Understanding Compute-Parameter Trade-offs in Sparse Mixture-of-Expert Language Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Scaling language model capacity is crucial for achieving better performance, as it allows these models to capture more complex patterns and representations. Empirically, increasing model size and compute improves outcomes; however, the relationship between model parameters and compute per example, and their combined contribution to capacity, is not yet fully understood. We explore this relationship through sparse Mixture-of-Expert models (MoEs), which allow scaling the number of parameters without proportionally increasing the FLOPs per example. We investigate how varying the sparsity level, i.e., the ratio of non-active to total parameters, affects model performance in terms of both pretraining and downstream objectives. We find that under different constraints (e.g. parameter and total training compute), there is an optimal level of sparsity that improves both training efficiency and model performance. These results provide a clearer understanding of the impact of sparsity in scaling laws for MoEs and complement existing works in this area, offering insights for designing more efficient architectures.

1 Introduction

Empirical scaling laws for language model pretraining [15, 14, 19, 23, 13, 4, 27, 17] have demonstrated that proportionally increasing model capacity, along with data and total compute budget, consistently decreases pretraining loss, improves downstream task performance [8, 3, 1] and unlocks emergent capabilities [24]. A recurring notion in these studies is that model capacity is well quantified by the total number of model parameters. However, the number of parameters is not the only means to increase model capacity— *compute per example (i.e., a fixed-sized input)*, measured in FLOPs, also plays a significant role. In fact, several mechanisms [22, 7, 25, 12, 6] allow for independent variation of the number of parameters or FLOPs per example within a model. For instance, Mixture-of-Experts (MoE) models[22] introduce "FLOP-free parameters" by leveraging sparsity, where only a subset of expert modules is activated for each input. Under specific conditions, the total number of parameters can serve as a reasonable relative estimator of FLOPs per example. Therefore, using the number of parameters as a measure of model capacity in scaling law studies is appropriate. However, in scenarios or for architectures where the number of parameters and FLOPs per example are not inherently linked, it is essential to jointly consider the effects of these variables on scaling model capacity. We thus ask *"Can we draw scaling laws for the optimal trade-off between parameter count and FLOPs per example?"* To address this question, we study sparse Mixture-of-Expert Transformers (MoEs) [22, 16, 10, 28, 18] in the context of language modeling.

Existing scaling law studies for MoEs, investigate the role of variables like number and granularity of experts, underlying dense model size and inference compute in predicting the performance of the models under different conditions such as training or inference compute optimality [9, 4, 27, 17]. In this paper, we focus on the interaction between FLOPs per example and total parameter count, and their impact on model performance in MoEs, through a large-scale empirical study.

We define sparsity as the ratio of inactive experts to the total number of experts, which indirectly controls FLOPs per example in MoEs. We evaluate loss and downstream metrics for different sparsities, model sizes, and compute budgets terms. Our findings are summarized as follows:

- **Effect of Sparsity on Scaling Laws for Optimal Model Size:** For any specific sparsity level, performance of the models as a function of their size exhibits parabolic behavior under a fixed training compute budget. i.e., the model reaches its optimal performance at a vertex, that indicates optimal model size. Under these conditions:
  - The optimal active number of parameters decreases as the sparsity level increases, leading to smaller FLOPs per example and more efficient inference even though the total number of parameters increases (see §2.1).
  - While the trend of increasing active number of parameters is similar across all training compute budgets; the optimal active number of parameters decrease more rapidly with sparsity as the training compute budget increases (see §3).
- **Optimal Sparsity for Fixed Model Size:** For any given number of parameters and under a fixed training compute budget, model performance as a function of sparsity exhibits a parabolic pattern, reaching its peak at an optimal sparsity level (see §2.2). Specifically, the optimal sparsity level:
  - Increases with the total number of parameters approaching 1.0 for larger models. i.e., if a model is relatively small for a given training compute budget, sparsifying it more than a threshold will hurt its performance. On the other hand, if a model is relatively large for a given compute budget, further sparsifying it helps as it leads to increase in the number of tokens the model is trained on under the given training budget constraints (see §2.2).
  - Decreases across all model sizes as the training compute budget decreases (see §D.1 and §D.2).
- **Effect of Sparsity on Downstream Performance:** Models with similar pretraining perplexity have similar downstream task performance regardless of sparsity. For reading comprehension tasks (e.g., CoQA [21], SQuAD [20]), denser models perform better, potentially due to their higher inference-time compute than a perplexity-matched sparse model. Alternative strategies to increase inference time compute dynamically [25, 12] may address this gap (see §4).

Ultimately, this paper highlights the crucial role of the sparsity variable in scaling laws for MoEs, emphasizing that the most efficient model configuration requires joint optimization of model size, total training budget and sparsity level and the optimal balance between FLOPs per example and parameter count in MoEs depends on both the computational constraints and the main objective.

2 The Interplay between Model Parameters and Sparsity in MoEs

Is there an optimal trade-off betweenparameter count and FLOPs per example in MoEs under the setting where the training compute budget is fixed?

Intuitively, under infinite data setting, scaling model capacity along with the training compute budget leads to performance improvements. Previous scaling law studies suggest that, conditioned on a training compute budget measured in FLOPs denoted by C, the optimal number of parameters, $N^*(C)$, exhibits a power-law relationship with C [14]:

$$N^*(C) = \arg\min_N \mathcal{L}(N; C) \propto C^a \tag{1}$$

Our goal is to study how to optimally trade-off FLOPs per example (i.e. a fixed-sized input) and total parameters as shown in Equation 2. Instead of FLOPs vs. parameters, we investigate the relationship between Sparsity S and total number of parameters N, as S is the variable in MoEs that indirectly impacts FLOPs per example.[1] Essentially, for models with the same N, the model with a higher S will have fewer active parameters N_a, resulting in fewer FLOPs per example. For more details on

[1]We use the active number of parameters as a proxy for FLOPs per example, as $6N_aD$ provides a good estimate of the total FLOP count for MoEs; see Appendix C for details.

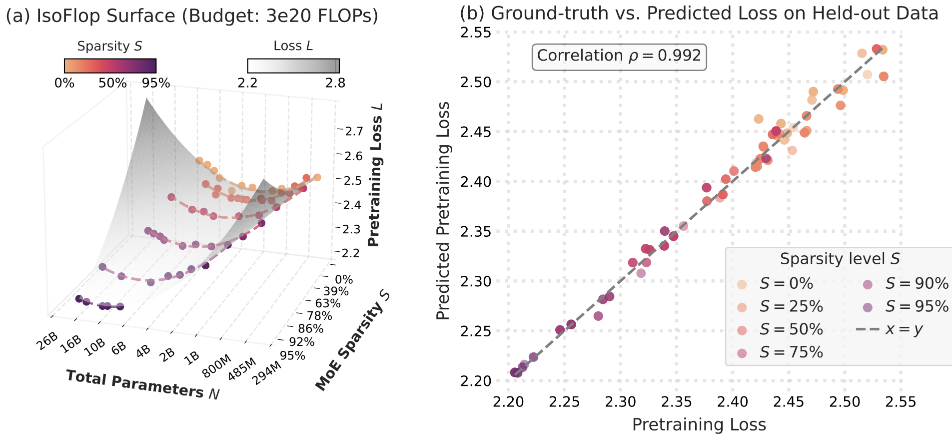

Figure 1: **IsoFLOP surface over observed pretraining loss L, model size N and sparsity S.** We fit a polynomial mapping N, S and their interaction to L using empirical data to obtain plot (a) from which we observe that for fixed compute budget the loss is decreasing with increased model sparsity. The plot on the right shows the goodness-of-fit from which we observe that the predictions and observed loss values are highly correlated with a small prediction error. (see Figure 6 in Appendix D.1 for other total training compute budgets.)

