Connito - Decentralized Composable MoE

Isabella Liu
isabella@connito.ai

George Kim
george@connito.ai

Abstract

Mixture-of-Experts (MoE) models provide a natural path for scaling open foundation models: they increase total parameter capacity while activating only a small subset of experts per token. However, adapting large MoE models remains difficult outside centralized high-bandwidth clusters, since standard distributed training methods require either dense synchronization, full-model residency, or expensive cross-device communication.

We introduce **Connito**, a decentralized framework for sparse MoE adaptation. Connito trains only a selected subset of target experts for a given domain, reducing the model state and communication payload required from each worker. This preserves part of the pretrained model's background capacity while keeping worker-side computation small.

Connito combines this sparse training architecture with a Proof-of-Loss incentive mechanism: workers submit updated target experts, validators evaluate their utility on held-out data, and useful updates are integrated back into the global MoE. Together, these components turn model improvement into a distributed process of expert-level contribution, evaluation, and integration. Connito offers a path toward continuously improving open MoE models whose capabilities compound through reusable expert updates rather than repeated centralized retraining.

1 Introduction

Frontier foundation models are increasingly expensive to train and adapt, often requiring centralized GPU clusters, high-bandwidth interconnects, and specialized engineering teams. This creates a barrier for open-model development: although open weights and open data are essential for broad scientific participation, the ability to improve frontier-scale models remains concentrated among a small number of organizations with sufficient compute and infrastructure. The ATOM Project argues that open model weights, data, and training processes are central artifacts for AI research, because they allow a wider community to inspect, reproduce, and improve foundation models rather than depending exclusively on closed systems (Wu, Rao, and W. Chen 2024).

Mixture-of-Experts (MoE) architectures offer a natural path toward more scalable open-model improvement. Unlike dense models, MoE models decouple total parameter capacity from active computation by routing each token to only a small subset of experts (Shazeer et al. 2017; Fedus, Zoph, and Shazeer 2022). This sparse structure makes MoE models inherently modular: experts can specialize in different domains, be selectively updated, and be composed back into a larger model (for a detailed survey of these modular characteristics and their historical milestones, see Appendix A). However, standard distributed training methods do not fully exploit this modularity. Dense data parallelism typically requires synchronizing a large fraction of the model state, while pipeline parallelism depends on transmitting activations and gradients between stages, which can become costly in open-internet settings with limited or variable bandwidth.

Connito is motivated by the observation that open-model scaling does not need to rely only on repeated centralized retraining. Instead, an MoE model can be improved through sparse, expert-level updates. For a target domain, only a small subset of experts may need to be trained, while the rest

of the model can remain frozen. Prior work such as ESFT shows that task-relevant experts can be selectively fine-tuned while preserving strong downstream performance (Wang et al. 2024). At the same time, training with only the selected experts can remove useful background computation that the original pretrained MoE would have provided.

Coordinating an open network of untrusted contributors requires a robust mechanism to evaluate their work. Building on this sparse training architecture, Connito introduces an **Intelligent Market** for decentralized MoE adaptation. Workers train selected target experts on local data and submit updated expert weights. Validators evaluate which submissions most improve the global model through a Proof-of-Loss mechanism, and useful updates are integrated back into the full foundation MoE. In this framework, the market is not merely selecting which experts are relevant to a dataset; it is selecting which worker contribution within a selected expert set provides the most valuable update to the global model. This turns open-model improvement into a distributed process of expert-level contribution, evaluation, and integration.

This paper makes the following contributions:

1. We propose a sparse MoE adaptation framework that trains only selected target experts, reducing the model state and communication payload required from each worker.
2. We define a decentralized Proof-of-Loss mechanism that scores worker submissions by their empirical contribution to held-out validation loss.
3. We establish a framework for how expert-level modularity enables an open model to improve continuously through reusable, composable expert updates.

2 Related Work

Connito builds on four lines of work: sparse MoE architectures, selective expert adaptation, composable expert training, and low-communication distributed optimization. Appendix A expands on these lines of work by characterizing four major milestones in the development of modular LLMs.

Sparse MoE architectures. Mixture-of-Experts models increase total parameter capacity while keeping per-token computation small by routing each token to a subset of experts (Shazeer et al. 2017; Fedus, Zoph, and Shazeer 2022). Recent MoE systems further separate shared and specialized computation. DeepSpeed-MoE studies residual and shared-expert variants for efficient MoE training and inference (Rajbhandari et al. 2022), while DeepSeekMoE introduces shared expert isolation to preserve common knowledge while encouraging routed experts to specialize (Dai et al. 2024). These architectures motivate Connito's view of MoE models as modular systems whose expert components can be selectively trained, compressed, and integrated.

Selective expert adaptation. Several methods show that MoE models can be adapted without updating all parameters (E. Hu et al. 2021). ESFT observes that downstream tasks activate only a small subset of experts and fine-tunes those task-relevant experts while freezing the rest of the model (Wang et al. 2024). DES-MoE extends this idea to multi-domain adaptation by using domain-specific expert activation patterns and gradient masking to reduce interference across domains (J. Li et al. 2025a). Connito follows this selective-update principle, but studies it in a decentralized setting where workers train only selected target experts.

Composable expert training. A second line of work studies how independently trained components can be combined into larger modular models. Branch-Train-Merge shows that expert language models can be trained independently and later combined through ensembling or parameter averaging (M. Li et al. 2022). Branch-Train-MiX constructs an MoE by training dense domain branches, installing their feed-forward blocks as experts, and learning a router over the resulting expert set (Sukhbaatar et al. 2024). FlexOLMo trains private experts against a shared public anchor, enabling experts to be plugged in or removed while maintaining compatibility (Shi et al. 2025). These works motivate Connito's use of a frozen shared anchor and expert-level integration.

Expert lifecycle management and MoE compression. Modular MoE models support expert-level lifecycle operations such as removal, pruning, skipping, and compression. FlexOLMo demonstrates expert opt-out at inference time (Shi et al. 2025), while task-specific expert pruning shows that

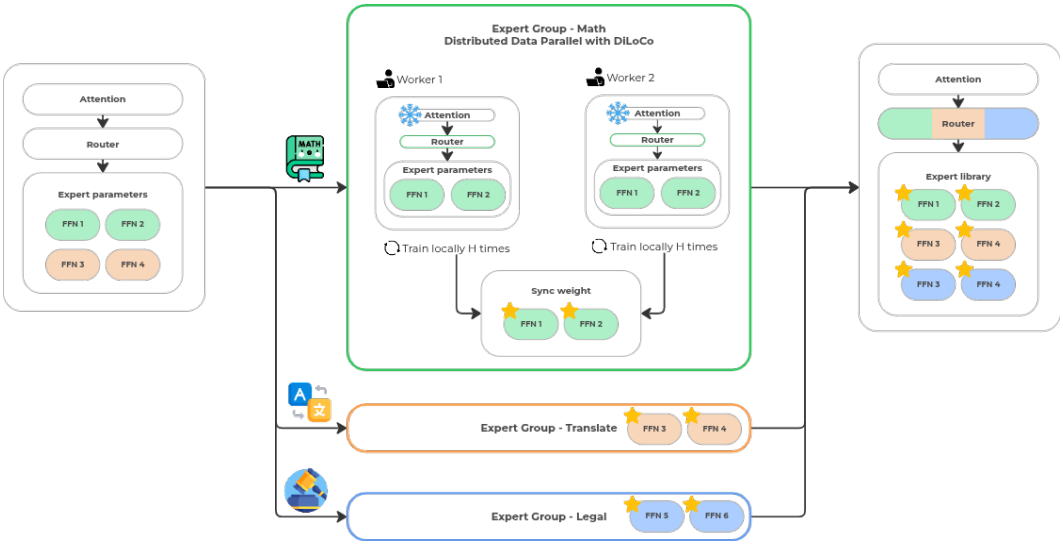


Figure 1: Overview of the Connito distributed training framework. The full MoE model is divided into task-specific expert groups (e.g., Math, Translate, Legal). Within each group, distributed workers perform multiple local optimization steps (H) on a selected subset of target experts (e.g., FFN 1 and FFN 2 for the Math domain) using a DiLoCo-style setup. Shared parameters such as attention layers remain frozen to ensure stability. After local training, the updated weights are synchronized across the worker group. Finally, the specialized experts are integrated into a global expert library for composable inference.

retaining only the most relevant expert per layer can preserve most downstream performance (T. Chen et al. 2022). Recent MoE compression methods further exploit expert redundancy through pruning, dynamic skipping, router-guided selection, and low-rank decomposition (Lu et al. 2024; Xie et al. 2024; Yang et al. 2024).

