The Thesis Committee for Jose L. Rojas
certifies that this is the approved version of the following thesis:

Analysis of Data Efficient and Budget Restricted Model Sparsity Training with Mixture of Experts

SUPERVISING COMMITTEE:

Atlas Wang, Supervisor

Aditya Akella, Reader

Analysis of Data Efficient and Budget Restricted Model Sparsity Training with Mixture of Experts

by

Jose L. Rojas

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

August 2024

Abstract

Analysis of Data Efficient and Budget Restricted Model Sparsity Training with Mixture of Experts

Jose L. Rojas, MSCompSci
The University of Texas at Austin, 2024

SUPERVISOR: Atlas Wang

Within machine learning, Mixture of Experts (MoE) has risen to become an architecture of choice in state of the art deep neural networks. MoEs have become popular due to their nature of exploiting conditional computing, conditionally activating certain neurons based on their inputs. The benefit of such an architecture is the ability to increase the model capacity linearly as a function of the number of experts chosen at training time while keeping the number of inference FLOPs constant. Pre-training MoE networks from scratch requires a large dataset and a lengthy training process to train the conditional gating networks. This thesis will investigate and analyze two data-efficient methods to train MoEs, namely Sparse Upcycling and MoEfication, that can reduce the amount of data and training time necessary. The contributions will include determining the best recipes for training and understand the benefits and drawbacks of each in comparison to training a dense model. Various experiments will be performed to validate techniques presented in the literature and document their effects on model performance and data efficiency. The key findings will show that utilizing these data-efficient techniques can leverage the expert specialization of MoEs in language models to garner improvements with accuracy metrics, reduced inference costs, and improved model compression.

Table of Contents

Chapter 1: Introduction

1.1 Preface

Mixture of Experts (MoE) has emerged as a powerful paradigm for enhancing model performance while constraining the computational complexity during inference. The core principle behind this architecture is to expand the model's parametric capacity by employing specialized neural structures, known as experts, that focus on distinct subsets of the input distribution through conditional computation. While MoE has demonstrated its potential in constructing extremely large models [Shazeer et al. (2017), Fedus et al. (2022b)], the training process documented in the literature can be resource-intensive [Fedus et al. (2022a)], often requiring substantial computational resources.

This thesis investigates two techniques for training MoE models with data-efficiency and computional budgets. By exploring various training methodologies, the research aims to provide an insightful summary of the strategies for optimizing the training process of MoEs, with a particular emphasis on their application to large language models (LLMs).

1.2 Motivations

Training machine learning models, especially large parametric models like Mixture of Experts (MoE), in a computationally budget-conscious manner offers several key advantages.

Firstly, MoE models are typically employed to scale up model capacity to billions or trillions of parameters [Fedus et al. (2022b)]. However, training such massive models requires immense computational resources that may not be available to researchers. By developing data-efficient and computationally efficient training techniques, it becomes feasible to train large MoE models using modest hardware resources and computational budgets resulting in reduced training time and costs.

Secondly, the development of state-of-the-art large language models (LLMs) and computer vision models has been limited to those fortunate organizations that have substantial resources and budgets. By enabling efficient training of MoE models on modest computational budgets, these techniques can democratize access to large models, fostering innovation and enabling a broader range of researchers and organizations to contribute to this area of research.

Additionally, the training of large neural networks is known to have a significant carbon footprint due to the immense computational resources required. Developing computationally efficient training methods for MoE models can help mitigate this environmental impact, aligning with the principles of green AI and sustainable machine learning practices.

Lastly, many real-world applications of machine learning models, such as edge computing, mobile devices, or embedded systems, operate under strict computational constraints. Developing techniques for smaller scale MoE models can enable their deployment in these resource-constrained environments, expanding their range of potential applications.

To summarize, while MoE models have primarily been explored for scaling up model capacity, the development of data-efficient and computationally efficient training techniques is crucial for making these models accessible and practical in scenarios with limited computational resources and budgets.

1.3 Challenges and Goals

Mixture of Experts has existed since the 1990s [Jacobs et al. (1991)]. However since 2017, new research was done to apply this approach to existing foundation models, particularly transformer based deep neural networks [Vaswani et al. (2017)]. Despite the renewed interest and novel approaches to utilize this architecture with the notions of scaling sparse models, a thorough review of various training recipes for building models with MoEs within limited computational budgets has not been provided in research literature. Bearing this in mind, the goal of this thesis is to compile, review, validate, and de-mystify the training of MoEs, with the primary means of doing so within a strict computational budget.

As a student researcher, the author of this thesis is constrained by computational resources in a realm where substatial amounts of GPU compute must be utilized. This poses a great challenge. Nonetheless, where there is a challenge, there are opportunities to learn from new techniques from a novel perspective. Additionally, this constraint will eliminate certain procedures that will be too resource intensive, thus reducing the scope of the thesis investigation appropriately.

Within machine learning research, an important consideration is the effects of hyperparameters on a particular model. An additional major challenge that runs parallel to the level of resource constraints is the sheer number of experiments that can be run to isolate the model behavior when a certain hyperparameter is modified. While there may be many interesting experiments to explore, due to time and resource constraints, a limited number of hyperparameters and combinations will need to be prioritized and investigated. The rationale behind such decisions will be elucidated within the relevant sections.

The questions that will be investigated and answered in this work are: 1) what data-efficient methods can be used to train an MoE within a restricted computional budget? 2) what are some design factors that should be considered in training the MoEs and their effects? 3) what are the advantages of using these data-effecient methods compared to training a dense model and when should they be used?

To summarize, the high level contributions of this thesis are to analyze data-efficient training processes that researchers have made on MoEs, validating the performance of these techniques, and determining when best to apply them. In this review, experiments will be performed to examine the factors that various hyperparameters have on the model and training processes. The challenges toward achieving this goal will include avoiding the constraints of large computational resources and combinatorial explosions of hyperparameters.

Chapter 2: Related Work

To better understand the experiments that are presented in this work, a foundation must be established regarding prior work in the research literature. This section will discuss and elaborate on the most salient research and topics.

2.1 Mixture of Experts

Mixture of Experts (MoE) was first introduced in Jacobs et al. (1991) as a concept derived from Gaussian Mixture Models (GMM) where a larger network is composed of distinct subnetworks, named "experts", that are merged together as a linear combination of their activations. Each expert is trained to specialize in a subregion of the multidimensional input space through the use of a routing network, also referred to as a gating network. The gating network could be an arbitrary function, but in most of the literature, it is a function based on the input into the expert networks. The gating network, which could also be a learnable function, maps subregions of input space into a selection process that determines the weights for each expert in the linear combination. A formulation is provided in the equation 2.1, where $G(x)_i$ is the gating network output and $E(x)_i$ is the expert output for each expert i.

$$y = \sum_{i=1}^{n} G(x)_i \cdot E(x)_i \tag{2.1}$$

2.2 Conditional Computing and Sparse Models

In the earlier works, inputs were passed through all of the expert networks and then combined. In the deep learning era, Shazeer et al. (2017) introduced the concept of the Mixture of Experts layer and sparsity utilizing conditional computation

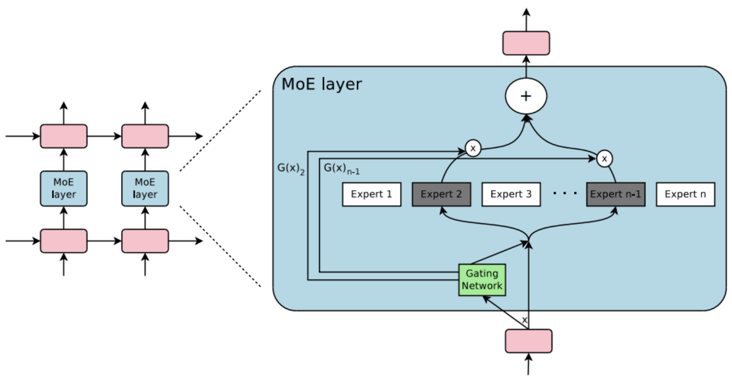

Figure 2.1: Conceptual Diagram of Sparse MoE [Source: Shazeer et al. (2017)]

[Bengio et al. (2013)]. Conditional computation seeks to reduce the overall inference cost of a model relative to the number of parameters. Instead of utilizing all of the parameters in the model, only sparse portions of the model are activated during inference. In the context of MoE models, a sparse mixture of experts only activates a subset of k experts out of n possible experts, typically by using the top k scoring weights generated by the gating network. The benefit of this technique is to allow for larger model capacity through training specialized experts while limiting the inference FLOP costs as only a specified number experts will be activated. The total number of sparse parameters activated in the model is referred to as the model's total number of effective parameters [Clark et al. (2022)].

