of consistently activated universal neurons and dynamically activated specialized neurons. Leveraging this discovery, we introduce ExpertWeaver, a training-free framework that partitions neurons according to their activation patterns and constructs shared experts and specialized routed experts with layer-adaptive configurations. Our experiments demonstrate that ExpertWeaver significantly outperforms existing methods, both as a training-free dynamic structural pruning technique and as a downcycling strategy for superior MoE initialization.

1 INTRODUCTION

The Mixture-of-Experts (MoE) architecture (Cai et al., 2025) effectively expands model capacity while maintaining efficiency by activating only sparse subsets of experts. However, training MoE models from scratch is rather expensive, driving growing interest in converting pretrained dense models into sparse MoE architectures. These efforts can be broadly categorized into two distinct lines of research based on target sparsity and objectives. The first one is *Dynamic Structural Pruning*, which converts dense models into MoE architectures with moderate sparsity (e.g., 25%) to balance performance and efficiency (He et al., 2024b; Pei et al., 2025; Nishu et al., 2025; Zheng et al., 2024). The second category is *Downcycling* (versus *upcycling* that expands models (He et al., 2024a; Komatsuzaki et al., 2022)), which initializes highly sparse MoEs (75%+ sparsity) by splitting pretrained dense MLPs into smaller experts, avoiding the prohibitive costs of training sparse MoEs from scratch (Zhang et al., 2021; Zhu et al., 2024b; Qu et al., 2024). However, existing methods break the inherent activation patterns, necessitating additional router training, and ignore layer-specific differences, leading to suboptimal expert construction. The detailed related work is shown in Appendix B.

In this work, we argue that the key to overcoming these limitations lies in the GLU mechanism (Dauphin et al., 2017; Shazeer, 2020), which provides a natural blueprint for dense-to-MoE conversion by revealing the model's intrinsic functional structure. The GLU mechanism employs additional gating weights to dynamically control neuron activations based on the input context, providing meaningful indicators of each neuron's role and importance for effective MoE expert construction. Our analysis of the GLU mechanism in §2.2 yields key insights that serve as the cornerstone of our method. Firstly, GLU gating signals naturally induce token-level activation sparsity (**Takeaway 1**). Moreover, these patterns reveal both universal neurons ideal for a shared expert and task-specific clusters suitable for routed experts (**Takeaways 2 & 3**). Additionally, by quantifying neuron spe-

cialization with the *Coefficient of Variation (CV)*, we find that the distribution of universal versus specialized neurons varies systematically across layers, indicating that different layers require different expert configurations rather than a one-size-fits-all approach (**Takeaway 4**). Together, these observations provide a complete blueprint for constructing a coarse-grained, expert-level MoE architecture from fine-grained, neural-level GLU activations.

Based on these findings, we propose **ExpertWeaver**, a novel, training-free method that effectively converts a dense model into MoE guided by GLU activation patterns. The process consists of three key stages. a) Firstly, the activation pattern of all neurons is captured by recording their GLU activations from a multi-task calibration dataset. b) Then, the CVs of these activations are calculated to perform a layer-aware configuration, determining the precise size of the shared expert and the pool of routed experts for each layer. c) Finally, neurons are partitioned according to layer-wise configuration by grouping the most consistently active neurons into a single *shared expert*, while remaining specialized neurons are clustered into *routed experts* based on their activation patterns. The MoE router is also constructed in a training-free manner from the GLU gating centroids. This entire process successfully "weaves" the neurons of a dense model into multiple experts, constructing a well-structured MoE architecture.

We conduct extensive experiments to validate the effectiveness of ExpertWeaver across two categories of methods. As a training-free dynamic structural pruning method, ExpertWeaver significantly outperforms existing structural pruning baselines on multiple benchmarks. Furthermore, when applied as a downcycling strategy, our method successfully initializes a highly sparse MoE model that, with limited continued pretraining, achieves superior performance compared to both established MoE and dense baselines with comparable parameters and training budgets. The main contributions of this paper are as follows:

- We systematically analyze GLU activation patterns in dense LLMs and find that their fine-grained activation patterns exhibit structural properties that provide a natural blueprint for constructing MoEs with coarse-grained activation.
- We propose ExpertWeaver, a novel training-free framework that leverages GLU activation patterns to convert dense models into MoE architectures.
- Comprehensive experiments demonstrate that ExpertWeaver significantly outperforms existing methods in both structural pruning and MoE downcycling across diverse downstream tasks.

2 PRELIMINARIES

2.1 BACKGROUND

Gated Linear Unit. Most current high-performance LLMs have adopted the GLU architecture, with SwiGLU being the most commonly used variant (Shazeer, 2020). A dense GLU layer processes an input x using three distinct weight matrices: $\boldsymbol{W}_{\text{gate}}$, $\boldsymbol{W}_{\text{up}}$, and $\boldsymbol{W}_{\text{down}}$. The entire computation can be expressed as:

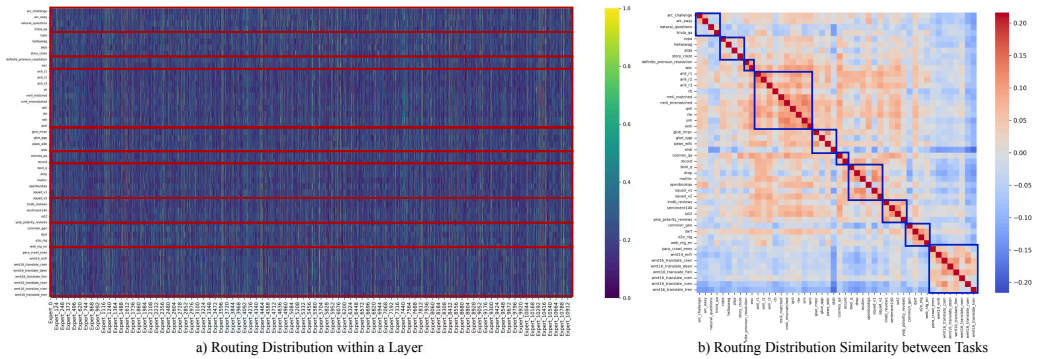
$$\boldsymbol{y} = (\text{Swish}(\boldsymbol{x}\boldsymbol{W}_{\text{gate}}) \odot (\boldsymbol{x}\boldsymbol{W}_{\text{up}}))\, \boldsymbol{W}_{\text{down}}.$$

The element-wise product $\odot$ combines the input projection with a dynamic gate that modulates neuron activation strength for each token. This gating mechanism provides dense GLU models with intrinsic dynamic sparsity, which we identify as the key to efficiently converting pretrained dense models into structured sparse MoEs.

Mixture-of-Experts. The MoE architecture replaces dense Feed-Forward Networks (FFNs) in LLMs with multiple smaller expert networks and a gating mechanism for sparse activation. Specifically, an MoE layer consists of N experts $(\boldsymbol{E}_1, \ldots, \boldsymbol{E}_N)$ and a router network $\boldsymbol{G}$ that dynamically selects which experts to activate for each input token:

$$\boldsymbol{y} = \sum_{i \in \text{TopK}(\boldsymbol{G}(\boldsymbol{x}))} \boldsymbol{g}(\boldsymbol{x})_i \boldsymbol{E}_i(\boldsymbol{x}); \quad \boldsymbol{E}_i(\boldsymbol{x}) = \left(\text{Swish}(\boldsymbol{x}\boldsymbol{W}^{(i)}_{\text{gate}}) \odot (\boldsymbol{x}\boldsymbol{W}^{(i)}_{\text{up}})\right) \boldsymbol{W}^{(i)}_{\text{down}}, \tag{1}$$

where $\boldsymbol{g}(x)_i$ represents the normalized gating weight for expert i, and each expert has its own parameters. By activating only a subset of experts per token, MoE maintains model capacity while significantly reducing computational costs.



a) Routing Distribution within a Layer

b) Routing Distribution Similarity between Tasks

Figure 1: **Neuron activation patterns across diverse tasks.** We visualize the middle layer's activation patterns from Qwen2.5-7B on a subset of Flan-v2. a) Activation distribution of neurons across different tasks. b) Activation distribution of neurons within individual task clusters, where tasks belonging to the same cluster are enclosed in boxes.

2.2 Motivation

As shown in §2.1, GLU and MoE represent two distinct granularities of activation patterns. GLU implements fine-grained activation at the neuron level through a dynamic gate that modulates neuron activation strength for each token. In contrast, MoE employs coarse-grained activation by routing entire tokens to selected experts, establishing macroscopic, structural activation at the expert level. This difference between neural-level and expert-level activations raises an intriguing question: *can the fine-grained neuron-level activation patterns of GLU be aggregated to inform the construction of a coarse-grained, expert-level activated MoE architecture?* Our exploratory analysis of the GLU activation patterns in pretrained dense LLMs yields four key insights that form the cornerstone of **ExpertWeaver**.

2.2.1 GLU as a Natural Sparsity Signal

As established in §2.1, the gating mechanism in a GLU, $\text{Swish}(\boldsymbol{x}\boldsymbol{W}_{\text{gate}})$, acts as a dynamic, data-dependent filter. To investigate the feasibility of leveraging inherent signals to directly enforce structured sparsity, we introduce AbsTopk-GLU, a simple modification that explicitly introduces sparsity into FFNs based on their gating activation scores:

$$\text{AbsTopk-GLU}(\boldsymbol{x}) = (\text{AbsTopK}(\text{Swish}(\boldsymbol{x}\boldsymbol{W}_{\text{gate}}), k)) \odot (\boldsymbol{x}\boldsymbol{W}_{\text{up}}). \tag{2}$$

Here, $\text{AbsTopK}(\boldsymbol{v}, k)$ preserves the top k values in $|\boldsymbol{v}|$ while zeroing out the rest, effectively retaining only the most activated neurons for each token.

