A Transparent MoE Architecture for Sequence-Level Routing

Björn Pettersson [1 2] **Hosung Lee** [1 2] **Doan Khue Do** [1 2]

Abstract

We wanted to find out if we could facilitate strong human like domain expertise based on weak domain specialization show by (Fan et al., 2024), but still use large unstructured datasets like OpenWebText for training. The purpose behind this was that if we could gather expertise within experts this could show potential in terms of interpret ability for alignment purposes and provide a strong platform for modularity, or adding new pretrained experts post general training. This report focuses on a first initial step in this direction exploring if pretraining domain experts on subject domain specific datasets and transferring into a Mixutre Of Experts network, before general training on a large text dataset would facilitate a stronger version of the weak domain specialization observed by (Fan et al., 2024). We found that while for our experiment model this was indeed the case, this expertise seamed to peak early after about 1200 iteration and then fall, this raises questions about weather this effect would remain for a very large number of iterations. It is however clear that weight initialization has some effect on expert specialization and if further developed may be a useful technique for facilitating expert specialization in sequence MoE models.

1. Introduction

While the widely used token level routing within mixture of experts MoE has been shown to be an effective way to scale up large langue models , while keeping inference costs down . The MoE architecture itself could hold interesting properties for interpret ability if expert specialization could be domain specific, rather than based on token properties, letting us see in real time what experts are used to answer different types of prompts based on routing behavior. However this types of analysis becomes useful to the extent that experts actually becomes topic experts. Projects like Steer Moe (Feng et al., 2025) show that analysis of routing patterns and dynamic altering of them in regards to alignment can be an effective way of changing model behavior during inference, strengthening the argument around why expert

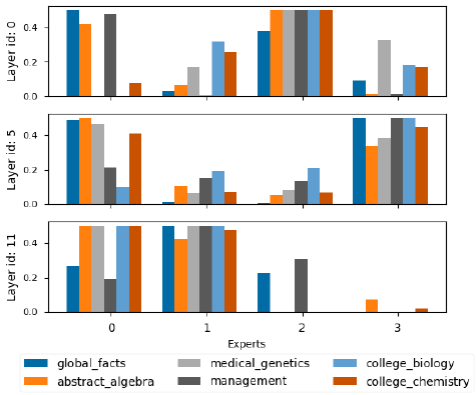

Figure 1. From (Fan et al., 2024) showing weak expert specialization organically emerging from sequence level routing

domain interpret ability might prove useful properties for this type of model behavior adjustments. Clear domain expert specialization could also show interesting properties in terms of model modularity and for mitigating catastrophic forgetting (Gururangan et al., 2021).

Gururangan et al. also show great potential for hard routing based on document content. However this approach heavily relies on labeled domain specific data, which is a major limitation for LLMs often trained on large unlabeled datasets such as openWebText.

Fan et al. (Fan et al., 2024) shows that sequence level routing organically gives rise to a weak form of this human like domain expertise, but not to an extent that would make this type of routing behavior interesting for an alignment point of view or as strong as seen by Gururangan et al..

We propose a middle ground where pretraining or weight up cycling of narrowly trained domain experts are combined in the MoE and then continuously trained on unstructured data. Our hypothesis is that initialization with pretrained weights may strengthen the weak organic domain specialization discovered by fan et al. and hence provide a step towards clearer more interpretable and scalable domain specialization in MoE architectures.

2. Related Works

The list of reviewed papers is shown on Table 1.

- (Fan et al., 2024) : Systematically explore how variations in the number of experts, gating mechanisms, and sparsity levels affect the model's performance and efficiency.
- (Sukhbaatar et al., 2024) : Introduce Branch-Train-MiX to combine multiple expert models with a MoE system, enabling the efficient training and utilization of specialized experts while maintaining the scalability and flexibility of the MoE architecture.
- (Gururangan et al., 2021) : Introduce DEMIX layers that allow for better specialization of models by domain, improving performance in complex, multi-domain tasks.
- (Nakamura et al., 2025) : Introduce a technique for training sparse MoE models named Drop-Up cycling, focusing on partial re-initialization to overcome training instability in sparse MoE systems.
- (Feng et al., 2025) : Propose Steer-MoE method that improves efficiency and performance of tasks requiring audio-language integration in a MoE framework.

