Nexus: Specialization meets Adaptability for Efficiently Training Mixture of Experts

Nikolas Gritsch[1,2] **Qizhen Zhang**[3†] **Acyr Locatelli**[2] **Sara Hooker**[1] **Ahmet Üstün**[1]
[1]Cohere For AI [2]Cohere [3]University of Oxford
{nikolasgritsch,acyr,sarahooker,ahmet}@cohere.com
qizhen.zhang@eng.ox.ac.uk

Abstract

Efficiency, specialization, and adaptability to new data distributions are qualities that are hard to combine in current Large Language Models. The Mixture of Experts (MoE) architecture has been the focus of significant research because its inherent conditional computation enables such desirable properties. In this work, we focus on "upcycling" dense expert models into an MoE, aiming to improve specialization while also adding the ability to adapt to new tasks easily. We introduce Nexus, an enhanced MoE architecture with *adaptive routing* where the model learns to project expert embeddings from domain representations. This approach allows Nexus to flexibly add new experts after the initial upcycling through separately trained dense models, without requiring large-scale MoE training for unseen data domains. Our experiments show that Nexus achieves a relative gain of up to 2.1% over the baseline for initial upcycling, and a 18.8% relative gain for extending the MoE with a new expert by using limited finetuning data. This flexibility of Nexus is crucial to enable an open-source ecosystem where every user continuously assembles their personalized MoE-mix according to their needs.

1 Introduction

In an era of bigger and bigger models [Canziani et al., 2016, Strubell et al., 2019, Rae et al., 2021, Raffel et al., 2020, Bommasani et al., 2022, Hooker, 2024], there are several key objectives driving state-of-art progress. Doing *more with less* by improving efficiency [Treviso et al., 2023] remains paramount, but in addition to efficiency the deployment of these models in the wild means that the ability to adapt to new data [Pozzobon et al., 2023b, Gururangan et al., 2020a, Jang et al., 2022, Jin et al., 2022], and specialization of compute [Zadouri et al., 2024, Shazeer et al., 2018, Riquelme et al., 2021, Du et al., 2022, Fedus et al., 2022] have gained renewed focus. While all these properties are desirable, a formidable challenge is designing architectures that can fulfill *all* of these requirements.

The Mixture-of-Expert (MoE) approach gained prominence because of its efficiency properties. In contrast to dense models which require significant compute to deploy, MoE approaches only activate a subset of the parameters for every single token. Intuitively, not all parameters are necessary for each request, as some parameters will specialize on certain tasks, and those unrelated to the current request can be ignored. However, while MoEs greatly improved efficiency, the ability to induce meaningful specialization has been more limited with observations that experts don't appear to exhibit dedicated expertise [Jiang et al., 2024, Zoph et al., 2022, Zadouri et al., 2023]. Furthermore, MoEs tend to suffer from severe training instabilities [Zoph et al., 2022].

Recent work has attempted to address both the training instabilities and the lack of specialization. These techniques often train completely separate experts and "upcycle" (combine) them into a single

[†]Work done at Cohere

Workshop on Adaptive Foundation Models at NeurIPS 2024.

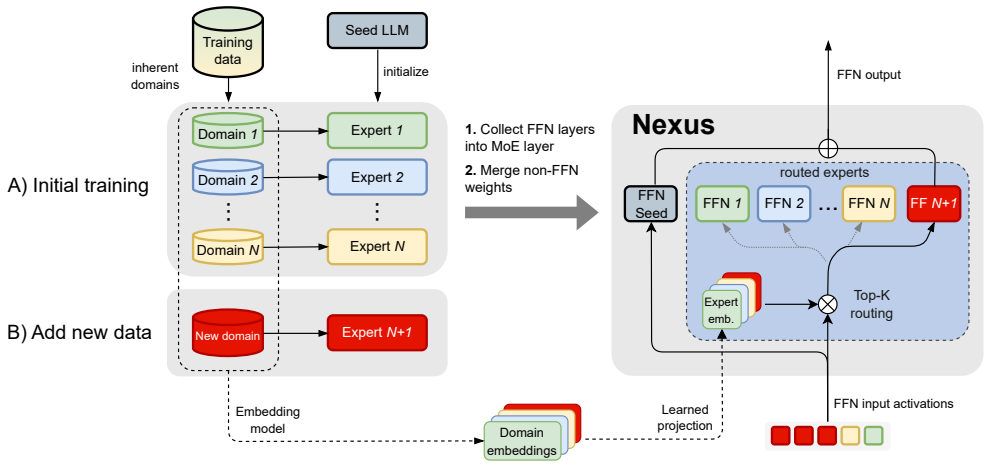

Figure 1: **The Nexus architecture: A)** Each expert is trained separately and an embedding of its training data is stored. The experts are combined by extracting their FFNs into the MoE layer, and finetuning the model on a mix of all domains. We perform top-1 routing based on the similarity of the input data with each of the transformed expert embeddings. The seed model FFN is used as the shared expert and always activated. **B)** Later, we can add a new expert by appending its training data embedding to the existing domain embeddings.

unified MoE model *after* dense training [Sukhbaatar et al., 2024]. This reduces the memory and communication cost, and improves ***efficiency*** during training as computations are more local and cross-device communication is reduced [Li et al., 2022, Gururangan et al., 2023]. Notably, the other major advantage of these approaches is the increase in ***specialization*** with separate experts that are trained on specific domains, making them clearly responsible for their human-interpretable subset of the data. On the other hand, MoEs with a standard router, which needs to be trained on a mix of all training data, are not designed to maintain domain specialization [Jiang et al., 2024].

However, efficiently integrating new experts into upcycled MoE models - a setting that is of great interest for ***adaptability*** objectives is far less studied. For most practitioners, given the scale of modern LLMs [Brown et al., 2020, Touvron et al., 2023, Kaplan et al., 2020, Anil et al., 2023] training MoEs repeatedly is an infeasible computational cost. Furthermore, most model development fails to take into account distribution drift in use cases, with limited flexibility and applicability across different tasks and domains [Pozzobon et al., 2023a, Gururangan et al., 2020b]. However, human language is shaped by a cumulative culture, constantly building upon itself and evolving over time [Silvey, 2016]. Also, specialized use cases such as multilingual, code and math often require tailored additional training.