Low-communication and decentralized training. Connito also relates to distributed optimization methods that reduce synchronization overhead. DiLoCo shows that language models can be trained on disconnected workers with many local steps between synchronization rounds (Douillard et al. 2024). Connito extends these approaches by exploiting MoE structure directly: rather than synchronizing dense model replicas, workers train and transmit sparse expert subsets, while validators select useful updates through Proof-of-Loss (Foundation 2023).

Overall, prior work establishes that MoE experts can specialize, be selectively updated, be composed across training runs, and be compressed or removed. Connito is the first to synthesize these modular ML techniques with a cryptoeconomic incentive layer. By turning expert-level contribution, evaluation, and integration into a trustless protocol, Connito transforms static model adaptation into a continuous, decentralized engine for open-model improvement.

3 Distributed Training Framework

Connito trains a sparse MoE model by updating only selected target experts while keeping the shared parameters, router, and training-time context fixed. This section describes the machine-learning workflow assuming honest workers; the incentive and adversarial setting is handled separately in Section 4. The goal is to reduce the model state and communication payload required from each worker while preserving enough background computation for stable fine-tuning.

Let the full pretrained MoE model be

$$\Theta_0 = \{\theta_0, \Psi_0, W_0\}, \tag{1}$$

where θ_0 denotes shared non-expert parameters such as attention layers, embeddings, and layer normalization; Ψ_0 denotes the pretrained router; and W_0 denotes the full collection of pretrained

expert weights:

$$W_0 = \{W^0_{\ell,i} \mid \ell = 1, \ldots, L,\ i = 1, \ldots, M\}. \tag{2}$$

Here, L is the number of MoE layers and M is the number of routed experts per layer.

3.1 Phase 1: Target Expert Selection

The first phase selects the experts to be updated for a target domain. Following ESFT and DES-MoE, we use pretrained routing behavior to identify domain-relevant experts (Wang et al. 2024; J. Li et al. 2025a). Given target data $\mathcal{D}_{\text{new}}$, the protocol computes a selection map

$$S = \{S_\ell\}_{\ell=1}^{L}, \tag{3}$$

where each $S_\ell \subseteq \{1, \ldots, M\}$ denotes the target experts selected at layer ℓ. The selection rule may use routing probability mass, activation frequency, or a differential score comparing $\mathcal{D}_{\text{new}}$ against a general-domain dataset $\mathcal{D}_{\text{gen}}$.

The selected set S defines which experts receive gradients. It does not define the entire forward-pass model: non-selected experts may still contain useful background computation, even if they are not worth updating for the target domain.

3.2 Phase 2: Sparse Local Optimization

The third phase performs local training on the selected target experts. This follows the local-training pattern of DiLoCo and Branch-Train-Merge, where workers perform multiple local steps before global synchronization (Douillard et al. 2024; M. Li et al. 2022).

Let the initial trainable state be defined as $\Phi_S^{(0)} = W_S^0$. At global step t, a worker receives the current trainable target expert state

$$\Phi_S^{(t)} = \{W_{\ell,i}^{(t)} \mid i \in S_\ell,\ \ell = 1, \ldots, L\}. \tag{4}$$

The worker then performs H local optimization steps on its local data partition $\mathcal{D}_{\text{local}}$ using $\widetilde{\Theta}_0$ as the forward-pass model:

$$\Phi_{S,i}^{(t,H)} \leftarrow \text{InnerOpt}\left(\Phi_S^{(t)}, \mathcal{D}_{\text{local}}, H; \widetilde{\Theta}_0\right). \tag{5}$$

After local training, worker i submits only the updated target expert weights $\Phi_{S,i}^{(t,H)}$. No updates are submitted for the shared parameters or router.

3.3 Phase 3: Frozen Routing Anchor

The routing function is inherited from the pretrained model and remains fixed during local training. This follows ESFT's frozen-router setup and is also aligned with FlexOLMo's anchor-based expert training, where independently trained experts remain compatible because they are trained against a shared reference (Wang et al. 2024; Shi et al. 2025).

For an input token representation $x \in \mathbb{R}^d$ at layer ℓ, the pretrained router computes

$$g_{\ell,i}(x) = \text{Softmax}\left(\text{TopK}(x\Psi_{\ell,0}, k)\right)_i. \tag{6}$$

During compressed local training, the available expert set consists of the selected target experts S_ℓ. Freezing Ψ_0 makes worker updates comparable because all workers optimize against the same routing distribution and shared anchor.

3.4 Phase 4: Global Integration

The final phase aggregates worker submissions and integrates the resulting target expert update back into the full pretrained MoE. Assuming honest workers, the aggregator receives $\{\Phi_{S,i}^{(t,H)}\}_{i=1}^{N}$ and computes a weighted update to the target expert state. In the simplest case, the updated target experts are averaged:

$$\Phi_S^{(t+1)} = \sum_{i=1}^{N} a_i \Phi_{S,i}^{(t,H)}, \tag{7}$$

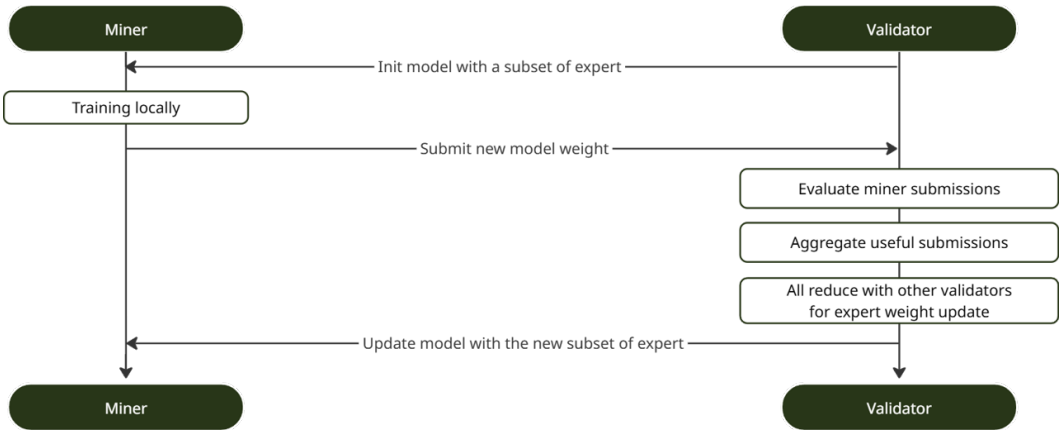


Figure 2: The decentralized training cycle in Connito. Validators initialize the network by distributing a specific subset of experts. Miners train these experts locally and submit their updated weights. Validators then evaluate the submissions via Proof-of-Loss, aggregate the most useful updates, synchronize the global model state across the validator network, and distribute the new expert subset for the next training cycle.

where a_i is an aggregation weight and $\sum_i a_i = 1$. Uniform averaging sets $a_i = 1/N$; validation-weighted aggregation can assign larger weights to submissions that improve held-out loss.

After training, the final target expert weights Φ_S^* are inserted back into the original full foundation model:

$$\Theta_{\text{final}} = \Theta_0[\Phi_S \leftarrow \Phi_S^*]. \tag{8}$$

Here, $\Theta_0[\Phi_S \leftarrow \Phi_S^*]$ denotes replacing the pretrained target experts in Θ_0 with their fine-tuned values.

4 Decentralized Incentive Mechanism

The distributed training framework in Section 3 assumes honest workers. In an open subnet, however, workers may have heterogeneous hardware, different data quality, and strategic incentives. Connito therefore separates the machine-learning update rule from the incentive layer. The role of the incentive mechanism is to evaluate submitted target-expert updates, reward useful work, and prevent low-effort or adversarial submissions from shaping the global model.

4.1 Miners and Validators

Connito separates participants into two roles. *Miners* perform local optimization on the selected target expert parameters Φ_S and submit updated target expert weights. *Validators* evaluate these submissions, assign rewards, and maintain the global model state.