Table 1. Paper Review on MoE Specialization and Routing Unit

PAPERS	ROUTING UNIT
FAN ET AL. (2024)	TOKEN/SEQUENCE
SUKHBAATAR ET AL. (2024)	TOKEN (TOP-2)
GURURANGAN ET AL. (2021)	SEQUENCE
NAKAMURA ET AL. (2025)	TOKEN (TOP-2)
FENG ET AL. (2025)	TOKEN

3. Method

3.1. Workflow Overview

Our study investigates whether sequence-level Mixture-of-Experts (MoE) architectures can exhibit meaningful domain specialization. To this end, we adopt a three-stage workflow to replicate the weak subject-dependent activation patterns reported by Fan et al. (2024) and to assess whether targeted specialization can amplify these effects under practical computational constraints. The overall pipeline consists of:

1. Baseline MoE training in general text (OpenWebText) to reproduce weak, originally emerging specialization.
2. Expert-wise narrow pre-training on domain-specific corpora (MMLU auxiliary train) to explicitly induce specialization.
3. Joint transfer learning on general data (OpenWebText) to examine whether induced specialization persists in routing behavior.

3.2. Baseline Model

The study by Fan et al. (Fan et al., 2024) demonstrated weak domain specialization in MoE models trained on sequence-level data, utilizing a modified GPT-2 Small architecture (nanoGPT with LoRA) (Karpathy, 2024). In this work, we aim to reproduce these findings on a scaled-down version, investigating whether subject-specific activation patterns indicative of weak specialization persist under tighter computational constraints.

While the original experiment was conducted on a single A100-SXM4-40GB GPU using the OpenWebText (Gokaslan & Cohen, 2019) dataset, our reproduction is adapted for an NVIDIA RTX 5080 (16GB VRAM). Consequently, we trained on a subset of OpenWebText. Furthermore, Fan et al. observed that routing behavior tends to stabilize early in the training process as shown in Figure. Leveraging this insight and aiming to derive a similar behavior, we reduced the training duration from 6,000 to 3,000 iterations, assuming that routing patterns can be effectively analyzed within this shorter time frame. A comprehensive comparison of hyperparameters and settings between the baseline and our implementation is presented in Table 2.

Table 2. Comparison between Fan et al. parameters with our experiment

Parameter	**Fan et al.**	**Ours**
Base model	GPT-2 Small	GPT-2 Small
Dropout	0.2	0.2
Learning Rate (LR)	9.6×10^{-4}	9.6×10^{-4}
LR (Minimum)	9.6×10^{-5}	9.6×10^{-5}
Weight Decay	0.5	0.5
iterations	6K	3K
Biases Enabled	True	True
Tokens / Iteration	1.05M	65,536
Gradient Accumulation	128	32
Batch Size	8	4
Sequence Length	1024	512
Total Tokens Seen	6B	$\approx$ 196M
No. of Experts (N)	4	4
Tokens / Parameter	20 (Chinchilla)	—
Routed Experts (K)	—	2
Load Balance Loss (λ)	0.01	0.0003
Tokenizer	GPT-2	GPT-2

Our implementation also builds upon the nanoGPT architecture, which integrates a LoRA extension. We introduced a `SequenceMoE` class to replace the standard `MLP` layers when the number of experts exceeds zero. Regarding the routing mechanism, we employ a top-k strategy with $k = 2$. For sequence-level routing specifically, we implemented a