In this work, we attempt to reconcile all three desirable properties: *efficiency, specialization, and adaptability.* We ask "*how can we adaptively combine separately trained specialized experts?*" To address this, we introduce **Nexus**, a novel MoE architecture that parameterizes the router based on domain-specific data by learning to project the embedding of each data domain to an expert embedding. This learnable projection for the router allows for the easy extension of the MoE model with new experts that are trained independently on new datasets of interest. This also avoids the difficulties of MoE training, as our learned router scales with the number of experts without needing to be trained from scratch, which enables adding or removing experts as desired.

Our experiments show that Nexus outperforms previous work when upscaling an MoE from separately trained specialized domain experts. Going beyond the single upscaling phase, Nexus can be efficiently extended with a new expert trained on a new domain, by finetuning it with much fewer tokens, compared to the finetuning after the initial upcycling.

In summary, our contributions are as follows: **1)** We present Nexus, a novel MoE framework designed to enhance sparse upcycling of specialized dense experts, while reducing the training cost of MoEs by facilitating easy adaptation to unseen data distributions. In Nexus, the traditional linear router from vanilla MoE models is replaced with routing based on the similarity of layer inputs to an expert embedding vector, derived from the average embedding of the corresponding expert dataset. **2)** Our

method outperforms the existing approach for upcycling specialized models into MoE, leading to 2.1% and 1.6% relative increase over the upcycled MoE (linear router) in 470M and 2.8B scales respectively. This enables performance increase in general tasks with 5.8% and 7.4% relative gains over the dense seed model at 470M and 2.8B respectively. **3)** Our method enables efficient adaptation to new domains by extending upcycled MoE with the new experts trained on unseen dataset. In this setting, Nexus outperforms the baseline MoE (linear router) when finetuning on the limited amount of data, leading 18.8% relative gain on the new domain with 1B finetuning tokens upon MoE extension. **4)** We show that our method is robust across different load balancing and data mixtures, and consistently outperforms the MoE with a linear router for specialized upcycling, confirming the benefits of the *adaptive* routing based on domain projections used in Nexus.

2 Adaptive Router for Upcycling Specialized Experts as MoE

The core component of an MoE model is the router, as it determines which experts to activate for any given input. In vanilla MoEs, the router is a learned linear layer that takes the token intermediate representations as input and computes the expert probabilities. However, this router does not necessarily learn specialization as MoEs are commonly trained using an auxiliary load balancing loss to improve training stability [Fedus et al., 2022, Jiang et al., 2024]. In Nexus, we propose a novel MoE router where per MoE block we learn a projection layer from given pre-computed domain embeddings to expert embeddings. We parametrize this projection layer P_r as a two-layer MLP with a SwiGLU activation function [Shazeer, 2020]:

$$\begin{aligned} e_i^{\text{expert}} &= P_r(e_i^{\text{domain}}) && \text{(Domain to Expert Embeddings)} \\ &= W_2 \cdot \text{SwiGLU}(W_1 \cdot e_i^{\text{domain}}) \end{aligned}$$

where $e_i^{\text{domain}} \in \mathbb{R}^d$, and $e_i^{\text{expert}} \in \mathbb{R}^h$ are the domain and expert embeddings for the ith domain respectively, with d and h as the domain embedding and the model dimensions. $W_1 \in \mathbb{R}^{2h \times d}, W_2 \in \mathbb{R}^{h \times h}$ are linear layers, and SwiGLU is defined as $\mathbb{R}^{2h} \to \mathbb{R}^h$. Given the expert embeddings e_i^{expert} and layer inputs $x \in \mathbb{R}^{s \times h}$, we then compute routing probabilities s_i using $\texttt{softmax}(x \cdot e_i^{\text{expert}})$.

Unlike the standard router, Nexus's router includes a stronger inductive bias through pre-computed domain embeddings[1] that enables expert embedding to specialize. Thus, $x \cdot e_i^{\text{expert}}$ gives a high value for input tokens that are closer to the domain of the corresponding expert. Notably, this router is particularly suited for the sparse upcycling setting where the dense experts are separately trained on different domains.

Upcycling dense experts as an MoE. After training dense expert models, we merge the individual experts into a unified MoE by appending their FFNs along a new dimension to create an MoE layer per Transformer block. Unlike Sukhbaatar et al. [2024], instead of using the original FFN of the seed model as one of the routed experts in an MoE layer, we use it as the "shared expert" FFN_s [Rajbhandari et al., 2022, Dai et al., 2024] to better preserve the previous capabilities in the MoE model. For all non-FFN parameters including the attention weights, we merge expert parameters using simple weight averaging:

$$\begin{aligned} \text{FFN}_{moe} &= \text{FFN}_s + [\text{FFN}e_1, \text{FFN}e_2, ..., \text{FFN}e_n] && \text{(MoE Layer FFNs)} \\ \phi_{moe} &= \frac{\sum_{i=1}^{n} \phi_i}{n} && \text{(Merge Non-FFN params.)} \end{aligned}$$

Efficient adaptation to new domains. An important advantage of method is that when a new data domain is present after MoE training, we use the learned projection P_r to compute expert embedding of the new domain as $e_{new} = P_r(d_{new})$. This enables to enhance the trained MoE model with additional dense experts, which are trained in the same way as the initial experts. The FFN parameters of the new expert are simply appended to the array of existing experts.

To adequately preserve the non-FFN parameters of existing experts, we perform a weighted average $\phi_f = (1 - \lambda) \cdot \phi_{moe} + \lambda \cdot \phi_{new}$ where ϕ_f, ϕ_e, and ϕ_{moe} are parameters of the final MoE, dense

[1]We used Cohere Embed v3 [Cohere, 2023] as an external embedding model to compute domain embeddings based on individual data sources. However, similar to [Gururangan et al., 2023], pre-training data can also be clustered and the centroids can be used for domain embeddings.