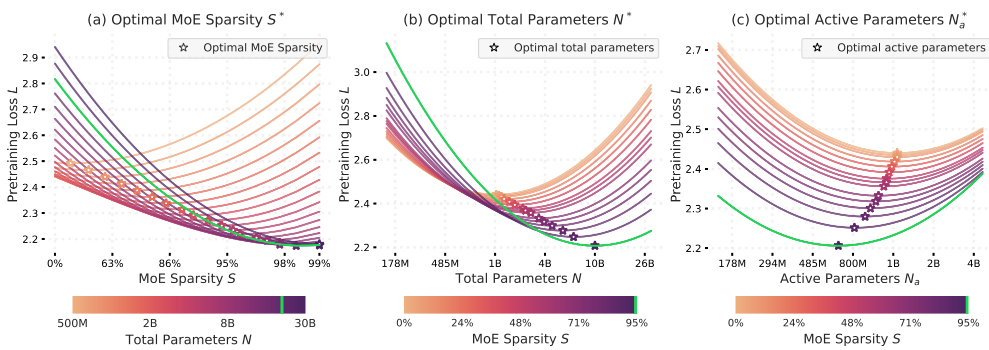

Figure 2: **IsoFLOP slices along Sparsity and Model Size.** We use fitted isoFLOP surfaces (Section 2) to analyze how sparsity **S** and model size **N** impact the loss **L** for a fixed compute budget. We identify optimal points by (a) fixing **N** and varying **S**, (b) fixing **S** and varying **N** and (c) fixing **S** and varying active parameters $\mathbf{N_a}$. Observe that (a) the optimal sparsity S increases with increasing model size N and converges to 1 while (b) and (c) show that the optimal model size N and active parameter count N_a increase and decrease respectively with increasing sparsity levels. (see Figure 7 in Appendix D.1 for other total training compute budgets.)

the notations and experimental settings see Appendix A and Appendix B.

$$(N^*, S^*) = \arg\min_{N,S} \mathcal{L}(N, S; C) \tag{2}$$

To simplify the problem of understanding the joint role of N and S in predicting $\mathcal{L}$, we break the problem, Equation 2, into two parts:

1. *"How does the sparsity level impact the scaling laws of the relationship between N and C for training-compute optimal models?"* To address this question in §2.1, we fix S and vary N, studying how optimal N and N_a change for different values of S:

$$N^* = \arg\min_{N} \mathcal{L}(N; C, S) \tag{3}$$

2. *"Is there an optimal balance between total and active number of parameters under fixed training-compute budget?"* To address this question in §2.2, we fix N and vary S, studying how optimal S changes across different values of N:

$$S^* = \arg\min_{S} \mathcal{L}(S; C, N) \tag{4}$$

As the first step, considering a fixed training compute budget C, we fit a 3D surface, referred to as the IsoFLOP surface, in Figure 1a, using a polynomial function, following approach II of Hoffmann et al. [14]. We include sparsity and fit a single IsoFLOP surface across all data points, rather than fitting separate curves for fixed sparsity levels or model sizes. We conducted a grid search to determine the optimal polynomial degree for N, S, and the interaction term $N \times S$, finding that a degree of $(2, 2, 2)$ resulted in the lowest cross-validation error. Both N and S are in log space (see Appendix B for more details). Figure 1b illustrates the goodness of fit, demonstrating a strong correlation and low predictive error.

As seen in Figure 1a, the IsoFLOP surface plot is parabolic along model size, suggesting that the findings of Hoffmann et al. [14] extend to MoEs across different sparsity levels, i.e., $\mathcal{L}(N; C, S)$ is parabolic, with its optimal solution located at the turning point. However, along sparsity, pretraining loss decreases monotonically, indicating that, for the same compute budget, sparser models achieve better pretraining loss. To better understand these observations, we examine slices of the IsoFLOP surface along the axes of S and N separately in §2.1 and §2.2, respectively.

2.1 Optimal Model Size for Fixed Sparsity Level

Here we examine how sparsity influences scaling laws governing the relationship between N and C for training-compute optimal models, i.e. how does N^*, for a given C, S (Equation 3), change as we increase S? Looking at slices of the IsoFLOP surface along the model size dimension, in Figure 2b and (c), we observe how the IsoFLOP curves shift along loss and model size. Considering the training-compute optimal model, for a fixed compute budget, loss decreases as we increase sparsity. Furthermore, while sparser models have larger N compared to denser models, as seen in Figure 2b, they have a smaller active parameter count N_a; hence, fewer FLOPs per example. More parameters in total increase the capacity of the sparser models to fit the data, while fewer FLOPs per example allow the model to be trained with more tokens, i.e., higher D, for the same compute budget.

2.2 Optimal Sparsity Level for Fixed Model Size

Understanding the dynamics between number of parameters and FLOPs per example is essential for training models with smaller inference cost under constraint training budget. This leads us to a fundamental question: Is there an optimal balance between the total and active number of parameters under a fixed training-compute budget? This section investigates this question. Specifically, we ask: Given N and C, How does S^* change as we increase N?

To address this, we look into slices of the IsoFLOP surface along sparsity, Figure 2a. As we can see in this figure, given a fixed compute budget for training and fixed model size there $\mathcal{L}(S; N, C)$ exhibits a parabolic profile, reaching its optimum value at the vertex where $S = S^*$. We observe in Figure 2a, generally, for smaller models, models with $N < N_{th}$, increasing the sparsity level, and for larger models, models with $N > N_{th}$, increasing sparsity has a positive impact. More accurately, for a fixed compute budget the optimal sparsity level increases with model size and converges to 1 as the model size grows (see Figure 8 in §D.2 in the Appendix for more details).

If the model size has more parameters than a threshold N_{th} it is favorable to sparsify as much as possible. Note that the model with the lowest loss is not the largest sparsest model, i.e., there is a compute optimal model size even after MoE sparsity is introduced, and increasing total number of parameters would lead to under-training if training compute budget is fixed.

These results highlight the importance of balancing the number of parameters with FLOPs per example. Intuitively, when the total number of parameters is small, higher sparsity results in fewer active parameters, and thus fewer FLOPs per example. We speculate that this reduction in FLOPs per example may lead to inefficiencies during both training and inference. Conversely, when the total number of parameters is large, a fixed compute budget may not allow sufficient training on enough tokens to make use of the model's additional capacity.

3 Impact of Training Compute Budget on the Interaction between Model Parameters and Sparsity

Does increasing compute budget impacts the interaction between the parameters and compute per example and how they contribute to model's capacity? In other words, does the recipe for optimally increasing model capacity change as we scale up the training budget?

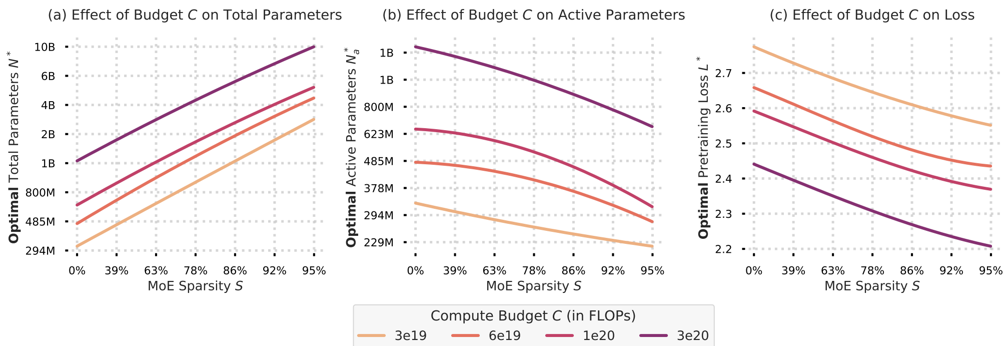

Figure 3: **Effect of compute budget on model size, number of active parameters and loss with sparsity.** Over all budgets considered, we observe that (a) the optimal model size N increases with sparsity, (b) the optimal number of active parameters N_a decreases with sparsity, and (c) the loss L decreases with sparsity.