Takeaway 1: The GLU's gating mechanism naturally induces structured sparsity within dense LLMs. As shown in Table 1, *AbsTopK-GLU*, which activates only 50% of neurons, consistently and significantly surpasses established unstructured pruning methods such as Wanda (Sun et al., 2023) and SparseGPT (Frantar & Alistarh, 2023). This performance highlights the inherent structural sparsity of dense models and validates the use of GLU gate scores as a reliable mechanism for controlling activation patterns. However, this fine-grained expert partitioning, where each neuron is treated as an individual expert, is impractical for hardware implementation. While it reduces theoretical FLOPs, the overhead of TopK selection and scattered memory access prevents real-world applications. Nonetheless, this experiment reveals the crucial insight that the GLU gating signal's control over neuron-level activation can be extended to manage coarse-grained experts, motivating our approach of grouping neurons into larger, hardware-friendly blocks.

Table 1: **Performance of AbsTopk-GLU vs. unstructured pruning at 50% sparsity.** (Numbers in parentheses denote the number of shots for evaluation; no annotation means zero-shot.)

Method	MMLU(5)	HellaSwag(10)	ARC-e	ARC-C(25)	PiQA	Avg.
LLaMA3-8B, 50% sparsity						
Dense	65.3	82.1	77.9	57.9	80.8	72.8
Wanda	55.8	75.0	72.0	51.3	77.3	66.3
SparseGPT	57.4	75.5	72.0	50.3	78.1	66.7
Magnitude	48.3	41.9	53.3	38.7	65.8	49.6
AbsTopk-GLU	**58.8**	**79.4**	**72.4**	**51.6**	**78.6**	**68.2**
Qwen2.5-7B, 50% sparsity						
Dense	74.2	80.3	77.8	63.8	80.0	75.2
Wanda	69.2	74.1	75.5	55.5	78.1	70.5
SparseGPT	69.8	75.9	74.7	56.7	78.3	71.1
Magnitude	64.2	49.5	46.3	33.6	63.8	51.5
AbsTopk-GLU	**70.8**	**78.6**	**76.6**	**59.7**	**78.9**	**72.9**

2.2.2 IDENTIFYING UNIVERSAL AND SPECIALIZED NEURONS VIA GLU ACTIVATION PATTERNS

To investigate the structure of GLU activation patterns, we analyzed activation patterns recorded for 5 few-shot samples per task in the Flan-v2 collection (48 tasks, 10 task clusters; details in Appendix F). The results in Figure 1 clearly illustrate two distinct and complementary phenomena, which inform the following observations.

Takeaway 2: There exists a core set of universally important neurons that are consistently activated across different tasks. As shown in Figure 1a), we observed that a consistent subset of neurons showed high activation regardless of the task domain. We hypothesize that these neurons encode task-agnostic knowledge. This observation motivates the design of MoE architectures that incorporate a *shared expert* specifically dedicated to capturing and leveraging this task-agnostic knowledge, thereby enhancing both parameter efficiency and overall performance.

Takeaway 3: Specialized neurons exhibit task-specific co-activation patterns. As illustrated in Figure 1b), the activation pattern similarity heatmap, generated from truncated and normalized neuron activations, reveals clear block-diagonal structures, where semantically related tasks show high similarity in specialized neuron activation patterns. This demonstrates that neurons naturally form co-activation clusters organized by task semantics, providing a principled approach for constructing *routed experts* through clustering based on multi-task activation patterns.

Takeaways 2&3 collectively provide a blueprint for dense-to-MoE conversion by forming shared experts from universal neurons and constructing routed experts through clustering specialized neurons based on their co-activation patterns.

2.2.3 LAYER-AWARE EXPERT ALLOCATION

Building on our discovery of universal and specialized neurons in §2.2.2, we next investigate how the ratio of universal to specialized neurons varies across layers. To quantify this layer-wise behavior, we measure each neuron's activation consistency using the Coefficient of Variation (CV), which is defined as the ratio of the standard deviation to the mean (Abdi, 2010):

$$\mathrm{CV}(a_j) = \frac{\sigma_j}{\mu_j + \epsilon}, \tag{3}$$

where $\mu_j = \mathbb{E}_{t\in\mathcal{T}}[\bar{a}_{j,t}]$ and $\sigma_j = \mathrm{STD}_{t\in\mathcal{T}}[\bar{a}_{j,t}]$ are the mean and standard deviation of the neuron's average absolute activation, $\bar{a}_{j,t}$, across the tasks in our calibration set $\mathcal{D}_{\text{calib}}$. A low CV indicates a *universal neuron* with consistent activation, while a high CV points to a *specialized neuron* that activates selectively.

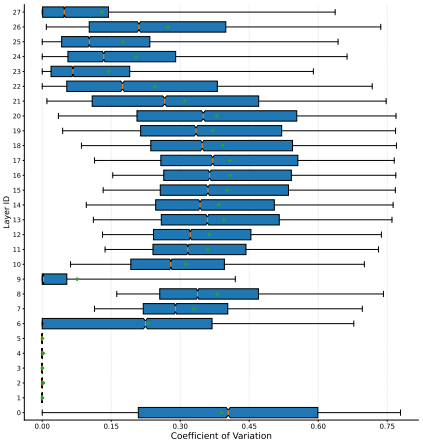


Figure 2: **Neuron Coefficient of Variation Across Layers**.

Takeaway 4: Layers exhibit different levels of neuron specialization. Figure 2 shows that boundary layers (shallow and deep) consistently have low CV scores, indicating universal neuron behavior, while middle layers exhibit diverse CVs with many highly specialized neurons. This layer-wise heterogeneity requires layer-specific expert configurations rather than uniform MoE conversion.

3 METHODOLOGY

This section introduces **ExpertWeaver**, a training-free framework that converts dense LLMs into efficient MoE architectures through a three-stage process: a) Capturing multi-task GLU activation patterns for each layer; b) Leveraging the CVs of these activations to determine the layer-specific ratio of shared to routed experts; c) Constructing shared experts from universal neurons, clustering specialized neurons into routed experts, and building the MoE router.

3.1 Capturing Multi-Task Gating Activation Patterns

A standard SwiGLU-based FFN layer (Shazeer, 2020) consists of three weight matrices: $W_{\text{gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$, $W_{\text{up}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$, and $W_{\text{down}} \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}$. Here, d_{model} is the model's hidden dimension and d_{ffn} is the dimension of the intermediate layer. Layer indices are omitted for notational simplicity. We define a **neuron slice** $\boldsymbol{s}_j$ for each neuron $j \in \{1, \ldots, d_{\text{ffn}}\}$ as:

$$\boldsymbol{s}_j = ((\boldsymbol{W}_{\text{gate}})_{:,j}, (\boldsymbol{W}_{\text{up}})_{:,j}, (\boldsymbol{W}_{\text{down}})_{j,:}).$$

The neuron slices are independent of each other. ExpertWeaver aims to partition the neuron slices $\{\mathbf{s}_j\}_{j=1}^{d_{\text{ffn}}}$ into different expert groups.

As established in §2.2, GLU activation patterns are a rich source of information about the model's inherent structure. We use a multi-task calibration set $\mathcal{D}_{\text{calib}}$ from Flan-v2 (42 tasks, 5 samples each) to capture robust activation signals. For each sample j in $\mathcal{D}_{\text{calib}}$, we compute the token-averaged gate activation for neuron i as $a_{ij} = \text{mean}(\text{Swish}(\boldsymbol{x}_j \boldsymbol{W}_{\text{gate}}))_i$, where $\boldsymbol{x}_j$ represents all tokens in sample j. This gives the activation profile of neuron i as $\boldsymbol{a}_i = [a_{i1}, a_{i2}, \ldots, a_{iM}]$ with $M = |\mathcal{D}_{\text{calib}}|$. We collect all activation profiles into a matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{d_{\text{ffn}}}]$, which serves as the primary signal for our subsequent process.

3.2 Layer-Aware Expert Allocation

Building on our observation that the ratio of universal to specialized neurons varies across layers (**Takeaway 4**), we propose a layer-adaptive allocation strategy that determines the shared expert ratio for each layer.

Quantifying Layer-wise Neural Specialization. First, we quantify the functional specialization of each layer ℓ. We compute the CV based on each neuron's activation pattern $\boldsymbol{A}_\ell$. A high CV indicates that a neuron is highly specialized, activating only for specific inputs. We then define the layer's overall specialization ratio, r_ℓ, as the fraction of neurons whose CV exceeds a CV threshold (τ) used to identify specialized neurons:

$$r_\ell = \frac{1}{d_{\text{ffn}}} \sum_{j=1}^{d_{\text{ffn}}} \mathbb{I}[CV_j > \tau], \qquad (4)$$

where $\mathbb{I}[\cdot]$ is the indicator function and τ is a specialization threshold.