Additional work has shown that sparsity in feed forward networks (FFNs) can be trained in an equivalent manner using MoEs. MoEfication [Zhang et al. (2021)] allows for sparse models to be reconstructed as MoEs by slicing their FFNs and running two algorithms to group neurons with similar activations together into the same experts and then separately training a router to select the experts based on the inputs. In SMoE-Dropout [Chen et al. (2023)], a similar process is employed to convert FFN into MoEs with a random routing function. Using curriculum training to increase the number of activated experts over time, this procedure produces an effect similar to random neuron dropout [Hinton et al. (2012)] and allows the network to be

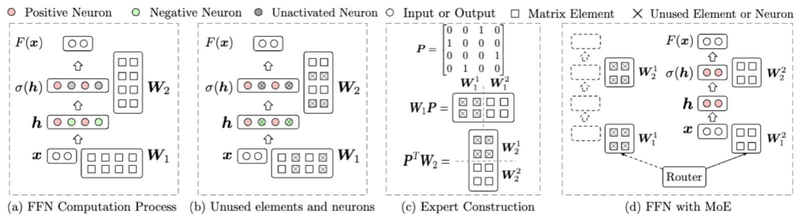

Figure 2.2: Conceptual Diagram of MoEfication [Source: Zhang et al. (2021)]

"self-slimming", creating a tradeoff between model accuracy and inference FLOPs.

2.3 Challenges with Sparse Mixture of Experts

Training sparse MoEs exhibits a variety of challenges. Unstable training is a common one [Zoph et al. (2022)] in which case MoEs can overfit as they learn a subset of the input data faster than a dense model would. Various papers [Fedus et al. (2022b)] discuss aspects of training imbalances when experts are overly selected. A capacity factor and load balancing are suggested workarounds for such problems. The gating network's ability to route inputs to the correct expert is critical, otherwise leading to poor specialization [Chen et al. (2023)]. This lack of specialization can be caused by a representational collapse [Chi et al. (2022)] in the routing network due the low rank nature of the parameter matrix $R^{N \times d}$, where $N \ll d$. Additionally, if the number of activated experts is low, the selection process tends to reinforce the routing network gradient towards the same experts being selected, reducing the specialization during training and increasing the load imbalance across experts.

Architecturally, MoEs can be complex to design and scale [Fedus et al. (2022a)]. Various papers discuss aspects of balancing experts across devices to maximize GPU throughput and expert specialization [Lepikhin et al. (2020); Dai et al. (2024)], however device sharding is not the focus of this particular thesis. However there are additional design choices that can add complexity that will be addressed here. Namely, the number of experts is a large consideration as adding experts will have diminishing returns [Komatsuzaki et al. (2023)]. Also the number of expert layers has been researched in a variety of papers, some recommending all the layers in the deep model [Jiang et al. (2024), xai-org] to others only recommending the last few layers [Komatsuzaki et al. (2023)]. These challenges will be addressed further in this work.

2.4 Gating Network Training

The gating network in MoEs are essentially important to enable specialization of the expert subnetworks and thus maximize the entire model's performance [Fedus et al. (2022a)]. By mapping the input space to the expert space appropriately during training, the experts can focus their parameters specifically on this subdomain, utilizing the extra model capacity provided by the additional experts efficiently.

As presented in SMoE [Shazeer et al. (2017)], the gating network can be learnable with weights using a matrix product and then sparsely activated with a TopK followed by a softmax, shown in Equation 2.2. The TopK only selects k experts out of N, while the rest receive zero weights. The softmax offers a non linear normalization to rescale the remaining experts, however this only properly provides a trainable gradient when k $>$ 1. In another work, switch routing [Fedus et al. (2022b)] was proposed which inverts the two operations and selects only one expert. This was a critical change as it allows sparse MoEs to have the same effective parameters as a dense model, shown in Equation 2.3.

$$G(x) = Softmax(KeepTopK(H(x), k)) \tag{2.2}$$

$$G(x) = KeepTopK(Softmax(H(x)), k) \tag{2.3}$$

$$KeepTopK(v, k)_i = \begin{cases} v_i, & \text{if } v_i \text{ is in the top k elements of v.} \\ -\infty, & \text{otherwise} \end{cases} \tag{2.4}$$

In a recent work, Mixtral [Jiang et al. (2024)] utilizes what will be called a "renormalization" gate in this thesis, one that peforms a normalization operation, after the TopK selection. This forces the experts to maximize the gradient signal through both the expert weights and the gating network, incorporating a faster training. However this only suitable when two or more experts are activated, as the Softmax function generates a zero gradient otherwise.

Alternatively to a trainable mapping of inputs to experts, referred to as "token-choice", there is another gating network named "expert-choice" [Zhou et al. (2022)] which seeks to assign a mapping of experts to tokens. This ensures all expert networks are utilized during training and inference by selecting preferential segments of the input space for specialization. Conditional computing for this gating strategy is an input filtering process; reducing the computational cost of utilizing all of the experts is achieved by learning which relevant subsections of the input to process per expert.

While the network can have additional layers for further specialization and stability [Chi et al. (2022); Shen et al. (2023a)], a core focus is placed on adjusting the objective function towards this end. Various auxiliary loss functions have been proposed: importance [Shazeer et al. (2017)], load balancing [Fedus et al. (2022b)], z-loss [Zoph et al. (2022)], and mutual information loss [Shen et al. (2023b)] are some examples, all which strive to address this problem. Another common technique to encourage specialization is using noise within the router as a form of regularization [Shazeer et al. (2017); Fedus et al. (2022b)]. X-layer [Chi et al. (2022)] uses low-rank adaptation and normalization layers as a means to address the representational collapse problem.

2.5 Data Efficient Training Methods

As training deep neural networks can be quite expensive, there is always a preference to train them as efficiently as possible. One approach is via continual learning [Gupta et al. (2023)]; instead of training a MoE model from scratch, one

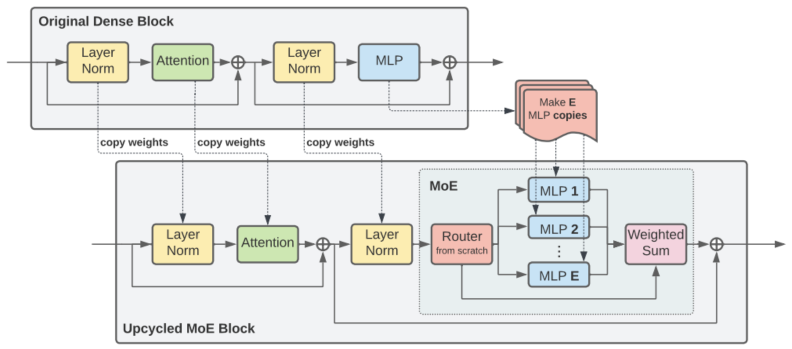

Figure 2.3: Conceptual Diagram of Sparse Upcycling [Source: Komatsuzaki et al. (2023)]

can be trained via partially trained dense model. This technique is referred to as "sparse upcycling" [Komatsuzaki et al. (2023)] and is performed by cloning the dense model MLP parameters N times to initialize the expert network weights, allowing the experts to diverge from a common dense checkpoint, saving pre-training time. The MoE specialization can then be further capitalized during fine-tuning, which is typically a much shorter training cycle.