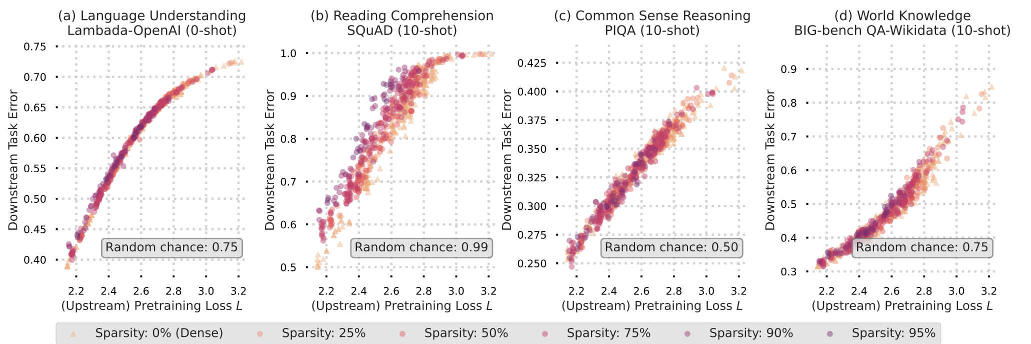

Figure 4: **Effect of sparsity on downstream vs upstream performance.** Downstream error shows a tight relationship with pretraining ("upstream") loss across downstream tasks across all sparsity levels.

To answer this question. in Figure 3 we illustrate the trends for changing the total number of parameters, N^*, the number of active parameters, N_a^*, and the loss, L^*, with sparsity level across different compute budgets.

Figure 3c shows that the optimal sparsity level approaches 1 across all compute budgets used in our experiments. There is no significant difference observed in the slope of the loss vs sparsity curves across different training compute budgets used in our experiments. This observation suggests that there is no diminishing effect of sparsity on the pretraining loss as we increase training compute budget, i.e., if there is no constraint on the model size, sparsity improves the performance of the model across all training budgets. As shown in §2.2, when model size in terms of total number of parameters is fixed, optimal sparsity level does now always approach 1.0, and it decreases as we increase the training compute budget (see Appendix D.2 in Figure 8).

Furthermore, as we see in Figures 3a and 3b, across all training compute budgets, we see a consistent trend of increasing N and decreasing N_a for compute optimal models as sparsity level increases. However, the rate of decreasing N_a, which can be interpreted as inference cost, increases as we increase training compute budget. This means, at larger training compute budgets, the benefit of reducing compute in terms of FLOPs per example amplifies. i.e., the gains of increasing sparsity level to reduce inference cost becomes more significant at larger training budgets.

4 Effect of MoE Sparsity on Downstream Task Performance

In this section, we study how sparsity affects the relationship between upstream and downstream performance of MoEs. In other words, does sparsity impact the relative gains from improvements in pretraining tasks on downstream tasks?

We use downstream tasks from the evaluation suite in `llm-foundry`[2] for benchmarking our pretrained models. The downstream task are devided into four pre-defined categories namely: language

[2]Github repository: https://github.com/mosaicml/llm-foundry

understanding, world knowledge, reading comprehension, and symbolic reasoning to help us systematically test whether the downstream vs upstream performance trend remains the same or is different as we vary sparsity values.

We observe from Figure 4a (language understanding), Figure 4c (common sense reasoning) and Figure 4d (world knowledge) that there is a tight relationship between upstream (pretraining) loss and downstream performance (error) across all these tasks. However, Figure 4b (reading comprehension) shows an example of a task where models with higher sparsity transfer worse compared to denser models. This decrease in the transfer performance of sparser models on these tasks maybe due to the lower inference-time compute in sparser models over their denser counterparts for similar pretraining loss. Further analysis in needed to verify this intuition. If fewer FLOPs per example is the reason behind worse transfer performance in sparser models, this effect might diminish at larger total training compute budget. Moreover, one can leverage approaches like chain-of-thought reasoning to independently increase FLOPs per example during inference time

While our results may indicate that there maybe no additional benefit obtained via sparsity in MoEs, we caution the reader that this suggestion maybe an artifact of the scale of our experiments. In the end, since, as shown in §2, sparser models are more efficient both in terms of training and inference cost (when measured in terms of theoretical FLOPs); we can reach a better pretraining performance with higher sparsity levels at a lower cost, which can translates to better downstream performance.

5 Conclusion

This paper underscores the role of sparsity in the scaling laws for Mixture-of-Expert Transformers (MoEs), showing that the most efficient model configuration depends on balancing model size, training compute, and sparsity level. The optimal recipe for balancing FLOPs per example and parameter count in MoEs depends on the objective as well other resource constraints. Our findings indicate that sparsity, as a knob that controls FLOPs per example in MoEs, is a powerful mechanism for optimizing model performance under constrained training compute budgets. By balancing the total number of parameters, compute, and sparsity, MoEs can be scaled more effectively. These insights provide valuable guidance for scaling language models, especially for MoEs, where the trade-offs between parameters and FLOPs must be carefully managed.

MoEs were originally introduced to allow increasing model capacity without a significant increase in inference cost. Our experiments show that under fixed total training compute budget increasing sparsity in MoEs leads to smaller FLOPs per example, higher number of parameters, and lower pretraining loss simultaneously. In other words, in the context of MoEs, if there are no constraints on the total number of parameters, increasing the capacity of the model through parameter count seem to be the optimal strategy if lower pretraining loss is the main goal. On the other hand, when comparing how well the pretraining performance transfers to various downstream tasks, denser models seem to be better on certain types of task that potentially rely on deeper processing of the input vs the knowledge stored in the parameters of the model. This potentially signals the importance of the role of FLOPs per example in increasing the capacity of the model during inference. Furthermore, under conditions where memory, i.e., number of total parameters, is a constraint, we find that there is an optimal sparsity value that depends both on the total number of parameters and total training compute budget.

It is also noteworthy that, in this paper, we have prioritized training compute-optimal models, in contrast to many published results on large language models (LLMs), which often rely on overtrained models. As a result, the performance of the models we use for the analysis in this paper is not directly comparable to those of other studies, where they overtrain smaller language models, to reduce the cost of inference relative to training.

Future work will examine the optimal balance of FLOPs per example and parameter count with more emphasis and in depth analysis on performance of the models on different types of downstream tasks, as well as investigating how the finding on the role of sparsity in MoEs extend to model architectures or approaches with different mechanisms to change FLOPs per example and number of trainable parameters of the models independently. More specifically, an interesting follow-up is to investigate the scaling behaviors of the models which allow negative sparsity values (through parameter sharing).

References

[1] BIG-bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

[2] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[4] A. Clark, D. d. l. Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud, G. v. d. Driessche, E. Rutherford, T. Hennigan, M. Johnson, K. Millican, A. Cassirer, C. Jones, E. Buchatskaya, D. Budden, L. Sifre, S. Osindero, O. Vinyals, J. Rae, E. Elsen, K. Kavukcuoglu, and K. Simonyan. Unified scaling laws for routed language models. In *Proceedings of the 39th International Conference on Machine Learning*. PMLR, 2022.

[5] T. Computer. Redpajama: An open source recipe to reproduce llama training dataset. https://github.com/togethercomputer/RedPajama-Data, Apr. 2023. Accessed: YYYY-MM-DD.

[6] R. Csord'as, K. Irie, J. Schmidhuber, C. Potts, and C. D. Manning. Moeut: Mixture-of-experts universal transformers. *ArXiv*, abs/2405.16039, 2024. URL https://api.semanticscholar.org/CorpusID:270063139.

[7] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[9] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. Bosma, Z. Zhou, T. Wang, Y. E. Wang, K. Webster, M. Pellat, K. Robinson, K. S. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. V. Le, Y. Wu, Z. Chen, and C. Cui. Glam: Efficient scaling of language models with mixture-of-experts. *ArXiv*, abs/2112.06905, 2021. URL https://api.semanticscholar.org/CorpusID:245124124.

[10] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), jan 2022. ISSN 1532-4435.

[11] T. Gale, D. Narayanan, C. Young, and M. Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. *Proceedings of Machine Learning and Systems*, 5, 2023.

[12] S. Goyal, Z. Ji, A. S. Rawat, A. K. Menon, S. Kumar, and V. Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

[13] T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, C. Hallacy, B. Mann, A. Radford, A. Ramesh, N. Ryder, D. M. Ziegler, J. Schulman, D. Amodei, and S. McCandlish. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv: Arxiv-2010.14701*, 2020.

[14] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. Rae, and L. Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

[15] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/pdf/2001.08361.pdf.