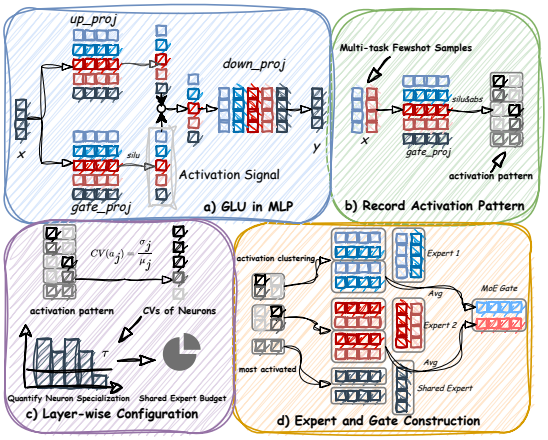


Figure 3: **The ExpertWeaver Framework.** a) The GLU in the MLP layer contains three weight matrices, where the same color denotes corresponding neuron slices. b) Neuron activation patterns are captured using a multi-task calibration dataset. c) The CVs are computed to determine the budget for shared vs. routed experts. (d) Neurons are clustered according to their activation patterns to form one shared expert and multiple routed experts.

Dynamic Allocation of Shared Expert Size. Next, we use this specialization ratio to determine the proportion of neurons, α_ℓ, to be allocated to the shared expert in that layer. The core principle is that more specialized layers (higher r_ℓ) require a smaller shared expert. We compute α_ℓ using a linear mapping:

$$\alpha_\ell = \alpha_{\max} - (\alpha_{\max} - \alpha_{\min}) \cdot r_\ell, \qquad (5)$$

where $\alpha_{\min}$ and $\alpha_{\max}$ define the bounds for the shared expert neuron ratio. In our framework, the total d_{ffn} neurons are distributed among N_e experts, giving each expert a fixed capacity of $d_{\text{expert}} = d_{\text{ffn}}/N_e$ neurons. The number of shared experts, $N_{se,\ell}$, is then determined by the number of full experts that can be formed from the allocated shared neurons by:

$$d_{s,\ell} = \text{round}(\alpha_\ell \cdot d_{\text{ffn}}), \qquad N_{se,\ell} = \text{round}(d_{s,\ell}/d_{\text{expert}}). \qquad (6)$$

The remaining $N_{re,l} = N_e - N_{se,\ell}$ experts are designated as routed experts. For each input token, we activate a total of k experts according to the required sparsity. This includes activating all $N_{se,\ell}$ shared experts, plus the top $k - N_{se,\ell}$ routed experts as determined by the router.

3.3 MoE Layer Construction

Once the layer-specific allocations of shared experts ($N_{se,\ell}$) and routed experts ($N_{re,l}$) are determined, we proceed to partition the FFN's neurons into *shared* and *routed* experts.

Constructing Shared Experts. The shared expert is formed from the most universally active neurons. We select the top $d_{expert} \cdot N_{se,\ell}$ neurons with the highest absolute average activation scores to form the shared neuron pool, indexed by $\mathcal{I}_s$. These neuron slices are then concatenated to form the weight matrices of a single, consolidated shared expert. Specifically, the weight matrices for the shared expert are constructed by concatenating the corresponding neural slices from the original dense layer:

$$\left(\boldsymbol{W}_{\text{gate}}^{(\text{s})}, \boldsymbol{W}_{\text{up}}^{(\text{s})}, \boldsymbol{W}_{\text{down}}^{(\text{s})}\right) = \left(\underset{j\in\mathcal{I}_s}{\text{CONCAT}}((\boldsymbol{W}_{\text{gate}})_{:,j}), \underset{j\in\mathcal{I}_s}{\text{CONCAT}}((\boldsymbol{W}_{\text{up}})_{:,j}), \underset{j\in\mathcal{I}_s}{\text{CONCAT}}((\boldsymbol{W}_{\text{down}})_{j,:})\right). \tag{7}$$

This shared expert is always activated, capturing and consolidating common knowledge across varying contexts.

Constructing Routed Experts. The remaining $N_{re,\ell} \cdot d_{\text{expert}}$ neurons, which form the specialized pool (indexed by $\mathcal{I}_{-s}$), are partitioned into $N_{re,\ell}$ routed experts. To group neurons that are frequently co-activated into the same expert, we employ a balanced K-Means clustering algorithm (Malinen & Fränti, 2014) on their activation pattern vectors $\{\boldsymbol{a}_i\}_{i\in\mathcal{I}_{-s}}$ (see Appendix G for details of balanced K-Means). This process partitions the set of specialized neurons into $N_{re,\ell}$ disjoint clusters $\{\mathcal{C}_1, \ldots, \mathcal{C}_{N_{re,\ell}}\}$, where each cluster contains exactly d_{expert} neurons and corresponds to a single routed expert. The weight matrices for the i-th expert are formed by concatenating the weights of all neurons in cluster $\mathcal{C}_i$:

$$\left(\boldsymbol{W}_{\text{gate}}^{(i)}, \boldsymbol{W}_{\text{up}}^{(i)}, \boldsymbol{W}_{\text{down}}^{(i)}\right) = \left(\underset{j\in\mathcal{C}_i}{\text{CONCAT}}(\boldsymbol{W}_{\text{gate},j}), \underset{j\in\mathcal{C}_i}{\text{CONCAT}}(\boldsymbol{W}_{\text{up},j}), \underset{j\in\mathcal{C}_i}{\text{CONCAT}}(\boldsymbol{W}_{\text{down},j})\right). \tag{8}$$

Constructing the MoE Router. We construct our MoE router in a training-free manner by leveraging the clustering structure established in the previous step. The key insight is that the original gating vector of each neuron, $(\boldsymbol{W}_{\text{gate}})_{:,j}$, controls that neuron's activation patterns. Therefore, the centroid of these vectors within each expert cluster naturally captures that expert's representative activation behavior. Thus, we construct the router $\boldsymbol{W}_{\text{router}} \in \mathbb{R}^{d_{\text{model}} \times N_{e,\ell}}$ by calculating a representative gating vector for each routed expert i through averaging the gating vector of all neurons assigned to that cluster:

$$\bar{\boldsymbol{w}}_{\text{gate}}^{(i)} = \frac{1}{|\mathcal{C}_i|} \sum_{j\in\mathcal{C}_i} (\boldsymbol{W}_{\text{gate}})_{:,j}, \qquad \boldsymbol{W}_{\text{router}} = \left[\bar{\boldsymbol{w}}_{\text{gate}}^{(1)}, \ldots, \bar{\boldsymbol{w}}_{\text{gate}}^{(N_{e,\ell})}\right]. \tag{9}$$

By directly using the original gating weights, this approach constructs the router without training while preserving the neuron activation patterns captured during pretraining.

3.4 ExpertWeaver in Practice: Dynamic Structural Pruning and Downcycling

3.4.1 Training-free Dynamic Structural Pruner

When operating at low sparsity levels without requiring additional training, ExpertWeaver can be viewed as a dynamic structural pruning method. For any given input $\boldsymbol{x}$, the router selects the top-k routed experts based on the logits from the reconstructed router, $\boldsymbol{T}(\boldsymbol{x}) = \text{TopK}(\boldsymbol{x}\boldsymbol{W}_g)$. The final output is the direct sum of the outputs from the always-active shared experts and these selected routed experts:

$$\boldsymbol{y} = \boldsymbol{E}_{\text{shared}}(\boldsymbol{x}) + \sum_{i\in\boldsymbol{T}(\boldsymbol{x})} \boldsymbol{E}_i(\boldsymbol{x}) \tag{10}$$

It is worth noting that, instead of using a weighted combination of expert outputs, our gate acts as a structured pruning mechanism, dynamically selecting which neurons to exclude from the forward pass. This preserves the integrity of the original neuron-level computations from the pretrained model, as the GLU mechanism within each activated expert still governs the precise activation values.

3.4.2 DOWNCYCLING DENSE LLMS INTO MOES

Model downcycling aims to convert large, pretrained dense models into computationally efficient MoEs, using the dense model's weights as a superior initialization to avoid the prohibitive costs of training from scratch. ExpertWeaver performs **downcycling** by converting dense models into sparse MoE architectures, followed by continued Pretraining (CPT) for further optimization.

Initialization. Since the sparsity in downcycling is often higher and configured according to downstream requirements, we use a *fixed, uniform shared expert ratio* across all layers. This provides a robust and well-structured starting point for the subsequent training phase.

Continued Pretraining. During CPT, we switch to a standard softmax router to provide greater optimization flexibility and enable experts to learn distinct specializations. The softmax router computes gating weights $\boldsymbol{g}(\boldsymbol{x}) = \text{Softmax}(\boldsymbol{x}\boldsymbol{W}_{\text{router}})$. The layer's output is a weighted combination of the shared expert and the top-k routed experts:

$$\boldsymbol{y} = \boldsymbol{E}_{\text{shared}}(\boldsymbol{x}) + \sum_{i \in \text{TopK}(\boldsymbol{g}(\boldsymbol{x}))} \boldsymbol{g}(\boldsymbol{x})_i \cdot \boldsymbol{E}_i(\boldsymbol{x}). \tag{11}$$

The total loss function consists of two components: the next-token prediction loss ($\mathcal{L}_{\text{NTP}}$) and an auxiliary load-balancing loss ($\mathcal{L}_{\text{LB}}$) to encourage a balanced distribution of tokens across experts.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{NTP}} + \lambda \mathcal{L}_{\text{LB}}; \quad \mathcal{L}_{\text{LB}} = N_{e,\ell} \cdot \sum_{i=1}^{N_{e,\ell}} f_i \cdot P_i, \tag{12}$$

where f_i is the fraction of tokens routed to expert i and P_i is its average router probability within a batch.