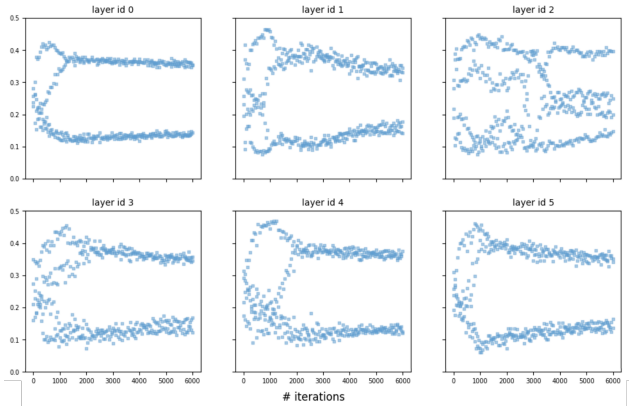

Figure 2. Figures from (Fan et al., 2024) showing expert activations from Layer-wise Sequence-level Top-2 routing with load balancing, explaining how routing behavior often stabilize early in training.

dual softmax mechanism, formulated as follows:

$$p_i(x) = \frac{e^{h_i(x)}}{\sum_j e^{h_j(x)}}, \quad y = \sum_{i=\tau} \frac{e^{p_i(x)}}{\sum_{j\in\tau} e^{p_j(x)}} E_i(x)$$

Here:

- x - is the input token or sequence embedding
- τ is the set of top-k indices ($K = 2$)
- $p_i(x)$ are the logits produced by the gating network as shown
- $E_i(x)$ is the output of the i-th expert
- y is the overall output, which is a weighted sum of the two selected experts.

The `SequenceMoE` module also handles load balancing, controlled by a hyperparameter λ. As indicated in the baseline study, load balancing is crucial for mitigating expert collapse and facilitating the early stabilization of experts—a key premise for the validity of our scaled-down experiment. Accordingly, we updated the total loss function as follows:

$$L_{total} = L_{LM} + \lambda L_{aux}$$

The original language modeling loss, cross entropy term L_{LM}, is measuring how well the model predicts the next token in a sequence:

$$L_{LM} = -\frac{1}{T}\sum_{t=1}^{T} \log(P(y_t \mid x_{1:t-1}))$$

Where $P(y_t \mid x_{1:t-1})$ is the probability assigned by the model to the correct next token y_t, and T is the total number of tokens being predicted (the sequence length). In the GPT class `Forward` function this is found as `main_loss = F.cross_entropy(logits.view(-1, logits.size(-1)), targets.view(-1), ignore_index=-1)`.

Where L_{aux} is a product of expert importance/traffic (P) and expert load/frequency of selection (F) summed over all experts (N):

$$L_{aux} = N \sum_{i=1}^{N} P_i F_i$$

Expert importance P_i, is the expected fraction of traffic routed to expert i. The average of the softmax probabilities $p_i(x)$ assigned to expert i across the entire batch B:

$$P_i = \frac{1}{\mid B \mid} \sum_{x\in B} p_i(x)$$

Expert Load (F_i) measures the actual fraction of sequences that selected expert i. It is the average of a binary indicator function ($1_{selected}$) across the batch B, which is 1 if expert i was one of the top K chosen experts for sequence x and 0 otherwise:

$$F_i = \frac{1}{\mid B \mid} \sum_{x\in B} 1_{selected}(x, i)$$

In our function `SequenceMoE.forward` we have P_i as `expert_importance`, F_i as `expert_load`, and L_{aux} as `load_balancing_loss`, and L_{total} as `total_loss`.

3.3. Proposed approach

3.3.1. Narrow Expert Pre-training

To strengthen domain specialization beyond the weak, organically emerging patterns in the baseline, we adopt a targeted narrow pre-training procedure. Each expert is isolated and trained exclusively on MMLU auxiliary train autolabelled (kz919, 2024), while all remaining experts are frozen.