[16] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

[17] J. Ludziejewski, J. Krajewski, K. Adamczewski, M. Pióro, M. Krutul, S. Antoniak, K. Ciebiera, K. Król, T. Odrzygóźdź, P. Sankowski, M. Cygan, and S. Jaszczur. Scaling laws for fine-grained mixture of experts. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=Iizr8qwH7J.

[18] N. Muennighoff, L. Soldaini, D. Groeneveld, K. Lo, J. Morrison, S. Min, W. Shi, P. Walsh, O. Tafjord, N. Lambert, Y. Gu, S. Arora, A. Bhagia, D. Schwenk, D. Wadden, A. Wettig, B. Hui, T. Dettmers, D. Kiela, A. Farhadi, N. A. Smith, P. W. Koh, A. Singh, and H. Hajishirzi. Olmoe: Open mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2409.02060.

[19] OpenAI. Gpt-4 technical report. *PREPRINT*, 2023.

[20] P. Rajpurkar, R. Jia, and P. Liang. Know what you don't know: Unanswerable questions for SQuAD. In I. Gurevych and Y. Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

[21] S. Reddy, D. Chen, and C. D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016.

[22] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

[23] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al. Gemini: A family of highly capable multimodal models, 2024. URL https://arxiv.org/abs/2312.11805.

[24] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD. Survey Certification.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

[26] M. Wortsman, P. J. Liu, L. Xiao, K. Everett, A. Alemi, B. Adlam, J. D. Co-Reyes, I. Gur, A. Kumar, R. Novak, et al. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

[27] L. Yun, Y. Zhuang, Y. Fu, E. P. Xing, and H. Zhang. Toward inference-optimal mixture-of-expert large language models. *arXiv preprint arXiv:2404.02852*, 2024.

[28] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, and W. Fedus. ST-MoE: designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

Appendices

A Preliminaries

In this section, we provide a brief overview of Mixture-of-Expert (MoE) Transformers.

A.1 Mixture-of-Expert (MoE) Transformers

Mixture-of-Experts Transformers modify the standard transformer architecture by introducing in the MLP layer. In this design, the experts are MLP (Multi-Layer Perceptron) modules that follow the attention mechanism and are selectively activated for each token. A gating mechanism determines which MLP experts are most relevant for each token, ensuring that only a subset of experts (top-k) is active at any given time, while the rest remain inactive. Below, we provide the notations used throughout the paper for various terms related to training MoEs.

Total and Active Parameters: In MoEs, we distinguish between total and active parameters, denoted by N and N_a, respectively. The total parameter count, N, includes all parameters of the network, encompassing both the experts and the rest of the architecture. The active parameter count, N_a, refers to the parameters associated with the active portion of the experts, along with the rest of the network that is always utilized.

Top-k Expert Selection: In MoEs, the gating mechanism assigns tokens to a subset of experts using a top-k selection process, where k denotes the number of experts activated for each token. The gate computes a relevance score for each expert, and the top k experts with the highest scores are selected and activated. This selective activation limits the computational overhead by ensuring that only a fraction of the experts are used per token.

Expansion Factor and Granularity: The expansion factor, typically denoted by E, represents the increase in model capacity due to the inclusion of multiple experts, measured as a multiplicative factor relative to the base dense model. The granularity, G, determines the size of each expert relative to the size of the MLP module in the base dense model. The total number of experts in the model is given by $E \times G$, where E scales the capacity and G controls the level of granularity.

Sparsity (S): In general, sparsity is defined as the ratio of inactive to total parameters. However, in the context of MoEs, we focus on the sparsity of the MLP modules specifically. Therefore, we define the sparsity level as the ratio of inactive to total experts, given by:

$$S = \frac{\text{number of non-active experts}}{\text{number of total experts}}. \tag{5}$$

This definition provides an interpretable measure of sparsity but cannot be directly used to calculate the active parameter count N_a due to the contribution of other parameters in the model that remain unsparsified.

B Experimental Setup

We train and evaluate auto-regressive sparse Mixture-of-Experts (MoE) language models of varying sizes and configurations on subsets of the RedPajamaV1 dataset [5]. The key variables we explore in our experiments are total model parameters N, training compute budget C, and the MoE sparsity S.

Pre-training data. Our models are pre-trained on subsets of the RedPajamaV1 dataset[3] [5], which attempts to replicate the LLaMA pre-training data recipe and comprises 1.2 trillion tokens from sources such as Common Crawl, C4, GitHub, and Wikipedia. In all our experiments, the effective dataset size is adjusted based on the training compute budget C and the model size N. We tokenize the data using the GPT-NeoX tokenizer [2], which has a vocabulary size of $50,432$ tokens.

Model and tokenizer. We use auto-regressive transformer-based MoE language models in order to study compute-parameter trade-offs by varying MoE sparsity. We use the Megablocks library [11] to train dropless MoEs in which the routing mechanism ensures that all tokens are efficiently routed without being dropped due to routing capacity constraints.

Optimizer and scheduler. We optimize our models using the scale-free Adam optimizer[4] with variable learning rate, a weight decay of 1×10^{-5}, and fixed Adam-specific parameters $\beta = (0.9, 0.95)$ and $\varepsilon = 1 \times 10^{-8}$. We use a learning rate scheduler consisting of a linear warm-up phase followed by a cosine decay. The warm-up phase increases the learning rate from 0 to the base learning rate over a fraction of the total training steps (selected from $\{0.1, 0.05, 0.02\}$). After warm-up, the learning rate decays following a cosine schedule for the remaining training steps.

Fitting IsoFLOP surfaces. Recall that in Section 2, we fit isoFLOP surfaces to predict pretraining loss L as a polynomial function of model size N and MoE sparsity S for a fixed training budget C. The polynomial function takes the form

$$L(N, S) = a\hat{N}^{\alpha} + b\hat{S}^{\beta} + c(\hat{N} \cdot \hat{S})^{\gamma} + d \tag{6}$$

where $\hat{N} = \log N$ and $\hat{S} = -\log(1 - S)$—we find that applying log transformations improves the fit of the resulting isoFLOP surface. Through a grid search over the polynomial coefficients $\alpha, \beta, \gamma \in \{0, 1, 2, 3, 4\}$, we found that the best fit was obtained for $\alpha = \beta = \gamma = 2$, i.e., a quadratic polynomial. We evaluate the fitted isoFLOP surfaces in Figure 1 by (a) re-running the fitting procedure $k = 100$ times on randomly sub-sampled data and (b) evaluating the Pearson correlation between the true and predicted pretraining loss values on a set of held-out data points.

Hyperparameters. We fix a subset of hyperparameters for which changing values in preliminary experiments (a) did not significantly improve pre-training loss, (b) the optimal value remained the same across several model configurations, or (c) in order to reduce the search space (i.e., limited compute resources). Specifically, we first opted to use z-router loss [28] and qk-normalization [26] in order to stabilize training for large MoEs. Second, we fixed MoE router jitter noise to 0, as it did not improve performance. We also fixed our batch size to 2048 for all model sizes.

We swept over hyperparameters that, when adjusted, (a) significantly improved pre-training loss and (b) the optimal values varied across different model configurations. We increase the MoE sparsity by decreasing the number of active experts and/or increasing the number of total experts. We also varied the MoE granularity [17], MoE load balancing regularizer, Adam learning rate, and linear warm-up steps (fraction) in order to improve pre-training loss. The table below summarizes our hyperparameter sweeps:

[3] GitHub repository: https://github.com/togethercomputer/RedPajama-Data
[4] Scale-free Adam: https://fabian-sp.github.io/posts/2024/02/decoupling/

Table 1: Hyperparameter configurations and search spaces

Hyperparameter	**Configuration**	**Search Space**
Sparsity Level	Tuned	$\{0, 25, 50, 75, 90, 95, 98\}\%$
Number of Total Experts	Tuned	Adjusted depending on sparsity
Number of Active Experts	Tuned	Adjusted depending on sparsity
Granularity	Tuned	$\{1, 2\}$
Learning Rate	Tuned	$[0.003, 0.002, 0.001]$
Load Balancing Factor	Tuned	$\{0.02, 0.05\}$
Warm-up Steps	Tuned	$\{2, 5, 10\}\%$
Batch Size	Constant	2048
Jitter Noise	Constant	0
z-Loss	Constant	0
z-Router Loss	Constant	0.001
QK Norm	Constant	Applied

C Estimating Mixture-of-Expert (MoE) FLOPs

Similar to prior work on scaling laws (e.g., [15, 14, 17]), we use theoretical FLOP estimates as proxies for training and inference costs of language models. In this section, we (a) outline our methodology for estimating FLOPs for MoEs and (b) show that the proposed estimator closely approximates empirical FLOPs of large-scale MoEs.