4 EXPERIMENTS

We evaluate ExpertWeaver in the following two scenarios. First, as a training-free dynamic structured pruner for low-sparsity settings, we benchmark it against some training-free baselines in Section 4.1. Second, as an initialization strategy for model downcycling in higher-sparsity settings, we demonstrate its advantages by comparing our resulting MoE models against similarly scaled baselines in Section 4.2.

4.1 EXPERTWEAVER FOR DYNAMIC STRUCTURAL PRUNING

Experimental Setup. We benchmark ExpertWeaver against training-free baselines including FLAP (You et al., 2024), LLM-Pruner (Ma et al., 2023), and CMoE (Pei et al., 2025) on Qwen2.5-7B and Llama3-8B models. Since CMoE is only implemented for Llama-series models, we only compared it on Llama3-8B. Following previous settings (Ma et al., 2023), we set the target sparsity level to 25% for all methods. Model performance is assessed on five widely-used benchmarks: MMLU, HellaSwag, ARC-e, ARC-c, and PIQA. For ExpertWeaver, we use our default hyperparameters: a shared expert ratio with $\alpha_{\min} = 0.2$ and $\alpha_{\max} = 0.7$, a specialization threshold of $\tau = 0.6$, and an expert granularity of 64. The specific layer-wise allocation between shared and routed experts is shown in Appendix L. To construct an effective calibration set that captures diverse multi-task activation patterns, we sample 5 instances from each of the 42 tasks in the Flan-v2 dataset, which spans 10 distinct task clusters (details in Appendix F).

Table 2: **Comparison with structured pruning methods under 25% sparsity.**

Method	MMLU(5)	HellaSwag(10)	ARC-e	ARC-c	PiQA	Avg.
LLaMA3-8B, 25% sparsity						
Dense	65.3	82.1	77.9	57.9	80.8	72.8
LLM-Pruner	24.2	51.3	58.9	32.4	74.4	48.2
FLAP	33.4	48.0	50.0	29.3	68.3	45.8
CMoE	41.6	65.9	63.1	41.5	73.9	57.2
ExpertWeaver	**47.0**	**69.8**	**64.4**	**44.3**	**76.3**	**60.4**
Qwen2.5-7B, 25% sparsity						
Dense	74.2	80.3	77.8	63.8	80.0	75.2
LLM-Pruner	55.9	72.2	71.0	49.1	**77.0**	65.0
FLAP	54.7	58.5	67.3	42.2	70.8	58.7
ExpertWeaver	**61.6**	**72.3**	**71.5**	**53.5**	76.3	**67.0**

Main Results. Table 2 summarizes the performance comparison between ExpertWeaver and other training-free structural pruning baselines at 25% sparsity. From these results, we observe the following key findings: (1) ExpertWeaver consistently outperforms all competing approaches, achieving a 5.6% relative improvement over CMoE on LLaMA3-8B and a 3.1% advantage over LLM-Pruner on Qwen2.5-7B. (2) Dense-to-MoE methods significantly surpass static pruning approaches like FLAP and LLM-Pruner. Rather than permanently removing parameters, these methods achieve sparsity by selectively activating parameter subsets based on input context, avoiding inevitable knowledge loss. (3) ExpertWeaver outperforms CMoE for three key reasons. First, our comprehensive analysis of GLU activation patterns ensures that the original activation structure remains intact during conversion. Second, our multi-task calibration set enables better expert clustering by capturing diverse neural co-activation patterns (ablation in Appendix E). Third, our layer-specific configuration allows adaptive MoE construction tailored to each layer's specialization characteristics. We also compared our method against other structural pruning techniques on more complex tasks and evaluated its performance on a reasoning model, as detailed in Appendix C.

Ablation Studies. As shown in Figure 4, we conduct a series of ablation studies to investigate the impact of key hyperparameters.

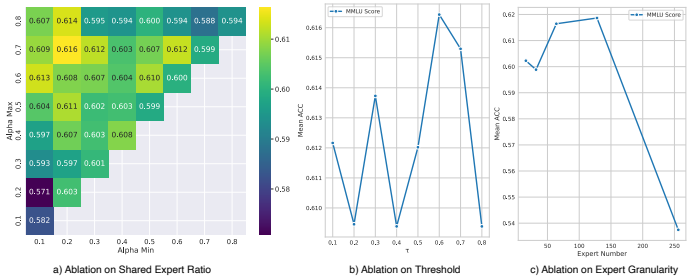


Figure 4: **Ablation studies on key hyperparameters of ExpertWeaver.** a) MMLU performance heatmap for different shared expert ratios, controlled by the hyperparameters $\alpha_{\min}$ and $\alpha_{\max}$ in Eq. 5. The diagonal where $\alpha_{\min} = \alpha_{\max}$ represents a static configuration with a uniform ratio across all layers. b) MMLU performance varying specialization threshold τ from Eq. 4. c) MMLU performance varying expert granularity.

a) Shared Expert Ratio: The heatmap in Figure 4a) explores the impact of the shared expert ratio, controlled by $\alpha_{\min}$ and $\alpha_{\max}$ from Eq. 5. The results reveal that our layer-aware dynamic allocation strategy significantly outperforms static configurations. The diagonal, where $\alpha_{\min} = \alpha_{\max}$, represents a static setting with a uniform ratio across all layers, and its performance is lower than that of the off-diagonal regions. The optimal performance is achieved within the range of $\alpha_{\min} \in [0.2, 0.4]$ and $\alpha_{\max} \in [0.5, 0.7]$, with the peak at $(0.2, 0.7)$. We adopted $\alpha_{\min} = 0.2$ and $\alpha_{\max} = 0.7$ as our default setting. **b) Specialization Threshold:** We study the impact of the specialization threshold τ from Eq. 4, which determines whether a neuron is classified as specialized based on its CV score. The model demonstrates robust performance across a range of τ values, with optimal results achieved at $\tau = 0.6$. We adopt $\tau = 0.6$ as the default configuration for all experiments. **c) Expert Granularity:** The total number of experts significantly influences performance. The MMLU score peaks when the number of experts is around 64 and 128, declining with either too few or too many experts. This suggests an optimal granularity that balances functional diversity and specialization efficiency. To prioritize inference efficiency while maintaining strong performance, we adopt 64 as our default expert granularity.

4.2 Downcycling with ExpertWeaver.

Experiment Setup. To evaluate ExpertWeaver as a downcycling strategy, we initialize two MoE variants. The first, ExpertWeaver (Qwen2.5-7B), is initialized from the dense Qwen2.5-7B model. It divides the MLP layers into 62 routed experts and 2 shared experts, activating 14 routed and 2 shared experts per token (7B total parameters, 3.5B active). This model undergoes CPT for 200B tokens on the FineWeb-Edu dataset (Lozhkov et al., 2024). For a fairer comparison with the OLMoE baseline, we also initialize a second variant, ExpertWeaver (OLMo-7B), from the OLMo-7B base model. This variant is continually pretrained for 200B tokens on the same dataset used for OLMoE. The performance of both models is evaluated on a wide range of tasks, including MMLU, HellaSwag, ARC-e, ARC-c, PIQA, WinoGrande, LogiQA, and SciQ, and compared against dense and MoE baselines with similar parameter counts and training budgets (see Appendix H for details). Following CPT, we conduct a two-stage supervised fine-tuning (SFT) process, similar to Llama-MoE-V2 (Qu et al., 2024), focusing first on general conversational abilities and then on code and math skills. The re-

Table 3: **Downcycling Performance Comparison.** Best results are in bold, second-best are underlined. Models in gray (Qwen2.5-{7B, 3B, 1.5B}, Llama-3.2-3B, and OLMoE) are for context, as they were trained on significantly more data. Our ExpertWeaver models (OLMo-based and Qwen2.5-based) were trained on 200B tokens. The OLMo-based version has 1B active parameters (7B-A1B), and the Qwen2.5-based version has 3.5B active parameters (7B-A3.5B). We compare against the 500B token checkpoint (*) of OLMoE. Blank entries for OpenMoE indicate unavailable results.