This dataset contains additional question and answers similar to the standard MMLU dataset, with a domain label. We use the concatenated question answer strings sorted by domain to create 4 separate train test splits for the subjects; college chemistry, global facts, management and medical genetics. This is a subset of the subjects chosen for evaluation by (Fan et al., 2024). The choice of pretraining and evaluating on the same domains is somewhat arbitrary, but helps us evaluate if there is any correlation between expert selection and pretraining.

During pretraining, routing is fixed such that all sequences from a given subject are directed to the corresponding expert. This procedure yields a set of domain-specialized

expert modules, each encoding a coherent subset of domain knowledge.

This design is motivated by evidence that domain-specific initialization can enhance modularity, interpret ability, and resistance to catastrophic forgetting (Gururangan et al. 2021), while also aligning with our goal of increasing transparency in MoE routing behavior. The result of this stage is an MoE architecture with experts that are deliberately and explicitly differentiated along domain boundaries.

3.3.2. Joint Transfer Training on General Data

Upon completion of narrow pre-training to induce expert specialization, we reintegrate the experts into the full MoE transformer and resume training on the OpenWebText dataset. In this phase, experts are released from domain-specific constraints; instead, the gating mechanism reverts to dynamic Top-2 sequence-level routing. The training configurations remain identical to the baseline setup described in Section 3.2.

To measure specialization in a controlled environment, we perform evaluations using held-out MMLU questions (Hendrycks et al., 2021), enabling a direct comparison against both the baseline model and the pre-training configuration. Furthermore, to monitor the training progression and validate the efficacy of our proposed method, we evaluate the model's performance and routing behaviors every 600 iterations.

4. Experimental Results

4.1. Baseline Model

As shown in Figure 3, our baseline model, which is a reproduction of the Sequence-Level model by (Fan et al., 2024) in a smaller scale, shows that expert activation patterns looks different upon evaluations with different MMLU domains. This aligns well with the work by Fan et al (Fan et al., 2024), and shows that we have successfully reproduced the weak domain specialization observed.

Further more, with Figure 4, we show that we were able to observe similar routing stabilization over time to Fan et al, where most of the routing behavior is learned in an earlier stage. However, we do note that a minority of experts still appear to be on a moving trajectory on the last portion of the train, which is also the case for the reference paper (Fan et al., 2024).

It can be understood to be the case where expert routing asymptotically seems to stabilize, and yet, at the same time some experts may move away from this asymptotic state later in the training. For our comparison, however, we are mainly interested in the difference between models, and therefore consider this limitation minor. Overall we

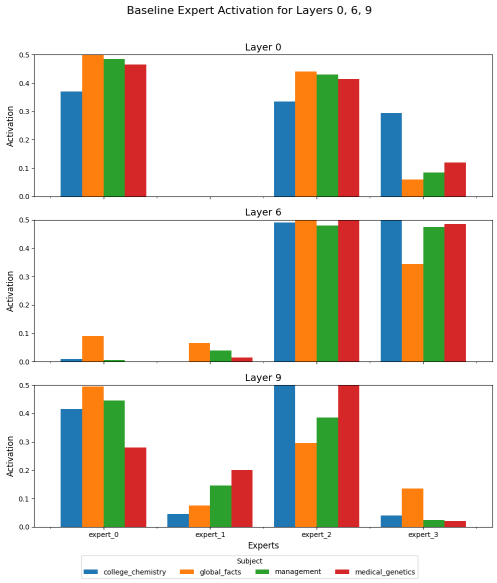

Figure 3. Expert activation on Layers 0, 6, 9 on our baseline model, resembling weak domain specialization similar to (Fan et al., 2024)

get a similar looking result to Fan et al, which is a good baseline to compare to. We can compare the results in figure 5 to those by Fan et al in figure 1. We have a significantly smaller lambda value than the original experiment, our smaller model is a probable cause for this. Lambda overall function in terms of load balancing and we see more convergence towards equal expert selection for a higher lambda value see appendix.