Setup and notation. Consider an MoE model with n_{layers} MoE layers, each with an embedding dimension of d_{model}. We denote the number of total experts and active experts in each MoE layer by E_{total} and E_{active} respectively. Following Ludziejewski et al. [17], we let G denote the MoE granularity, which defaults to 1 and controls the size of each expert relative to the size of a feed-forward layer in an equivalent dense transformer. In our experiments, we use a vocabulary size $n_{\text{vocab}} = 50,432$ context length n_{ctx} of 2048 and use GLU modules (Gated Linear Units) [22] over feed-forward modules as the architecture of choice for MoE experts. We also set the (a) hidden dimension of each GLU expert d_{ffn} to $4 \cdot d_{\text{model}}$ and (b) instantiate MoEs where the number of attention heads n_{heads} times the dimensionality for each head d_{head} equals d_{model}, i.e., $n_{\text{heads}} d_{\text{head}} = d_{\text{model}}$.

Estimating module-specific FLOPs. To estimate the FLOPs of a given MoE model, we first individually estimate the FLOPs per token incurred by a forward *and* backward pass through every module in MoEs. Then, we aggregate these estimates to obtain the final estimator for the FLOPs per token incurred by a forward *and* backward pass through the model.

Like in prior work [15, 14], we take a two-step approach to estimate module-specific FLOPs. Given a module, we first estimate the number of parameters in the module and then scale this with an appropriate constant corresponding to the number of add-multiply operations per parameter through a forward and backward pass of the given module. We also omit non-leading terms such as non-linearities, biases, and layer normalization in our estimation. We estimate the FLOPs per token for attention modules, MoE routers, MoE experts, and the final un-embedding layer as follows:

1. **Attention module.** We estimate the FLOPs incurred via the QKV (and final) projections, attention logits, and attention values of all heads in a multi-head attention module as follows.
   - *QKV (and final) projections.* These projections involve $4 \cdot d_{\text{model}} n_{\text{heads}} d_{\text{heads}} = 4d_{\text{model}}^2$ parameters. Following Kaplan et al. [15], we use the multiplicative constant $C = 6$ to account for the add-multiply operations per parameter in a forward and backward pass through linear modules, resulting in a FLOPs-per-token estimate of $4 \cdot C \cdot d_{\text{model}}^2$.
   - *Attention logits.* The FLOPs required to compute the attention logits for all n_{ctx} tokens equals $C \cdot n_{\text{ctx}}^2 d_{\text{model}}$ FLOPs, making the FLOP-per-token estimate equal to $C \cdot n_{\text{ctx}} d_{\text{model}}$.
   - *Attention values.* The computation of attention values requires a per-token weighted sum over n_{ctx} d_{model}-dimensional vectors, making the estimate $C \cdot n_{\text{ctx}} d_{\text{model}}$.
2. **MoE module.** Given an MoE layer, we estimate the FLOPs incurred by its router and all experts separately.
   - *Router.* The MoE routing linearly maps a d_{model}-dimensional token embedding to a E_{total}-dimensional logit vector, which is subsequently used to map the token to E_{active} active experts. Following Ludziejewski et al. [17], we use a multiplicative constant $R = 14$ that accounts for the add-multiply-route operations per router parameter. The resulting FLOP estimate equals $R \cdot d_{\text{model}} E_{\text{total}}$
   - *Experts.* Each MoE experts corresponds to a GLU module [22] with $d_{\text{ffn}} = 4 \cdot d_{\text{model}}$. Since there are E_{active} active experts with granularity G, each involving three linear projections, this results in a FLOP estimate of ${}^{1}\!/_{G} \cdot 3 \cdot E_{\text{active}} \cdot C \cdot d_{\text{model}} d_{\text{ffn}} = {}^{12C}\!/_{G} \cdot E_{\text{active}} \cdot d_{\text{model}}^2$.
3. **Un-embedding layer.** The un-embedding linear layer maps the final d_{model}-dimensional embedding of a token to n_{vocab}-dimensional logits, making the FLOPs-per-token $C \cdot n_{\text{vocab}} d_{\text{model}}$.

Estimating MoE FLOPs. We can aggregate the module-level FLOP estimates described above to estimate the FLOPs per token required for a single forward and backward pass through a given MoE model as follows:

$$n_{\text{layer}} \left(4Cd_{\text{model}}^2 + 2Cd_{\text{model}} n_{\text{ctx}} + {}^{12C}\!/_{G} E_{\text{active}} d_{\text{model}}^2 + Rd_{\text{model}} E_{\text{total}} \right) + Cn_{\text{vocab}} d_{\text{model}}$$

When $E_{\text{total}}/d_{\text{model}}$ is small, which is typically the case in practice, the FLOPs induced by MoE routing can be ignored as they contribute negligibly to the estimator. This allows us to simplify the estimator to:

$$\text{MoE FLOPs per token} \coloneqq C \cdot n_{\text{layers}} d_{\text{model}}^2 \left(4 + \frac{2n_{\text{ctx}}}{d_{\text{model}}} + \frac{12E_{\text{active}}}{G} + \frac{n_{\text{vocab}}}{d_{\text{model}} n_{\text{layers}}}\right) \tag{7}$$

Evaluating $6N_aD$ as a FLOPs-per-token estimator in MoE Models For standard dense transformers, the FLOPs are often estimated as $6ND$ [15, 14]. Given that D is fixed and not adjusted dynamically, N can serves as a reliable relativeestimator of FLOPs per token for dense transformer models.

To adapt the $6ND$ estimator for MoE models, we replace N with N_a (the active number of parameters)—the number of parameters used in every forward and backward pass. In Figure 5, we evaluate the accuracy of the $6N_aD$ estimator by plotting the ratio between the MoE FLOPs estimator described in Equation 7 and $6N_aD$ as a function of model size N and a fixed context length $D = 2048$. The results show that, across all sparsity levels, the ratio remains close to one, and the gap between the two estimators decreases as model size N increases.

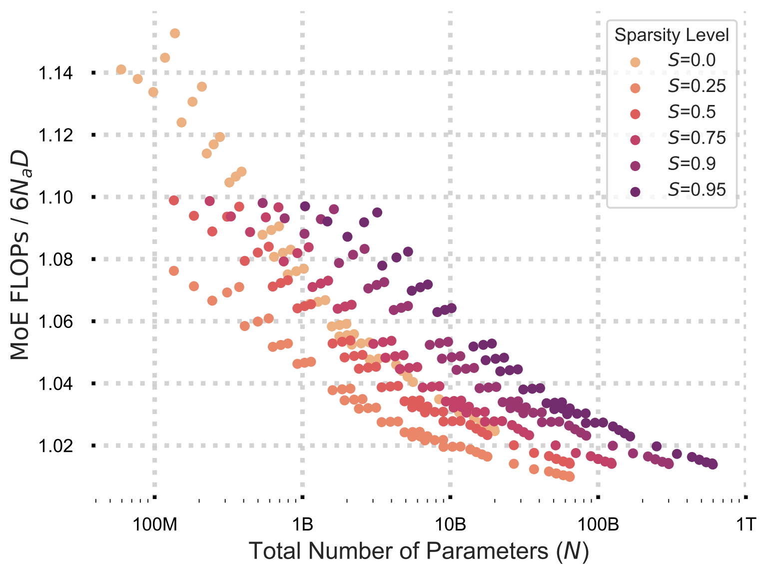

Figure 5: **Accuracy of $6N_aD$ FLOPs Estimator for MoEs**. Ratio of the MoE FLOPs estimator (Equation 7) to the $6N_aD$ estimator as a function of the total number of parameters, for a fixed context length of $D = 2048$, used in our experiments.