Model	MMLU(5)	HellaSwag(10)	ARC-e	ARC-c(25)	PIQA	WinoGrande	LogiQA	SciQ	Average
Dense Models									
Qwen2.5-7B	74.1	80.2	77.5	63.7	79.7	73.2	36.4	95.2	72.5
Qwen2.5-3B	65.6	74.6	73.9	56.5	78.8	68.1	33.5	95.2	68.3
Qwen2.5-1.5B	60.9	68.0	72.5	54.9	75.9	63.8	31.9	93.4	65.2
Llama-3.2-3B	56.1	76.4	71.6	50.5	77.4	69.9	30.6	92.7	65.7
OLMo-7B	30.7	77.1	68.7	45.1	79.6	66.5	27.5	88.6	60.8
OPT-2.7B	25.8	61.4	54.4	34.0	74.8	60.8	25.8	78.9	52.0
Pythia-2.8B	26.8	60.7	58.8	36.7	73.6	59.6	28.1	83.2	53.4
INCITE-Base-3B	27.2	64.7	61.7	40.3	73.9	63.5	27.5	85.6	55.6
Open-LLaMA-3B-v2	26.8	71.4	63.3	40.1	77.9	63.1	28.1	88.0	57.3
Sheared-LLaMA-2.7B	27.3	71.0	63.3	41.6	76.9	65.0	28.3	87.5	57.6
Gemma-2-2b	**53.0**	69.0	36.9	52.6	67.5	51.9	22.7	75.8	53.7
SmolLM2-1.7B	50.4	72.6	**73.4**	53.2	76.0	**65.8**	**30.1**	84.3	63.2
MoE Models									
LLaMA-MoE-v1-3.5B	26.8	73.3	65.6	44.2	77.9	65.5	29.7	87.6	58.8
OpenMoE-3B-9B	-	56.5	50.6	33.3	65.7	51.9	-	-	-
OLMoE-1B-7B	53.8	79.6	76.3	55.6	80.1	68.4	29.3	94.9	67.2
OLMoE-1B-7B*	28.4	70.2	71.0	43.9	77.1	63.5	28.7	88.5	58.9
LLaMA-MoE-v2-3.5B	40.9	53.7	57.0	40.2	67.9	56.1	30.7	**88.8**	54.4
ExpertWeaver-E64-A14-S2$_{(OLMo-7B)}$	45.0	61.2	69.3	38.8	74.5	62.1	28.5	91.8	58.9
ExpertWeaver-E64-A14-S2$_{(Qwen2.5-7B)}$	45.6	**73.7**	72.4	**56.3**	**78.0**	65.3	29.0	87.7	**63.5**

sulting instruction-tuned model is then compared with other instruction-tuned baselines on MMLU, ARC-c, GSM8K, HumanEval, and IFEval. Further training details are available in Appendix J.

Main Results. Table 3 presents the results of comparing ExpertWeaver with other models of comparable parameters and training budgets, yielding the following observations. (1) Among models with comparable training budgets and parameter counts, our ExpertWeaver$_{(Qwen2.5-7B)}$ achieves the best average performance (63.5), outperforming the strongest MoE baseline (OLMoE-1B-7B*) by 4.6 points. (2) To ensure a fair comparison, we also downcycled OLMo-7B. After just 200B tokens of continued pretraining, our ExpertWeaver$_{(OLMo-7B)}$ achieves a score of 58.9. This performance represents 97.35% of the original OLMo-7B model's performance (60.5). Notably, this result not only nears the performance of the original model but also matches the performance of the OLMoE-1B-7B* model, which was trained from scratch on more than double the data (500B tokens). While a slight difference in scores may be attributed, in part, to the potentially lower quality of the Dolma dataset used by OLMo compared to OLMoE-Mix (Muennighoff et al., 2024), this result strongly highlights the effectiveness of our ExpertWeaver method. It demonstrates the capacity to efficiently downcycle a large dense model into a high-performing MoE architecture with only a minimal amount of continued training data (200B tokens). (3) Compared to its dense counterpart, ExpertWeaver$_{(Qwen2.5-7B)}$ retains 87.6% of the base Qwen2.5-7B's performance while activating only a quarter of the MLP parameters. This remarkable performance retention, achieved with a modest 200B tokens of training, strongly validates the effectiveness of our downcycling strategy. Furthermore, when compared to dense models with similarly activated parameter counts like Qwen2.5-3B and Llama-3.2-3B, our model achieves 93.0% and 96.7% of their respective performance. Given that these models were trained on massive datasets (18T and 9T tokens, respectively), this result further demonstrates that our downcycling method is a highly efficient path to creating performant MoE models.

Table 4: **Instruct Model Performance Comparison**

Method	MMLU(5)	ARC-C(25)	GSM8K	HumanEval	IFEval	Avg.
LLaMA-MoE-3B-7B	28.24	44.03	4.62	12.02	28.10	23.40
OLMoE-1B-7B	53.79	55.63	40.94	40.48	35.49	45.27
LLaMA-MoE-v2	40.90	40.20	55.00	**51.20**	**36.00**	44.66
ExpertWeaver-Instruct	**50.60**	**69.80**	**57.10**	50.20	33.10	**52.16**

SFT Performance. As shown in Table 4, our instruction-tuned model, ExpertWeaver-Instruct, achieves superior performance against other MoE baselines. It achieves a top-ranking average score of 52.16, establishing a clear lead over comparable MoE models like OLMoE-1B-7B (45.27) and LLaMA-MoE-v2 (44.66). This result demonstrates the effectiveness of the ExpertWeaver methodology in creating a robust foundation for supervised fine-tuning. We also show the serving efficiency of the ExpertWeaver-Instruct model in Appendix D.

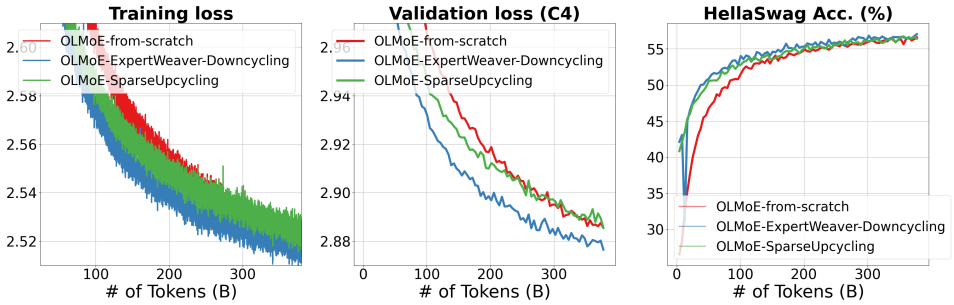


Figure 6: **Comparison of Downcycling, Upcycling, and From-Scratch Training.** Comparison of training loss, evaluation loss, and downstream task performance using the same OLMoE model configuration under three different MoE initialization paradigms.

Compare with other MoE initialization methods. Figure 5 shows the training loss for the first 5,000 steps (approximately 20B tokens), highlighting the effectiveness of our ExpertWeaver as a downcycling strategy. Compared to both random and Llama-MoE initializations, our method consistently yields a lower training loss throughout the entire training process. This demonstrates that ExpertWeaver provides a more effective starting point for the MoE, enabling faster convergence and superior final performance. The persistent gap between our loss curve (green) and the others validates that our strategy better preserves the foundational knowledge of the dense model during conversion.

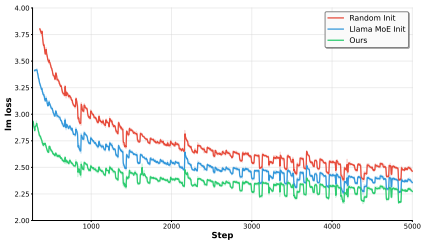


Figure 5: **Training Loss Comparison with Different MoE Initialization Strategies.**

A Direct Comparison of Downcycling, Upcycling, and From-Scratch Training. To provide a fairer comparison and demonstrate the effectiveness of ExpertWeaver, we used OLMo-1.3B (pre-trained on 1T tokens) as the base model for our downcycling process. For direct contrast, we also performed sparse upcycling (Muennighoff et al., 2024), training a 575M OLMo model for 1T tokens to yield a 1.3B OLMoE model (with 676M activated parameters). Both our downcycling method and the upcycling approach are compared against a baseline trained from scratch. Detailed model configurations are provided in Appendix I. As illustrated in Figure 6, which plots the training loss, evaluation loss, and downstream task performance, the results lead to several key observations. Initially, both the downcycling and upcycling strategies exhibit faster convergence compared to the from-scratch baseline. However, over a longer training period, the performance of the upcycled model regresses toward that of the from-scratch baseline. In contrast, our downcycling approach with ExpertWeaver consistently achieves the best performance and convergence, maintaining its superiority throughout the entire training process.

These findings suggest that the superiority of downcycling stems from two key factors. First, by starting with a larger, more capable dense model, downcycling inherits a richer feature space. Second, since the experts are constructed without parameter duplication, they offer more diverse and specialized pathways, leading to a higher ceiling for long-term optimization. In contrast, upcycling methods, which often initialize experts by duplicating weights, are constrained by this parameter redundancy and the limited capacity of the smaller base model. This can lead to the model quickly falling into a local optimum, resulting in a long-term performance disadvantage compared to downcycling.

5 Conclusion

In this paper, we investigate the problem of converting dense models into high-performance MoEs. We show that GLU activation patterns provide a rich source for identifying latent neuron specialization, which naturally enables effective expert construction. Based on this observation, we introduce ExpertWeaver, a training-free method that partitions neurons into shared and routed experts in a layer-wise manner based on their activation patterns. Experiments demonstrate that ExpertWeaver significantly outperforms existing methods in both zero-shot pruning for inference efficiency and MoE initialization for model downcycling.

REFERENCES

Hervé Abdi. Coefficient of variation. *Encyclopedia of research design*, 1(5):169–171, 2010.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. A survey on mixture of experts in large language models. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *European Conference on Computer Vision*, pp. 370–387. Springer, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Cisco. INCITE-Base-3B Model.
urlhttps://huggingface.co/Cisco/incite-base-3b-v1, 2023.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

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

Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023. URL `https://github.com/openlm-research/open_llama`.

Ethan He, Abhinav Khattar, Ryan Prenger, Vijay Korthikanti, Zijie Yan, Tong Liu, Shiqing Fan, Ashwath Aithal, Mohammad Shoeybi, and Bryan Catanzaro. Upcycling large language models into mixture of experts. *arXiv preprint arXiv:2410.07524*, 2024a.