Training proceeds in synchronous communication cycles. Each cycle contains a training window, a commit window, and a submit window. During training, miners optimize the selected target experts on local data. During commit, miners publish a cryptographic commitment to their checkpoint. During submit, miners reveal the corresponding checkpoint for evaluation. Submissions outside the expected window are rejected, keeping all participants aligned to the same global training state.

4.2 Proof-of-Loss Evaluation

Connito scores miner submissions using *Proof-of-Loss*: validators measure whether a submitted target-expert update improves the global model on a held-out validation subset. Let $\Phi_S^{(t)}$ denote the current global target expert state and let $\Phi_{S,i}^{(t,H)}$ denote miner i's submitted target expert weights after local training. The validator evaluates the validation loss after applying the submitted update to a

copy of the model:

$$\mathcal{L}_i = \mathcal{L}_{\text{val}}\left(\Theta_0[\Phi_S \leftarrow \Phi_{S,i}^{(t,H)}]\right). \tag{9}$$

The miner's utility is measured relative to the current global model:

$$u_i = \max\left(0,\ \mathcal{L}_{\text{val}}\left(\Theta_0[\Phi_S \leftarrow \Phi_S^{(t)}]\right) - \mathcal{L}_i\right). \tag{10}$$

Only positive loss reduction is rewarded. Submissions that do not improve validation loss receive zero utility.

The validation subset is deterministically sampled from the target distribution for each cycle. This ensures that validators evaluate the same objective and reduces the incentive to optimize for unrelated public benchmarks. In practice, this validation procedure is the core mechanism that turns local expert training into a competitive update-selection process.

4.3 Reward Assignment and Global Update Selection

Connito assigns rewards by ranking miners within each cycle by validation loss. Because each cycle uses a randomly sampled validation subset, raw loss values are not directly comparable across cycles; rankings provide a more stable relative signal. Miners in the top-K receive emissions, with higher-ranked submissions receiving more points.

Let $\mathcal{R}_t(i)$ be the rank of miner i in cycle t, where rank 1 denotes the lowest validation loss. Rewards from a validator are assigned by a decreasing rank function:

$$r_i^{(t)} = f(\mathcal{R}_t(i)), \tag{11}$$

where f is positive for top-K miners and zero otherwise.

Model integration is independent of reward assignment. The protocol may reward the top-K miners by rank, while aggregating a selected subset $\mathcal{T}_t$ of top-ranked submissions into the global model. Following DiLoCo-style outer optimization, integration is performed on pseudo-gradients computed from weight displacements, rather than on the submitted weights directly.

Let $\mathcal{T}_t$ denote the set of top-ranked submissions in cycle t. For each selected miner $i \in \mathcal{T}_t$, the aggregator computes a pseudo-gradient from the submitted target expert weights:

$$G_i^{(t)} = \Phi_S^{(t)} - \Phi_{S,i}^{(t,H)}. \tag{12}$$

The selected pseudo-gradients are then merged by weighted averaging:

$$G^{(t)} = \sum_{i \in \mathcal{T}_t} a_i G_i^{(t)}, \qquad \sum_{i \in \mathcal{T}_t} a_i = 1. \tag{13}$$

The outer optimizer is applied to the merged pseudo-gradient, not directly to the submitted weights:

$$\Phi_S^{(t+1)} \leftarrow \text{OuterOpt}\left(\Phi_S^{(t)}, G^{(t)}\right). \tag{14}$$

This design allows the subnet to reward multiple useful contributors while integrating only the update that provides the strongest measured improvement in a given cycle. The broader choice between top-ranked integration and weighted aggregation is an implementation parameter; both use the same Proof-of-Loss scoring signal.

4.4 Commit-Reveal and Evaluation Randomness

Permissionless training requires protection against free-riding, copying, and post-hoc optimization against the validation set. Connito uses a commit-reveal mechanism to bind miners to a checkpoint before the evaluation data is fully determined. During the commit window, miner i publishes

$$c_i = \text{SHA256}\left(\text{checkpoint}_i\right). \tag{15}$$

During the submit window, the miner reveals the checkpoint, and validators verify that its hash matches c_i.

The evaluation seed is generated from validator-provided randomness:

$$s_t = \mathrm{SHA256}\left(\mathrm{sort\_and\_concat}(s_1, \ldots, s_V)\right), \tag{16}$$

where s_v is validator v's random seed for the cycle. The seed s_t determines validator assignment and the held-out validation subset. Because no single miner controls all validator seeds, miners cannot reliably know the exact evaluation partition before committing to their checkpoint.

4.5 Security Goals

The incentive mechanism is designed around three practical security goals:

1. **Utility alignment:** rewards should track measurable improvement on held-out validation data from the target domain.
2. **Copy resistance:** commit-reveal should prevent miners from copying another miner's checkpoint after observing submissions.
3. **Evaluation unpredictability:** validation partitions should be deterministic for validators but difficult for miners to predict before committing.

Together, these mechanisms allow Connito to coordinate open participation without assuming that every worker is honest or equally capable. Miners compete to produce useful expert updates, validators measure the empirical value of those updates, and the global model evolves through submissions that improve held-out loss.

5 System Analysis

Connito is structurally designed to reduce the cost of distributed MoE adaptation by exploiting expert-level sparsity. In dense distributed training, each worker typically maintains and synchronizes a large fraction of the full model state. Connito alters the memory and communication bounds of this process by separating three distinct sources of efficiency: sparse payload communication, reduced optimizer-state memory, and update-filtering via Proof-of-Loss.

5.1 Communication Complexity and Sparse Upload

In Connito, the trainable and transmitted payload is strictly restricted to the selected target expert subset Φ_S. As a result, the communication cost per round scales with the size of Φ_S rather than the full pretrained model Θ_0.

Let $P_{\text{full}} = |\Theta_0|$ be the total parameter count of the MoE, and $P_S = |\Phi_S|$ be the parameter count of the selected target experts. The communication payload per worker per round is reduced proportionally by $O(P_S/P_{\text{full}})$. Furthermore, because workers perform H local optimization steps before committing an update, the amortized communication cost per local gradient step becomes $O(P_S/H)$. This compression is structural—arising from the MoE decomposition itself—and is fully complementary to quantization, gradient sparsification, and reduced synchronization frequency.

5.2 Worker-Side VRAM

Instead of instantiating all omitted non-target experts, workers use the compressed training model $\widetilde{\Theta}_0 = \{\theta_0, \Psi_0, W_S^0\}$.

Because θ_0, and Ψ_0 remain frozen, they do not receive gradients and require no optimizer-state memory (e.g., Adam moments). This reduces the active expert state required during local training while preserving part of the background computation that would otherwise be provided by the full frozen expert pool.

5.3 Proof-of-Loss as an Optimization Filter

Beyond hardware efficiency, Connito's incentive layer treats decentralized training as an update-selection problem. Proof-of-Loss evaluates submitted target expert weights by their effect on held-out validation loss. This mechanism provides a clear, empirical coordination signal for a heterogeneous

worker network, allowing the system to reward useful local work and systematically reject updates that do not improve the global model.

6 Future Outlook

Connito targets organizations that need deeper model customization than generic foundation models, prompt engineering, or lightweight PEFT methods can reliably provide. In domains such as legal, finance, healthcare, coding, compliance, and enterprise knowledge work, customers often need models to internalize domain-specific schemas, terminology, workflows, and reasoning patterns. Connito packages this need as a managed adaptation workflow: customers provide domain objectives or data, the network produces validated expert updates, and the resulting capabilities become reusable components rather than isolated one-off fine-tunes.

The long-term goal is to make open-model improvement compounding. As more domains are served, validated expert updates can form a shared expert library: a pool of specialized model components that can be selected, reused, and further improved for future tasks. This creates a flywheel in which each deployment can lower the cost and improve the quality of later deployments. Rather than treating model customization as bespoke consulting or repeated centralized retraining, Connito aims to turn it into durable infrastructure for continuously improving, community-built MoE models.

References

Chen, Tianyu et al. (2022). *Task-Specific Expert Pruning for Sparse Mixture-of-Experts*. arXiv: 2206.00277 [cs.LG]. URL: https://arxiv.org/abs/2206.00277.