The previously mentioned MoEfication [Zhang et al. (2021)] technique is another example of a data efficient training method for MoEs. The training weights of each FFN in the MoE layer can be repurposed, divided across N number of experts. As a Sparse MoE is a linear combination of the expert weights, the gating network learns what portions of the pre-trained weights are the most specialized to reduce the loss and these networks are further refined. Given the experts are only using a fraction of their original FFN weights, this model is expected to only approximate dense model performance.

Parameter-efficient fine-tuning (PEFT) [Xu et al. (2023)] has recently risen as a dominant technique to fine-tune large language models. These techniques typically rely on adapters, such as low-rank adaptation (LoRA) [Hu et al. (2021)], to transform a large weight matrix into a smaller pair of matrices that can be multiplied together to reduce the parameter count and offer memory-efficient fine-tuning. The idea has been applied to MoEs by using a mixture of adapters with a routing network to adapt the MLP parameters of any particular layer [Wang et al. (2022)]. Other variations on this design include Mixture-of-Vectors and Mixture of LoRA Experts [Zadouri et al. (2023)]. All of these techniques can be viewed as a lightweight replacement for a full MoE training.

2.6 Implementations in LLMs Models

Up to this point, only the general structure for MoEs have been referenced, but it is important to note their concrete applications. LSTMs were first used to implement the MoE layers in modern deep learning [Shazeer et al. (2017)]. Soon after, MoEs were applied to the transformer model by replacing the MLP parameters after the self-attention heads into MoEs subnetworks. While convolutional neural networks also have been demonstrated using MoEs [Chowdhury et al. (2023)], the transformer remains the most popular network to employ this architecture.

Large language models have experienced the most success by adopting the MoE architecture. Due to the large parameter counts required to scale up emergent behaviors [Wei et al. (2022)], MoEs offer an efficient path for training and inference savings. Various state of the art LLM implementations with MoEs include: Mixtral [Jiang et al. (2024)], Llama-MoE [Zhu et al. (2024)], Grok [xai-org], DeepSeek [Dai et al. (2024)], ModuleFormer [Shen et al. (2023b)], and also GPT-4 has been speculated to use MoEs. In many of these models, the number of activated experts is larger than 1 ($k > 1$). Others experiment similarly with “thinner” experts as in MoEfication. A core question that will be presented in this paper will focus around these particular characteristics.

Chapter 3: Methodology

In the following section, the experiments that will be performed will be described, along with their motivations and rationalizations behind the choices made. These experiments will form the basis of validation tests to determine the best recipes for MoE training within the context of a resource constrained environment.

3.1 Data Efficiency

Model efficiency was a primary motivation for this project. LLMs and Vision models require an enormous amount of computation to train from scratch. Running many hyperparameter search experiments would not be possible under our budget constraints as even the smallest LLM could take weeks to pre-train. Therefore one method that could be tested is "sparse upcycling" [Komatsuzaki et al. (2023)] to recycle the pre-training stage of the model. Any experiments that would be performed would be based on improvements from the original checkpoint that was selected.

To compare the benefit of MoEs, a dense baseline will be established that will allow comparisons of MoE sparse models against additional pre-training via a continued learning process. Any sparse models will be trained through a MoE specific process while the dense baseline will follow the original pre-training recipes. Given that a pre-training checkpoint needs to be selected as the starting point, a natural set of questions that needs to be answered is which checkpoint is best? Will the most fully trained checkpoint be the best or will it be an earlier checkpoint?

Within the Sparse Upcycling paper [Komatsuzaki et al. (2023)], the authors suggest that a semi-trained checkpoint is the best choice, not continuing to train an MoE from the most fully trained one. The rationale for this is fairly intuitive: the experts in the MoE layers need to diverge to specialize and if the model is fully trained, the experts will be in a narrow optimization valley and will not have as much breadth in the subregion of optimal solutions to explore, thus an earlier checkpoint is better for wider specialization. To test if an earlier checkpoint is better, one set of experiments will compare a semi-trained checkpoint that is under-trained against one that is the final checkpoint.

As an alternative to upcycling, MoEfication can also utilize a pretrained model to generate a MoE. A pre-trained checkpoint will be used to initialize the experts, but with this technique, the effective parameters will be scaled down by evenly distributing all of the MLP parameters randomly into individual experts with a reduced hidden parameter size for each expert MLP. To ensure a fair comparison between both MoEfication and Sparse Upcycling, the same baseline checkpoint will be adopted for both.

3.2 Model Selection

Selecting a pre-trained LLM model would be a daunting task as there are many to choose from. To narrow the field, having a historical set of checkpoints would be beneficial for analysis of the effect of pretraining degree on expert specialization. Many released pretrained models only offer a single checkpoint, with the exception of a few research training suites. Pythia [Biderman et al. (2023)] is one such suite that contains models of various sizes and checkpoints at every 1000 steps across 143,000 training steps. A smaller model size of 70 million parameters will be chosen for most experiments as the modest model size trains quickly and offers some larger potential for improvement in downstream task performance over the baseline through expert specialization.

3.3 MoE Design Decisions

Mixture of Experts has an extensive number of design decisions that affect the quality and performance of the model while also considering the efficiency of the inference: the number of experts, number of expert layers and their arrangement, and the number of activated experts. To analyze these factors, a subset of permutations based on recommendation from various papers will be explored. For the number of experts, powers of 2 from 2 to 8 will be utilized to understand the width scaling effect of expert count on performance. For the expert layers, a comparison of utilizing all of the transformer layers or the last two layers will be performed to understand the depth scaling effect of MoEs in a LLM. While staggering layers is a recommendation configuration [Fedus et al. (2022a)], due to the reduced size of the baseline model, only consecutive arrangements of MoE layers are considered. For the number of activated experts, the k hyperparameter a comparison of 1, 2, and 6 will be explored to understand the effect of doubling the effective parameter count at the cost of higher inference FLOPs and training time.

3.4 Gating Strategies

Gating strategies are likely the most important factor in the MoE architecture, deserving its own discussion. The gating strategy is crucial as to how well the experts will specialize on the training data. The simplest gating variant is the switch gate where only one expert is chosen (k=1). Utilizing this strategy with upcycling offers the benefit of keeping the effective parameter count constant compared to the dense model for a fair analysis. This gating strategy, along with others that engage only a subset of experts named “token choice”, can require careful load balancing. Such techniques that will be analyzed will be gate jittering which involves adding noise to the softmax activation of the gate, which increases the entropy, and auxiliary losses such as the load balance, mutual information, and z-losses.

Expert choice [Zhou et al. (2022)] is another strategy, however it engages all experts but assigns a non-exclusive subset of the input to every expert. The advantage of such an approach is the lack of having to load balance the experts with a secondary objective function. Both “token choice” and “expert choice” strategies will be analyzed along with their hyperparameters.

3.5 Dataset Selection

The choice of dataset for this investigation will need to be small due to the training time constraints. Many LLMs are trained on several hundred billion tokens which would require a lengthy training cycle. Pythia suite is one such example [Biderman et al. (2023)], being trained on 300 billion tokens using the Pile corpus [Gao et al. (2020)]. Reducing the amount of training data could have a negative impact on the model's quality. Instead of focusing on the data quality problem in this thesis, a research effort has reduced the Pile corpus into a high quality dataset, named MiniPile [Kaddour (2023)]. This dataset is representative of the Pile dataset with less than 1% of the data, roughly 1.5 billion tokens. Utilizing continual learning, the MiniPile dataset can be used with the data efficient techniques described previously to maximize the model quality with the least amount of training tokens.