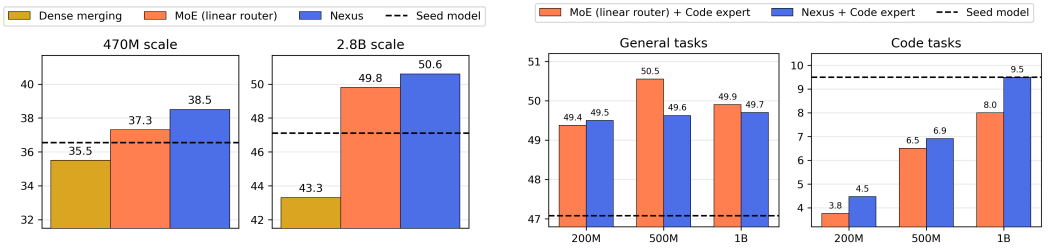

Figure 2: **(a)** Downstream performance at 470M and 2.8B scales. We report the average performance on Knowledge, Science, Reasoning, and MMLU. **(b)** Extending upcycled MoE models with the Code experts. After initial upcycling, we extended `Nexus` and `MoE(linear router)` using an independently trained dense Code expert and finetuned the resulting models small number of tokens (200M, 500M, and 1B finetuning tokens) as described in 2.

expert, and initial MoE model and $\lambda = 1/(n+1)$. This enables *efficiently adapting* Nexus to new domain by extending it with the new dense expert trained independently. After extending the MoE with a new expert, we perform a lightweight finetuning with a limited number of tokens.

3 Experiments

Our experimental setup includes 3 phases: **1) Training specialized expert LMs**, where we train separate dense models on different data domains, **2) MoE training**, where we combine those dense models into an MoE model and train it on a mix of all domains, and **3) Extending the MoE model with new experts**, where after phase 2, we add a new dense expert to the model to demonstrate how it can adapt to new data. More details about the seed model, data mix, and training procedure can be found in Appendix B. Figure 1 shows the architecture of Nexus and how a new expert can be added.

We compare our experiments against **1) Dense Merging** where all separately pre-trained experts and the seed model merged into a dense Transformer via equal weight averaging, and **2) MoE (Linear Router)** which is an MoE with a standard linear router that is upcycled from dense experts, to evaluate Nexus's novel router for upcycling. For a fair comparison, we also train this MoE model on the same datasets and for the same number of tokens as our method, and use the same architectural modifications such as shared experts.

For the downstream evaluation, we measure the performance of each model on 15 tasks from five evaluation categories (**Knowledge**, **Science**, **Reasoning**, **General Language Understanding**, and **Code**) that reflect different capabilities based on the tasks and the datasets used in the benchmarks. Appendix C details the evaluation datasets for each category.

4 Results and Discussion

4.1 Main Results for Upcycled Models

We first compare `Nexus` to the upcycled baselines `MoE with linear router` and `dense merging`. Here, we ask "*How does our MoE upcycling recipe with adaptive routing compare against baseline upcycling approaches?*"

470M parameter seed model. Table 2 (Appendix D) shows performances of upcycled models including `Nexus` where a 470M seed model is used to train dense experts. Both `Nexus` and the upcycled `MoE(linear router)` consist of 1 shared and 6 routed experts, corresponding to a total number of 1.3B parameters where 605M parameters are activated per input for top-2 routing (1 expert always activated, 1 chosen by the router). The `dense merging` baseline is created by averaging the weights of all dense experts and the seed model, and therefore has the same number of parameters as the seed model.

Compared to the `seed model`, `Nexus` performs better in all evaluation categories with a 5.8% relative gain on average (38.5 vs 36.4). Compared to upcycled models, `Nexus` outperforms `MoE(linear router)` in 3 out of 4 categories with 3.2% relative gain (38.5 vs 37.3) on average, and beats `dense`

merging by 8.5% overall relative increase (38.5 vs 35.5). Notably, while both upcycled MoEs outperform the seed model, dense merging underperforms on average, showing the benefits of MoE upcycling over parameter averaging.

2.8B parameter seed model. Next, we experiment by upcycling dense models with 2.7B parameters to validate if the results from the 470M seed model hold at a larger scale. Table 1 compares Nexus with MoE(linear router) and dense merging. Both Nexus and MoE(linear router) use 1 shared expert and 4 routed experts in these experiments, corresponding to 4.3B active parameters per input (top-2) out of 9.1B total parameters.

Our results show that Nexus leads to higher upcycling results compared to the baselines at the 2.8B scale, confirming the previous findings. Nexus enables a 7.4% relative gain over the seed model and outperforms the MoE(linear router) with a 1.6% relative increase (50.6 vs. 49.8). Nexus outperforms the best baseline in 3 out of 4 task categories and achieves the highest increase in *knowledge* tasks with 22.5% and 5.6% relative to seed model and MoE (linear router).

Similar to the 470M experiments, both Nexus and MoE(linear router) outperform the dense merging baseline. We relate this to potential cross-task interference between diverse specialized experts (including the seed model as an additional expert), leading to poor performance by applying a simple weight averaging.

4.2 Extending the Upcycled MoE model with a New Expert

To support fully modular and efficient training of MoEs, besides upcycling the existing expert models, it is crucial for an adaptive method to have the ability to continuously extend the upcycled MoE with new experts trained using previously unseen data domains. To evaluate this, we train a dense CODE expert and extend the upcycled MoEs (both Nexus and MoE(linear router)) as described in Section 2. We perform a small-scale finetuning of up to 1B tokens after extending the models. Figure 2b shows both the general performance and the target code performance at 200M, 500M, and 1B finetuning tokens. Here, we ask "*Can we continuously upcycle dense models into an MoE without requiring large-scale MoE training each time?*"

Performance on the new domain. As shown in Figure 2b (right), Nexus outperforms the MoE(linear router) for 200M, 500M and 1B finetuning tokens with 18.4%, 6.2% and 18.8% relative gains respectively. Unlike MoE(linear router), where the router weights are reset after extending the MoE layers, Nexus uses the information that is available about the new domain by mapping the domain embedding to a new expert embedding for the router, and therefore finetunes the router weights without a restart.

Comparison with the dense models. Nexus reaches the code performance of the seed model while retaining superior performance on general tasks. In comparison to the seed model and the dense code expert (trained for 8B code-only tokens on top of the seed model), although the dense code expert still performs higher than both upcycled MoEs with a score of 14.3, its performance on general tasks is far inferior (42.1). Our method also achieves up to 18.8% relative gains over the MoE(linear router). These results show that with a fraction of the original upcycling budget (1B vs 40B tokens for initial upcycling, and 1B vs 8B tokens for code expert training), Nexus can acquire a new capability.

Performance on general tasks. As a proxy for the knowledge for previously learned domains, Figure 2b (left) shows the average performance of Nexus and MoE(linear router) in general tasks. Although there is a slight drop on the general tasks for Nexus compared to initial upcycling (a relative decrease of 1.9%), the competitive performance is maintained across different numbers of finetuning tokens. We relate this to the composition of the finetuning mix where we use a high percentage of the code data (50% of the code and 50% of the previous domains).