Dai, Damai et al. (2024). *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. arXiv: 2401.06066 [cs.CL]. URL: https://arxiv.org/abs/2401.06066.

Douillard, Arthur et al. (2024). *DiLoCo: Distributed Low-Communication Training of Language Models*. arXiv: 2311.08105 [cs.LG]. URL: https://arxiv.org/abs/2311.08105.

Fedus, William, Barret Zoph, and Noam Shazeer (2022). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. arXiv: 2101.03961 [cs.LG]. URL: https://arxiv.org/abs/2101.03961.

Foundation, Opentensor (2023). *BitTensor: A Peer-to-Peer Intelligence Market*. Tech. rep. Bittensor Foundation. URL: https://bittensor.com/whitepaper.

Hu, Edward et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

Hu, Gang et al. (2025). *FFT-MoE: Efficient Federated Fine-Tuning for Foundation Models via Large-scale Sparse MoE under Heterogeneous Edge*. arXiv: 2508.18663 [cs.LG]. URL: https://arxiv.org/abs/2508.18663.

Hu, Guimin et al. (2026). *What Gets Activated: Uncovering Domain and Driver Experts in MoE Language Models*. arXiv: 2601.10159 [cs.CL]. URL: https://arxiv.org/abs/2601.10159.

Li, Junzhuo et al. (2025a). *Dynamic Expert Specialization: Towards Catastrophic Forgetting-Free Multi-Domain MoE Adaptation*. arXiv: 2509.16882 [cs.LG]. URL: https://arxiv.org/abs/2509.16882.

– (2025b). *Dynamic Expert Specialization: Towards Catastrophic Forgetting-Free Multi-Domain MoE Adaptation*. arXiv: 2509.16882 [cs.LG]. URL: https://arxiv.org/abs/2509.16882.

Li, Margaret et al. (2022). *Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models*. arXiv: 2208.03306 [cs.CL]. URL: https://arxiv.org/abs/2208.03306.

Lu, Xudong et al. (2024). "Not All Experts are Equal: Efficient Expert Pruning and Skipping for Mixture-of-Experts Large Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6159–6172. DOI: 10.18653/v1/2024.acl-long.334. URL: https://aclanthology.org/2024.acl-long.334/.

Rajbhandari, Samyam et al. (2022). *DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale*. arXiv: 2201.05596 [cs.LG]. URL: https://arxiv.org/abs/2201.05596.

Shazeer, Noam et al. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. arXiv: 1701.06538 [cs.LG]. URL: https://arxiv.org/abs/1701.06538.

Shi, Weijia et al. (2025). *FlexOlmo: Open Language Models for Flexible Data Use*. arXiv: 2507.07024 [cs.CL]. URL: https://arxiv.org/abs/2507.07024.

Sukhbaatar, Sainbayar et al. (2024). *Branch-Train-MiX: Mixing Expert LLMs into a Mixture-of-Experts LLM*. arXiv: 2403.07816 [cs.CL]. URL: https://arxiv.org/abs/2403.07816.

Wang, Zihan et al. (2024). *Let the Expert Stick to His Last: Expert-Specialized Fine-Tuning for Sparse Architectural Large Language Models*. arXiv: 2407.01906 [cs.CL]. URL: https://arxiv.org/abs/2407.01906.

Wu, Xiaofeng, Jia Rao, and Wei Chen (2024). *ATOM: Asynchronous Training of Massive Models for Deep Learning in a Decentralized Environment*. arXiv: 2403.10504 [cs.DC]. URL: https://arxiv.org/abs/2403.10504.

Xie, Yanyue et al. (2024). *MoE-Pruner: Pruning Mixture-of-Experts Large Language Model using the Hints from Its Router*. arXiv: 2410.12013 [cs.CL]. URL: https://arxiv.org/abs/2410.12013.

Yang, Cheng et al. (2024). "MoE-I2: Compressing Mixture of Experts Models through Inter-Expert Pruning and Intra-Expert Low-Rank Decomposition". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10456–10466. DOI: 10.18653/v1/2024.findings-emnlp.612. URL: https://aclanthology.org/2024.findings-emnlp.612/.

Yuan, Yichao, Lin Ma, and Nishil Talati (2025). *MoE-Lens: Towards the Hardware Limit of High-Throughput MoE LLM Serving Under Resource Constraints*. arXiv: 2504.09345 [cs.DC]. URL: https://arxiv.org/abs/2504.09345.

Zhang, Qizhen, Prajjwal Bhargava, et al. (2025). *BTS: Harmonizing Specialized Experts into a Generalist LLM*. arXiv: 2502.00075 [cs.CL]. URL: https://arxiv.org/abs/2502.00075.

Zhang, Qizhen, Nikolas Gritsch, et al. (2024). *BAM! Just Like That: Simple and Efficient Parameter Upcycling for Mixture of Experts*. arXiv: 2408.08274 [cs.LG]. URL: https://arxiv.org/abs/2408.08274.

Zhou, Yanqi et al. (2022). *Mixture-of-Experts with Expert Choice Routing*. arXiv: 2202.09368 [cs.LG]. URL: https://arxiv.org/abs/2202.09368.

A Model Modularity in Practice

We organize the literature around four milestones that collectively characterize modular large language models. We begin with Milestone 0, which establishes the core properties and empirical advantages of Mixture-of-Experts (MoE) architectures. Milestone 1 then examines selective adaptation, showing how partial expert updates enable efficient fine-tuning without modifying the full model. Building on this, Milestone 2 focuses on composable specialization, demonstrating how experts trained on different domains can be combined to improve cross-domain performance. Milestone 3 addresses the practical interfaces required for expert composition, including routing mechanisms and learned connectors that coordinate expert interaction. Finally, Milestone 4 explores expert lifecycle management, covering pruning, removal, and replacement of experts to support long-term model evolution.

A.1 Milestone 0: Demonstration of MoE Characteristic

A.1.1 Shared Expert (Hub)

A common design pattern in Mixture-of-Experts (MoE) models is to include a *shared expert*: a dense MLP (or a small set of experts) that is applied to *all* tokens, in addition to the sparsely routed experts. This "shared + routed" structure is often used to improve training stability and preserve a strong general-purpose pathway, while still allowing routed experts to specialize. Variants of this idea appear in several MoE systems, including Residual-MoE (Rajbhandari et al. 2022) and DeepSeekMoE (Dai et al. 2024).

Shared experts play a critical role in our modular model by providing a common representational space that aligns and integrates outputs from multiple experts.

Residual-MoE (DeepSpeed). Microsoft proposes *Residual-MoE* (Rajbhandari et al. 2022), where each MoE layer consists of a fixed dense MLP (the shared expert) together with a sparse set of routed experts. For each token, the model computes the shared expert output and the output of a single routed expert selected by the router, then combines them additively in a residual-style form. Intuitively, the dense pathway provides a stable general representation, while the routed expert acts as an "error-correction" term that captures token-specific specialization:

$$\mathrm{MoE}_\ell(h^\ell) = E_0(h^\ell) + E_k(h^\ell), \quad k = \arg\max_i g_i(h^\ell). \tag{17}$$

In their experiments, only a single expert is selected from the M available experts to enable direct comparison with top-2 MoE baselines.

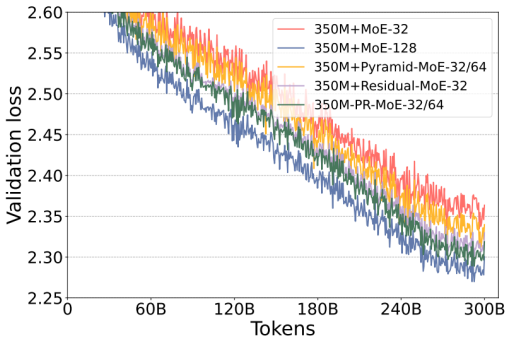


Figure 3: The plot shows that Residual MoE-32 (gray), which uses a shared expert and an expert selected by the router, is significantly more parameter-efficient than the standard MoE-32 (red) with top-2 expert selection.