D Additional Analysis

D.1 Interplay between parameters and FLOPs per example

Recall that in Section 2, we showed that isoFLOP curves were predictive of pretraining loss for different parameter counts and sparsity levels. In this section, we show similar results with additional training compute budgets.

1. In Figure 6, we first show that IsoFLOP surfaces mapping model size N and sparsity level S to pre-training loss L are predictive for all training compute budgets that we consider, ranging from 3e19 to 1e21 FLOPs.
2. In Figure 7, we analyze the fitted IsoFLOP surfaces (one for each training budget) and find that the (a) effect of model size N on optimal MoE sparsity S^* and (b) the effect of MoE sparsity S on the optimal total and active parameters, $N*$ and N_a^*, is similar for all training budgets.

D.2 Effect of training budget and model size on optimal MoE sparsity

Recall that Section 3, we demonstrated how the relationship between optimal total parameters $N*$, optimal active parameters $N*_a$, and optimal pretraining loss L predictably changes as a function of sparsity S and training budget C. In this section, we use the fitted isoFLOP surfaces to analyze how the optimal MoE sparsity S^* changes as a function of total parameters N and training budget C, as shown in Figure 8. Our main findings are:

- Across all training budgets (ranging from 3e19 to 3e20 FLOPs), increasing the total parameters N leads to an increase in the optimal sparsity level S^*.
- For a fixed model size (i.e., total parameters N), increasing the training budget C generally reduces the optimal sparsity level S^*.
- The relationship between model size N and optimal $S*$ is not linear. For smaller models (up to about $500 \cdot 10^6$ parameters), the optimal sparsity remains at 0 (i.e., dense) for most compute budgets.

D.3 Effect of sparsity on downstream task performance

In Section 4, we analyzed the relationship between upstream pre-training loss and downstream task performance across different MoE sparsity levels. We found that language understanding and world knowledge tasks generally showed a strong correlation between upstream and downstream performance, while reading comprehension tasks seemed to favor denser models to some extent.

In this section, we provide additional plots for a broader range of tasks within each category to further support our findings. We consider the following tasks:

- **Common Sense Reasoning**: PIQA, CommonSenseQA, OpenBookQA, COPA
- **Language Understanding**: LAMBADA, HellaSwag, Winograd, Winogrande
- **Reading Comprehension**: SQuAD, CoQA, BoolQ
- **World Knowledge**: TruthfulQA, ARC-Easy, ARC-Challenge

Figure 9 shows the relationship between upstream pre-training loss and downstream task performance for these additional tasks. Each row corresponds to a task category and each subplot represents a different task, with points colored according to MoE sparsity S. The x-axis represents the upstream pre-training loss, while the y-axis shows the downstream task performance metric (usually accuracy or error rate). These results supplement our main findings from Section 4:

- We observe consistent trends across tasks within each category, with language understanding and world knowledge tasks showing strong correlations between upstream and downstream performance regardless of sparsity.
- Reading comprehension tasks continue to show a slight advantage for denser models, while common sense reasoning tasks (which can be considered part of the symbolic problem-solving category) show more varied relationships between upstream and downstream performance.

D.4 Comparing IsoFLOP Surface Analysis with Independent 2d IsoFLOPs

Recall that in Section 2, we used IsoFLOP surfaces that predict pre-training loss across varying parameter counts and sparsity levels to understand how optimal sparsity and optimal model size depend on each other.

In this section, we evaluate whether these findings remain consistent when we do not rely on fitted IsoFLOP surfaces. Specifically, similar to Approach II in Hoffmann et al. [14], we directly fit univariate quadratic functions that map model size N to pre-training loss L, independently for each sparsity level and training compute budget. We then assess these univariate fits to determine whether our findings in Section 2 hold.

- In Figure 10, each row shows how the optimal total and active parameters change as a function of MoE sparsity for fixed training budgets. As in our findings from Section 2 (Figure 2), increasing sparsity increases the optimal total parameters while decreasing the optimal active parameters. Moreover, larger compute budgets still result in higher optimal total and active parameters, regardless of the sparsity level.
- Furthermore, in Figure 11, we observe that across all training compute budgets, increasing sparsity reduces the optimal pre-training loss. This is consistent with the trends identified in Section 3 (Figure 3), thereby validating our earlier results.

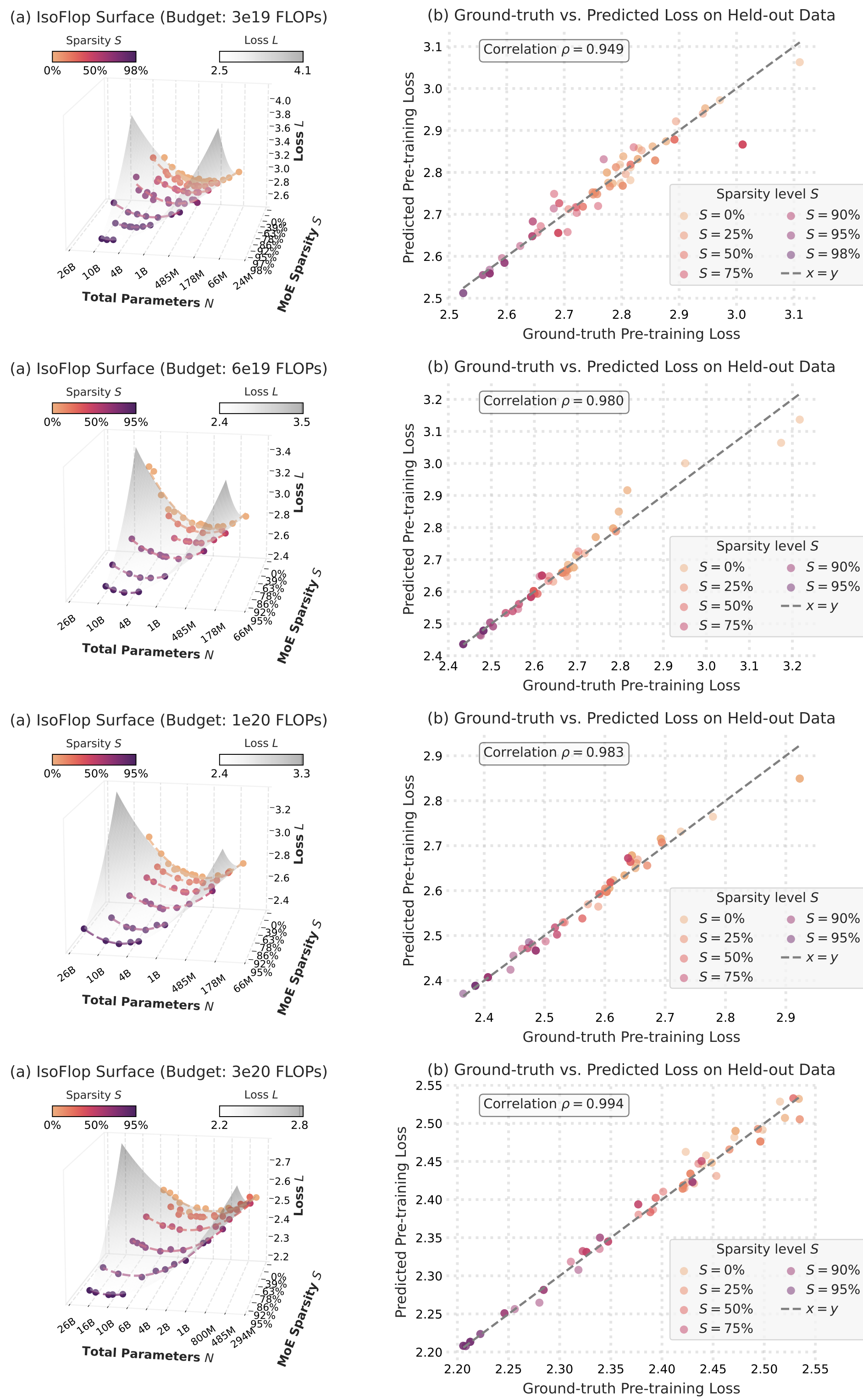

Figure 6: **IsoFLOP surfaces over total parameters N, MoE sparsity S, and pretraining loss L for different compute budgets**. The rows correspond to IsoFLOP surface fitted using models trained with a budget of 3e19, 6e19, 1e20, 3e20, and 1e21. The subplots on the left visualize IsoFLOP surfaces mapping total parameters N and sparsity level S to pretraining loss L. The subplots on the right correlate the ground-truth pretraining loss with the estimated pretraining loss on held-out data. Taken together, these results show that isoFLOP surfaces are accurate proxies for understanding how model size and MoE sparsity jointly impact pretraining loss.