5 Conclusion

We propose Nexus, a new LLM framework that enables efficient upcycling of specialized dense experts into a sparsely activated MoE model. We show that individual experts in our method retain their specialization after upcycling, and that our router based on expert embeddings outperforms previous approaches for combining dense experts. Furthermore, the model can be extended efficiently with new dense experts after the initial training phase, saving much compute compared to re-training the upcycled model or training from scratch.

References

R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, E. Chu, J. H. Clark, L. E. Shafey, Y. Huang, K. Meier-Hellstern, G. Mishra, E. Moreira, M. Omernick, K. Robinson, S. Ruder, Y. Tay, K. Xiao, Y. Xu, Y. Zhang, G. H. Abrego, J. Ahn, J. Austin, P. Barham, J. Botha, J. Bradbury, S. Brahma, K. Brooks, M. Catasta, Y. Cheng, C. Cherry, C. A. Choquette-Choo, A. Chowdhery, C. Crepy, S. Dave, M. Dehghani, S. Dev, J. Devlin, M. Díaz, N. Du, E. Dyer, V. Feinberg, F. Feng, V. Fienber, M. Freitag, X. Garcia, S. Gehrmann, L. Gonzalez, G. Gur-Ari, S. Hand, H. Hashemi, L. Hou, J. Howland, A. Hu, J. Hui, J. Hurwitz, M. Isard, A. Ittycheriah, M. Jagielski, W. Jia, K. Kenealy, M. Krikun, S. Kudugunta, C. Lan, K. Lee, B. Lee, E. Li, M. Li, W. Li, Y. Li, J. Li, H. Lim, H. Lin, Z. Liu, F. Liu, M. Maggioni, A. Mahendru, J. Maynez, V. Misra, M. Moussalem, Z. Nado, J. Nham, E. Ni, A. Nystrom, A. Parrish, M. Pellat, M. Polacek, A. Polozov, R. Pope, S. Qiao, E. Reif, B. Richter, P. Riley, A. C. Ros, A. Roy, B. Saeta, R. Samuel, R. Shelby, A. Slone, D. Smilkov, D. R. So, D. Sohn, S. Tokumine, D. Valter, V. Vasudevan, K. Vodrahalli, X. Wang, P. Wang, Z. Wang, T. Wang, J. Wieting, Y. Wu, K. Xu, Y. Xu, L. Xue, P. Yin, J. Yu, Q. Zhang, S. Zheng, C. Zheng, W. Zhou, D. Zhou, S. Petrov, and Y. Wu. Palm 2 technical report, 2023.

J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program synthesis with large language models, 2021.

Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models, 2022. URL `https://arxiv.org/abs/2108.07258`.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. 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. M. 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, 2020.

A. Canziani, A. Paszke, and E. Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv e-prints*, page arXiv:1605.07678, May 2016.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021. URL `https://arxiv.org/abs/2107.03374`.

E. Choi, H. He, M. Iyyer, M. Yatskar, W.-t. Yih, Y. Choi, P. Liang, and L. Zettlemoyer. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*, 2018.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Cohere. Introducing cohere embed v3, 2023. URL https://cohere.com/blog/introducing-embed-v3.

D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

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. 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, 2022.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2022.

T. Gale, D. Narayanan, C. Young, and M. Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems*, 5:288–304, 2023.

S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.740. URL https://aclanthology.org/2020.acl-main.740.

S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020b.

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

S. Gururangan, M. Li, M. Lewis, W. Shi, T. Althoff, N. A. Smith, and L. Zettlemoyer. Scaling expert language models with unsupervised domain discovery, 2023. URL https://arxiv.org/abs/2303.14177.

H. Hazimeh, Z. Zhao, A. Chowdhery, M. Sathiamoorthy, Y. Chen, R. Mazumder, L. Hong, and E. H. Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning, 2021.

D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

S. Hooker. On the limitations of compute thresholds as a governance strategy, 2024. URL https://arxiv.org/abs/2407.05694.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021.

J. Jang, S. Ye, S. Yang, J. Shin, J. Han, G. Kim, S. J. Choi, and M. Seo. Towards continual knowledge learning of language models, 2022. URL https://arxiv.org/abs/2110.03215.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

X. Jin, B. Y. Lin, M. Rostami, and X. Ren. Learn continually, generalize rapidly: Lifelong knowledge accumulation for few-shot learning, 2022. URL https://arxiv.org/abs/2104.08808.

M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

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, 2020.

A. Komatsuzaki, J. Puigcerver, J. Lee-Thorp, C. R. Ruiz, B. Mustafa, J. Ainslie, Y. Tay, M. Dehghani, and N. Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints, 2023.

T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, M. Kelcey, J. Devlin, K. Lee, K. N. Toutanova, L. Jones, M.-W. Chang, A. Dai, J. Uszkoreit, Q. Le, and S. Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/lewis21a.html.

M. Li, S. Gururangan, T. Dettmers, M. Lewis, T. Althoff, N. A. Smith, and L. Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models, 2022. URL https://arxiv.org/abs/2208.03306.

R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder: may the source be with you!, 2023.

A. Matton, T. Sherborne, D. Aumiller, E. Tommasone, M. Alizadeh, J. He, R. Ma, M. Voisin, E. Gilsenan-McMahon, and M. Gallé. On leakage of code generation evaluation datasets. *arXiv preprint arXiv:2407.07565*, 2024.

T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260.

M. Muqeeth, H. Liu, Y. Liu, and C. Raffel. Learning to route among specialized experts for zero-shot generalization. *arXiv preprint arXiv:2402.05859*, 2024.

O. Ostapenko, Z. Su, E. M. Ponti, L. Charlin, N. L. Roux, M. Pereira, L. Caccia, and A. Sordoni. Towards modular llms by building and reusing a library of loras. *arXiv preprint arXiv:2405.11157*, 2024.

J. Parmar, S. Prabhumoye, J. Jennings, M. Patwary, S. Subramanian, D. Su, C. Zhu, D. Narayanan, A. Jhunjhunwala, A. Dattagupta, V. Jawa, J. Liu, A. Mahabaleshwarkar, O. Nitski, A. Brundyn, J. Maki, M. Martinez, J. You, J. Kamalu, P. LeGresley, D. Fridman, J. Casper, A. Aithal, O. Kuchaiev, M. Shoeybi, J. Cohen, and B. Catanzaro. Nemotron-4 15b technical report, 2024. URL https://arxiv.org/abs/2402.16819.