Shared Expert Isolation (DeepSeekMoE). DeepSeekMoE introduces *Shared Expert Isolation* (Dai et al. 2024), which can be interpreted as a hub-style mechanism that separates common knowledge from specialized expertise. Concretely, each MoE layer designates K_s experts as *shared experts* that

are always activated for every token, independent of the router. The remaining experts are treated as routed specialists. This design aims to compress broadly useful "common knowledge" into the shared experts, reducing redundancy among routed experts and encouraging the routed experts to specialize on more distinctive patterns:

$$\mathrm{MoE}_\ell(h^{(\ell)}) = \sum_{i=1}^{K_s} E_{\ell,i}(h^{(\ell)}; W_{\ell,i}) + \sum_{i \in \mathrm{TopK}_{k-K_s}(p_\ell(\cdot|h^{(\ell)}))} p_\ell(i \mid h^{(\ell)})\, E_{\ell,i}(h^{(\ell)}; W_{\ell,i}). \tag{18}$$

Here, the experts with indices $i \leq K_s$ correspond to the shared experts (the "hub"), while the remaining experts are selected by top-k routing.

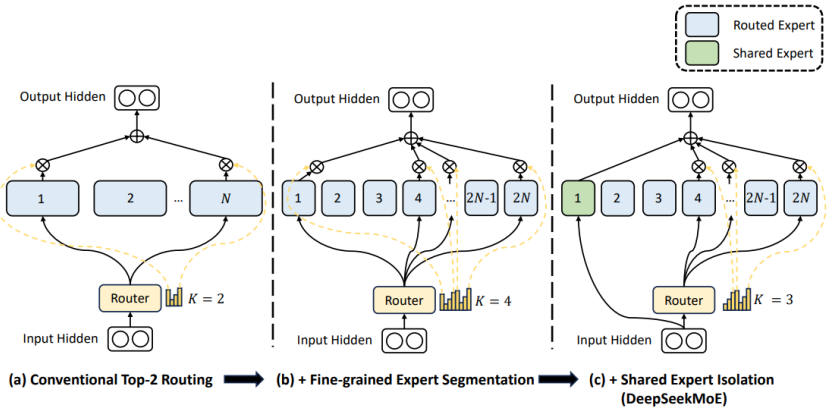


Figure 4: Illustration of DeepSeekMoE. Subfigure (a) showcases an MoE layer with the conventional top-2 routing strategy. Subfigure (b) illustrates the fine-grained expert segmentation strategy, where the model is decomposed into many smaller experts rather than a few large ones. Subsequently, subfigure (c) demonstrates the integration of the shared expert isolation strategy, constituting the complete DeepSeekMoE architecture. It is noteworthy that across these three architectures, the number of expert parameters and computational costs remain constant.

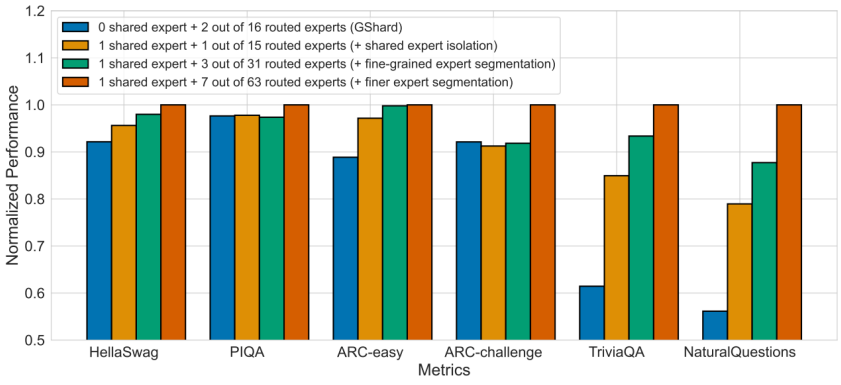


Figure 5: Ablation studies for DeepSeekMoE. All compared models have the same number of parameters and activated parameters. We can find that fine-grained expert segmentation and shared expert isolation both contribute to stronger overall performance.

Other occurrences. Similar shared-expert patterns have also been reported in other recent MoE architectures, such as Qwen3 MoE, although the exact implementation details may differ across model families.

A.1.2 The Role of Experts in LLMs

Recent work analyzes router behavior in Mixture-of-Experts (MoE) language models to characterize the functional roles of individual experts. In Guimin Hu et al. 2026, two primary roles are identified: *domain experts* and *driver experts*. Domain experts are detected using an entropy-based specialization metric that measures how concentrated an expert's routing distribution is over specific domains. Driver experts are identified via a causal-effect metric that perturbs routing probabilities and measures the resulting change in the model's output distribution.

Empirical results show that only a small fraction of experts exhibit strong domain specialization, while most experts display weak or negligible domain-specific behavior. Increasing router weights for identified domain and driver experts improves accuracy across models.

The identification of driver experts would be essential to our project, since it helps us to identify and build up the required shared experts from a model.

A.1.3 Low Expert Utilization

Several studies examine how experts are utilized during inference and find that MoE models rely on a small subset of experts. Yuan, Ma, and Talati 2025 analyzed routing distributions across domains and tracing expert contributions to the residual stream using early decoding, it shows that a single expert is often sufficient to approximate next-token prediction outputs. Overall, the results indicate that many experts contribute minimally to final predictions, motivating pruning and sparsification strategies that preserve accuracy while improving efficiency. And we can see similar level of low expert utilization from Wang et al. 2024

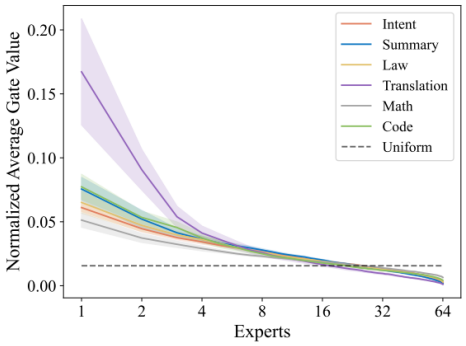


Figure 6: Top Expert distribution for specific tasks from Deepseek ESFT paper. Shaded areas represent variance across layers. The figure shows that few experts handle most gate values, highlighting expert specialization for different tasks.

This result is important because it suggests that fine-tuning and inference often do not require the full set of experts, enabling simpler, lower-cost distributed training designs with reduced overhead.

A.1.4 Relationships Between Experts

Recent analyses from Wang et al. 2024 further explore relationships between experts by examining co-activation and routing correlations within MoE layers. The results reveal structured patterns of redundancy and cooperation, where subsets of experts exhibit correlated behavior while others act more independently.

Understanding these relationships reveals expert redundancy and specialization boundaries, which can guide structured pruning, expert merging, and model compression. More broadly, it reinforces that MoE architectures naturally support distributed training by enabling modular updates and coordination across experts.

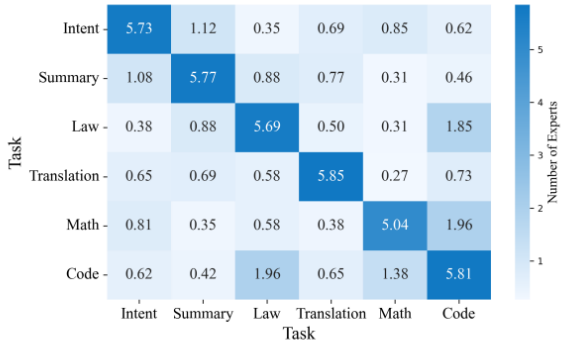


Figure 7: The average number of shared Top-6 routed experts across tasks from Deepseek ESFT paper. The values are averaged by layer, indicating that the sets of experts used for the same task are consistent while different tasks are distinct.

A.2 Milestone 1: Selective Adaptation via Partial Expert Updates

A defining advantage of modular architectures, such as Mixture-of-Experts (MoE) models, is their ability to adapt by updating only a subset of parameters—typically experts or routing components—rather than fine-tuning the entire model. Prior work shows that selectively adapting experts can improve domain specialization, reduce interference between tasks, and significantly lower training cost.

A.2.1 Domain-Composable Adaptation via Routing and Gradient Control

Dynamic Expert Specialization (DES-MoE). DES-MoE studies multi-domain adaptation under catastrophic forgetting and proposes mechanisms that encourage domain specialization while minimizing interference across domains. It introduces an Adaptive Lightweight Router (ALR) trained with a distillation objective, which keeps the fine-tuned router close to the pretrained routing behavior early in training and gradually allows domain-specific routing as adaptation proceeds. In addition, DES-MoE builds an expert–domain correlation map by tracking expert activation frequencies during a warm-up phase, then masks gradients so that domain updates primarily modify experts most associated with that domain. Together, these techniques enable stable domain adaptation and encourage experts to remain reusable across domains (J. Li et al. 2025b). Table 8 shows the effectiveness on the progressive parameter specialization schedule finetuning method compared to other finetuning methods like LoRA and ESFT(that will be mentioned in the later section).