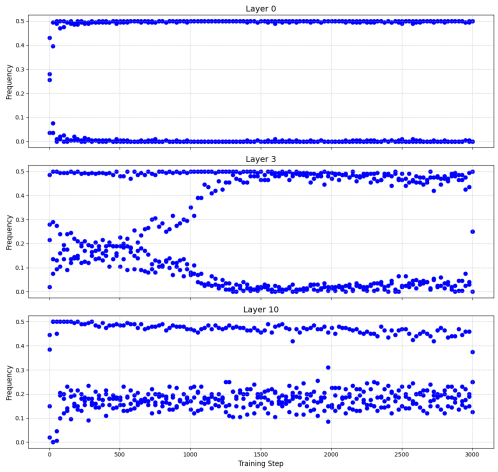

Figure 4. Expert Activation Frequency on Layer 0, 3, and 10 of the baseline model, resembling similar behaviors introduced in (Fan et al., 2024)

4.2. Transfer Learning Evaluation

In this section, we investigate whether our transfer model exhibits stronger domain specialization compared to Fan et al.. To this end, we compare the expert activation patterns of the transfer model, trained for 3000 iterations, with those of the baseline model obtained in the previous section. The results indicate that, after transfer, our model begins to show a more visible domain-dependent routing behavior, suggesting stronger specialization across the experts. Figure 5 reports the activation distribution across the four experts in layer 0 for several evaluation subjects, which we use as a proxy to quantify and visualize this emerging domain specialization.

In the baseline model (top panel), routing remains relatively balanced across experts, with weak domain specialization among experts. In contrast, the proposed model (bottom panel) exhibits subtle but systematic shifts in these patterns: certain subjects (e.g., *management*, *medical_genetics*) activate specific experts more frequently, while others redistribute their activations across different experts. Although this specialization is not extreme, the consistent, subject-dependent routing tendencies indicate that stronger expert specialization emerges in our model. To further analyze this, we propose an evaluation metric to numerically assess this behavior, which will be presented in the next section.

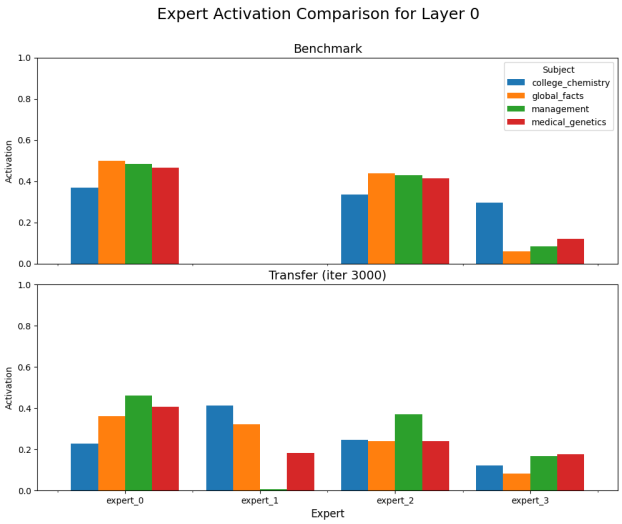

Figure 5. Expert activation results of Layer 0 on our baseline(top) and our proposed specialized model(bottom)

4.3. Evaluation metrics

Fan et al. visually inspects their plots to state weather they have weak specialization on the basis that the activation vector, consisting of activation percentages across experts given one subject, differs between subjects. This is clearly true in their case, however when measuring weather we have more or less of this weak specialization it is useful to use a numerical evaluation of this difference between activation vectors.

For this purpose we choose to sum the pairwise cosine distance between each of the 4 subject vectors for a given layer and to then average this number across our 12 layers. This give us a metric of how different the activation vectors or patterns are depending on the subject.

For robustness of the analysis we also include the Jensen Shanon Divergence between the activation vectors. For more details see the appendix.