L. Pozzobon, B. Ermis, P. Lewis, and S. Hooker. Goodtriever: Adaptive toxicity mitigation with retrieval-augmented models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5108–5125, Singapore, Dec. 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.339. URL https://aclanthology.org/2023.findings-emnlp.339.

L. Pozzobon, B. Ermis, P. Lewis, and S. Hooker. Goodtriever: Adaptive toxicity mitigation with retrieval-augmented models, 2023b. URL https://arxiv.org/abs/2310.07589.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners, 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. van den Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. de Masson d'Autume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. de Las Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, and G. Irving. Scaling Language Models: Methods, Analysis & Insights from Training Gopher, 2021.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/rajbhandari22a.html.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34:8583–8595, 2021.

S. Roller, S. Sukhbaatar, A. Szlam, and J. Weston. Hash layers for large sparse models, 2021.

K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019.

M. Sap, H. Rashkin, D. Chen, R. LeBras, and Y. Choi. Socialiqa: Commonsense reasoning about social interactions, 2019. URL https://arxiv.org/abs/1904.09728.

N. Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

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, 2017.

N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. Hechtman. Mesh-tensorflow: Deep learning for supercomputers, 2018.

Y. Shen, Z. Zhang, T. Cao, S. Tan, Z. Chen, and C. Gan. Moduleformer: Modularity emerges from mixture-of-experts. *arXiv e-prints*, pages arXiv–2306, 2023.

C. Silvey. Speaking our minds: Why human communication is different, and how language evolved to make it special, by thom scott-phillips, 2016.

D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, and N. Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp, 2019. URL https://arxiv.org/abs/1906.02243.

S. Sukhbaatar, O. Golovneva, V. Sharma, H. Xu, X. V. Lin, B. Rozière, J. Kahn, D. Li, W.-t. Yih, J. Weston, et al. Branch-train-mix: Mixing expert llms into a mixture-of-experts llm. *arXiv preprint arXiv:2403.07816*, 2024.

A. Talmor, J. Herzig, N. Lourie, and J. Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *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 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

M. Treviso, J.-U. Lee, T. Ji, B. v. Aken, Q. Cao, M. R. Ciosici, M. Hassid, K. Heafield, S. Hooker, C. Raffel, P. H. Martins, A. F. T. Martins, J. Z. Forde, P. Milder, E. Simpson, N. Slonim, J. Dodge, E. Strubell, N. Balasubramanian, L. Derczynski, I. Gurevych, and R. Schwartz. Efficient Methods for Natural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics*, 11:826–860, 07 2023. ISSN 2307-387X. doi: 10.1162/tacl_a_00577. URL https://doi.org/10.1162/tacl_a_00577.

B. Wang. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

T. Zadouri, A. Üstün, A. Ahmadian, B. Ermiş, A. Locatelli, and S. Hooker. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning, 2023. URL https://arxiv.org/abs/2309.05444.

T. Zadouri, A. Üstün, A. Ahmadian, B. Ermis, A. Locatelli, and S. Hooker. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EvDeiLv7qc.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence?, 2019.

Q. Zhang, N. Gritsch, D. Gnaneshwar, S. Guo, D. Cairuz, B. Venkitesh, J. Foerster, P. Blunsom, S. Ruder, A. Ustun, and A. Locatelli. Bam! just like that: Simple and efficient parameter upcycling for mixture of experts, 2024. URL https://arxiv.org/abs/2408.08274.

Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon. Mixture-of-experts with expert choice routing, 2022.

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, 2022.

S. Zuo, X. Liu, J. Jiao, Y. J. Kim, H. Hassan, R. Zhang, T. Zhao, and J. Gao. Taming sparsely activated transformer with stochastic experts, 2022.

A Nexus routing algorithm

```
def router(self, inputs, domain_embeddings):
    # domain_to_expert_ffn learns projection domain to expert
    embeddings
    # domain_embeddings: [e_dim x n_experts]
    # expert_embeddings: [h_dim x n_experts]
    expert_embeddings = self.domain_to_expert_ffn(self.
    domain_embeddings)

    # router probs: [batch, seq, n_experts]
    router_probs = nn.softmax(inputs @ expert_embeddings)

    # Top-1 gate for routed experts
    index, gate = nn.topk(1, router_probs)

    # routed_experts_ffns: An MoE layer with FFN experts
    # routed_expert_out: [batch, seq, h_dim]
    # shared_expert_out: [batch, seq, h_dim]
    routed_expert_out = self.routed_expert_ffns[index](input)
    shared_expert_out = self.shared_expert_ffn(input)

    return shared_expert_out + gate * routed_expert_out
```

Figure 3: **Router layer in Nexus:** PyTorch-like pseudo-code illustrating a router layer, which consists of a 2-layer MLP network (`domain_to_expert_ffn`) to project domain embeddings to expert embeddings, shared and routed expert FFNs, and sparse Top-k gating. Note that the expert embeddings are independent of the input and could be precomputed once and stored during inference.

B Experimental setup details

1. Training specialized expert LMs. For training the dense specialized experts, we use the sub-datasets from the SlimPajama dataset [Soboleva et al., 2023], a 627B token English-language corpus assembled from web data of various sources. We initialize four dense experts from the weights of the seed model and train them on the ARXIV, BOOKS, C4, GITHUB, STACKEXCHANGE, and WIKIPEDIA domains.[2] As the seed model, we use a 470M and 2.8B parameters decoder-only autoregressive Transformer models [Radford et al., 2019] that are trained with a standard language modeling objective for 750B tokens. We train dense experts for 25 and 40 billion tokens for 470M and 2.8B seed models respectively. We use parallel attention layers, [Anil et al., 2023, Wang, 2021], SwiGLU activation [Shazeer, 2020], no biases in dense layers, and a byte-pair-encoding (BPE) tokenizer with a vocabulary size of 256,000. During training, we use a linear warmup (10% of total steps) to a maximum learning rate of 1e-3 and a cosine decay schedule to 3e-4.