	CLUEWSC	**TriviaQA**	**IFEval**	**MMLU**	**CEval**	**HellaSwag**	**ARC**	**Average**
Vanilla LM	81.5	67.7	42.5	57.5	59.9	74.0	53.7	62.4
Single-Domain Fine-Tuning (each model per domain)								
FFT	80.9 ± 1.1	65.9 ± 0.7	34.2 ± 4.1	55.5 ± 1.0	58.8 ± 0.9	67.9 ± 3.8	48.4 ± 2.4	58.8 ± 1.3
LoRA	74.3 ± 7.7	63.4 ± 5.4	38.7 ± 2.5	55.5 ± 1.2	57.0 ± 1.5	72.8 ± 1.9	51.8 ± 2.3	59.1 ± 2.5
ESFT-Token	80.9 ± 0.9	66.7 ± 1.8	40.7 ± 1.3	57.1 ± 0.5	59.6 ± 0.8	72.3 ± 3.6	52.9 ± 1.5	61.5 ± 1.1
ESFT-Gate	81.4 ± 1.1	66.5 ± 2.3	40.2 ± 1.5	57.0 ± 0.4	59.5 ± 0.8	68.2 ± 9.9	51.5 ± 3.1	60.6 ± 2.3
Mixed-Domain Fine-Tuning (unified multi-domain data)								
FFT (Mixed)	76.2	61.3	30.8	53.1	55.4	65.7	45.9	55.5
LoRA (Mixed)	73.5	60.8	34.6	54.3	54.9	70.2	48.7	56.7
ESFT-Token (Mixed)	78.4	63.5	36.1	55.7	56.3	69.8	49.2	58.4
ESFT-Gate (Mixed)	79.1	64.2	37.9	56.0	57.1	68.5	50.3	59.0
DES-MoE	**81.7**	**67.3**	**42.9**	**58.2**	**60.5**	**73.3**	**53.1**	**62.4**

Figure 8: General ability evaluation under mixed-domain fine-tuning form DES-MoE paper. Their method maintains original capabilities through dynamic expert isolation, while conventional approaches suffer from catastrophic forgetting when trained on mixed-domain data.

A.2.2 Dynamic Routing

Expert Choice Routing Expert Choice Routing redefines the routing paradigm by allowing each expert to select its top-k tokens, instead of requiring each token to choose a fixed number of experts (Zhou et al. 2022). This design eliminates load imbalance and provides experts with predictable work allocation, leading to more efficient utilization of compute resources. As a result, Expert Choice supports a variable number of experts per token and scales effectively as the total number of experts increases. Empirically, it achieves more than $2\times$ faster training convergence than Switch (top-1) and GShard (top-2) routing under the same compute budget, while maintaining strong model performance.

A.3 Milestone 2: Composable Specialization Across Experts

A second defining property of modular architectures is *compositionality*: experts trained independently on different domains can be combined to improve performance on the union of those domains, without retraining a single monolithic model. Recent MoE research explores this idea from multiple angles, including domain-aware expert adaptation, expert-level parameter-efficient fine-tuning, and explicit expert merging.

A.3.1 Task-Specific Expert Selection and Reuse

Expert-Specialized Fine-Tuning (ESFT). DeepSeek introduces Expert-Specialized Fine-Tuning (ESFT), motivated by the observation that MoE routing distributions are highly concentrated: for a given task, only a small subset of experts processes most tokens, and the identity of these experts varies substantially across tasks (Wang et al. 2024). ESFT selectively fine-tunes only the task-relevant experts in place while freezing all other experts and shared modules. This approach substantially reduces training time and storage (reported reductions up to $\sim$90%) while often matching or surpassing full fine-tuning on benchmarks such as GSM8K and MMLU. ESFT also outperforms LoRA-style PEFT baselines in this sparse setting, highlighting that MoE structure enables more natural modular adaptation (Wang et al. 2024).

	Math Ability		**Code Ability**		**Specialized Tasks**				
	MATH	GSM8K	Humaneval	MBPP	Intent	Summary	Law	Translation	Average
Vanilla Model	19.6	55.9	42.1	44.6	16.8	58.6	17.1	14.5	33.6
FFT	**23.4**	**66.4**	42.1	42.2	**78.8**	**69.4**	47.0	**38.4**	**51.0**
LoRA	20.6	58.9	39.6	**44.8**	67.8	64.7	39.7	23.1	44.9
ESFT-Token (Ours)	22.6	**66.0**	41.5	42.6	75.6	65.4	45.7	36.2	49.4
ESFT-Gate (Ours)	23.2	64.9	**43.3**	41.8	**78.6**	65.8	**49.1**	35.2	50.2

Figure 9: Main performance comparison across methods and tasks from Deepseek ESFT paper. This method provides a strong balance of performance across diverse tasks, rivaling FFT and surpassing LoRA, particularly in specialized task domains.

A.3.2 Expert Merging from Independently Trained Branches

Branch-Train-MiX (BTX). Branch-Train-MiX (Sukhbaatar et al. 2024) proposes an explicit expert-merging pipeline that constructs an MoE model from independently trained dense branches. Starting from a dense seed model (e.g., Llama-2 7B), it trains N branch models $\{E_{\ell,1} \forall \ell \in L, \ldots, E_{\ell,N} \forall \ell \in L\}$ in parallel, where each branch is trained on a domain dataset D_i using a standard language modeling objective. The resulting MoE is assembled by (i) extracting the feed-forward (FFN) weights from each branch and installing them as experts, and (ii) averaging the remaining non-FFN parameters across branches. Finally, an MoE fine-tuning stage learns token-level routing over the assembled experts. This design enables embarrassingly parallel training and demonstrates that experts trained in isolation can be combined into a single model that selects different experts at inference time at figure 11. Increasing the number of experts increases overall model capacity while keeping the number of active experts fixed, improving the efficiency–capacity tradeoff.

We discuss BTS in more detail in a later section; however, in figure 16 it demonstrates that composing independently trained experts can improve cross-domain performance.

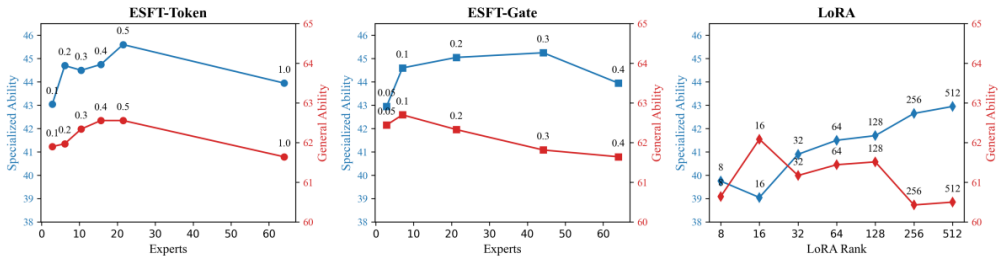


Figure 10: Comparison of three methods of Deepseek ESFT under different training efficiency settings on the Math task. The x-axis shows the average trainable experts per layer for ESFT and rank for LoRA, indicating the ratio of trained parameters. The y-axis represents specialized and general ability. Markers on the lines indicate p or rank values. ESFT consistently outperforms LoRA in both specialized and general ability

Figure 11: BTX routing decisions of the tokens at various layers to different experts (Wiki, Math, Code, LLaMa-2 7B) for different downstream tasks. The tasks are aggregated by domain: Code (Human Eval, MBPP), Math (GSM8K, MATH), World knowledge (Natural Questions, TriviaQA), and Reasoning (ARC-Easy, ARC-Challenge, SIQA, PIQA, and WinoGrande). We observe that Top-2 routing with load balancing (top) ensures a more uniform distribution of the load between experts compared to Top-2 without load balancing (bottom). On the other hand, even when experts are trained independently on disjoint domains, experts from other domains can still be automatically utilized.