4.4. Experiment Model

We obtain evaluations every 600 iterations of our experiment model up to 3000 iterations. Generally we observe similar result to the base model with some weak specialization seemingly appearing by subject specific expert selection patterns. See figure 5. It is visually difficult to determine the level of expert domain specialization we therefore continue to analyze this numerically in the following section.

4.5. Comparison Analysis

Given our results for our routing patterns find use the average pairwise cosine and Jensen–Shannon Divergence to investigate weather our pretrained model show more specialization than our benchmark see table 4.5.

Metric	CosDist	JSM
Bench	0.066874	0.033323
eval_1200	0.102351	0.047709
eval_1800	0.078089	0.038246
eval_2400	0.085084	0.043014
eval_3000	0.087617	0.042046
eval_600	0.054552	0.032388

Table 3. Evaluation Metrics Comparison

We plot the values in Table 4.5 in Figure 6 to get a picture of how further training on OpenWebText after transfer of pretrained experts affect the domain specialization of the experts. We plot the corresponding metric for expert specialization average pairwise cosine distance or Jensen–Shannon Divergence against the flat rate of expert specialization for the benchmark model after 3000 iterations.

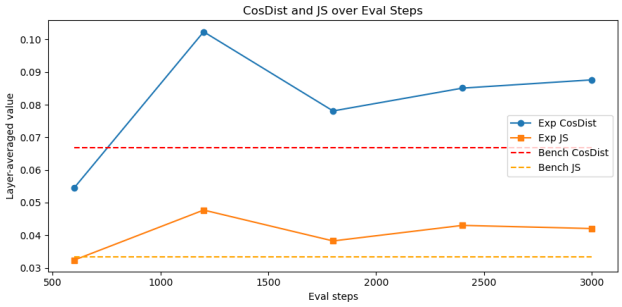

Figure 6. Analysis of metric for specialization over training steps, flat lines are benchmark model at 3000 steps.

5. Discussion

We observe that expert specialization peaks at 1200 iterations and then falls afterward. We also note that our pre-trained model, after only about 700 iterations, show stronger expert specialization than our benchmark.

We hypothesize that we can explain the peak and drop at 1200 iterations in terms of specialization with the fact that the router has to learn to use the experts for some number of iterations, and therefore the specialization increases. However, we suspect that training on the general dataset, OpenWebText, may dilute the expertise introduced by the pre-training over time and hence result in the eventual drop.

If it holds, this would disqualify the idea that we can create a "path of least resistance" where pretraining naturally leads to more of similar capabilities gathering in the related experts when general training is started after transferring the pretrained expert weights.

Another consideration in this regard is the datasets and evaluations used. It is possible that the subdomains used for pre-training were too narrowly scoped compared to the diversity of OpenWebText, yielding an initialization that was ill-suited for the broader data distribution. Consequently, a general training phase on a combined set of these pre-training subsets might have provided a superior initialization for inducing expert specialization. Alternatively, when utilizing OpenWebText, it might have been more effective to pre-train on broader generalized categories—such as STEM, Law, Medicine, or Code—where clearer domain boundaries are naturally expected within the corpus.

To generalize these findings, we hypothesize that given the large scale of the training dataset, pre-training primarily serves to establish distinct distributions for initialization values. Consequently, our experiment provides information on whether initializing experts with divergent distributions—as opposed to uniform ones—effectively promotes greater specialization.

Further more, our desire to see predictable specialization based on pretraining seems very unlikely. Especially with the substantial depth of the network. Intuitively, it may be advantageous for the model to have all kinds of mixed capabilities in different layers, leading us away from clear predicable specialization.

Overall we can conclude that while it seems like our experiment model does show more specialization because of its pre-training, it is unclear if initializing a general training on a large dataset such as OpenWebText will give rise to more expertise over a very large number of iterations. Consequently, the current degree of expert specialization seems too weak to satisfy the requirements for both transparency and modularity.