To test the wider generalization capabilities of the MoE paradigm, additional fine-tuning should be done on out-of-distribution data. To test the sparse models performance with fine-tuning, a subset of FLAN [Wei et al. (2021)] will be used to investigate how well the model can learn an out-of-distribution dataset. This dataset is chosen as it is commonly utilized for instruction tuning [Shen et al. (2023a)] an LLM.

3.6 Task Selection

The traditional metrics used for analyzing neural network performance is accuracy on some test dataset. MiniPile offers a test dataset consisting of 10,000 samples, whereas Pile offers no additional test split, which is another advantage of utilizing a smaller representative dataset. Within the context of LMs, validation loss and perplexity are also key metrics that will be measured as a proxy metric that correlates to the test dataset accuracy.

Fine-tuning is an important part of the model training lifecycling to adapt pre-trained models to new data distributions. Instruction tuning, which is an intermediate form of fine-tuning for LLMs, has become a standard practice and shown great promise with MoEs [Shen et al. (2023a)]. This intermediate fine-tuning will be examined by utilizing an instruction dataset, as mentioned previously. The accuracy of the LLM will be used as the adaptation metric. The goal will be to determine if the additional instruction tuning within a constrainted budget has an impact on the data-efficient training dynamics of the MoE and if it also helps increase downstream task performance.

As the attention of this study is on MoEs applied to LMs, there are additional downstream tasks that are typically used to evaluate a model's generalization performance. A few of these have been used widely in literature and are chosen because they tend to scale up well based on parameter count: PiQA [Bisk et al. (2019)], HellaSwag [Zellers et al. (2019)], SciQ [Welbl et al. (2017)], and Lambada [Paperno et al. (2016)]. These tasks will be evaluated in a zero-shot learning [Wei et al. (2021)] fashion to measure base model and fine-tuned performance.

3.7 Optimization

Training any deep neural network would require considering many other hyperparameters such as the learning rate, weight decay, dropout rate, batch size, optimizer, model precision, and training steps. To reduce the complexity of the analysis, these optimization parameters are mostly inherited from the Pythia suite. The optimizer algorithm was updated from Adam to AdamW [Loshchilov and Hutter (2017)] with a weight decay of 0.01. For the learning rate during continued pre-training, a linear decay with values in the range of 5e-4 and 4e-4 which approximates part of the cosine decay curve that is utilized in the original pre-training of the model that will be the baseline and foundation for continued learning. For fine-tuning, a smaller linear decay learning rate of 4e-4 to 1e-4 is chosen. Dropout is not used as it is not recommended in the first couple of epochs of training when there are limited tokens [Xue et al. (2023)]. For training steps, 1000 steps is used for most experiments, unless otherwise stated. This lines up with approximately 1 epoch of training on MiniPile. For fair comparisons of behavior across baseline dense and sparse models, It is important to note these core parameters will remain constant.

3.8 Hardware and Software Resources

To explore the viability of Mixture of Experts in a constrained resource environment, the resources that are available must be first discussed. In the VITA lab at UT Austin, 4 RTX6000 GPUs were allocated to this project. Additionally, 2 A100 GPUs were infrequently used in the experiments, available from the UT TACC allocation pool. With these resources, a 70M parameter LLM model could be trained using data-efficient practices in under 4 hours for up to 1,000 training steps. The computation budget for the model training needed to be low due to the limited hardware resources and number of experiments.

For training the MoE models, a custom framework was developed using Pytorch Lightning for general deep neural network training code and HuggingFace for foundation model code. LM Eval Harness [Biderman (2023)] was used to evaluate zero-shot performance.

Chapter 4: Experimental Results

4.1 Pretrained Checkpoint Experiments

Deciding on which pretrained checkpoint to use, dense model checkpoints were evaluated at various training steps for validation accuracy and zero-shot downstream task performance. Figure 4.1 shows these metrics across training steps. The graph show the model accuracy peaks around 25,000 training steps while zero-shot performance for many tasks peaks between 25,000 to 50,000 training steps. As explained in the Sparse Upcycling [Komatsuzaki et al, 2022], an undertrained checkpoint is more desirable than a fully trained checkpoint as the starting point for MoE training. The motivation here is to allow some training runway for the model's parameters to diverge as it achieves peak performance. Given this information, a 20,000 step checkpoint is chosen as the baseline checkpoint for sparse and dense models because it is below the lower end of the performance peak.

To validate the reasoning of choosing this checkpoint further, experimental results for sparse and dense models at 20k, 100k, and 143k training steps were evaluated for their peak zero-shot performance. The results for these experiments are listed in Table 4.1. From the experiments, the 20k training checkpoint performs best.

Model	Steps	Accuracy %			
		SciQ	PiQA	Lambada	Average
Dense	20000	**69.3**	60.1	**21.07**	**50.16**
Sparse 2x	20000	**69.5**	59.68	**20.52**	**49.9**
Dense	100000	66.6	**60.3**	18.98	48.62
Sparse 2x	100000	67	**60.1**	18.86	48.65
Dense	143000	63	59.92	17.17	46.69
Sparse 2x	143000	63.4	59.58	17.12	46.7

Table 4.1: Pythia 70M Zero-Shot Performance vs training steps, sparse/dense

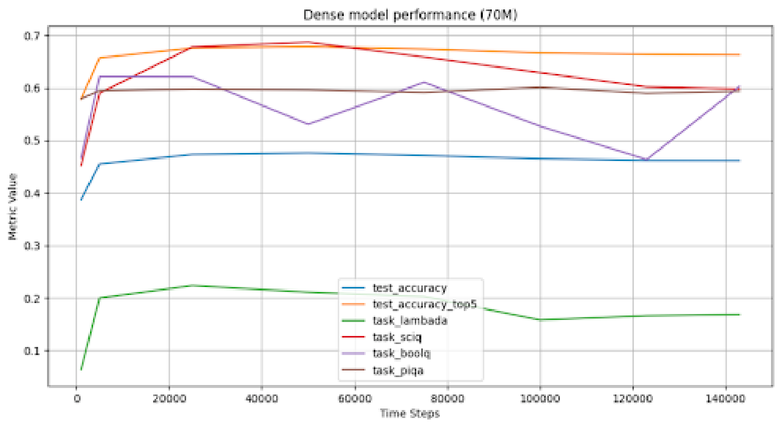

Figure 4.1: Pythia 70M Dense model performance across training checkpoint

4.2 Mixture of Experts Hyperparameter Experiments

4.2.1 Number of Expert Layers

The number of expert layers in a transformer-based LM has an effect on the performance of the models. These results are displayed in Table 4.2, using an 8 expert upcycled sparse MoE model trained on MiniPile for 1000 steps. This experiment shows that performance increases with additional expert layers, however with diminishing returns. The last two layers as expert layers approach offers a good trade-off between increased overhead of the MoE architecture and additional performance increases, which is utilized in the remainder of the experiments. This experiment was not performed with the MoEfication approach, however the assumption is this observation will also hold as an additive effect.

Number of Layers	Top 1 Accuracy	Top 5 Accuracy
0 (dense)	47.75	68.0
1 (last)	47.83	68.11
2 (last-2)	47.91	68.24
6 (all layers)	**48.05**	**68.41**

Table 4.2: Number of Expert Layers vs Accuracy (MiniPile), 8 expert sparsely upcycled

4.2.2 Number of Experts per Layer

The number of experts in each layer is an important factor to consider in both Upcycled and MoEfication MoE models as it affects the models overall parameter or effective parameter counts. For upcycling, the number of total parameters is increased as the number of experts increases, whereas for MoEfication the total capacity stays the same, but the effective parameter count decreases while k=1 for both. Table 4.3 shows the results of accuracy performance as a function of the number of experts in the model, again trained on 1000 steps using MiniPile, with a Last-2 configuration for MoE layers.