A.3.3 Composable Experts for Private or Federated Settings

FlexOLMo. FlexOLMo (Shi et al. 2025) proposes an MoE-based framework that enables multiple data owners to train experts independently on private datasets (without data sharing or joint training) and later plug those experts into a single MoE model. A key component is a domain-informed router as shown in equation 19: experts are trained relative to a shared public anchor $\hat{r}_{\ell,\text{hub}}$, and each expert learns both FFN parameters and a router embedding $\hat{r}_{\ell,i}$ that reflects which tokens it should handle independently. Because all experts are trained with the same anchor, their router embeddings become comparable across domains, enabling expert **opt-in/opt-out** at inference without retraining. Empirically, FlexOLMo shows that performance improves as more experts are activated, up to a small number (e.g., four experts), after which gains saturate, indicating that sparse inference can remain efficient while benefiting from composed specialization.

	GSM8K	Flores En/Ru	Flores Ru/En	Ru-MGSM
Dense models				
Seed Model	10.5	22.8	32.8	12.8
Math Expert	*__20.5__	10.2	28.9	10.8
Russian Expert	9.48	*__32.3__	__34.6__	9.60
Seed Model (DM)	12.6	24.8	32.8	__14.0__
Expert upcycling				
BTX Sample	15.6	29.9	34.3	17.6
BTX Soft	17.6	30.6	34.5	17.6
BAM	__19.3__	30.9	34.5	*__18.4__
Expert merging				
Model Soup	17.5	14.7	32.3	13.2
BTM	20.5	*__32.3__	34.6	9.60
Expert Routing	9.48	*__32.3__	34.6	9.60
BAM Adapters	15.2	31.0	34.3	15.6
BTS	13.3	31.9	*__34.7__	16.0

Figure 12: Cross capability performance. We evaluate the seed model, Russian-language, and Math experts on the Russian subset of MGSM. Their performance was compared with expert merging and expert upcyling baselines trained with small amounts of in-domain data on Russian mathematics. We also continued pretraining the strongest dense model, the seed model on the same in domain data. We call the resulting baseline "Seed Model Data Matched (DM)". BTS outperforms the data-matched seed model, and achieves the best cross capability performance among all expert merging methods. This demonstrates that with only a small amount of in-domain data, BTS models can effectively learn how to combine expert capabilities.

Instead of learning r_ℓ jointly, FlexOlmo decomposes it into expert-specific router embeddings and builds the merged router by concatenating the independently learned rows $\{r_{\ell,i}\}_{i=1}^{n}$ together with the shared public row $r_{\ell,\text{pub}}$.

$$r_\ell = \begin{bmatrix} \hat{r}_{\ell,\text{hub}} \\ \hat{r}_{\ell,1} \\ \vdots \\ \hat{r}_{\ell,n} \end{bmatrix}, \qquad \hat{r}_{\ell,i} \in \mathbb{R}^d, \tag{19}$$

Federated MoE Fine-Tuning (FFT-MoE). FFT-MoE (Gang Hu et al. 2025) extends composable expert adaptation to heterogeneous federated learning. Instead of using LoRA adapters, it inserts sparse MoE adapters into a frozen foundation model as show in figure 13. Each client trains a lightweight router to activate a personalized subset of experts, while keeping the overall adapter structure compatible for aggregation. Importantly, FFT-MoE supports heterogeneous compute by allowing clients to choose different sparsity levels K_n: high-capacity clients can activate more experts, while resource-constrained clients activate fewer, without breaking model-size consistency. This adaptive activation improves scalability and efficiency while enabling collaborative training across heterogeneous devices. Figure 14 demonstrates the effectiveness of FFT-MoE in accuracy relative to fine-tuning with LoRA.

A.4 Milestone 3: Practical Interfaces for Expert Composition

Beyond the existence of specialized experts, modular language models may require *practical interfaces* that enable experts to be reliably selected, combined, and coordinated at inference time. Such interfaces determine how information flows between experts and can take the form of routing mechanisms, learned connector layers, or explicit constraints on expert interaction. This section shows that when attention parameters are jointly trained or unfrozen, expert composition is not merely

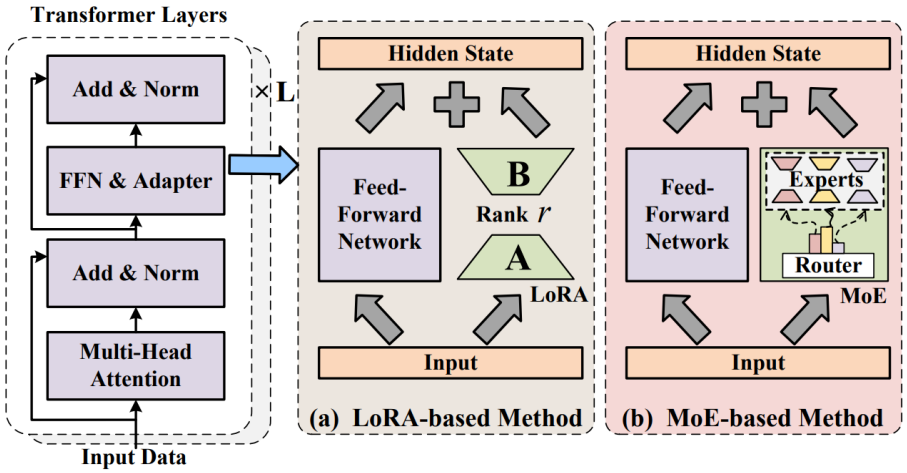
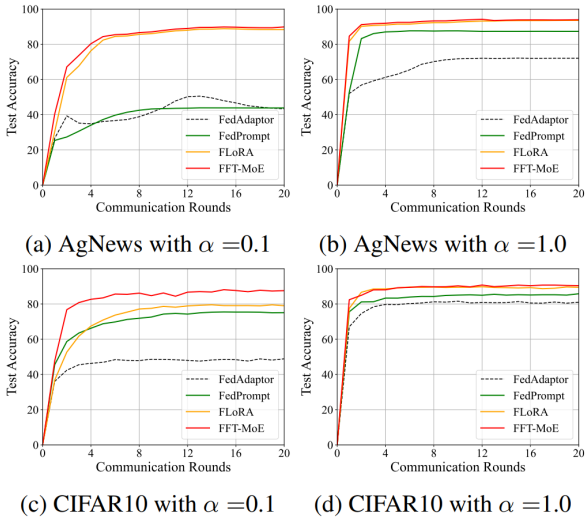


Figure 13: LoRA-Based Tuning vs MoE-based Tuning



(a) AgNews with α =0.1 (b) AgNews with α =1.0

(c) CIFAR10 with α =0.1 (d) CIFAR10 with α =1.0

Figure 14: Table from FFT-MoE paper to illustrate the effectiveness of training with FFT-MoE than LoRA.

a routing problem and often benefits from additional learned structure to harmonize independently trained components. In our setting, however, we do not plan to train or modify the attention layers, and therefore do not require these more complex composition interfaces. Nonetheless, these studies demonstrate that expert composition can be extended beyond routing alone, highlighting a viable path for future expansion should attention-level adaptation become necessary.

A.4.1 Stitching-Based Expert Composition

BTS: Harmonizing Specialized Experts into a Generalist LLM. Branch-Train-Stitch (BTS) presents an alternative to standard MoE routing by introducing a learned *stitching layer* that connects independently trained experts without token-level routing, as illustrated in figure 15 (Zhang, Bhargava, et al. 2025). In contrast to our approach, BTS fine-tunes both feed-forward and attention parameters. To manage the added complexity of unfreezing attention, BTS relies on intermediate connector layers rather than dynamic per-token expert selection to align and merge expert representations, mitigating mismatches that arise from independent expert training. This learned interface enables effective

cross-domain composition as shown in figure 17. Empirically, according to figure 16,BTS achieves the strongest generalist performance among several baselines, including the seed model, individual experts, expert merging, and expert upcycling, and is the only method to outperform all individual experts on certain tasks.