References

Fan, D., Messmer, B., and Jaggi, M. Towards an empirical understanding of moe design choices. arXiv preprint arXiv:2402.13089, 2024.

Feng, R., Zhang, B., Liang, S., and Yuan, Z. Steer-moe: Efficient audio-language alignment with a mixture-of-experts steering module. arXiv preprint arXiv:2510.13558, 2025.

Gokaslan, A. and Cohen, V. OpenWebText Corpus. Brown University, 2019. URL https://huggingface.co/datasets/Skylion007/openwebtext. Hugging Face Dataset. Original work retrieved from http://Skylion007.github.io/OpenWebTextCorpus.

Gururangan, S., Lewis, M., Holtzman, A., Smith, N. A., and Zettlemoyer, L. Demix layers: Disentangling domains for modular language modeling. arXiv preprint arXiv:2108.05036, 2021.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring Massive Multitask Language Understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2009.03300.

Karpathy, A. nanoGPT: The simplest, cleanest code of a gpt model, 2024. URL https://github.com/karpathy/nanoGPT. Source code. Accessed: 2025-12-04.

kz919. MMLU auxiliary train auto labelled. Hugging Face, 2024. URL https://huggingface.co/datasets/kz919/mmlu-auxiliary-train-auto-labelled. Derived dataset based on the MMLU benchmark (Hendrycks et al., 2021).

Nakamura, T., Akiba, T., Fujii, K., Oda, Y., Yokota, R., and Suzuki, J. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. arXiv preprint arXiv:2502.19261, 2025.

Sukhbaatar, S. et al. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. arXiv preprint arXiv:2403.07816, 2024.

Appendix

A. The Modified Forward Function

We feed in a batch x, which contains B samples of T tokens with embedding dimension C. First, we average over the dimension T, to get a batch of B average tokens of embedding dimension C. Based on these average tokens, our router assigns weights: we pass them through our dense FFN router layer to obtain the logits, which are then normalized with softmax to obtain the routing probabilities for each average token (the first part of the "double softmax"). Next, we use the `topk` function to obtain the selected top k experts and their routing weighting/probability, and we then normalize this routing probability over the k selected experts so it sums to 1 (the second part of the "double softmax").

With these routing decisions fixed, we then loop through the batch B times, and obtain, this time, the actual sequence (T, C) for each sample. In an inner loop, we loop through each of our selected experts (from earlier based on our average tokens), but now we route the actual sequences. For each sequence we then have a weighted average between the output from the top k experts (`output_i`), where the weights are given by the corresponding routing probabilities. We gather all the weighted average logits from passing the sequences through the expert FFNs (all `output_i`) into a final output matrix; this is the updated x vector after passing it through the MoE layer. This vector is later passed through a language modelling head (see the GPT class), that is, a linear layer to project the x vector from the embedding dimension C up to the vocabulary size. Finally, we take the softmax over this final vector and choose the next token with the highest probability.

Since running the above proposed code litterally is computationally expensive and we are limited to a 16gb gpu, we implemented an optimiseation. The code is almost fucntionally the same: a Sequence-level Mixture-of-Experts that (1) routes a sequence to top-k experts, (2) runs experts on the full sequence, (3) aggregates weighted expert outputs, and (4) returns an auxiliary load-balancing loss. For our purposes we consider this tradeoff close enough and not detremental to our results.

The general optimisation of model.py vectorizes the expert computation by bucketing sequences per expert and running each expert once on its assigned mini-batch, while moe-model.py runs experts per-sequence (nested loop over batch then top-k), causing repeated small forward passes. This vectorization reduces duplicated work and greatly improves throughput on GPUs when multiple sequences share experts.