2. MoE training. After the training of dense expert models, we merge them into a unified MoE by appending their FFNs along a new dimension to create an MoE layer per Transformer block. For the *shared expert* in our MoE layer, we use the original FFN layer of the seed model to better preserve the previous capabilities in the MoE model. For all non-FFN parameters including the attention weights, we merge expert parameters using simple weight averaging, following Sukhbaatar et al. [2024]. After the MoE model is created, we continually train it for an additional 25B and 40B tokens respectively for the 470M and 2.8B experiments, on a mix of all domain and original pre-training datasets, using the same training hyperparameters as in the single expert training. Finally, we train the MoE models using an additional 1B tokens by upweighting the original pre-training dataset as it includes high-quality data sources such as instruction-style datasets using a cosine learning rate decay to 3e-5 [Parmar et al., 2024].

3. Extending the MoE model with new experts. After adding a new expert as defined in Section 2, we finetune the extended MoE model for up to 1 billion tokens using a uniformly sampled data mix consisting of 50% the previous domains and pre-training data and 50% the new domain. For the

[2]We exclude the Github and StackExchange datasets from SlimPajama in order to ablate adding a new expert model using the CODE domain

	Know.	Science	Reason.	MMLU	Code (excl. in upcyc.)	Avg. (w/o Code)
SEED MODEL (2.8B)	27.1	62.0	**63.8**	35.4	**8.4**	47.1
Upcycled Models						
DENSE MERGING	17.6	60.3	59.2	36.0	3.4	43.3
MOE (LINEAR ROUTER)	31.5	66.5	62.9	38.6	2.6	49.8
NEXUS	**33.2**	**67.3**	62.6	**39.4**	2.7	**50.6**

Table 1: **Downstream task results for Nexus with a 2.8B parameter seed model:** Our approach outperforms the baselines in 3 out of 4 evaluation categories. `Dense merging` corresponds a dense model with 2.8B parameters, while both `Nexus` and `MoE (linear router)` have 4.3B active and 9.1B total parameters. Note that the trained models show severe forgetting on Code benchmarks, as we exclude Code data on purpose during the upcycling phase to simulate extending models with a new dataset in Section 4.2.

new expert (CODE), we train a dense model using code documents from StarCoder [Li et al., 2023] with the same settings as for the training of the initial experts. As the 470M scale MoE did not have sufficient instruction following capabilities to attempt the code benchmarks, we only tested extending the MoEs with a new expert on the 2.8B scale.

C Evaluation details

For the downstream evaluation, we measure the performance of each model on 15 tasks[3] from five evaluation categories that reflect different capabilities based on the tasks and the datasets used in the benchmarks:

- **Knowledge**: To measure question-answering capabilities based on world knowledge and web documents such as Wikipedia, we report the performance on OpenBookQA [Mihaylov et al., 2018], Natural Questions [Kwiatkowski et al., 2019], TriviaQA [Joshi et al., 2017], QUAC [Choi et al., 2018] (all 0-shot) and SQuAD (4-shot) [Rajpurkar et al., 2016].
- **Science**: For measuring knowledge in science-oriented academic benchmarks, we use ARC-Easy, ARC-Challenge [Clark et al., 2018], SciQ [Welbl et al., 2017] (all 0-shot).
- **Reasoning**: For reasoning abilities, we use CommonSenseQA [Talmor et al., 2019], SIQA [Sap et al., 2019], PIQA [Bisk et al., 2020], WinoGrande [Sakaguchi et al., 2019], and HellaSwag [Zellers et al., 2019] (all 0-shot).
- **General Language Understanding**: We use MMLU (5-shot) [Hendrycks et al., 2021] to test general language understanding.
- **Code**: For code generation, we evaluate models on MBPP [Austin et al., 2021], LBPP [Matton et al., 2024] and HumanEval-Pack [Chen et al., 2021] that includes Cpp, Javascript, Java, Go, Python, and Rust (all 0-shot).

D Results for the 470M parameter model

Table 1 and 2 show detailed results for 2B and 470M parameter models respectively.

E Expert Specialization

To measure the specialization in our MoE, we take a closer look at how the MoE experts are activated for samples of separate domains. We compute average routing frequencies across all Transformer layers in Figure 4, where the labels on the x-axis represent which domain the tokens are coming from, and the colored bars show the routing frequencies for each of the experts trained on one of the domains. Since we select only one routed expert per token in each MoE layer, and expert

[3]We did not include ARC-Challenge and Natural Questions in 470M experiments as some model variants were unable to achieve non-random performance.

	Know.	Science	Reason.	MMLU	Avg.
SEED MODEL (470M)	14.0	51.4	50.5	29.8	36.4
Upcycled Models					
DENSE MERGING	10.9	52.0	50.3	27.8	35.5
MOE (LINEAR ROUTER)	13.4	**55.0**	51.3	29.6	37.3
NEXUS	**16.7**	**55.0**	**52.3**	**29.8**	**38.5**

Table 2: **Downstream task results for Nexus with a 470M parameter seed model:** Our approach outperforms baselines in all downstream benchmarks. `Dense merging` corresponds a dense model with 470M parameters, while both `Nexus` and `MoE (linear router)` consist of 605M active and 1.3B total parameters.

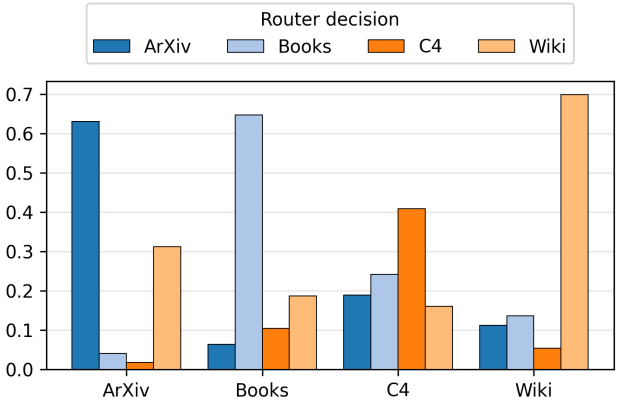

Figure 4: **Average routing probabilities for each expert per domain in Nexus:** We compute the average routing probabilities across Transformer blocks for 512 samples per domain (from the 2.8B experiment). The x-axis denotes the samples' domain and the colored bars show the routing probabilities for the corresponding expert. We show the domains that are used to train specialized experts.