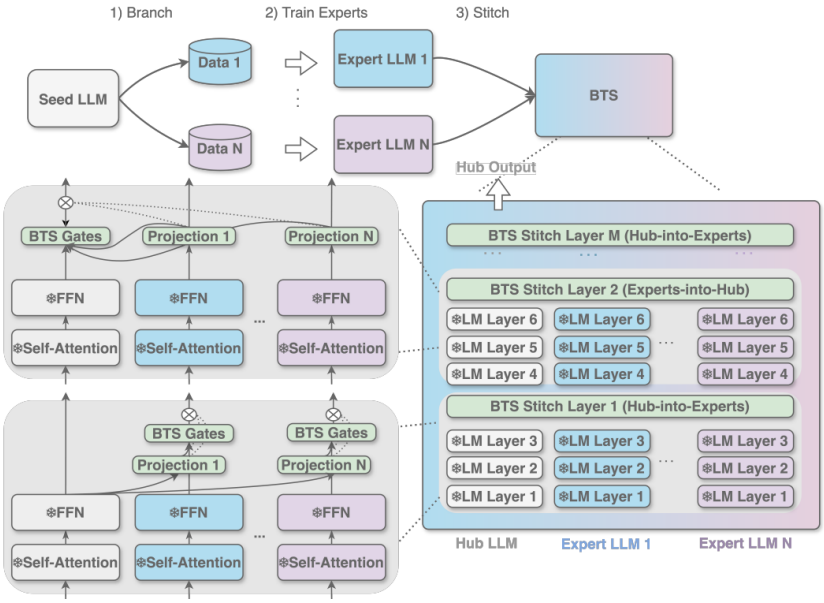


Figure 15: Overview of the BTS algorithm. BTS operates in three phases. Different colors correspond to different expert domains. 1) Branch: They begin with a pretrained seed model and create N copies of it. 2) Train Experts: Each copy is independently pretrained on its respective data mixture, resulting in specialized expert models. 2) Stitching: Stitch layers are inserted throughout the layers, alternating between the Experts-into-Hub stitch layer and the Hub-into-Experts stitch layer. Only the stitch layers are updated during this training phase. The BTS model always have a Experts-into-Hub stitch layer as the last layer, as the hub output is returned as the final BTS output.

A.4.2 Parameter Upcycling with Learned Connectors

BAM: Parameter Upcycling for Mixture-of-Experts Cohere's BAM upcycles multiple domain-specialized dense models into a unified MoE by promoting both feed-forward and attention layers to expert modules (Zhang, Gritsch, et al. 2024). During merging, all non-expert parameters are averaged across branches, while expert parameters are preserved. Unlike stitching-based approaches, BAM relies on token-level routing, and the resulting MoE model requires an additional fine-tuning stage to train the router and coordinate the upcycled experts. This design demonstrates that jointly upcycling attention and feed-forward components enables effective expert composition with minimal architectural changes.

A.5 Milestone 4: Expert Lifecycle Management

A final advantage of modular architectures is support for *expert lifecycle operations*, like expert pruning. These operations enable models to evolve over time while preserving efficiency, controllability, and deployment practicality.

A.5.1 Expert Removal and Data Opt-Out

FlexOLMo (Shi et al. 2025) provides a simple mechanism for dataset-level opt-out by allowing experts trained on specific data sources to be removed at inference time. For example, in figure 18 it shows that excluding the news expert leads to an expected performance drop on news-related tasks, while performance on unrelated tasks remains largely unchanged. This demonstrates that indepen-

	GSM8K	Flores En/Ru	Flores Ru/En	Ru-MGSM
Dense models				
Seed Model	10.5	22.8	32.8	12.8
Math Expert	*20.5	10.2	28.9	10.8
Russian Expert	9.48	*32.3	34.6	9.60
Seed Model (DM)	12.6	24.8	32.8	14.0
Expert upcycling				
BTX Sample	15.6	29.9	34.3	17.6
BTX Soft	17.6	30.6	34.5	17.6
BAM	19.3	30.9	34.5	*18.4
Expert merging				
Model Soup	17.5	14.7	32.3	13.2
BTM	20.5	*32.3	34.6	9.60
Expert Routing	9.48	*32.3	34.6	9.60
BAM Adapters	15.2	31.0	34.3	15.6
BTS	13.3	31.9	*34.7	16.0

Figure 16: Cross capability performance. We evaluate the seed model, Russian-language, and Math experts on the Russian subset of MGSM. Their performance was compared with expert merging and expert upcyling baselines trained with small amounts of in-domain data on Russian mathematics. We also continued pretraining the strongest dense model, the seed model on the same in domain data. We call the resulting baseline "Seed Model Data Matched (DM)". BTS outperforms the data-matched seed model, and achieves the best cross capability performance among all expert merging methods. This demonstrates that with only a small amount of in-domain data, BTS models can effectively learn how to combine expert capabilities.

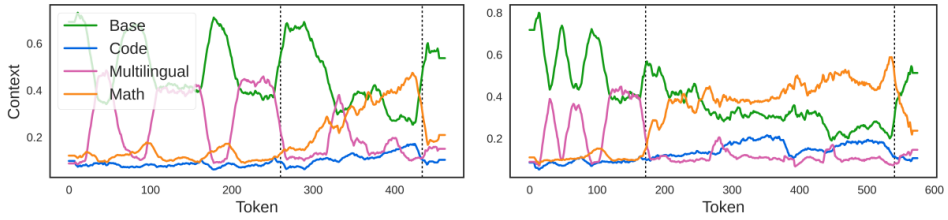


Figure 17: Visualization of the gate values of BTS's final stitch layer for context-switching sequences at inference time. These sequences are constructed by concatenating question-answer examples from Flores (3-shot), GSM8K (2-shot), and TriviaQA (2-shot), in that order, with dotted lines indicating task transitions. Each plot corresponds to a different randomly sampled prompt. This visualization highlights BTS's ability to dynamically adjust expert utilization based on token-level context.

dently trained experts can be treated as removable modules, enabling controllable specialization and supporting practical data governance constraints.

A.5.2 Expert Pruning and Model Densification

Recent work (T. Chen et al. 2022) also shows that MoE models can be pruned into smaller architectures while retaining most of their downstream performance. In task-specific expert pruning, the model is fine-tuned and progressively pruned so that only the most "professional" expert in each MoE layer is retained. As shown in figure 19, after pruning, the resulting single-expert (dense) model preserves approximately 99.3% of the performance of the original full MoE model across multiple downstream tasks, indicating substantial redundancy among experts and supporting efficient expert lifecycle management.

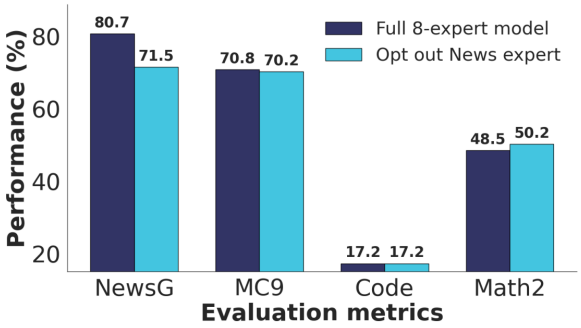


Figure 18: Experiment from Flexolmoe to opting out of news expert. Removing the news expert reduces performance on NewsG with minimal impact on other tasks.

Table 1: **The performance on the GLUE tasks for the dense models and the MoE models**, measured on the development sets. We report the average results by a set of seeds. **AVG** denotes the average score of all the tasks. Task-specific expert pruned MoE models with only single expert could achieve about 0.84 avg glue score than dense models.

Settings	CoLA	STS-B	MNLI-m	MNLI-mm	SST-2	QQP	QNLI	MRPC	AVG
Dense-ft	57.23	89.18	85.87	85.87	92.87	91.20	92.23	88.07	85.32
MoE-ft	63.75	89.37	86.93	86.93	94.03	91.37	92.90	89.13	86.80
Two-pass-staged-drop	58.32	88.64	86.30	85.90	90.00	90.50	91.40	86.90	84.75
Two-pass-eager-drop	60.89	89.02	86.10	86.20	92.60	90.60	91.50	87.72	85.58
MoE-staged-pruning	56.20	89.00	85.33	85.33	92.33	90.07	91.97	86.83	84.63
MoE-eager-pruning	**61.47**	**89.43**	**86.10**	**86.10**	**93.17**	**91.20**	**92.53**	**89.30**	**86.16**

Figure 19: The performance on the GLUE tasks for the dense models and the MoE models. AVG denotes the average score of all the tasks. Task-specific expert pruned MoE models with only single expert could achieve about 0.84 avg glue score than dense models.