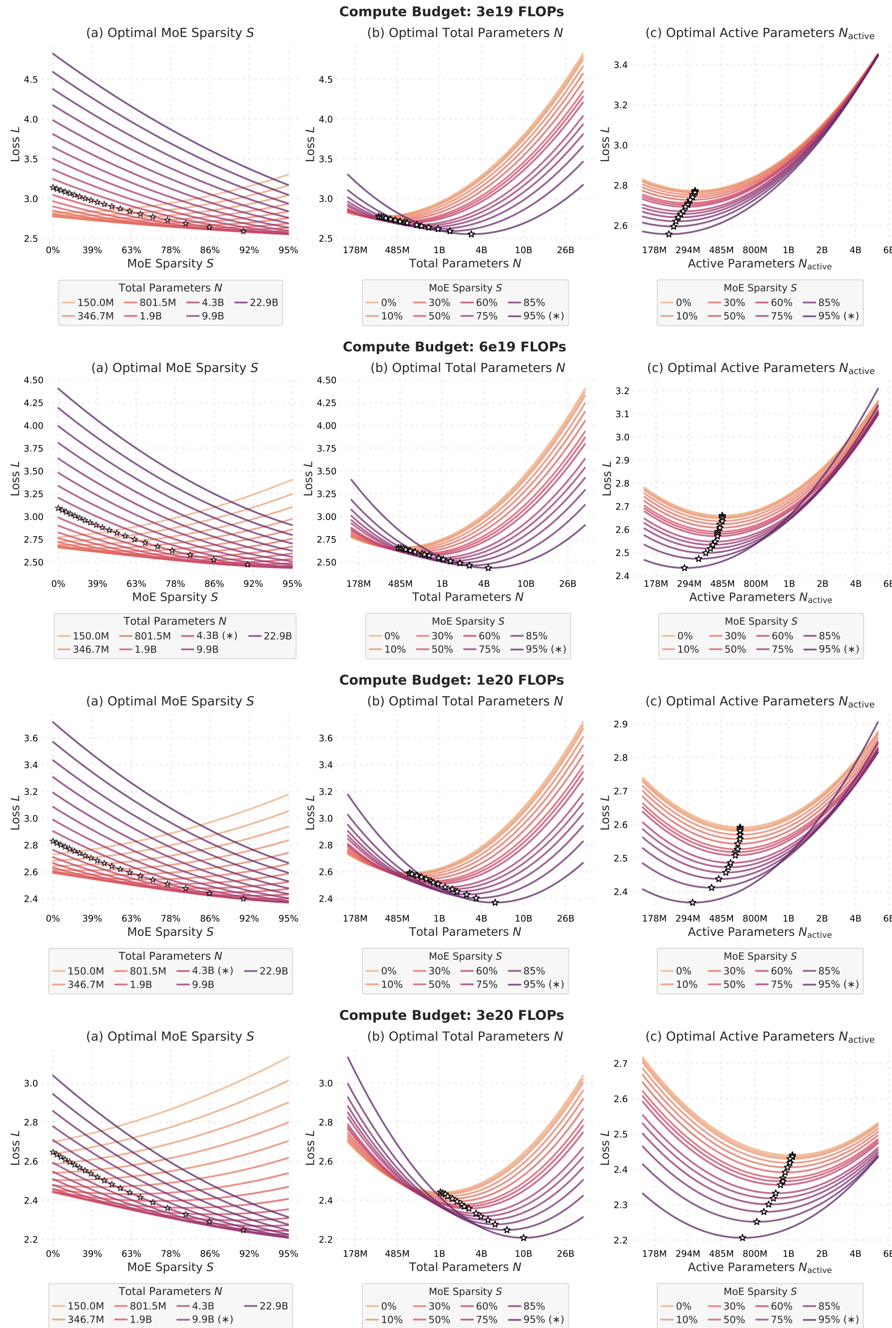

Figure 7: **Optimal MoE configurations predictably change with training compute budget.** Each row corresponds to an analysis of how optimal MoE sparsity S^*, total parameters N^*, and active parameters N_a^* change for a given training budget. The subplots on the left show that (a) increasing the training budget increases the model size N (denoted with black dots) with the minimum pretraining loss and (b) for models smaller than a threshold (which increases with training budget), dense models (i.e., 0% sparsity) fare better than sparse MoEs. The subplots in the second and third panel show that (a) increasing MoE sparsity increases the optimal total parameters N^* and decreases the optimal active parameters N_a^*. In both cases, for a fixed sparsity level, increasing the budget shifts increases the optimal total and active parameters.

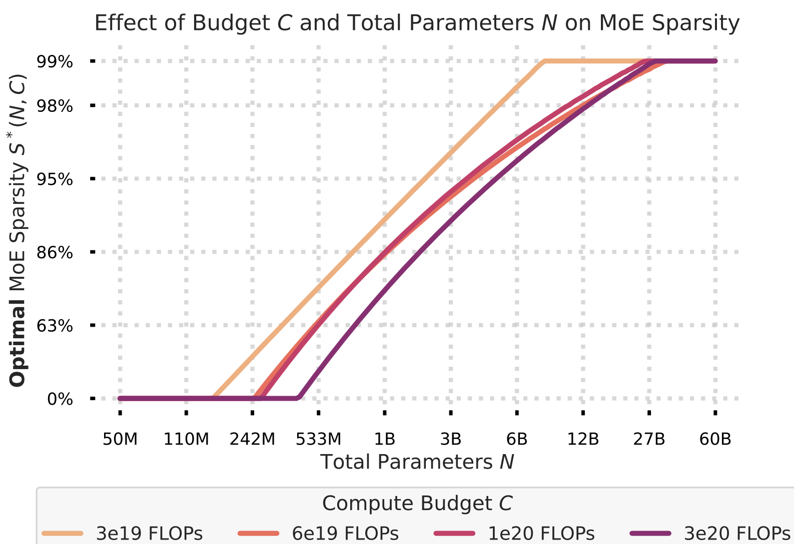

Figure 8: **Effect of training budget C and total parameters N on MoE sparsity.** This figure shows how the optimal MoE sparsity S^* changes with respect to the total number of parameters N and the training budget C. The x-axis represents the total parameters N on a logarithmic scale and the y-axis shows the optimal MoE sparsity S^*.