Model	Experts	MLP Param	MLP Effective Param	Top 1 Acc.	Top 5 Acc.	Change
Dense	0	12.5×10^6	12.5×10^6	47.75	68.0	-
Upcycled	2	16.7×10^6	12.5×10^6	47.81	68.09	+0.06
	4	25.1×10^6	12.5×10^6	47.84	68.18	+0.09
	8	41.9×10^6	12.5×10^6	**47.94**	**68.26**	**+0.19**
	16	75.5×10^6	12.5×10^6	47.94	68.25	+0.19
MoEfication	2	12.5×10^6	10.5×10^6	**47.21**	**67.36**	**-0.54**
	4	12.5×10^6	9.4×10^6	46.92	67.06	-0.83
	8	12.5×10^6	8.9×10^6	46.65	66.79	-1.1
	16	12.5×10^6	8.6×10^6	46.51	66.62	-1.24

Table 4.3: Number of Experts per Layer vs MoE architecture vs Accuracy (MiniPile)

For upcycled models, the data shows diminishing returns in accuracy as the number of experts per layer increases and the model capacity increases, with 8 experts being the peak in accuracy. With MoEfication, the model accuracy diminishes as the number of effective parameters is reduced while the total capacity remains constant. The peak performance is demonstrated by the lowest number of experts.

Model	Experts	K	MLP Effective Param	Top 1 Acc.	Top 5 Acc.
Upcycled	8	1	12.5×10^6	47.94	68.26
	8	2	16.7×10^6	47.92	68.25
	16	1	12.5×10^6	47.94	68.25
	16	2	16.7×10^6	**47.95**	**68.31**
MoEfication	2	1	10.5×10^6	**47.21**	**67.36**
	4	1	9.4×10^6	46.92	67.06
	4	2	10.5×10^6	47.03	67.25
	8	1	8.9×10^6	46.65	66.79
	8	2	9.4×10^6	46.92	67.09
	16	1	8.6×10^6	46.51	66.62
	16	4	9.4×10^6	46.85	67.02

Table 4.4: Effect of increasing K on various model types and expert counts

4.2.3 Effective Parameters with K adjustment

The k hyperparameter is the expert selection parameter in the TopK function, which controls how many experts are selected per layer based on the gate scoring function. Table 4.4 shows the effect on performance as K is increased. For upcycling, only 8 and 16 expert models are tested. For MoEfication 4, 8, 16 models are tested with k=2 and k=4 to compare the performance between configurations with similar numbers of effective parameters. A renormalization gating strategy with a load balance loss was used in these experiments.

There are several observations to be made here. Firstly, increasing k in the upcycled models had little effect on performance; only a slight increase was observed in the 16 expert models and no increase in the 8 expert models. This can be explained by the expert specialization metric, expert similarity, which leads to a low specialization rate in upcycled models, particularly when k is higher than 1. Having more experts leads to a more differentiated set of model parameters, so seeing an effect is only likelier with a higher number of experts. This will be explained in further detail in the analysis section.

Secondly, MoEfication trained models have a more substantial change as k is increased. These models have highly differentiated expert parameters from their initial inception, thus leading to more differentiation and contribution to overall performance by taking the weighted averages across more experts. Thirdly, the models with different expert counts but similar effective parameters due to increasing the k factor do perform similarly in accuracy, however there is generally a loss in performance as experts MLP density diminishes. This will be discussed in more detail in the analysis section.

4.2.4 Gating Experiments

Routing Strategies are at the heart of the MoE model; the scoring function provided by the gating network determines which experts are selected. The experiments here examine switch, renormalized, and expert choice gating with auxiliary losses such as noise jittered load balancing, mutual information, and z-loss. Several points should be made first. Switch gating is only designed to work with k = 1 while the renormalized gating is designed to work with k > 1. Also z-loss is meant to support load balancing when k > 1. Thus the experiments will have only certain MoE configurations that are compatible with the gating strategy being tested. Table 4.5 summarizes the experiments.

Gating Type	Auxiliary Loss	Mean Top 1 Acc. (std dev)	Mean Top 5 Acc. (std dev)
Switch (k=1)	Load Balance	47.37 (1.3)	67.65 (1.2)
	Mutual Information	**47.44 (1.3)**	**67.77 (1.2)**
Renorm (k¿1)	Load Balance	47.42 (0.54)	67.68 (0.63)
	Load Balance + Z-Loss	**47.91 (0.06)**	**68.24 (0.08)**
Expert-Choice	-	47.25 (1.2)	67.56 (1.2)

Table 4.5: Gating Strategies and Auxiliary Losses vs Performance

The results shown are an average over various hyperparameters, such as expert count, noise jittering, and auxiliary loss scale. These factors affect the gating strategy concretely. For expert choice, the capacity factor is held to 1.0.

From the data, the auxiliary loss functions such as mutual information and the additional z-loss perform better than the standard load balancing. These findings align with both recommendations found in the auxiliary losses' original respective papers [Shen et al. (2023b), Zoph et al. (2022)]. Comparing gating types, the switch and renormalization routings outperform expert-choice. Expert-choice has a natural load-balancing effect with regards to experts, but it performs this load balancing by subsampling the input tokens. There is no data collected to offer a convincing explanation of why expert-choice performs slightly worse, but it can be speculated that at this parameter level, all the tokens are necessary, and subsampling the input tokens leads to suboptimal performance compared to the switch routing.

4.2.5 Out-of-Distribution Fine-Tuning

Fine-tuning is an important part of the life-cycle of a pre-trained model. The following experiment fine-tunes the dense and sparse models continually trained on the subset of MiniPile and tests their adaptation capabilities to an out-of-distribution dataset, FLAN. Table 4.6 shows the results of these experiments. The fine-tuning involves 500 steps of training, equivalent to 2 epochs, on a subset of FLAN published on HuggingFace. This approach remains consistent with the theme of data-efficient tests for MoE training. The baseline model in this experiment refers to training directly on FLAN without MiniPile pre-training.

Model	Experts	Top 1 Acc. FLAN mini (% change)	Top 5 Acc. FLAN mini (% change)	Top 1 Acc. MiniPile (% change)	Top 5 Acc. MiniPile (% change)
Baseline	0	47.64 (+0.08)	66.24 (+0.07)	43.18 (0)	63.81 (-0.11)
Dense	0	47.56	66.17	43.18	63.92
Upcycled	2	47.69 (+0.13)	66.31 (+0.14)	43.28 (+0.1)	64.0 (+0.08)
	4	47.88 (+0.32)	66.52 (+0.35)	43.33 (+0.15)	64.08 (+0.16)
	8	47.99 (+0.43)	66.58 (+0.41)	43.44 (+0.26)	64.22 (+0.3)
	16	**48.27 (+0.71)**	**66.88 (+0.71)**	**43.72 (+0.54)**	**64.45 (+0.53)**
MoEfication	2	**47.09 (-0.47)**	**65.61 (-0.56)**	**42.98 (-0.2)**	**63.48 (-0.44)**
	4	46.88 (-0.68)	65.32 (-0.85)	42.59 (-0.59)	63.03 (-0.89)
	4 (k=2)	47.06 (-0.5)	65.54 (-0.63)	42.96 (-0.22)	63.44 (-0.48)
	8	46.72 (-0.84)	65.14 (-1.03)	42.44 (-0.74)	62.81 (-1.11)
	8 (k=2)	46.88 (-0.68)	65.33 (-0.84)	42.57 (-0.61)	63.03 (-0.89)
	16	46.6 (-0.96)	64.99 (-1.18)	42.28 (-0.9)	62.61 (-1.31)
	16 (k=4)	46.83 (-0.73)	65.26 (-0.91)	42.46 (-0.72)	62.91 (-1.01)

Table 4.6: Subset FLAN Fine-Tuning on dense/sparse models vs performance

There are several essential observations to be made regarding the outcome of this experiment. Regarding the upcycled models, firstly, the baseline outperformed on the FLAN fine-tuning while it underperformed on the MiniPile validation. The dense models parameters are trained on a higher-quality version of MiniPile, however the FLAN data is out-of-distribution, thus the dense model's additional training was not as helpful as utilizing the more generalized pre-training checkpoint. Secondly, all of the upcycled sparse checkpoints outperformed the dense and baseline models on FLAN and MiniPile validation accuracy. Furthermore, the upcycling shows continued benefits as the expert count increases, suggesting the out-of-distribution training is a prime candidate for MoE specialization. Finally, while all fine-tuned upcycled models performed worse than their original MiniPile performance (see Table 4.3), the sparse models were able to retain more performance between both tasks. These key observations will be explored further in the analysis section.