B. The λ selection

An important hyperparameter in sequence-level MoE training is the weight λ applied to the load-balancing loss. This auxiliary term encourages the router to distribute traffic more evenly across experts, mitigating collapse into a single dominant expert. In our experiment, we try three values of λ including 0.01, 0.0001, and 0.0003. Across all tested settings, λ = 0.0003 consistently achieves the most desirable routing behavior, as shown in Fig. x. This value of λ provides the best balance between preventing expert collapse and avoiding over-regularization. When λ is too small (e.g., 0.0001), the load-balancing loss becomes too weak to counteract the router's natural tendency to favor a single expert, leading to uneven traffic distribution. Conversely, with a larger λ (e.g., 0.001), the router becomes overly constrained, which disrupts specialization and prevents experts from converging to meaningful behaviors.

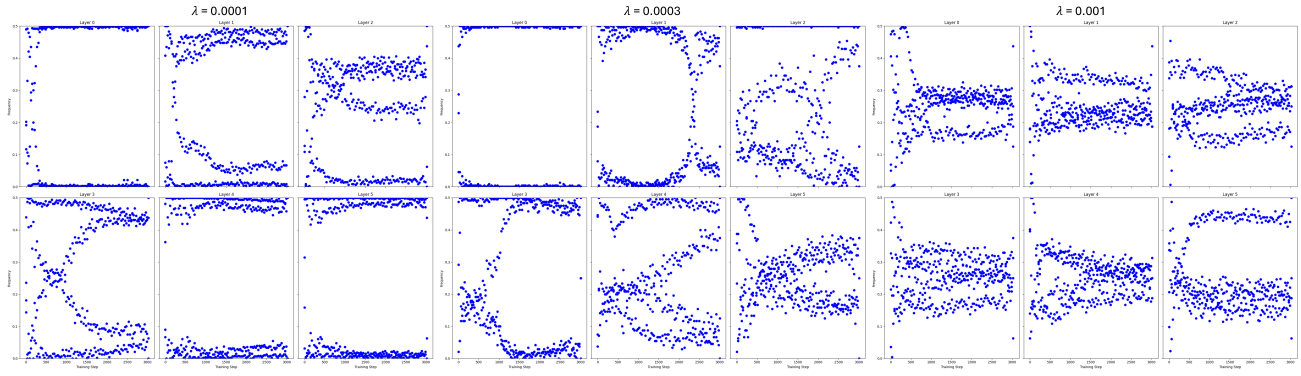

Figure 7. Expert Activation Frequency of the first six layers with three different values of λ

C. Evaluation Metrics

To quantify the degree of domain specialization in expert routing behavior, we compute four complementary metrics on the grouped activation frequencies per domain across all transformer layers.

C.1. Average Pairwise Cosine Distance

For each layer, we compute the mean activation vector for each domain, yielding a matrix $\mathbf{M} \in \mathbb{R}^{D \times E}$ where D is the number of domains and E is the number of experts. The average pairwise cosine distance measures how dissimilar these domain-specific activation patterns are:

$$\text{CosDist} = \frac{2}{D(D-1)} \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} \left(1 - \frac{\mathbf{m}_i \cdot \mathbf{m}_j}{\|\mathbf{m}_i\| \|\mathbf{m}_j\|}\right) \tag{1}$$

Higher values indicate greater differentiation in expert usage across domains, suggesting stronger specialization.

C.2. Average Pairwise Jensen-Shannon Divergence

The JS divergence measures distributional similarity between domain activation patterns:

$$\text{JS}(\mathbf{p}_i, \mathbf{p}_j) = \frac{1}{2} D_{KL}(\mathbf{p}_i \| \mathbf{m}) + \frac{1}{2} D_{KL}(\mathbf{p}_j \| \mathbf{m}) \tag{2}$$

where $\mathbf{m} = \frac{1}{2}(\mathbf{p}_i + \mathbf{p}_j)$ and D_{KL} is the Kullback-Leibler divergence. We report the average over all pairs:

$$\text{JS} = \frac{2}{D(D-1)} \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} \text{JS}(\mathbf{p}_i, \mathbf{p}_j)^2 \tag{3}$$

Higher JS divergence indicates more distinct routing strategies per domain.