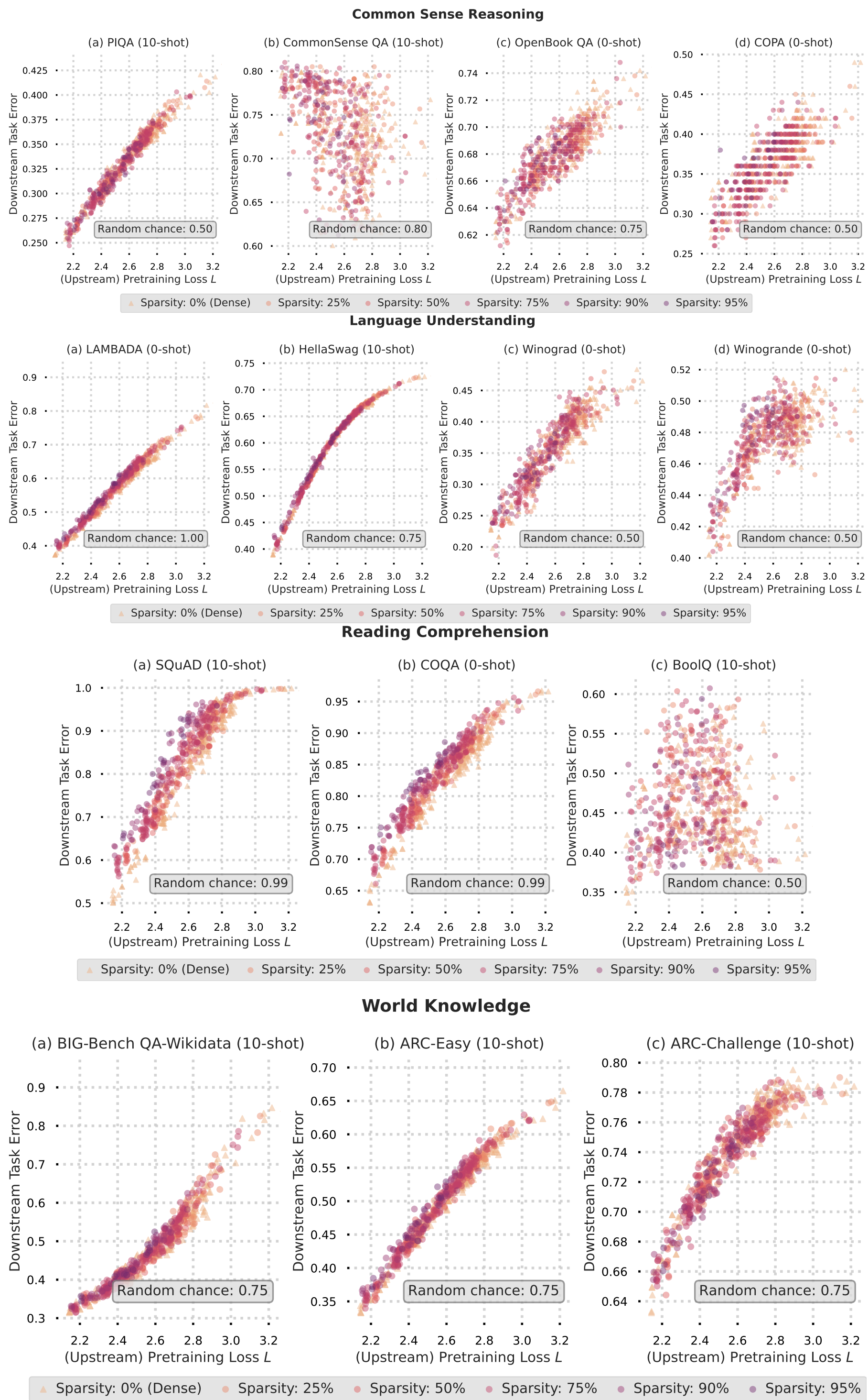

Figure 9: **Downstream task performance vs. upstream pre-training loss.** Each subplot shows the relationship between upstream pre-training loss (x-axis) and downstream task performance (y-axis) for a specific task. Similar to our results in Section 4, we find that the MoE sparsity level does not change the relationship between upstream pre-training loss and downstream task performance.

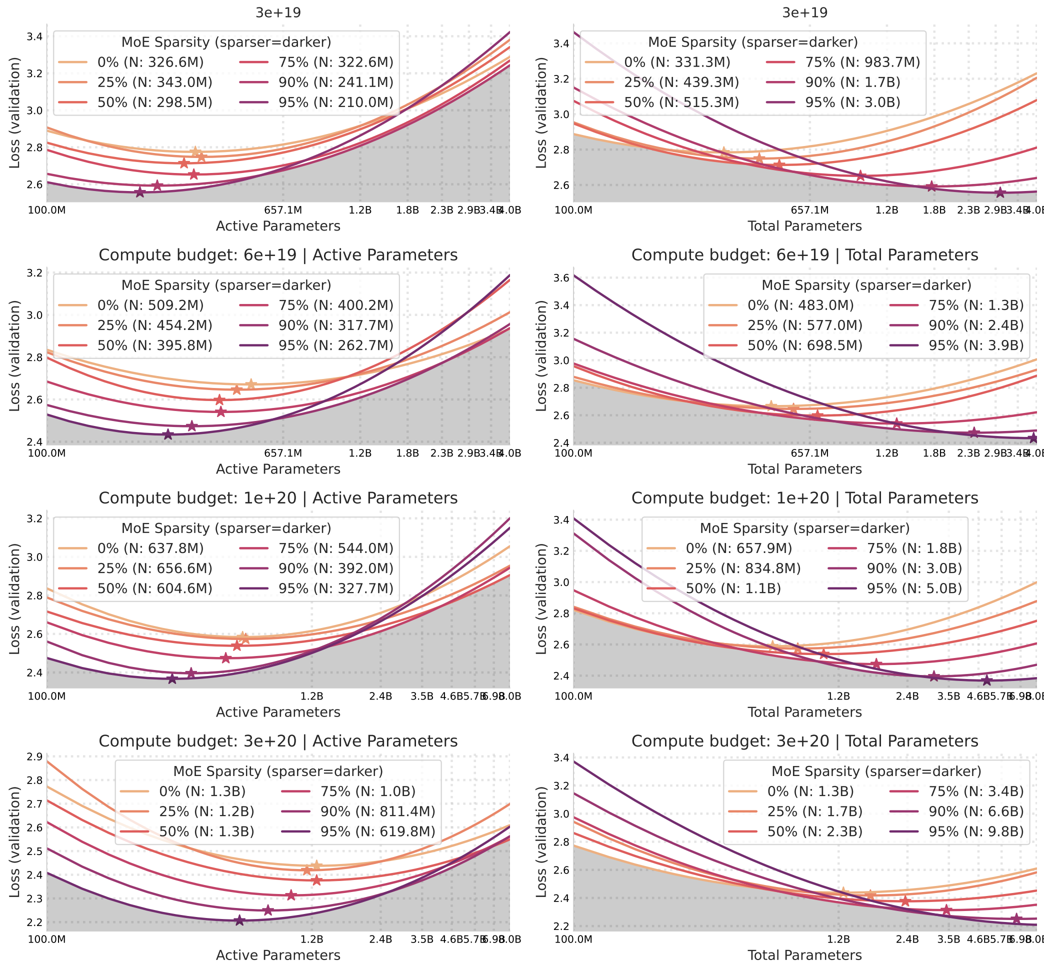

Figure 10: **Effect of MoE sparsity on optimal total and active parameters across different training compute budgets.** Each row shows the change in total and active parameters as a function of sparsity level for fixed training budgets. Increasing sparsity leads to an increase in the optimal total parameters while reducing the optimal active parameters, consistent with our findings in Section 2 (Figure 2). Larger training compute budgets result in higher optimal (total and active) parameters across all sparsity levels.

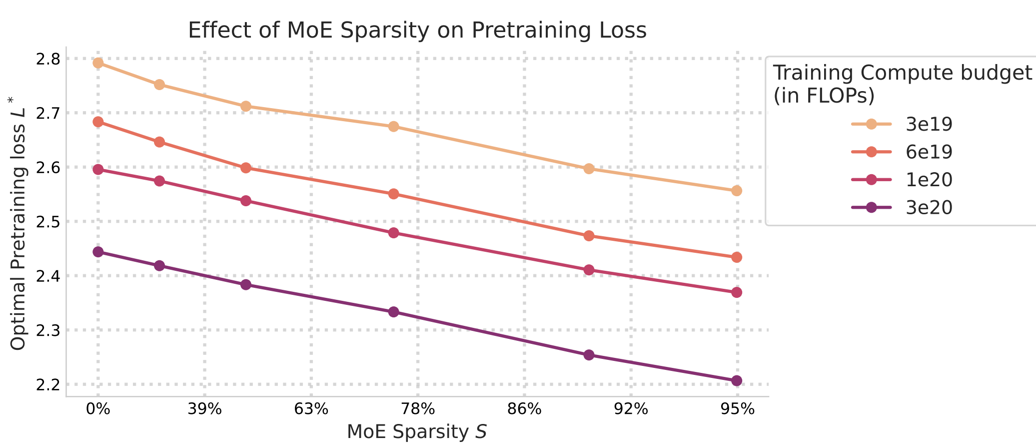

Figure 11: **Effect of MoE sparsity on pretraining loss across different training compute budgets**. As sparsity increases, the validation loss decreases for all compute budgets, with larger budgets (darker lines) achieving lower losses at each sparsity level. This trend is consistent with the findings from Section 3, demonstrating that increasing sparsity reduces the optimal pretraining loss across all compute budgets.