For MoEfication models, the models continue to approximate the dense model performance, all with reduced accuracy on FLAN and MiniPile. The performance diminishes as the expert count increases, with the exception of the higher k variants, which approximate their counterparts with lower expert counts. This is expected as the MoEfication models have increasingly fewer effective parameters as the expert count increases. To explore the potential benefits, the effective parameter and accuracy tradeoff will be discussed further in the analysis section.

4.2.6 Zero-Shot Downstream Performance

Zero-shot performance is a key metric in most language model tests. As part of the training process of the sparse models, it would be beneficial to understand the effects of sparsity on zero-shot performance. It should be noted that few-shot performance was attempted with this model, however in all cases few-shot performance declined, thus only zero-shot performance numbers are provided. A possible explanation for this is the low parameter level of this model may not allow for in-context learning, thus the irrelevant tokens provided by few-shot training examples can distract the model on the primary example being evaluated.

Table 4.7 below reports the outcome of the zero-shot experiments. As part of the experiments, models both trained with in-distribution data via MiniPile (ID) and additional out-of-distribution FLAN data (OOD) are grouped. There are numerous observations to be made about task-specific performance.

- SciQ performs better using MoEfication than with Upcycling under ID models, particularly with higher expert counts, indicating this task has an affinity to lower parameter counts (an example of an over-parameterized task).
- SciQ performance generally improves with additional FLAN training, however

all models seem to have saturated performance at around 73% across upcycling and MoEfication techniques.

- PiQA has a strong affinity for extra capacity and specialization, as the upcycling technique in both ID and OOD models dominates the performance, and increases with expert count (an example of an under-parameterized task).
- Lambada has a strong affinity for ID sparse models with k ¿ 1, indicating this task improves as more experts are averaged together.
- Lambada generally underperforms under FLAN fine-tuning.
- HellaSwag is unaffected by sparsity at this parameter level in either ID or OOD modalities, however performs better with additional fine-tuning on FLAN.

Model	Experts	SciQ	PiQA	Lambada	Hellaswag
ID Models					
Baseline	0	67.0 (-2.3)	60.39 (+0.71)	21.46 (+0.39)	26.91 (-0.4)
Dense	0	69.3	59.68	21.07	**27.31**
Upcycled	2	69.5 (+0.2)	60.12 (+0.42)	20.52 (-0.55)	27.23 (-0.08)
	4	69.3 (0)	60.28 (+0.58)	19.67 (-1.4)	27.17 (-0.14)
	8	**70.1 (+0.8)**	60.61 (+0.93)	21.12 (+0.05)	27.28 (-0.03)
	8 (k=2)	69.3 (0)	60.23 (+0.55)	20.57 (+0.5)	27.23 (-0.08)
	16	68.7 (-0.6)	60.5 (+0.82)	19.4 (-1.67)	27.28 (-0.03)
	16 (k=2)	66.0 (-3.3)	**60.77 (+1.09)**	**23.0 (+1.93)**	27.0 (-0.31)
MoEfication	2	70.9 (+1.6)	59.79 (+0.11)	20.24 (-0.83)	27.12 (-0.19)
	4	69.3 (0)	59.3 (-0.38)	19.5 (-1.57)	27.15 (-0.16)
	4 (k=2)	70.4 (+1.1)	**59.96 (+0.28)**	20.99 (-0.08)	27.15 (-0.16)
	8	**71.3 (+2.0)**	59.52 (-0.16)	20.65 (-0.42)	27.05 (-0.26)
	8 (k=2)	70.4 (+1.1)	59.09 (-0.59)	21.6 (+0.53)	27.2 (-0.11)
	16	70.5 (+1.2)	59.19 (-0.49)	19.6 (-1.47)	27.07 (-0.24)
	16 (k=4)	70.8 (+1.5)	58.98 (-0.7)	**21.21 (+0.14)**	27.17 (-0.14)
OOD Models					
Baseline	0	73 (-0.1)	60.94 (+0.82)	19.16 (-0.13)	27.15 (-0.3)
Dense	0	73.1	60.12	**19.29**	27.45
Upcycled	2	72 (-1.1)	61.04 (+0.92)	18.95 (-0.34)	27.35 (-0.1)
	4	72.7 (-0.4)	60.99 (+0.87)	19.02 (-0.27)	27.38 (-0.07)
	8	73.2 (+0.1)	61.21 (+1.09)	19.02 (-0.27)	27.45 (0)
	8 (k=2)	**73.9 (+0.8)**	**61.64 (+1.52)**	19.25 (-0.04)	27.39 (-0.06)
	16	72.2 (-0.9)	61.26 (+1.14)	19.05 (-0.24)	**27.57 (+0.12)**
	16 (k=2)	72.9 (-0.2)	61.15 (+1.03)	19.21 (-0.08)	27.33 (-0.12)
MoEfication	2	71.9 (-1.2)	59.90 (-0.22)	18.54 (-0.77)	**27.25 (-0.2)**
	4	72.0 (-1.1)	59.85 (-0.27)	18.08 (-1.21)	27.24 (-0.21)
	4 (k=2)	**73.5 (+0.4)**	**61.26 (+1.14)**	18.97 (-0.32)	27.11 (-0.34)
	8	73.0 (-0.1)	59.52 (-0.5)	18.44 (-0.84)	27.24 (-0.21)
	8 (k=2)	73.0 (-0.1)	59.09 (-1.03)	18.98 (-0.31)	27.08 (-0.37)
	16	72.1 (-1.0)	60.77 (+0.65)	18.26 (-1.03)	27.24 (-0.21)
	16 (k=4)	71.1 (-2.0)	59.79 (-0.33)	18.84 (-0.45)	27.15 (-0.3)

Table 4.7: Zero-Shot performance across model types and fine-tuning domains

These observations show several factors that can influence downstream task specific performance, which will be further discussed in the analysis section.

Chapter 5: Conclusions

5.1 Analysis

To further analyze the experimental results from the previous section, some additional elaboration on various topics will be presented in the following section.

5.1.1 Understanding MoE Specialization and Optimization

MoE layers are designed to conditionally process inputs across a subset of experts. Influencing factors for designing MoE layers include the number of experts, the k selection hyperparameter, and gating strategies. Due to the conditional and discrete nature of the gating process using the TopK function, backpropagation through the network can lead to instabilities in the training process where certain experts can be trained at different rates based on the training data and the other factors mentioned. To understand this better, a metric to compare experts, expert similarity, will be used which uses a mean cosine similarity calculation to compare the parameter weights of all experts in a layer. With this metric, a high expert similarity would mean a layer with low expert specialization and a low expert similarity would signify a layer with high expert specialization. The reasoning is based on the assumption that experts that are specialized should be dissimilar with respect to their parameters.

Figure 5.1 shows the change in expert similarity to illustrate the effect of various design choices on the rate of specialization in the network during training. The models are all sparsely upcycled, thus their similarity all begin at 1 and then diverge towards 0 as they begin to specialize on input data distributions. Using switch gating with different numbers of experts and auxiliary loss hyper-parameters will generate expert similarity curves with varying rates of change.