FFN layers are inherited from dense experts, average routing frequencies present a good proxy for specialization of each of the experts. Here, we ask "*can Nexus retain a high degree of specialization after upcycling?*"

Routing for the upcycled experts. As shown in Figure 4, we find that the expert trained on the corresponding domain always receives the highest share of the tokens from that domain, confirming that `Nexus` retains the specialization from the specialized dense models. Concretely, this specialization is higher for ArXiv, Books, and Wikipedia with 63.0%, 64.7%, and 69.8% respectively. Interestingly, tokens from C4 are routed only 40.9% of the time to the C4 expert and distributed to the other experts approximately 20% for each one. We relate this to the broad coverage of the C4 dataset, which potentially includes samples closer to other domains and also a large percentage of the C4 used in the MoE training phase (proportional to its size in the SlimPjama dataset). Especially the latter factor pushes tokens from C4 to be distributed to the other experts due to the load balancing factor.

Specialized routing for the new expert. Next, we measure expert specialization for the newly added expert on the new code domain. Figure 5 shows the average routing probability per expert for sampled code tokens. We compute routing probabilities on the `Nexus` model with the code expert after 1B finetuning tokens (See Section 4.2 for details). Here, we see clearly that code tokens are routed to the code expert 69.1% of the time on average. This shows that Nexus not only retains the specialization for the initial upcycling but also exhibits a high degree of specialization for a newly added expert for its own domain.

F Ablations

Mixture-of-expert models are known to be sensitive to the choice of load balancing loss factor [Fedus et al., 2022, Zoph et al., 2022] and sampling weights for each data domains during training. As additional ablations, we run two new sets of experiments at 470M scale, one with a lower load balancing factor and the other one with equal weighting of each domain during training (whereas originally the weights were proportional to the share of tokens of that domain in SlimPajama). Figure 6 compares `Nexus` and `MoE(linear router)` in terms of their downstream performances for these

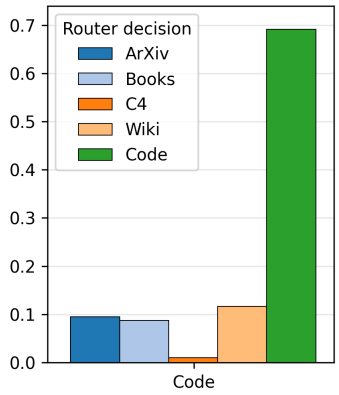

Figure 5: **Average routing probabilities per expert for the new domain in extended Nexus:** We show the routing probabilities for code tokens after extending MoE (1B finetuning).

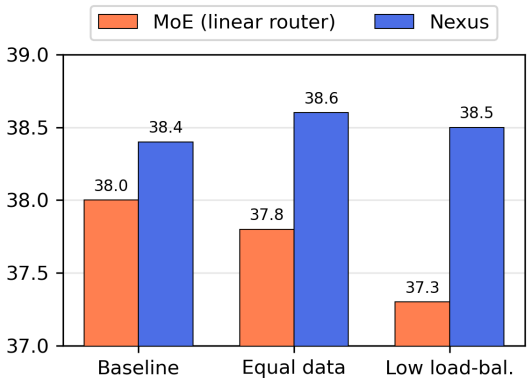

Figure 6: **Comparison between Nexus and the baseline in different load balancing and data sampling setups:** We compare Nexus and MoE (linear router) by lowering load balancing loss factor and uniformly sampling the data domain during training in isolation.[4]

ablations. Finally, in this section, we also visualize domain and projected expert embeddings to see if the relationship between embeddings is preserved after the learned projection.

Lowering the load balancing loss factor. In Figure 6 (baseline vs low load-bal.), we compare two `Nexus` models with the corresponding `MoE(linear router)` baselines where we use load balancing loss factor of 0.05 and 0.0005 for each set of experiments. We find that using a significantly lower factor for the load balancing loss hurts `MoE(linear router)` performance by approximately 2% relative drop while `Nexus` shows a robust performance across both load balancing factors. We hypothesize that because the expert embeddings in our router are always based on the domain representations, we achieve more stable distribution of tokens even if the load balancing loss is weighted extremely low.

Changing the training data composition. Next, we compare our default of sampling specialized domain data proportional to the size of the domain (total amount of tokens in SlimPajama), with a uniform sampling over all domains. Figure 6 (baseline vs equal data) shows the downstream performances for both `Nexus` and `MoE(linear router)`. Although sampling uniform sampling domains' data does not significantly impact the downstream performance for both models, we find that it helps `Nexus` to improve specialization for all the domains in terms of expert routing probabilities (Figure 8, Appendix H). In particular, compared to the size proportional sampling, tokens from the C4 domain are routed more accurately (27.6% vs 71.1%) when data is equally sampled.

Domain embeddings before and after projection. Finally, in Figure 7, we visualize cosine similarities between domains and the projected expert embeddings from the last Transformer block, in our main upcycling experiments at the 470M scale. Comparing the embeddings before and after mapping, we find that the router's learned projection preserves the main relationship between domains. For instance, relatively high cosine similarity between Books & C4, and StackExchange & GitHub exist both between their domain embeddings and the projected expert embeddings. Interestingly, while preserving the main relationships, we also find that the learned projection pushes expert embeddings further away from each other, potentially due to our choice of only activating a single expert per token besides the shared expert.

[4]We report the average performance on Knowledge, Science, Reasoning, and MMLU.

G Comparison of domain embeddings and expert embeddings

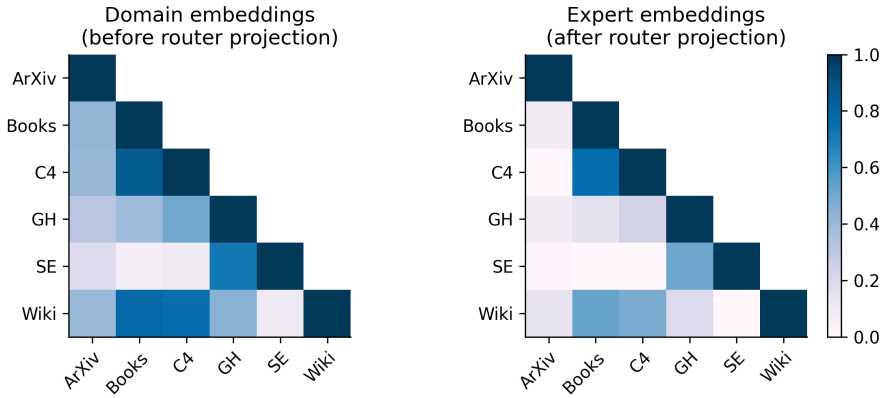

Figure 7: **Domain and the projected expert embeddings for Nexus:** We visualize cosine similarities between domains and the projected expert embeddings from the last Transformer block that are obtained in 470M experiments. Our projected router maintains the relative similarity between the original domains (e.g. Books & C4, Github & StackExchange) after the router's projection.