Zexin He, Zexi Zhang, Jun-Jie Li, Kuan-Chieh Chen, and Yu-Chiang Frank Wang. Tomoe: A training-free and ood-robust method for moe-based model merging. *arXiv preprint arXiv:2404.08789*, 2024b.

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.

HuggingFaceTB. SmolLM2-1.7B Model.
urlhttps://huggingface.co/HuggingFaceTB/SmolLM2-1.7B, 2024.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*, 2022.

Jijie Li et al. Infinity instruct: Scaling instruction selection and synthesis to enhance language models. *arXiv preprint arXiv:2506.11116*, 2025.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Xinyin Ma, Yixin Hou, Peng Gui, Yubo Li, Tianyu Gao, Wanli Che, Zhi Sun, and Ting Liu. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Mikko I Malinen and Pasi Fränti. Balanced k-means for clustering. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pp. 32–41. Springer, 2014.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. Olmoe: Open mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2409.02060.

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

Kumari Nishu, Sachin Mehta, Samira Abnar, Mehrdad Farajtabar, Maxwell Horton, Mahyar Najibi, Moin Nabi, Minsik Cho, and Devang Naik. From dense to dynamic: Token-difficulty driven moefication of pre-trained LLMs. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*, 2025. URL https://openreview.net/forum?id=IBHtKmJME0.

Zehua Pei, Lancheng Zou, Hui-Ling Zhen, Xianzhi Yu, Wulong Liu, Sinno Jialin Pan, Mingxuan Yuan, and Bei Yu. Cmoe: Converting mixture-of-experts from dense to accelerate llm inference. *arXiv preprint arXiv:2502.04416*, 2025.

Xiaoye Qu, Daize Dong, Xuyang Hu, Tong Zhu, Weigao Sun, and Yu Cheng. Llama-moe v2: Exploring sparsity of llama from perspective of mixture-of-experts with post-training. *arXiv preprint arXiv:2411.15708*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Qwen Team et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3), 2024.

The Gemma Team and Google. Gemma 2: Unlocking the power of open models. *arXiv preprint arXiv:2406.15559*, 2024.

Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023. URL https://huggingface.co/datasets/teknium/OpenHermes-2.5.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Ya-Chu You, Kuan-Chieh Chen, and Yu-Chiang Frank Wang. Flap: A training-free and efficient fine-tuning framework for language-and-vision models. *arXiv preprint arXiv:2404.08789*, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*, 2021.

Haizhong Zheng, Xiaoyan Bai, Xueshen Liu, Zhuoqing Morley Mao, Beidi Chen, Fan Lai, and Atul Prakash. Learn to be efficient: Build structured sparsity in large language models. *Advances in Neural Information Processing Systems*, 37:101969–101991, 2024.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*, 2023a.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023b.

Fuzhao Zhu, Wen-Bin Huang, , Zhi-Hong Chen, , Jian-Fang Wang, , Yao Fu, , and Ke-Tan Chen. Openmoe: An early-stage open-source mixture-of-experts language model. *arXiv preprint arXiv:2401.03969*, 2024a.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *arXiv preprint arXiv:2406.16554*, 2024b.

A THE USE OF LLMS

LLMs were used to improve writing clarity and assist with code development. Specifically, LLMs assisted in improving the clarity, fluency, and grammatical correctness of the manuscript, including rephrasing sentences and ensuring consistent terminology. Additionally, LLMs helped generate auxiliary code and scripts for data processing, experimental setup, and result visualization. However, the core research ideas, technical contributions, experimental design, and scientific conclusions are entirely the intellectual contribution of the human authors. All LLM-generated content underwent thorough human review and verification to ensure technical accuracy, scientific rigor, and alignment with our research objectives.

B Related Work

Structural Pruning. Structural pruning improves LLM efficiency by removing entire components like neurons or attention heads. To avoid costly retraining, recent training-free methods use importance scores to identify and remove structures. For example, LLM-Pruner (Ma et al., 2023) analyzes gradients, while FLAP (You et al., 2024) measures output feature stability. These methods perform *static* pruning by permanently removing weights. In contrast, methods that convert dense models into MoEs, such as ToMoE (He et al., 2024b), can be considered a form of *dynamic* structural pruning. Since the goal of converting a dense model to an MoE aligns with that of structural pruning (reducing computational cost while preserving performance), we also categorize ExpertWeaver as a dynamic structural pruning method. It achieves this by dynamically selecting a sparse subset of neuron blocks for each token at inference time.

Model Upcycling. Model *upcycling* (Team et al., 2024; Muennighoff et al., 2024; He et al., 2024a; Nakamura et al., 2025; Komatsuzaki et al., 2022) offers a cost-effective strategy for creating larger sparse models by merging smaller, pretrained dense models, thus avoiding the expense of from-scratch training. A common technique is to replicate the FFN layers from one or more dense models to form the experts of a new, larger MoE, followed by a fine-tuning phase to learn the routing logic. While effective, this paradigm faces two emerging challenges. First, the concentration of research and computational resources on frontier models means that the capabilities of state-of-the-art large models are advancing at a pace that smaller models struggle to match. Upcycling from smaller, less capable models may therefore not match the performance of downcycling from a larger, more advanced one. Second, because upcycling often relies on parameter duplication to initialize experts, it can lead to a lack of diversity that predisposes the model to fall into local optima during long-term optimization. In contrast, downcycling, as implemented by ExpertWeaver, leverages the rich, non-redundant internal structure of a single large model, providing a more robust foundation for sustained performance gains.

Model Downcycling. Model *downcycling* converts large pretrained dense models into computationally efficient MoEs, aiming to retain performance while gaining inference speed through FFN neuron partitioning. Moefication (Zhang et al., 2021) pioneered this field by splitting FFN parameters into functional partitions as experts with learned routers, but was originally designed for ReLU-based networks and struggles with modern GLU-based architectures. LTE (Zheng et al., 2024) trains efficiency-aware models to amplify inherent activation sparsity through efficiency loss penalties, but requires substantial computational overhead during the training phase and does not directly leverage inherent model structures for expert creation. ToMoE (He et al., 2024b) uses dynamic structural pruning with frozen weights to discover expert structures via learned routing, but still requires substantial training overhead for the routing modules. Llama-MoE (Zhu et al., 2024b; Qu et al., 2024) partitions FFN parameters through clustering followed by extensive continual pre-training with 200B tokens, but ignores the model's internal structural patterns, resulting in poor MoE performance that fails to preserve the original dense model's capabilities. CMoE (Pei et al., 2025) achieves training-free conversion using balanced clustering with analytically constructed routers, yet lacks detailed activation signal analysis and employs fixed configurations across layers, leading to suboptimal expert partitioning that fails to capture layer-specific specializations. ExpertWeaver introduces a novel training-free dense-to-MoE technique that leverages intrinsic GLU activation patterns to form experts and construct routers, enabling both training-free applications and supporting efficient downcyling for further CPT.

C Comparision on Complex Tasks and Reasoning Models

To compare the capabilities of different methods in more complex scenarios, we first investigate the performance of Qwen2.5-7B on GSM8k and HumanEval under various sparsities. Furthermore, to assess performance on models specialized for reasoning, we explore the effectiveness of these methods on the DeepSeek-R1-Distill-Qwen-7B model, evaluating it on the GPQADiamond and LiveCodeBench benchmarks.

The results, presented in Table 5, reveal the limitations of static structural pruning on complex tasks. At 25% sparsity on GSM8k, static methods like LLM-Pruner and FLAP almost completely fail; the

dash (-) for LLM-Pruner indicates its accuracy dropped to zero. In stark contrast, ExpertWeaver, which retains all model parameters, maintains strong performance. This advantage holds at 12.5% sparsity, where ExpertWeaver continues to significantly outperform static methods on both GSM8k and HumanEval. The same trend is observed on the specialized reasoning model, DeepSeek-R1-Distill-Qwen-7B, where ExpertWeaver again achieves the highest scores on GPQADiamond and LiveCodeBench. This demonstrates that ExpertWeaver's dynamic pruning approach is substantially more effective at preserving critical reasoning and coding capabilities, especially at higher sparsity levels where static methods suffer from irreversible information loss.

Table 5: **Training-Free Pruning on Code Benchmarks.** We compare ExpertWeaver against static pruning methods on several code-related benchmarks. The base models used for each benchmark are specified in the table.

	GSM8k		HumanEval		GPQADiamond	LiveCodeBench
Base Model	Qwen2.5-7B				DeepSeek-R1-Distill-Qwen-7B	
Sparsity	25%	12.5%	25%	12.5%	12.5%	
LLM-Pruner	2.0	-	14.6	10.4	33.3	6.1
FLAP	14.8	-	57.6	9.8	36.3	2.3
ExpertWeaver	34.9	18.9	64.6	32.9	37.3	16.8

D Serving Efficiency

To investigate the serving efficiency of our model, we conduct a comprehensive benchmark comparing ExpertWeaver (E64-A14-S2, derived from Qwen2.5-7B) with the dense Qwen2.5-7B model using the vLLM framework. All tests are run on a single GPU with 'tensor_parallel_size=1' and GPU memory utilization set to 90% to maximize the memory allocated to the KV Cache. We simulate a high-load scenario by sending 1024 random requests with an average input length of 512 tokens at an infinite rate, evaluating the models' peak performance at a maximum concurrency of 128 sequences. The evaluation spans three generations of NVIDIA GPUs—A100 (Ampere) and H100 (Hopper)—to ensure a comprehensive and fair assessment.