Generally, with proper load balancing and a batch size much larger than the expert count, as the expert count increases, the rate of similarity divergence increases.

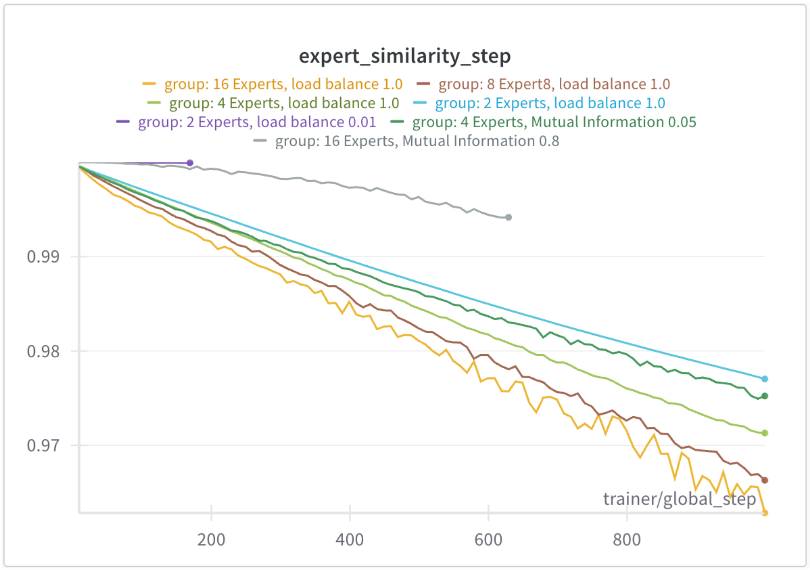

Figure 5.1: Expert Similarity of upcycled models, with varying hyperparameters

This is explained by understanding that if load balancing is treated as a uniform distribution, then the optimization process will force all expert parameters to slightly shift in different directions leading to a faster rate of divergence when more experts are involved in the selection process.

Another effect that is noteworthy is the increasing amount of noise as more experts are introduced and as divergence increases. As the divergence level and expert count increases, there is a higher likelihood of the gating process to assign clusters of tokens to specialized experts and less to others, diminishing the equality in assignment to lower the language modeling loss. This conflicts with the load balancing loss objective which eventually grows larger and balances the gating selection, thus resulting in a noisy oscillation process over time.

Expert similarity does not reflect the level of load balancing. In a 2 expert layer, if only one expert is chosen constantly, the expert similarity will decline steadily at the rate of the gradient training signal. To fine tune the load balancing, a heatmap is used to visualize the expert selection process. Figure 5.2 illustrates two different load balancing auxiliary losses at two different time points. The top row is an 8

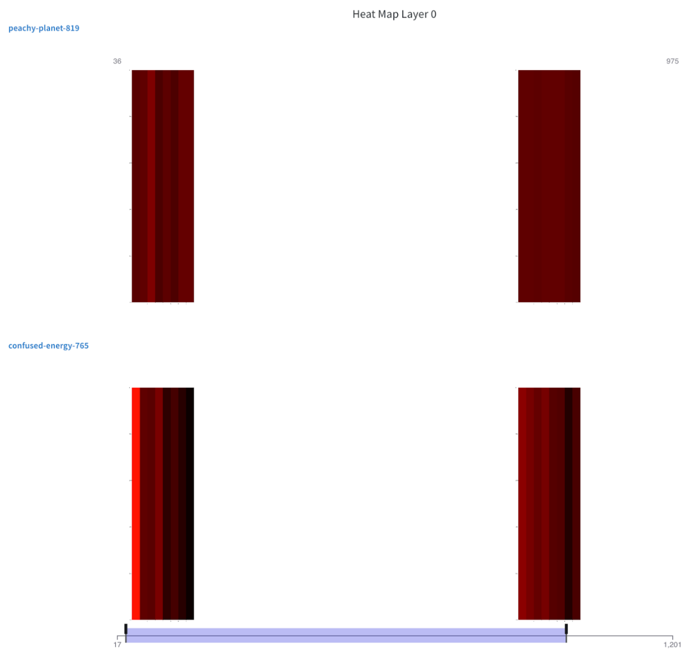

Figure 5.2: Expert Layer Selection Heatmap

expert standard load balancing auxiliary loss and the bottom row is an 8 expert mutual information loss. The right side shows the start of training and the left shows the end of training. Each heatmap shows 8 segments with the higher saturation indicating a higher degree of selection.

The heatmap demonstrates the differences in the two auxiliary losses. At the start, the both auxiliary losses have some imbalance, with the mutual information having a strong bias to the first expert. At the end of training, the load balancing loss has a fairly uniform selection distribution, but the mutual information loss is reflective of a Guassian distribution as this loss is a minimization and maximization of cross entropy of the selection with respect to the input distribution.

Using these two forms of visualization, it can be concluded that reducing the expert similarity is essential to expert specialization which maximizes the advantage of the MoE layer, but it can be complex to fine-tune based on various design choices. The load balancing is a critical factor for maximizing the expert capacity while training the MoE. The experimental results have shown which gating strategies and auxiliary losses are best, however each requires tuning. These visualization tools can be utilized as guides to monitor expert specialization behavior and tune the hyperparameters to maximize the capacity and performance of the MoE network.

5.1.2 Understanding Trade-Offs in Data-Efficient MoE Training Techniques

The two approaches for training MoEs with data-efficient techniques presented in this paper have been Sparse upcycling and MoEfication. Both leveraged pre-trained models to avoid a lengthy full training process. The key question that should be answered is how is each approach useful and when should it be applied?

Sparse upcycling, as previously described, duplicates in-place MLP parameters in a transformer layer to initialize a variable number of experts, with experts all starting with perfect expert similarity. MoEfication on the other hand uses all of MLP parameters and randomly allocates an equal portion across experts. This rationing of the parameters scales down the effective parameter count, reducing the number of inputs into each expert by the expert count factor. Thus the upcycling technique increases the total model capacity through parameter duplication while retaining the effective parameter count constant whereas MoEfication holds the model capacity constant, while reducing the effective parameter count through parameter rationing.

The two techniques have opposing objectives. Upcycling is applicable when increasing model capacity is essential due to under-parameterization. MoEfication is useful when a model is overparameterized and model compression is the objective. The experiments offer some clear evidence of the benefits for both of these techniques.

For upcycling, FLAN adaption showed an increasing trend in performance as more experts were added up to the maximum 16 experts, whereas in MiniPile training, the performance increase peaked with 8 experts. Additionally PiQA zero-shot performance increased as expert count increased in both cases under ID and

OOD models, outperforming dense and baseline models. It would be recommended to use upcycling as a method to grow the capacity of a pretrained model so it can be adapted to a new distribution of data while retaining the original pretraining parameters. As also shown in the PiQA example, OOD fine-tuning improved the upcycled sparse model performance, even above a baseline directly trained on the OOD distribution, proving the benefit of the intermediate fine-tuning. Additionally as seen in Table 4.6, OOD fine-tuning performed better on sparse upcycled models, above a directly fine-tuned baseline in both the OOD accuracy and the ID accuracy, demonstrating the retention capabilities of both data distributions.

A noteworthy complexity of upcycling is when it should be used in the training process. As seen with MiniPile training, performance increases are modest compared to FLAN training. Also the evidence presented for expert similarity divergence shows greater specialization over time. Upcycling needs to be utilized early in the pre-training and intermediate fine-tuning phases to leverage and maximize the specialization offered by the MoE layers. If the dense model is seen as the average of all possible expert models, then a fully trained dense model would be difficult to improve. With an overtrained checkpoint using upcycling, the experts will converge towards the dense model's performance as there is less of a training loss to optimize and utilize for expert parameter divergence. Thus upcycling needs a large training horizon to allow for continued performance improvements.