H Routing Probabilities for Upcycling Ablations

Figure 8 shows the expert routing probabilities for Nexus for all three settings described in Section F.

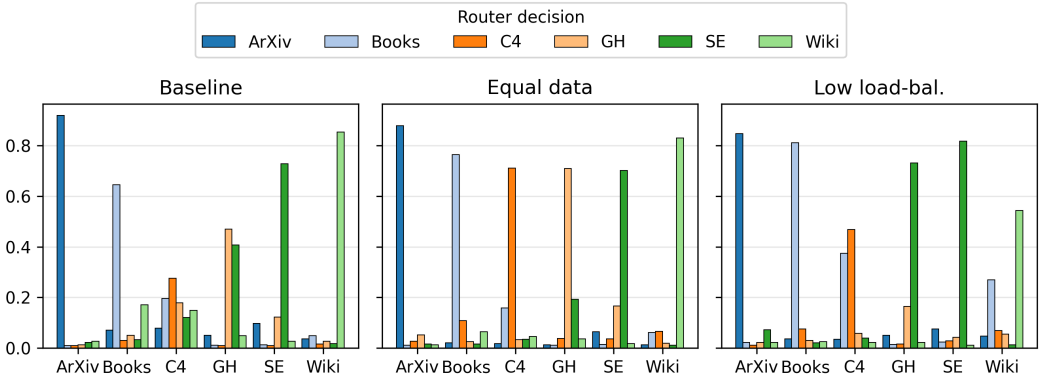

Figure 8: **Average routing probabilities for each expert per domain in different upcycling setting**: We show expert routing probabilities for Nexus for all three settings described in Section F.

I Related Work

Routing Variants of MoEs. The most common MoE architecture [Shazeer et al., 2017, Lepikhin et al., 2020, Fedus et al., 2022] employs a linear router with a top-k routing scheme, where k typically equals 1 or 2. In this standard routing schema, only the k experts with the highest router gate values are activated. There is substantial research proposing alternatives to top-k expert assignments [Hazimeh et al., 2021, Lewis et al., 2021, Roller et al., 2021, Zhou et al., 2022, Zuo et al., 2022]. DeepSeek-MoE [Dai et al., 2024] introduces a routing variant where a number of experts are "*shared*" and always assigned to all tokens. Our work also adopts this approach for our general base expert. However, these efforts primarily focus on improving the general performance and/or training stability of MoEs. In contrast, our work puts emphasis adaptability and extensibility.

	MoE (Vanilla)	BTM (Merge)	BTX (Linear router)	NEXUS (Ours)
Dense experts are trained independently (upcycling)	✗	✔	✔	✔
Experts are specialized in different domains	✗	✔	✔	✔
Experts are chosen by a learned router per input token	✔	✗	✔	✔
Router is adaptive via learned projection for new domains	✗	✗	✗	✔

Table 3: **A comparison of existing approaches with Nexus:** Unlike the vanilla MoE architecture [Shazeer et al., 2017, Fedus et al., 2022] the Branch-Train-Merge [BTM; Li et al., 2022] and the Branch-Train-Mix [BTX; Sukhbaatar et al., 2024] approaches train experts separately in different domains, reducing the training cost and improving specialization. However, they either merge the experts during inference or learn an MoE router layer from scratch, where prior domain information is not used. Our approach trains the MoE router based on domain information, maintaining the specialization and enabling efficient extension of the MoE with a new expert after training.

Efficient MoE Training by Re-Using Existing Dense Models. Training MoEs from scratch is computationally expensive [Gale et al., 2023, Fedus et al., 2022] and often challenging due to training instabilities [Zoph et al., 2022]. Alternatively, recent works have explored re-using existing dense models to initialize MoEs. Sparse Upcycling [Komatsuzaki et al., 2023] re-uses a single dense model to initialize the MoE by replicating the FFN weights in an MoE layer. The router is initialized randomly, and all other parameters are copied directly from the dense model. BTX [Sukhbaatar et al., 2024] extends this approach by upcycling not from a single dense model, but from multiple specialized dense expert models. Furthermore, BAM [Zhang et al., 2024] expands BTX to upcycle not only FFN experts but also attention experts. Our work also leverages this approach by reusing specialized dense experts for an MoE, while extending it further to facilitate on-the-fly adaptations for new experts specialized in unseen data domains.

Efficient MoE Architectures. Zadouri et al. [2024] proposes replacing traditional MoE's computation-heavy feed-forward network (FFN) experts with more efficient experts comprised of smaller vectors and adapters, which are activated in parallel to a single dense FFN. This lightweight architecture necessitates only a limited number of parameter updates when finetuning, offering efficiency advantages. However, unlike our approach, it does not leverage existing specialized dense models and lacks a notion of specialized experts, which are central to our method. Similar to our work, Muqeeth et al. [2024] and Ostapenko et al. [2024] study combining separately trained experts into a unified model. However, they focus on parameter-efficient adapters such as LoRA [Hu et al., 2021] and supervised finetuning. In this work, we focus on efficiently pre-training fully-fledged MoE models via upcycling.

Adaptive MoEs and Ensemble Models. ModuleFormer [Shen et al., 2023] also aims to produce adaptable MoEs. The authors achieve adaptability by freezing existing MoE parameters while only training newly added modules with optimization constraints to the router. Unlike our work, ModuleFormer does not leverage existing expert dense seed models for efficiency gains, nor does it have a notion of specialization which is central to our work. Similar to our work, DEMix [Gururangan et al., 2021] independently trains different FFN experts on specialized data domains, with each expert functioning as a domain-specific module. Modules can be added on-the-fly for adaptability. Followup works BTM and C-BTM [Li et al., 2022, Gururangan et al., 2023] extend DEMix to create adaptive ensemble models. However, all three works use a router requiring a forward pass for every expert at inference instead of sparsely activating them, which significantly increases inference costs, especially with a large number of experts. Unlike these approaches, our router cost is approximately the same as standard top-k routing during both training and inference, offering a more scalable solution for adaptability.