The results, presented in Table 6, show ExpertWeaver demonstrates superior performance on both platforms. It achieves higher throughput (RPS, OTPS, and TTPS) and lower latency for both the first token (TTFT) and subsequent tokens (TPOT). This clear-cut advantage across all metrics confirms the inference efficiency of the ExpertWeaver.

Table 6: **Inference throughput comparison across different GPU architectures.** RPS: Requests Per Second. OTPS: Output Tokens Per Second. TTPS: Total Tokens Per Second. TTFT: Time To First Token. TPOT: Time Per Output Token. ITL: Inter-Token Latency.

GPU	Method	RPS ↑	OTPS ↑	TTPS ↑	TTFT ↓ (ms)	TPOT ↓ (ms)	ITL ↓ (ms)
A100	ExpertWeaver	22.88	2928.86	14644.31	440.6	40.3	40.3
	Qwen2.5-7B	18.85	2413.22	12066.10	675.4	47.8	48.2
H100	ExpertWeaver	44.04	5637.42	28187.10	542.2	9.8	9.8
	Qwen2.5-7B	41.41	5299.96	26499.78	1404.5	18.5	18.5

E Ablation Studies on the Calibration Set

To investigate the impact of the calibration set on ExpertWeaver's performance, we conducted two ablation studies, with results presented in Table 7.

Impact of Data Quantity and Diversity. We first analyzed the sensitivity to the calibration set's size and diversity. We created several calibration sets by randomly sampling different proportions

(25%, 50%, 75%, and 100%) of the tasks from our default Flan-v2 collection. For each selected task, we used 10 samples, meaning that as the proportion increases, both the number of tasks (diversity) and the total number of samples (quantity) grow. The results show that performance generally improves with a larger and more diverse calibration set, with the best result (67.0) achieved using 100% of the tasks. Notably, using just 50% of the tasks already yields a strong average score of 66.1, which is over 98% of the final performance. This demonstrates that while diversity is beneficial, ExpertWeaver is data-efficient and does not require an excessively large calibration set to achieve robust performance.

Impact of Data Source. We also compared our default multi-task calibration set (Flan-v2) against a variant calibrated solely on a general-domain corpus (C4), denoted as ExpertWeaver$_{C4}$. The results clearly demonstrate the benefit of using a multi-task dataset. Our default ExpertWeaver achieves a superior average score of 67.0, outperforming ExpertWeaver$_{C4}$ (65.8). This suggests that capturing a wide range of activation patterns from diverse tasks is crucial for identifying a truly robust and generalizable functional structure within the dense model, leading to a more effective expert partition.

Table 7: **Ablation Studies on the Calibration Set.** We analyze the impact of calibration set size, diversity, and source.

Method	MMLU	HellaSwag(10)	ARC-e	ARC-c(25)	PiQA	Avg.
Qwen2.5-7B, 25% sparsity						
Dense	74.2	80.3	77.8	63.8	80.0	75.2
Ablation on Data Source						
ExpertWeaver$_{Flan}$ (default)	61.6	72.3	71.5	53.5	76.3	67.0
ExpertWeaver$_{C4}$	61.2	71.6	72.1	49.1	74.8	65.8
Ablation on Data Quantity & Diversity						
ExpertWeaver$_{Flan25\%}$	59.2	69.0	68.7	47.3	75.4	63.9
ExpertWeaver$_{Flan50\%}$	60.6	71.2	71.5	51.7	75.5	66.1
ExpertWeaver$_{Flan75\%}$	60.1	71.3	72.5	46.8	77.3	65.6
ExpertWeaver$_{Flan100\%}$	61.6	72.3	71.5	53.5	76.3	67.0

F Details of the Calibration Set

To construct a diverse, multi-task calibration set for analyzing neuron activation patterns, we sampled from the Flan-v2 collection (Chung et al., 2024). Flan-v2 is a large-scale dataset consisting of a mixture of publicly available NLP datasets that have been formatted into an instruction-tuning style. This diversity makes it an ideal source for a calibration set intended to capture a wide range of functional specializations.

For our calibration set, $\mathcal{D}_{\text{calib}}$, we selected a representative subset of 48 distinct tasks from 10 different task clusters within Flan-v2. For each of these 48 tasks, we randomly sampled 5 few-shot examples, resulting in a total of 240 samples in our calibration set. This carefully curated subset ensures that our analysis of neuron activation patterns is based on a broad and balanced distribution of tasks, from reading comprehension and summarization to commonsense reasoning and natural language inference.

Table 8 lists the 10 task clusters and the 48 specific tasks used in our calibration set.

G Balanced K-Means Clustering

We employ balanced K-Means clustering to partition specialized neurons $\mathcal{I}_r$ into $N_{re,\ell}$ routed experts of equal capacity d_{expert}. Given activation pattern vectors $A_r = \{\boldsymbol{a}_i\}_{i \in \mathcal{I}_r}$, the algorithm finds clusters $\mathcal{C}_1, \ldots, \mathcal{C}_{N_{re,\ell}}$ that solve:

Table 8: Tasks from Flan-v2 used in the calibration set.

Task Cluster	Description	Datasets
Reading Comprehension	Answers questions based on provided passages.	squad_v1, squad_v2, drop, duorc, quac, record
Summarization	Creates a shorter version of a document.	xsum, cnn_dailymail, samsum, multi_news
Translation	Translates text across multiple languages.	wmt14_en-fr, wmt14_en-de, wmt14_en-ro
Commonsense Reasoning	Understands everyday scenarios.	boolq, piqa, siqa, cosmos_qa, hellaswag, winogrande
Natural Language Inference	Determines logical relationship between sentences.	mnli, qnli, rte, wnli, anli
Coreference Resolution	Identifies expressions referring to the same entity.	wsc, dpr, winogender
Sentiment Analysis	Determines sentiment polarity.	imdb, sentiment140, yelp_polarity
Question Answering	Answers questions without external knowledge.	arc, openbookqa, race, trivia_qa
Paraphrase Detection	Generates alternative phrasings of sentences.	mrpc, qqp
Structure-to-Text	Generates text from structured data.	common_gen, e2e_nlg, dart

$$\begin{aligned} \min_{\mathcal{C}_1,\ldots,\mathcal{C}_{N_{re,\ell}}} \quad & \sum_{k=1}^{N_{re,\ell}} \sum_{i\in\mathcal{C}_k} ||\boldsymbol{a}_i - \boldsymbol{\mu}_k||^2 \\ \text{s.t.} \quad & |\mathcal{C}_k| = d_{\text{expert}} \quad \forall k \in \{1,\ldots,N_{re,\ell}\} \\ & \mathcal{C}_j \cap \mathcal{C}_k = \emptyset \quad \forall j \neq k \\ & \bigcup_{k=1}^{N_{re,\ell}} \mathcal{C}_k = \mathcal{I}_r \end{aligned} \tag{13}$$

where $\boldsymbol{\mu}_k$ is the centroid of cluster $\mathcal{C}_k$.

Algorithm:

1. **Initialize**: Sample $N_{re,\ell}$ centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{N_{re,\ell}}$ from A_r.
2. **Assign**: Solve the minimum-cost perfect matching problem with fixed centroids. We use a greedy approximation: iteratively assign neurons to the nearest available cluster slot.
3. **Update**: Recompute centroids as:

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{d_{\text{expert}}} \sum_{i\in\mathcal{C}_k} \boldsymbol{a}_i \quad \forall k \tag{14}$$

4. **Iterate**: Repeat steps 2-3 until convergence.

This ensures functionally coherent and structurally uniform experts for efficient MoE implementation.

H Details of the compared baselines

H.1 Baselines for Structured Pruning

For the structured pruning evaluation, we compare ExpertWeaver with the following training-free methods:

- **LLM-Pruner** (Ma et al., 2023) is a task-agnostic, training-free structural pruning method that identifies and removes redundant structures by analyzing gradient information and parameter magnitudes. It aims to preserve the model's generalization capabilities by focusing on the interconnectedness of model components.
- **FLAP** (You et al., 2024) is a training-free pruning framework that prunes large language models at the FFN-layer level. It uses a metric based on output feature stability to identify and remove less important neurons, offering a computationally efficient alternative to methods that require gradient computation.
- **CMoE** (Pei et al., 2025) is a training-free method that converts dense models into MoEs using balanced clustering. It constructs routers analytically but uses a fixed configuration across all layers, which may not capture layer-specific functional specializations.

H.2 Baselines for Model Downcycling

In the model downcycling experiments, we compare our ExpertWeaver-initialized model against a variety of both dense and MoE models with comparable parameter counts and training budgets.

Dense Models. We include several strong, publicly available dense models as baselines:

- **OPT-2.7B** (Zhang et al., 2022), **Pythia-2.8B** (Biderman et al., 2023), **INCITE-Base-3B** (Cisco, 2023), **Open-LLaMA-3B-v2** (Geng & Liu, 2023), and **Sheared-LLaMA-2.7B** (Xia et al., 2023) are all well-established language models in the 2.7B-3B parameter range.
- **Gemma-2-2B** (Team & Google, 2024) is a recent, highly-performant model from Google.
- **SmolLM2-1.7B** (HuggingFaceTB, 2024) is another strong baseline known for its efficiency and performance at a smaller scale.