For MoEfication, the model compression evidence is also available within our experimental results. Within the context of zero-shot performance, both the SciQ and HellaSwag models retained their performance despite reducing the MoE MLP layers by a factor of up to 8 using MoEfication. However, model compression is ideal as a final phase of downstream fine-tuning, not as part of a pre-training or intermediate fine-tuning phase such as is the case with upcycling.

To summarize the performance and trade-offs of these data-efficient techniques, Table 5.1 below shows the effects of each technique on model parameter size, performance, and effective parameters. If the goal is to reduce the model footprint without much loss in performance, MoEfication is a beneficial technique. The magnitude of the effective parameter reduction is larger than that of the performance declines. This leads to a lower inference cost in terms of FLOPS. If greater model capacity is necessary, upcycling could be a beneficial technique for MoE training as it can leverage the greater amount of parameters while keeping inference costs equal to a similarly trained dense model. However upcycling comes with a massive memory footprint increase. The accuracy gains may not justify this extra cost. While the accuracy gains reported here are low, one should realize these experiments are only testing under a small training horizon. Thus, with further training and varied data, the performance gains could be substantially higher [Komatsuzaki et al. (2023)].

Model	Total # Parameters	Total # Effective Parameters	Accuracy (rel change %)
Dense	7×10^7	7×10^7	47.56
Upcycled 16x model	1.3×10^8 (+190%)	7×10^7 (+0.0%)	48.27 (+1.01%)
MoEfication 16x model	7×10^7 (+0.0%)	6.6×10^7 (-5.6%)	46.6 (-2%)

Table 5.1: Data-efficient model architecture, performance vs model parameters, Last-2 16 experts, FLAN

5.1.3 MoE Design Recommendations

Based on the experimental evidence provided, some recommendations can be offered for designing MoEs.

- Expert Layer Count: Last-2 is a good choice to minimize model size while maximizing performance.
- Expert Count Per Layer: For either upcycled and MoEfication MoE training, 8 experts appears to be a good compromise to maximize performance. Models such as Mixtral and Grok utilize this expert count.
- Top-K factor: K=1 is a recommended configuration as it performs generally well while keeping the effective parameter count equal. $K > 1$ can boost performance for certain tasks (see Lambada), but this will increase inference and training costs.
- Expert Width vs Expert Count: For MoEfication, increasing the expert count leads to thinner experts and increasing the k factor can recover some performance as the expert count declines. However, given the evidence, selecting wider and fewer experts as possible is recommended when attempting model compression for the best performance.
- Gating Strategy: Switch with k=1 or Renormalization gates with k=2 are recommended with Mutual Information Loss or Load Balancing and Z-loss respectively. Tuning any jitter and auxiliary loss scales hyperparameters is critical for maximizing performance.

5.2 Summary

Mixture of Experts as part of a deep learning model architecture is a useful and growingly popular technique. Through the experiments presented in this paper, two data-efficient methodologies of how to train these networks for stability and performance are fully implemented and tested. Insights were gained on how to design and apply these techniques with the proper goals, either regarding model capacity growth or model compression at different phases to the training cycle. Additionally further MoE design factors were explored and tested. Final recommendations were made based on the experimental evidence.

The final take-away of this paper is MoEs typically require large training datasets to fully capitalize on their sparse conditional computation, an exploration into their behaviors is still possible with a constrained computation budget. These training experiments ran for 1 to 4 GPU hours, a tiny fraction of the total cost for a large language model. Regardless of the limitations in training budget, the experimental evidence validates the expectation in performance advantages and scalability of MoEs. With the knowledge presented, other researchers can utilize this prescribed methodology to build, train, and fine-tune their own MoEs either from scratch or leveraging pre-trained checkpoints with data-efficiency.

References

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`.

Stella Biderman. Evaluating llms — eleutherai. *EleutherAI*, Feb. 15 2023. URL `https://www.eleuther.ai/projects/large-language-model-evaluation`.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, apr 3 2023. URL `https://arxiv.org/abs/2304.01373`.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL `https://arxiv.org/abs/1911.11641`.

Tianlong Chen, Zhenyu Zhang, Ajay Jaiswal, Shiwei Liu, and Zhangyang Wang. Sparse moe as the new dropout: Scaling dense and self-slimmable transformers. https://arxiv.org/abs/2303.01610, mar 2 2023.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, Xian-Ling Mao, Heyan Huang, and Furu Wei. On the representation collapse of sparse mixture of experts. https://arxiv.org/abs/2204.09179, apr 20 2022.

Mohammed Nowaz Rabbani Chowdhury, Shuai Zhang, Meng Wang, Sijia Liu, and Pin-Yu Chen. Patch-level Routing in Mixture-of-Experts is Provably Sample-efficient for Convolutional Neural Networks. https://arxiv.org/abs/2306.04073, jun 7 2023.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models, 2022. URL https://arxiv.org/abs/2202.01169.

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

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning, 2022a. URL https://arxiv.org/abs/2209.01667.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022b. URL https://arxiv.org/abs/2101.03961.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. https://arxiv.org/abs/2101.00027, dec 31 2020.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L. Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. Continual pre-training of large language models: How to (re)warm your model?, 2023. URL https://arxiv.org/abs/2308.04014.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. https://arxiv.org/abs/1207.0580, jul 3 2012.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 2 1991. [Online; accessed 2024-08-02].

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. https://arxiv.org/abs/2401.04088, jan 8 2024.

Jean Kaddour. The minipile challenge for data-efficient language models, 2023. URL `https://arxiv.org/abs/2304.08442`.

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, 2023. URL `https://arxiv.org/abs/2212.05055`.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. https://arxiv.org/abs/2006.16668, jun 30 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. https://arxiv.org/abs/1711.05101, nov 14 2017.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. https://arxiv.org/abs/1606.06031, jun 20 2016.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. https://arxiv.org/abs/1701.06538, jan 23 2017.

Sheng Shen, Le Hou, Yanqi Zhou, Nan Du, Shayne Longpre, Jason Wei, Hyung Won Chung, Barret Zoph, William Fedus, Xinyun Chen, Tu Vu, Yuexin Wu, Wuyang Chen, Albert Webson, Yunxuan Li, Vincent Zhao, Hongkun Yu, Kurt Keutzer, Trevor Darrell, and Denny Zhou. Mixture-of-Experts meets instruction tuning:a Winning combination for large language models. https://arxiv.org/abs/2305.14705, may 24 2023a.

Yikang Shen, Zheyu Zhang, Tianyou Cao, Shawn Tan, Zhenfang Chen, and Chuang Gan. Moduleformer: Modularity emerges from mixture-of-experts. https://arxiv.org/abs/2306.04640, June 7 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://arxiv.org/abs/1706.03762, jun 12 2017.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adaptations for parameter-efficient model tuning, 2022. URL `https://arxiv.org/abs/2205.12410`.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. https://arxiv.org/abs/2109.01652, sep 3 2021.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022. URL `https://arxiv.org/abs/2206.07682`.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. https://arxiv.org/abs/1707.06209, July 19 2017.

xai-org. Github - Xai-org/grok-1: Grok open release. https://github.com/xai-org/grok-1. [Online; accessed 2024-08-02].

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-Efficient fine-tuning methods for pretrained language models: A critical review and assessment. https://arxiv.org/abs/2312.12148, dec 19 2023.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis, 2023. URL `https://arxiv.org/abs/2305.13230`.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning, 2023. URL `https://arxiv.org/abs/2309.05444`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? https://arxiv.org/abs/1905.07830, may 19 2019.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer Feed-forward Layers are Mixtures of Experts. https://arxiv.org/abs/2110.01786, oct 5 2021.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-Experts with expert choice routing. https://arxiv.org/abs/2202.09368, feb 18 2022.

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, 2024. URL `https://arxiv.org/abs/2406.16554`.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-MoE: Designing stable and transferable sparse expert models. https://arxiv.org/abs/2202.08906, feb 17 2022.