MoE Models. We also compare against several existing MoE models:

- **LLaMA-MoE-v1** (Zhu et al., 2024b) and **LLaMA-MoE-v2** (Qu et al., 2024) are MoE variants of the LLaMA architecture. They are created by partitioning the FFN parameters of a dense model into experts and then applying extensive continued pretraining.
- **OpenMoE** (Zhu et al., 2024a) is an open-source MoE model series.
- **OLMoE** (Muennighoff et al., 2024) is a family of open language models with a MoE architecture, developed by AI2. We compare against a checkpoint trained on 500B tokens to ensure a fair comparison with our model's training budget.

H.3 Baselines for Instruct Model Performance

For evaluating the performance of our instruction-tuned model, **ExpertWeaver-Instruct**, we compare it against the following instruction-tuned MoE baselines:

- **LLaMA-MoE-3B-7B** and **LLaMA-MoE-v2** are the instruction-tuned versions of the LLaMA-MoE models described above.
- **OLMoE-1B-7B** is the instruction-tuned version of the OLMoE model.

I Model Architecture for OLMo Downcycling Experiment

The model architecture for the OLMo Downcycling experiment is shown in Table 9.

Table 9: **Model Configurations of OLMo Downcycling Experiment**

	OLMo 575M	OLMo 1.3B	OLMoE 1.3B-A676M
Model Dimension	2048	2048	2048
FFN Dimension	1024	8192	1024
Attention Heads	16	16	16
Key/Value Heads	16	16	16
Layers	16	16	16
Vocabulary Size	50280	50280	50280
Weight Typing	True	True	True
Context Length	4096	4096	4096
Expert Granularity	-	-	2 in 8

J Training Configurations

This section provides the detailed configurations for both the continued pre-training (CPT) and supervised fine-tuning (SFT) phases.

J.1 CONTINUED PRE-TRAINING (CPT)

The CPT phase was conducted on the ExpertWeaver-E64-A14-S2 model for 200 billion tokens using the FineWeb-Edu dataset (Lozhkov et al., 2024) using 128 H100 GPUs. Key hyperparameters are listed in Table 10. We use megatron-swift[1] for model training.

Table 10: Hyperparameters for Continued Pre-training.

Hyperparameter	Value
Optimizer	AdamW
Learning Rate	4e-4
Min Learning Rate	4e-5
Global Batch Size	1024
Sequence Length	4096
Training Iterations	50,000
Warmup Iterations	100
Auxiliary Loss Coeff (λ)	0.01

J.2 SUPERVISED FINE-TUNING

The SFT process consists of two stages with identical hyperparameters. Following the methodology of (Muennighoff et al., 2024), we use a fixed learning rate of 2e-5, a global batch size of 128, and do not use an auxiliary loss for 2 epochs. The training for both stages was conducted on 64 H100 GPUs.

Stage 1: General Conversational Tuning. This stage focuses on general conversational abilities, using a dataset mixture of LIMA (Zhou et al., 2023a), OpenHermes (Teknium, 2023), ShareGPT (Chen et al., 2024), and BAAI Infinity Instruct (Li et al., 2025).

Stage 2: Code and Math Tuning. This stage hones the model's capabilities in code and mathematics, using a dataset mixture of BAAI (Li et al., 2025) and MetaMathQA (Yu et al., 2023), with a small amount of conversational data from Stage 1.

K EVALUATION DATASETS

We evaluate our models on a comprehensive suite of benchmarks to assess their capabilities across various reasoning and knowledge domains.

- **MMLU** (Hendrycks et al., 2020) (Measuring Massive Multitask Language Understanding) is a broad benchmark designed to measure knowledge acquired during pre-training. It covers 57 subjects across STEM, humanities, social sciences, and more, making it a robust test of world knowledge and problem-solving ability.
- **HellaSwag** (Zellers et al., 2019) is a commonsense reasoning benchmark that challenges models to complete a sentence by choosing the most plausible ending from four options. It is designed to be difficult for models that rely on superficial statistical patterns.
- **ARC** (Clark et al., 2018) (AI2 Reasoning Challenge) is a question-answering dataset containing grade-school level science questions. We use both the Easy (ARC-e) and Challenge (ARC-c) sets, which are designed to be answerable with simple retrieval or multi-hop reasoning, respectively.
- **PIQA** (Bisk et al., 2020) (Physical Interaction Question Answering) is a commonsense reasoning benchmark focused on physical interactions. It presents two possible solutions to everyday situations, and the model must choose the more physically plausible one.
- **WinoGrande** (Sakaguchi et al., 2021) is a large-scale dataset for commonsense reasoning, formulated as a Winograd Schema Challenge. It requires resolving pronouns in ambiguous sentences, which is challenging for models without a deep understanding of context.

[1] https://swift.readthedocs.io/en/latest/Megatron-SWIFT/Quick-start.html

- **LogiQA** (Liu et al., 2020) is a dataset designed for logical reasoning. It consists of reading comprehension questions from professional logic exams, requiring the model to perform complex logical operations.
- **SciQ** (Welbl et al., 2017) is a question-answering dataset containing science exam questions from various domains, primarily focused on physics, chemistry, and biology.
- **GSM8K** (Cobbe et al., 2021) (Grade School Math 8K) is a dataset of high-quality, linguistically diverse grade school math word problems, designed to test multi-step mathematical reasoning.
- **HumanEval** (Chen et al., 2021) is a benchmark for evaluating code generation. It consists of 164 programming problems with function signatures, docstrings, and unit tests to assess the functional correctness of the generated code.
- **IFEval** (Zhou et al., 2023b) (Instruction Following Evaluation) is a benchmark for evaluating a model's ability to follow instructions. It consists of a set of prompts with explicit constraints that the model's response must adhere to.

L Layer-wise Expert Configuration

Figure 7 shows the exact configuration of ExpertWeaver at 25% sparsity. This configuration is derived from the layer-aware expert allocation strategy in § 3.2. The resulting U-shaped distribution, with more shared experts in the initial and final layers and more routed experts in the middle layers, demonstrates ExpertWeaver's ability to automatically tailor expert composition to each layer's specific needs, contrasting with the uniform approaches used by other methods.

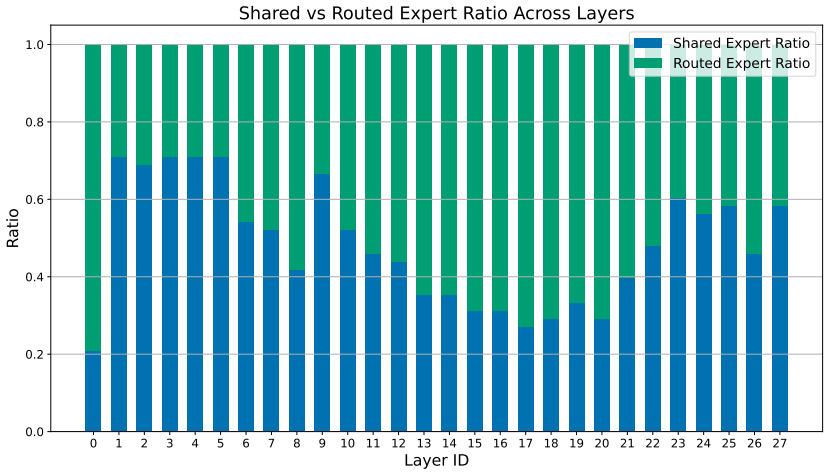


Figure 7: **Shared vs. Routed Expert Ratio Across Layers.** The figure shows the layer-wise configuration of shared and routed expert ratios for Qwen2.5-7B, as determined by ExpertWeaver.

M Expert Specialization

Figure 8 presents a detailed visualization of the expert routing patterns within our ExpertWeaver model. To create these heatmaps, we sampled 20 instances from subsets of the Red Pajama dataset (such as github and arxiv) at different model layers. The results shows that, in the shallow layers (e.g., Layer 0), tokens from various domains tend to activate a broad range of experts. Although many experts are utilized, the activation patterns between tasks are still distinguishable. The routing in deeper layers (e.g., Layer 23) becomes highly concentrated and specialized, with tokens from a specific domain consistently routed to a small and distinct set of experts. This results also reveal that shallow layers are responsible for processing common, foundational knowledge, while experts in deeper layers undergo functional differentiation to efficiently handle domain-specific information.

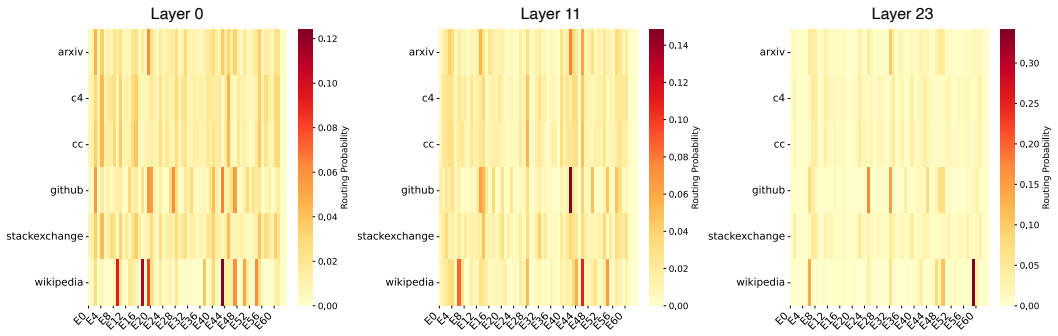


Figure 8: **Expert Specialization in the ExpertWeaver Model.**